ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΛΟΠΟΝΝΗΣΟΥ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ

ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

"Ανάπτυξη front-end web εφαρμογών με React JS και Angular JS"

ΝΙΚΟΛΑΟΣ ΣΤΑΥΡΟΠΟΥΛΟΣ (2564)

ΑΛΕΞΑΝΔΡΟΣ ΚΙΑΜΟΣ (2470)

ΕΠΙΒΛΕΠΩΝ: ΧΡΙΣΤΟΔΟΥΛΟΥ ΣΩΤΗΡΙΟΣ

ΠΑΤΡΑ 2023

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή

Πάτρα, Ημερομηνία

ΕΠΙΤΡΟΠΗ ΑΞΙΟΛΟΓΗΣΗΣ

1. Ονοματεπώνυμο, Υπογραφή

2. Ονοματεπώνυμο, Υπογραφή

3. Ονοματεπώνυμο, Υπογραφή

Υπεύθυνη Δήλωση Φοιτητών

Βεβαιώνουμε ότι είμαστε συγγραφείς αυτής της εργασίας και ότι κάθε βοήθεια την οποία είχαμε για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης έχουμε αναφέρει τις όποιες πηγές από τις οποίες κάναμε χρήση δεδομένων, ιδεών ή λέξεων, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επίσης βεβαιώνουμε ότι αυτή η εργασία προετοιμάστηκε από εμάς προσωπικά ειδικά για τη συγκεκριμένη εργασία.

Η έγκριση της διπλωματικής εργασίας από το Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Πανεπιστημίου Πελοποννήσου δεν υποδηλώνει απαραιτήτως και αποδοχή των απόψεων των συγγραφέων εκ μέρους του Τμήματος.

ΠΕΡΙΛΗΨΗ

Αντικείμενο της παρούσας πτυχιακής εργασίας είναι η εκμάθηση της front-end JavaScript βιβλιοθήκης React και η ανάπτυξη μιας front-end εφαρμογής ιστού για τη διαχείριση των εκπαιδευτών ενηλίκων. Πρόκειται για μία SPA εφαρμογή που θα μπορεί να χρησιμοποιηθεί ως ψηφιακή υπηρεσία στο GOV.GR με αποτέλεσμα να ενισχύσει τις υπηρεσίες του.

Με την αυξανόμενη ζήτηση για δια βίου μάθηση και επαγγελματική κατάρτιση, η αποτελεσματική διαχείριση των εκπαιδευτών ενηλίκων είναι ζωτικής σημασίας. Στόχος της εφαρμογής ιστού είναι να διευκολύνει τους διαχειριστές με τη διαχείριση των εκπαιδευτών ενηλίκων μέσα από μια φιλική και εύχρηστη διεπαφή χρήστη που ανταποκρίνεται δυναμικά σε διάφορες οθόνες και συσκευές. Οι διαχειριστές μέσω αυτής έχουν τη δυνατότητα να εισάγουν εκπαιδευτές, να προβάλλουν καθώς και να τροποποιούν τα στοιχεία των εκπαιδευτών αποτελεσματικά. Η εφαρμογή προβάλλει μία λίστα με τους συνολικούς εισαχθέντες εκπαιδευτές και διαθέτει προς χρήση προηγμένα φίλτρα όπως αναζήτηση βάσει στοιχείων, επιλογή πιστοποιημένων ή μη πιστοποιημένων εκπαιδευτών και επιλογή δήμου. Οι διαχειριστές αξιοποιώντας τα φίλτρα μπορούν να βρίσκουν συγκεκριμένους εκπαιδευτές γρήγορα, εξοικονομώντας χρόνο και φόρτο.

Για την υλοποίηση της εφαρμογής ιστού μελετήθηκαν και χρησιμοποιήθηκαν πολλές ανοιχτού κώδικα γλώσσες προγραμματισμού και τεχνολογίες ανάπτυξης εφαρμογών ιστού. Αυτές ονομαστικά είναι οι ακόλουθες: React, HTML5, CSS3, JavaScript, JSX, Bootstrap, Material UI, Σύστημα Σχεδιασμού GOV.GR, React Router, npm, Visual Studio Code, Git, GitLab, REST, JSON, Axios, mockAPI.

Λέξεις-κλειδιά: React, JavaScript frameworks, Front-end web programming, Gov.gr εφαρμογές, Διαχείριση εκπαιδευτών ενηλίκων

ABSTRACT

The subject of this thesis is the learning of the front-end JavaScript library React and the development of a front-end web application for the management of adult trainers. It is a SPA application that will can be used as a digital service at GOV.GR, thus enhancing its services.

With the increasing demand for lifelong learning and vocational training, effective management of adult trainers is vital. The aim of the web application is to facilitate administrators with the management of adult trainers through a friendly and easy-to-use user interface that responds dynamically to various screens and devices. Through it, administrators have the ability to create trainers, read as well as update trainer data effectively. The application displays a list of the total created trainers and has advanced filters for use such as search by data, selection of certified or non-certified trainers and municipality selection. Administrators utilizing the filters can find specific trainers quickly, saving time and effort.

For the implementation of the web application, many open source programming languages and web application development technologies were studied and used. These are nominally the following: React, HTML5, CSS3, JavaScript, JSX, Bootstrap, Material UI, Σύστημα Σχεδιασμού GOV.GR, React Router, npm, Visual Studio Code, Git, GitLab, REST, JSON, Axios, mockAPI.

Keywords: React, JavaScript frameworks, Front-end web programming, Gov.gr applications, Management of adult trainers

ΕΥΧΑΡΙΣΤΙΕΣ

Αρχικά, θα θέλαμε να ευχαριστήσουμε θερμά τον επιβλέποντα επίκουρο καθηγητή μας κ. Χριστοδούλου Σωτήριο για την πολύτιμη βοήθεια, υποστήριξη και καθοδήγηση που μας προσέφερε σε όλη τη διάρκεια της εκπόνησης της εργασίας αυτής. Επίσης, θα θέλαμε να ευχαριστήσουμε τον υποψήφιο διδάκτορα κ. Τόγια Κωνσταντίνο για τις σημαντικές συμβουλές και την καθοδήγηση στην ανάπυξη της εφαρμογής. Τέλος, θα θέλαμε να ευχαριστήσουμε τις οικογένειες μας που μας στήριξαν με κάθε δυνατό μέσο καθ' όλη τη διάρκεια των σπουδών μας.

Περιεχόμενα

Πίνακας εικόνων

Κεφάλαιο 1 - Εισαγωγή

1.1 Γενικό πεδίο

Ο παγκόσμιος ιστός είναι η δημοφιλέστερη και η πιο χρήσιμη υπηρεσία του διαδικτύου, η οποία αποτελείται από δισεκατομμύρια ιστοσελίδες και πολλές εφαρμογές ιστού. Οι χρήστες αποκτούν πρόσβαση σε αυτές μέσω του προγράμματος περιήγησης για να αντλήσουν πληροφορίες, να πραγματοποιήσουν ηλεκτρονικές παραγγελίες, να επικοινωνήσουν μεταξύ τους, να ψυχαγωγηθούν και να εργαστούν απομακρυσμένα. Συνεπώς, ο παγκόσμιος ιστός βοηθάει τους χρήστες στην επίλυση διάφορων προβλημάτων με την προϋπόθεση όμως ότι αξιοποιείται σωστά και υπεύθυνα.

Αρχικά, οι ιστοσελίδες που αναπτύσσονταν ήταν κυρίως στατικές, ωστόσο με την πάροδο του χρόνου και τη δημιουργία της JavaScript άρχισαν να αναπτύσσονται σε μεγάλο βαθμό δυναμικές και διαδραστικές. Η αυξημένη δημοτικότητα και η ευρεία χρήση της JavaScript οδήγησε στη δημιουργία των JavaScript frameworks, τα οποία είναι συλλογές βιβλιοθηκών κώδικα JavaScript και βοηθούν τους προγραμματιστές να αναπτύσσουν ταχύτερα και ευκολότερα επεκτάσιμες διαδραστικές εφαρμογές ιστού. Γενικά, τα τελευταία χρόνια αναπτύσσονται συνεχώς με τη χρήση των JavaScript frameworks εφαρμογές ιστού που διευκολύνουν τη διαχείριση σε διάφορους τομείς.

1.2 Παρουσίαση του προβλήματος

Ο τομέας της εκπαίδευσης ενηλίκων και επαγγελματικής κατάρτισης στην Ελλάδα είναι σημαντικός, λόγω του ότι οι ενήλικες αναζητούν ευκαιρίες συνεχούς μάθησης στα προγράμματα επαγγελματικής κατάρτισης για να αντιμετωπίσουν τις προκλήσεις της σύγχρονης αγοράς εργασίας. Ως εκ τούτου, υπάρχει αυξανόμενη ζήτηση για εκπαιδευτές ενηλίκων και η ανάγκη για την αποτελεσματική διαχείριση τους. Γενικά, οι παραδοσιακές μέθοδοι διαχείρισης των εκπαιδευτών έχουν αποδειχθεί μη αποδοτικές, χρονοβόρες και επιρρεπείς σε σφάλματα.

Για την αντιμετώπιση του παραπάνω ζητήματος, στη συγκεκριμένη πτυχιακή εργασία αναπτύσσεται με τη χρήση της JS βιβλιοθήκης React μία front-end εφαρμογή ιστού για την αποτελεσματική διαχείριση των εκπαιδευτών ενηλίκων, η οποία θα ενσωματωθεί στις

ψηφιακές υπηρεσίες του GOV.GR. Η εν λόγω εφαρμογή απευθύνεται στους διαχειριστές και στόχος της είναι να τους προσφέρει μέσω μιας εύχρηστης και responsive διεπαφής χρήστη ένα σύνολο προηγμένων λειτουργιών που διευκολύνουν και απλοποιούν τη διαχείριση των εκπαιδευτών ενηλίκων. Οι διαχειριστές μέσω αυτών μπορούν να εισάγουν νέους εκπαιδευτές, να προβάλλουν και να επεξεργάζονται τα στοιχεία των εκπαιδευτών με αποδοτικό τρόπο. Συγκεκριμένα, στη λειτουργία της εισαγωγής και επεξεργασίας εμφανίζονται κατάλληλες φόρμες, στις οποίες εφαρμόζεται επικύρωση για τη μείωση των σφαλμάτων. Η εφαρμογή παρέχει μία σελιδοποιημένη λίστα με τους συνολικούς εισαχθέντες που συνοδεύεται από προηγμένα φίλτρα (αναζήτηση με βάση συγκεκριμένα στοιχεία, επιλογή δήμου, επιλογή πιστοποιημένων ή μη πιστοποιημένων εκπαιδευτών, ταξινόμηση) για τη βελτίωση της χρηστικότητας. Αυτά τα φίλτρα δίνουν τη δυνατότητα στους διαχειριστές να αναζητούν συγκεκριμένους εκπαιδευτές εύκολα και γρήγορα. Επιπλέον, δεδομένου ότι είναι μία responsive εφαρμογή ιστού, οι διαχειριστές μπορούν να διαχειρίζονται τους εκπαιδευτές από διάφορες συσκευές, εξαλείφοντας την ανάγκη για φυσική παρουσία σε γραφεία. Τέλος, όλες οι παραπάνω δυνατότητες μπορούν να βελτιώσουν σημαντικά την παραγωγικότητα και να εξοικονομήσουν χρόνο.

Η υλοποίηση της εφαρμογής ιστού έγινε με τη χρήση πολλών γλωσσών προγραμματισμού και σύγχρονων τεχνολογιών ανάπτυξης εφαρμογών ιστού. Ειδικότερα αξιοποιήθηκε ως κύρια η JavaScript βιβλιοθήκη React. Ωστόσο, μόνη της δεν αρκεί και για αυτό το λόγο χρησιμοποιήθηκαν επιπλέον οι γλώσσες HTML5, CSS3, JavaScript, το CSS framework Bootstrap, οι βιβλιοθήκες Digigov CSS, Material UI, React Router, Axios, το πρότυπο JSON, το εργαλείο mockAPI, το πρόγραμμα επεξεργασίας πηγαίου κώδικα VS code, ο διαχειριστής πακέτων npm, το κατανεμημένο σύστημα ελέγχου εκδόσεων Git και η πλατφόρμα GitLab.

1.3 Δομή εργασίας

Η συγκεκριμένη εργασία αποτελείται από επτά κεφάλαια και το παράρτημα.

Σε αυτό το κεφάλαιο περιγράφεται το αντικείμενο της πτυχιακής εργασίας και παρουσιάζεται το πρόβλημα που επιλύει.

Στο δεύτερο κεφάλαιο παρουσιάζονται οι βασικές front-end τεχνολογίες HTML, CSS και JavaScript.

Στο τρίτο κεφάλαιο γίνεται μία εισαγωγή στα JavaScript frameworks και αναφέρονται τα πλεονεκτήματα τους. Έπειτα, πραγματοποιείται η σύγκριση των δημοφιλέστερων front-end JavaScript frameworks με κριτήρια τις δυνατότητες που παρέχουν, την απόδοση, την καμπύλη μάθησης και τη δημοτικότητα τους. Τέλος, περιγράφεται η χρησιμότητα της JavaScript βιβλιοθήκης React καθώς και οι λειτουργίες που προσφέρει.

Στο τέταρτο κεφάλαιο αναλύονται οι React φόρμες. Αρχικά, αναφέρονται μερικές διαφορές μεταξύ των React και HTML φορμών. Στη συνέχεια αναλύονται οι τρόποι διαχείρισης των δεδομένων μιας φόρμας στην React, καθώς και οι τρόποι επικύρωσης της φόρμας από την πλευρά του πελάτη. Τέλος, αναφέρονται ονομαστικά μερικές εξωτερικές βιβλιοθήκες φόρμας.

Στο πέμπτο κεφάλαιο γίνεται αναφορά στα frameworks, στις βιβλιοθήκες, στις πλατφόρμες και στα εργαλεία που αξιοποιήθηκαν για την ανάπτυξη της front-end εφαρμογής ιστού.

Στο έκτο κεφάλαιο πραγματοποιείται η ανάπτυξη της front-end εφαρμογής ιστού. Ειδικότερα, αρχικά πραγματοποιείται η ανάλυση και ο σχεδιασμός της. Στη συνέχεια, αναφέρονται όλες οι γλώσσες και οι τεχνολογίες ανάπτυξης web εφαρμογών που χρησιμοποιήθηκαν. Τέλος, πραγματοποιείται η παρουσίαση της εφαρμογής σε επιτραπέζιους υπολογιστές (desktops) και έπειτα σε κινητά τηλέφωνα (mobile phones).

Στο έβδομο κεφάλαιο αναφέρονται τα τελικά συμπεράσματα της πτυχιακής, καθώς και μερικές πιθανές μελλοντικές επεκτάσεις της εφαρμογής.

Τέλος, στο παράρτημα παρατίθεται ο συνολικός κώδικας της εφαρμογής και επεξηγείται.

Κεφάλαιο 2 – Βασικές Τεχνολογίες Front-end

Οι **front-end** προγραμματιστές ασχολούνται με την ανάπτυξη της διεπαφής χρήστη (user interface) που προβάλλουν και αλληλεπιδρούν οι χρήστες.

2.1 HTML

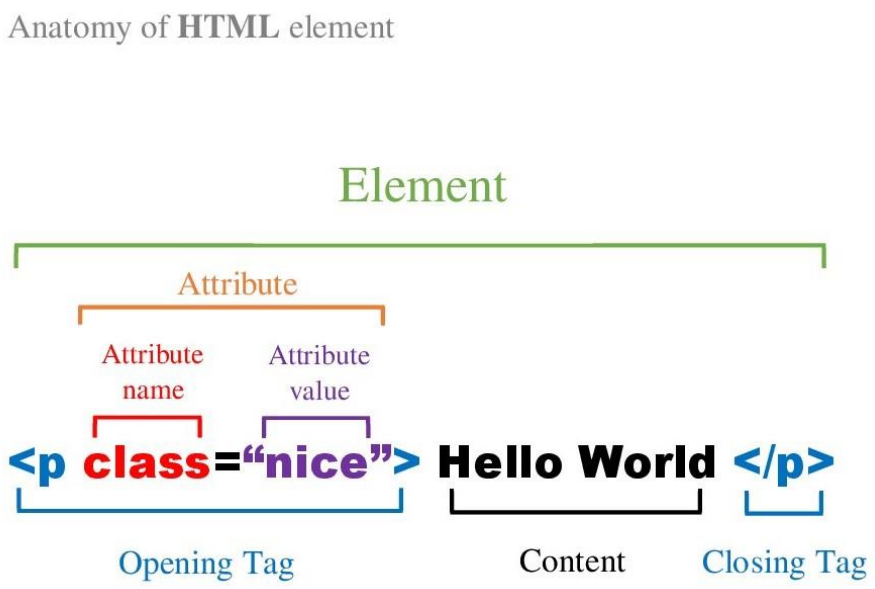
Η **HTML** (**HyperText Markup Language**, Γλώσσα Σήμανσης Υπερκειμένου) είναι η κύρια γλώσσα σήμανσης για τη δημιουργία ιστοσελίδων, η οποία καθορίζει τη δομή του περιεχομένου μιας ιστοσελίδας. Αποτελείται από μία σειρά **στοιχείων HTML** (elements), τα οποία αποτελούνται από ετικέτες HTML (tags) που περιλαμβάνονται εντός των συμβόλων "<" (μικρότερο) και ">" (μεγαλύτερο), για παράδειγμα ** ** [1] [2].

Οι **ετικέτες HTML** (tags) συνήθως γράφονται ανά ζεύγη, για παράδειγμα **<h1>** και **</h1>** όπου η πρώτη ετικέτα του ζεύγους είναι η ετικέτα έναρξης και η δεύτερη η ετικέτα λήξης. Η ετικέτα έναρξης μπορεί να περιλαμβάνει **ιδιότητες** (attributes) στοιχείου, οι οποίες παρέχουν πρόσθετες πληροφορίες για τα στοιχεία και είναι σε ζεύγη ονόματος/τιμής, για παράδειγμα **<img src="image1.jpg">**. Οι ετικέτες μπορεί να είναι είτε με κεφαλαία είτε με πεζά γράμματα (not case-sensitive). Ανάμεσα στις ετικέτες, τοποθετείται το περιεχόμενο (content) του στοιχείου, όπου είναι συνήθως κείμενο. Η HTML επιτρέπει την προσθήκη εικόνων, την ενσωμάτωση διαφόρων αντικειμένων, τη μορφοποίηση κειμένου, τη δημιουργία συνδέσμων, τη δημιουργία πινάκων καθώς και την εμφάνιση διαδραστικών φορμών.

Βασικές ετικέτες της HTML:

- **<html>…</html>**, ορίζει την αρχή και τέλος της ιστοσελίδας.
- **<head>…</head>**, ορίζει πληροφορίες (μεταδεδομένα) σχετικά με την ιστοσελίδα, οι οποίες δεν εμφανίζονται από τον περιηγητή.
- **<title>…</title>**, ορίζει τον τίτλο της ιστοσελίδας και εμφανίζεται στη γραμμή τίτλου του περιηγητή.
- **<body>…</body>**, ορίζει το περιεχόμενο της ιστοσελίδας.
- ** … **, ορίζει μια παράγραφο.

- **…**, εμφανίζει το κείμενο εντός της ετικέτας με έντονη γραφή.
- **<form>…</form>**, ορίζει μια φόρμα για συμπλήρωση στοιχείων του χρήστη.
- **<table>…</table>**, ορίζει έναν πίνακα.
- ****, ορίζει μια εικόνα.
- **…**, ορίζει έναν υπερσύνδεσμο.



Εικόνα 2.1 Ανατομία ενός HTML στοιχείου

Ο **περιηγητής ιστού** (web browser) διαβάζει τα HTML έγγραφα και τα αναπαριστά ως ιστοσελίδες. Επίσης οι ετικέτες HTML δεν εμφανίζονται από τον περιηγητή, αλλά τις χρησιμοποιεί για να παρουσιάσει το περιεχόμενο της ιστοσελίδας στον χρήστη. Τα αρχεία HTML αποθηκεύονται με επέκταση **.htm** ή .**html**.

Η HTML εφευρέθηκε από τον φυσικό Tim Berners-Lee, ο οποίος ήταν ο εφευρέτης του Παγκόσμιου Ιστού (World Wide Web). Ο Tim Berners-Lee δημιούργησε την HTML 1.0 το 1991. Η τελευταία σημαντική και μεγάλη έκδοση είναι η HTML5, η οποία δημοσιεύθηκε για πρώτη φορά το 2008 και συστάθηκε από το W3C (World Wide Web Consortium, Κοινοπραξία του Παγκόσμιου Ιστού) το 2014. Το 2017 δημοσιεύθηκε η HTML 5.2 ως σύσταση του W3C.

Εικόνα 2.2 Λογότυπο HTML5

Η **HTML5** εισήγαγε νέα χαρακτηριστικά. Οι στόχοι της ήταν να είναι κατανοητή και ευανάγνωστη από τον άνθρωπο, τον περιηγητή ιστού και να υποστηρίζει τα πολυμέσα (βίντεο, ήχος) χωρίς πρόσθετα (plugins). Προστέθηκαν νέα σημασιολογικά στοιχεία για τον διαχωρισμό των τμημάτων του εγγράφου για παράδειγμα **<header>…</header>**, νέοι τύποι στοιχείων στις φόρμες για παράδειγμα **<input type="tel"/>**, νέες ιδιότητες ετικετών και νέες δυνατότητες σχεδίασης γραφικών για παράδειγμα **<canvas>…</canvas>**. Εκτός από τη σχεδίαση γραφικών με καμβά, υποστηρίζει και τα διανυσματικά γραφικά SVG (Scalable Vector Graphics). Επίσης, η HTML5 είναι πιο φιλική προς τις κινητές συσκευές από τις προηγούμενες εκδόσεις.

2.2 CSS

Η **CSS** (**Cascading Style Sheets** – διαδοχικά φύλλα στυλ) είναι μια γλώσσα που χρησιμοποιείται για την περιγραφή της εμφάνισης και της μορφοποίησης ενός εγγράφου γραμμένου σε μια γλώσσα σήμανσης, όπως η HTML [3].

Η CSS έχει σχεδιαστεί κυρίως για τον καθορισμό στυλ στις ιστοσελίδες μας και για να διαχωρίσει το περιεχόμενο από τη μορφοποίηση, συμπεριλαμβανομένων στοιχείων που αφορούν στη διάταξη, τις γραμματοσειρές, τα χρώματα. Ο διαχωρισμός αυτός προσφέρει βελτίωση της προσβασιμότητας του περιεχομένου, ευκολότερη συντήρηση, μεγαλύτερη ευελιξία, σαφήνεια και έλεγχο.

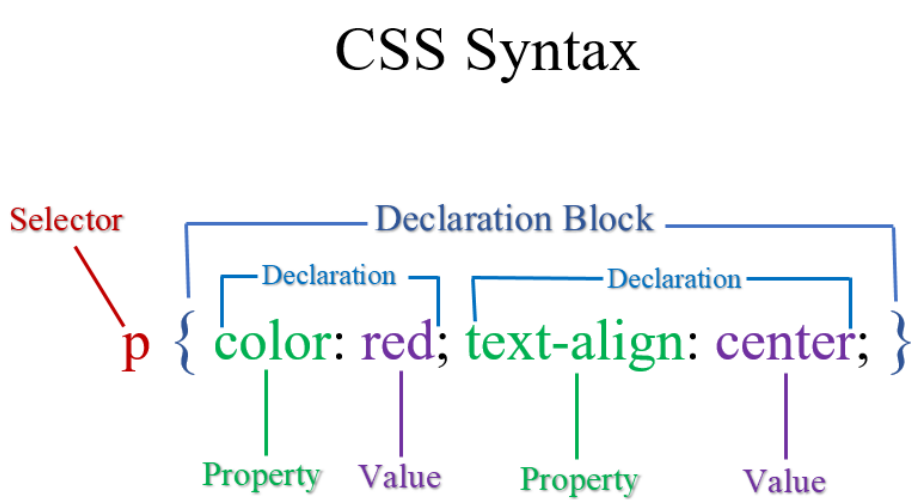
Πριν τη δημιουργία της CSS, υπήρχε ένα μεγάλο πρόβλημα. Το πρόβλημα ήταν ότι η HTML δεν προοριζόταν να περιέχει ετικέτες για τη μορφοποίηση του περιεχομένου. Έτσι, όταν προστέθηκαν ετικέτες για τη μορφοποίηση περιεχομένου στην προδιαγραφή της HTML 3.2, η ανάπτυξη ιστοσελίδων έγινε περίπλοκη. Επομένως, ο διαχωρισμός που επιτρέπει η CSS έλυσε το παραπάνω πρόβλημα [4].

Η CSS προτάθηκε το 1994 από τον Håkon Wium Lie. Η πρώτη έκδοση της είναι η CSS1, η οποία κυκλοφόρησε το 1996 ως σύσταση του W3C. Οι δυνατότητες που παρείχε η CSS1 ήταν υποστήριξη ιδιοτήτων γραμματοσειράς, καθορισμός χρώματος στοιχείων και φόντου, στοίχιση διάφορων στοιχείων και καθορισμός περιθωρίου ανάμεσα στα στοιχεία. Η έκδοση CSS2 αναπτύχθηκε και δημοσιεύθηκε από το W3C το 1998. Η CSS2 παρείχε νέες δυνατότητες όπως την τοποθέτηση στοιχείων (positioning), νέες ιδιότητες γραμματοσειράς και εισήγαγε την έννοια των media types [5].

Εικόνα 2.3 Ανεπίσημο λογότυπο CSS3

Η τελευταία έκδοση είναι η **CSS3**, όπου τα πρώτα προσχέδια δημοσιεύθηκαν το 1999. Η CSS3 διαιρείται σε τμήματα (modules), όπου κάθε τμήμα προσθέτει νέες δυνατότητες. Μερικές δυνατότητες που προστέθηκαν στην CSS3 είναι η υποστήριξη γραμματοσειρών Ιστού (Web Fonts), υποστήριξη κινούμενων σχεδίων (animations), υποστήριξη μεταβάσεων (transitions), υποστήριξη δισδιάστατων και τρισδιάστατων μετασχηματισμών (2D and 3D transforms), στρογγυλοποίηση γωνιών, νέες ιδιότητες και νέα χρώματα. Επίσης, η CSS3 εισήγαγε τα media queries, τα οποία επεκτείνουν τα media types της CSS2 και είναι μια δημοφιλής τεχνική για την κατασκευή ιστοσελίδων που προσαρμόζονται σε διάφορες συσκευές (κινητά, tablets, laptops, desktops).

Η σύνταξη της CSS είναι εύκολη. Ένα αρχείο CSS αποτελείται από ένα σύνολο κανόνων. Ένας **κανόνας** αποτελείται από τον **επιλογέα** (selector) και ένα **μπλοκ από δηλώσεις** (declaration block). Ο επιλογέας δηλώνει το στοιχείο HTML που θα μορφοποιηθεί και μέσα στο μπλοκ ορίζονται δηλώσεις, οι οποίες χωρίζονται με ερωτηματικό. Μια **δήλωση** αποτελείται από το **όνομα ιδιότητας** (property) και μια **τιμή** (value), οι οποίες χωρίζονται μεταξύ τους με άνω και κάτω τελεία [6].



Εικόνα 2.4 Σύνταξη ενός κανόνα CSS

Όπως είπαμε, οι **επιλογείς** (selectors) καθορίζουν το στοιχείο HTML που θα μορφοποιηθεί. Οι **τύποι επιλογέων** που χρησιμοποιούνται περισσότερο είναι ο **επιλογέας στοιχείου** (element selector), ο **επιλογέας κλάσης** (class selector) και ο **επιλογέας id** (id selector). Ο επιλογέας κλάσης δηλώνεται με τελεία (.) και ένα όνομα. Τα στοιχεία που θέλουμε να μορφοποιήσουμε πρέπει να έχουν το χαρακτηριστικό class ίσο με το συγκεκριμένο όνομα. Ο επιλογέας κλάσης μπορεί να εφαρμοστεί σε πολλά στοιχεία. Για παράδειγμα, ο επιλογέας κλάσης .center θα εφαρμοστεί στα στοιχεία που περιλαμβάνουν το χαρακτηριστικό class="center". Από την άλλη, ο επιλογέας id δηλώνεται με το σύμβολο hash (#) και ένα όνομα. Αντίστοιχα, τα στοιχεία που θέλουμε να μορφοποιήσουμε πρέπει να έχουν το χαρακτηριστικό id ίσο με το συγκεκριμένο όνομα. Επίσης, το id ενός στοιχείου είναι μοναδικό σε μια σελίδα. Για παράδειγμα, ο επιλογέας id #left θα εφαρμοστεί στο μοναδικό στοιχείο που περιλαμβάνει το χαρακτηριστικό id="left" [7].

Υπάρχουν τρεις τρόποι εφαρμογής κανόνων CSS [8]:

1. **Ένθετο Στυλ (Inline Style)**. Οι κανόνες CSS ορίζονται απευθείας στο αρχείο HTML. Αυτός ο τρόπος μπορεί να χρησιμοποιηθεί για την εφαρμογή ενός συγκεκριμένου στυλ σε ένα συγκεκριμένο στοιχείο HTML. Για να χρησιμοποιήσουμε αυτό τον τρόπο, πρέπει να προσθέσουμε το style χαρακτηριστικό (attribute) στο συγκεκριμένο στοιχείο που θέλουμε να μορφοποιήσουμε. Το χαρακτηριστικό style μπορεί να περιέχει πολλές ιδιότητες CSS.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Inline Style</title>
  </head>
  <body>
    <h1 style="color: red; text-align: center">Hello World</h1>
    <p style="color: green">My paragraph.</p>
  </body>
</html>
```

2. **Εσωτερικό φύλλο στυλ (Internal Style Sheet).** Οι κανόνες CSS ορίζονται απευθείας μέσα στο αρχείο HTML, μέσα στην HTML ετικέτα **<style>**, η οποία ορίζεται εντός της ετικέτας **<head>**. Αφορά το στυλ για μια σελίδα, οπότε κάθε σελίδα πρέπει να ορίζει τους δικούς της κανόνες.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Inline Style Sheet</title>
    <style>
      h1 {
        color: red;
        text-align: center;
      }
      p {
        color: green;
      }
    </style>
  </head>
  <body>
    <h1>Hello World</h1>
    <p>My paragraph.</p>
```

```
  </body>
</html>
```

3. **Εξωτερικό φύλλο στυλ (External Style Sheet).** Είναι ο πιο δημοφιλής τρόπος και συνιστάται από τους προγραμματιστές επειδή διαχωρίζει τους κανόνες CSS από την HTML σήμανση. Οι κανόνες ορίζονται σε ξεχωριστό αρχείο CSS, το οποίο αποθηκεύεται με επέκταση **.css**. Ο διαχωρισμός αυτός κάνει τον κώδικα πιο ευανάγνωστο και μπορεί να τροποποιηθεί πιο εύκολα. Για να μορφοποιηθεί μια σελίδα HTML, πρέπει να κάνει αναφορά στο εξωτερικό αρχείο CSS. Η αναφορά αυτή γίνεται μέσα στο αρχείο HTML και συγκεκριμένα ορίζουμε τη εντολή **<link rel="stylesheet" href="CssFile.css">** εντός της ετικέτας **<head>**, όπου το χαρακτηριστικό href είναι το μονοπάτι που βρίσκεται το αρχείο CSS που θέλουμε να συνδέσουμε. Επίσης, ένα αρχείο CSS μπορεί να συνδεθεί και να μορφοποιήσει πολλές σελίδες.

test.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>External Style Sheet</title>
    <link rel="stylesheet" href="test.css" />
  </head>
  <body>
    <h1>Hello World</h1>
    <p>My paragraph.</p>
  </body>
</html>
```

test.css

```
h1 {
  color: red;
  text-align: center;
}
p {
  color: green;
}
```

2.3 JavaScript

Εικόνα 2.5 Ανεπίσημο λογότυπο JavaScript

Η **JavaScript** (JS) είναι μια δημοφιλής δυναμική γλώσσα προγραμματισμού, η οποία χρησιμοποιείται για τη δημιουργία εφαρμογών ιστού (web applications) και αποτελεί μία από τις βασικές τεχνολογίες του Παγκόσμιου Ιστού. Αναφέρεται ως αντικειμενοστραφής γλώσσα προγραμματισμού που βασίζεται σε πρωτότυπα (prototype-based) επειδή χρησιμοποιεί αντικείμενα για την εκτέλεση πολλών εργασιών. Είναι μία γλώσσα σεναρίων (scripting language), δηλαδή βασίζεται σε σενάρια (scripts) [9], τα οποία διερμηνεύονται από τη μηχανή JavaScript που διαθέτουν όλοι οι περιηγητές ιστού (browsers), ορισμένοι διακομιστές (servers) και διάφορα προγράμματα εφαρμογών. Επίσης, συμμορφώνεται με το πρότυπο ECMAScript.

Η JavaScript θεωρείται γλώσσα υψηλού επιπέδου και μπορεί να υποστηρίξει διαφορετικά προγραμματιστικά παραδείγματα (multi-paradigm). Γενικά, υποστηρίζει το συναρτησιακό (functional), αντικειμενοστραφή (object-oriented) και προστακτικό (imperative) στυλ προγραμματισμού. Εκτός από τα παραπάνω στυλ προγραμματισμού, υποστηρίζει και τον οδηγούμενο από γεγονότα προγραμματισμό (event-driven programming).

Η JavaScript καθορίζει τη συμπεριφορά των ιστοσελίδων και μετατρέπει τις στατικές HTML σελίδες σε δυναμικές και διαδραστικές. Βελτιώνει τις ιστοσελίδες προσφέροντας μια καλύτερη εμπειρία στους χρήστες. Η JavaScript μπορεί να χειριστεί το HTML DOM (Document Object Model), δηλαδή μπορεί να τροποποιήσει το περιεχόμενο, τις ιδιότητες της HTML και το στυλ εμφάνισης της CSS. Έχει τη δυνατότητα να προσθέσει και να διαγράψει HTML στοιχεία,

καθώς και να αντιδράσει σε ενέργειες του χρήστη. Επίσης, χρησιμοποιείται για την επικύρωση (validation) των δεδομένων που εισάγουν οι χρήστες στις φόρμες.

Ένα πρόγραμμα JavaScript αποτελείται από ένα σύνολο εντολών. Οι εντολές στην JavaScript ονομάζονται **δηλώσεις** (statements) και εκτελούνται σειριακά. Οι δηλώσεις συνιστάται να διαχωρίζονται με το σύμβολο ";". Γενικά, οι δηλώσεις αποτελούνται από τιμές, εκφράσεις, τελεστές, λέξεις-κλειδιά και σχόλια [10].

Για να ενσωματώσουμε κώδικα JavaScript σε ένα έγγραφο HTML, πρέπει να χρησιμοποιήσουμε την ετικέτα (tag) **<script>**. **Υπάρχουν τρεις τρόποι εισαγωγής κώδικα JavaScript σε ένα έγγραφο HTML** [11]:

1. **Εισαγωγή κώδικα JavaScript στην ετικέτα <head>…</head>.** Με αυτόν τον τρόπο, ο κώδικας JavaScript πρέπει να τοποθετηθεί εντός της ετικέτας **<script>…</script>**.

```
<!DOCTYPE html>
<html>
  <head>
    <title>JavaScript Script In Head Tag</title>

    <script>
      function myFunction() {
        let heading1 = document.getElementById("heading1");
        heading1.innerHTML = "Heading Changed";
      }
    </script>

  </head>
  <body>
    <h1 id="heading1">Hello World</h1>
    <button onclick="myFunction()">Change Heading</button>
  </body>
</html>
```

2. **Εισαγωγή κώδικα JavaScript στην ετικέτα <body>…</body>**. Με αυτόν τον τρόπο, ο κώδικας JavaScript πρέπει να τοποθετηθεί εντός της ετικέτας **<script>…</script>**. Γενικά, είναι προτιμότερο να τοποθετούμε την ετικέτα **<script>** στο τέλος της ετικέτας **<body>** παρά στην ετικέτα **<head>**. Έτσι, βελτιώνεται η

ταχύτητα εμφάνισης επειδή η εκτέλεση του κώδικα JavaScript καθυστερεί την εμφάνιση της σελίδας.

```
<!DOCTYPE html>
<html>
  <head>
    <title>JavaScript Script In Body Tag</title>
  </head>
  <body>
    <h1 id="heading1">Hello World</h1>
    <button onclick="myFunction()">Change Heading</button>

    <script>
      function myFunction() {
        let heading1 = document.getElementById("heading1");
        heading1.innerHTML = "Heading Changed";
      }
    </script>

  </body>
</html>
```

3. **Κώδικας JavaScript σε εξωτερικό αρχείο (External JavaScript).** Είναι προτιμότερος από τους άλλους τρόπους επειδή ο κώδικας JavaScript διαχωρίζεται από την HTML σήμανση. Ο διαχωρισμός αυτός κάνει τον κώδικα πιο ευανάγνωστο και μπορεί να τροποποιηθεί πιο εύκολα. Ο κώδικας JavaScript ορίζεται σε εξωτερικό αρχείο χωρίς τη χρήση HTML ετικετών, το οποίο αποθηκεύεται με επέκταση **.js**. Ένα εξωτερικό αρχείο μπορεί να χρησιμοποιηθεί σε πολλές σελίδες. Για να χρησιμοποιήσουμε το εξωτερικό αρχείο JS, πρέπει να γίνει αναφορά σε αυτό. Η αναφορά αυτή γίνεται μέσα στο αρχείο HTML χρησιμοποιώντας την ετικέτα **<script>** με το χαρακτηριστικό (attribute) **src**, όπου το src θα είναι ίσο με το μονοπάτι του εξωτερικού αρχείου ή με μία διεύθυνση URL.

test.html

```
<!DOCTYPE html>
<html>
  <head>
```

```
    <title>External JavaScript</title>
  </head>
  <body>
    <h1 id="heading1">Hello World</h1>
    <button onclick="myFunction()">Change Heading</button>

    <script src="test.js"></script>
  </body>
</html>
```

test.js

```
function myFunction() {
  let heading1 = document.getElementById("heading1");
  heading1.innerHTML = "Heading Changed";
}
```

Αρχικά, ο κώδικας JavaScript εφαρμοζόταν σε ένα πρόγραμμα περιήγησης ιστού (web browser) από την πλευρά του πελάτη (client) και για αυτό το λόγο χαρακτηρίστηκε ως client-side γλώσσα προγραμματισμού. Βέβαια, από την αρχή χρησιμοποιήθηκε και από την πλευρά του διακομιστή (server-side) χωρίς όμως μεγάλη επιτυχία. Η χρήση της JavaScript από την πλευρά του διακομιστή άρχισε να γίνεται δημοφιλής με τη δημιουργία του Node.js [12].

Η JavaScript δημιουργήθηκε από τον Brendan Eich το 1995 και η δημιουργία της προήλθε από την επιθυμία να γίνουν οι ιστοσελίδες από στατικές σε δυναμικές. Τον Νοέμβριο του 1996, η Netscape υπέβαλλε την JavaScript στην Ecma International (οργανισμός προτύπων για συστήματα πληροφοριών και επικοινωνιών) ως πρότυπο και αυτό είχε ως αποτέλεσμα την τυποποιημένη μορφή που ονομάζεται ECMAScript (ES). Η πρώτη έκδοση της ήταν η ECMAScript 1 (ES1), η όποια κυκλοφόρησε τον Ιούνιο του 1997 [13]. Από τότε έχουν δημοσιευθεί αρκετές εκδόσεις που διορθώνουν τις προηγούμενες και εισάγουν νέες δυνατότητες. Μετά την έκδοση ECMAScript 6 (ES6) που κυκλοφόρησε το 2015, η προδιαγραφή ECMAScript εκδίδεται ετησίως και εμπεριέχει τη χρονολογία στο όνομα της, για παράδειγμα ECMAScript 2016, ECMAScript 2017, … , ECMAScript 2021.

Κεφάλαιο 3 – JavaScript Frameworks

3.1 Εισαγωγή στα JavaScript Frameworks

Όπως αναφέρθηκε στο προηγούμενο κεφάλαιο, η JavaScript είναι μια δημοφιλής γλώσσα προγραμματισμού που χρησιμοποιείται για την ανάπτυξη εφαρμογών ιστού. Η ευρεία χρήση της JavaScript οδήγησε στη δημιουργία των JavaScript frameworks, τα οποία αποτελούνται από ένα σύνολο βιβλιοθηκών κώδικα JavaScript και διευκολύνουν τους προγραμματιστές να δημιουργήσουν επεκτάσιμες διαδραστικές εφαρμογές ιστού [14]. Υπάρχουν αρκετά front-end αλλά και back-end JavaScript frameworks, για παράδειγμα μερικά front-end είναι τα Angular, React, Vue.js και μερικά back-end είναι τα Express.js, Sails.js. Ειδικότερα, τα front-end φημίζονται για την ανάπτυξη εφαρμογών μιας σελίδας (single-page applications). Τα περισσότερα είναι δωρεάν και ανοιχτού κώδικα. Το κάθε JS framework έχει τους δικούς του κανόνες, οδηγίες που πρέπει να ακολουθούνται ώστε να χρησιμοποιηθούν αποτελεσματικά και γρήγορα [15]. Για τη χρήση ενός JS framework, προαπαιτείται η εξοικείωση με τις βασικές γλώσσες HTML, CSS και κυρίως JavaScript. Να αναφερθεί ότι οι εφαρμογές που δημιουργούνται με JS frameworks μπορούν να δημιουργηθούν και με «καθαρή» JavaScript (Vanilla JavaScript), ωστόσο προτιμάται η χρήση των JS frameworks για τη δημιουργία σύνθετων εφαρμογών.

Τα JavaScript frameworks καθορίζουν τη δομή των εφαρμογών ιστού και προσφέρουν αρκετά πλεονεκτήματα στους προγραμματιστές. Συγκεκριμένα, παρέχουν λειτουργίες, ένα σύνολο εργαλείων και προγραμμένο κώδικα JavaScript για την ολοκλήρωση των προγραμματιστικών εργασιών τους. Έτσι, αποφεύγεται η σύνταξη κώδικα από την αρχή με αποτέλεσμα να εξοικονομείται χρόνος και να επιταχύνεται η ανάπτυξη των εφαρμογών ιστού. Χρησιμοποιούν τα components, τα οποία είναι μικρά επαναχρησιμοποιήσιμα κομμάτια κώδικα και έτσι αποφεύγεται η επανάληψη συγγραφής ενός υπάρχοντα κώδικα. Ο κώδικας στα JS frameworks είναι πιο ευανάγνωστος και πιο δομημένος. Ένα ακόμα χαρακτηριστικό που παρέχουν είναι το routing (δρομολόγηση). Βέβαια, μερικά δεν διαθέτουν routing και έτσι χρειάζεται η εγκατάσταση πρόσθετων βιβλιοθηκών που θα τους προσφέρει τη λειτουργία αυτή.

Η δημιουργία των διεπαφών χρήστη (UIs) στα front-end JavaScript frameworks γίνεται δηλωτικά, δηλαδή οι προγραμματιστές συντάσσουν κώδικα που περιγράφει το πώς πρέπει να φαίνεται η διεπαφή χρήστη και όχι βήμα – βήμα το πώς θα δημιουργηθεί όπως στην «καθαρή»

JavaScript. Γενικά, ο συνολικός κώδικας είναι λιγότερος στις εφαρμογές που έχουν αναπτυχθεί με ένα JS framework σε σύγκριση με τον κώδικα στις εφαρμογές που έχουν αναπτυχθεί με «καθαρή» JavaScript. Ορισμένα JS frameworks χρησιμοποιούν ειδικές γλώσσες, οι οποίες είναι παραλλαγές JavaScript ή HTML και διευκολύνουν την ανάπτυξη με αυτό το framework. Βέβαια, δεν απαιτείται η χρήση τους. Ένα παράδειγμα είναι το JSX που χρησιμοποιείται στην React.

3.2 Σύγκριση δημοφιλέστερων front-end JavaScript Frameworks

Εν έτη 2022 υπάρχουν αρκετά front-end JavaScript Frameworks, ωστόσο τα τρία πιο δημοφιλή που χρησιμοποιούνται περισσότερο και υπερισχύουν στην αγορά εργασίας είναι τα **Angular**, **React** και **Vue**. Τα τρία αυτά frameworks χρησιμοποιούνται για την ανάπτυξη εφαρμογών μιας σελίδας και βασίζονται σε components. Είναι ικανά να δημιουργήσουν εφαρμογές ιστού μικρού ή μεσαίου ή μεγάλου μεγέθους. Ωστόσο, το Angular προτείνεται για μεγάλες εταιρικές εφαρμογές, ενώ η React και το Vue για μικρές ή μεσαίες εφαρμογές. Ακόμα, έχουν κοινά χαρακτηριστικά αλλά και διαφορές μεταξύ τους. Για αυτό τον λόγο θα γίνει μια σύγκριση των τριών [16] [17].

Το **Angular** είναι ένα πλήρες front-end JavaScript framework [18], το οποίο δημιουργήθηκε το 2009 από την Google και συγκεκριμένα από τον Miško Hevery. Κυκλοφόρησε ως έργο ανοιχτού κώδικα για πρώτη φορά το 2010 με την ονομασία AngularJS και βασιζόταν σε JavaScript. Το 2016 η Google κυκλοφόρησε το Angular 2, όπου μετονομάστηκε από AngularJS σε «σκέτο» Angular. Το Angular 2 βασίζεται σε TypeScript και είναι μία πλήρης επανεγγραφή του AngularJS. Μέχρι σήμερα, η τελευταία έκδοση του είναι η Angular 14 και κυκλοφόρησε στις 2 Ιουνίου του 2022. Διαθέτει μια συλλογή από ενσωματωμένες βιβλιοθήκες που εξυπηρετούν πολλές λειτουργίες, όπως δρομολόγηση (routing), διαχείριση κατάστασης (state management), διαχείριση φορμών (forms management), επικοινωνία πελάτη-διακομιστή (client-server communication). Επιπλέον, παρέχει το επίσημο Angular CLI και μια σουίτα εργαλείων για να βοηθήσει τους προγραμματιστές με την ανάπτυξη των Angular εφαρμογών. Έτσι, δεν χρειάζεται η εγκατάσταση πρόσθετων βιβλιοθηκών.

Η **React** είναι μία βιβλιοθήκη JavaScript [19], η οποία δημιουργήθηκε από την Meta (πρώην Facebook) και συγκεκριμένα από τον Jordan Walke. Κυκλοφόρησε για πρώτη φορά το 2013

ως έργο ανοιχτού κώδικα. Μέχρι σήμερα, η τελευταία έκδοση είναι η React 18.2.0 που κυκλοφόρησε στις 14 Ιουνίου του 2022. Δεν είναι ολοκληρωμένο framework και δεν διαθέτει ενσωματωμένες βιβλιοθήκες. Έτσι, χρειάζεται η εγκατάσταση πρόσθετων βιβλιοθηκών από τρίτους (third-party libraries) για διάφορες λειτουργίες όπως δρομολόγηση (routing), διαχείριση κατάστασης (state management), επικύρωση φορμών (forms validation), δημιουργία και αποστολή HTTP αιτημάτων. Επίσης, δεν περιλαμβάνει επίσημο CLI αλλά χρησιμοποιείται το εργαλείο create-react-app.

Το **Vue** είναι ένα προοδευτικό (progressive) front-end JavaScript framework [20], το οποίο δημιουργήθηκε από τον Evan You, έναν πρώην προγραμματιστή της Google και κυκλοφόρησε για πρώτη φορά το 2014. Η τελευταία έκδοση μέχρι σήμερα είναι η 3.2.37 και κυκλοφόρησε στις 6 Ιουνίου του 2022. Προσφέρει περισσότερα από την React αλλά λιγότερα από το Angular. Παρέχει το επίσημο Vue CLI, ένα σύνολο εργαλείων και βιβλιοθηκών για την υποστήριξη λειτουργιών όπως δρομολόγηση και διαχείριση κατάστασης. Ωστόσο, δεν διαθέτει ενσωματωμένη διαχείριση φορμών και επικοινωνία πελάτη-διακομιστή.

Η React και το Angular έχουν μεγάλη υποστήριξη από την κοινότητα λόγω του Facebook και της Google αντίστοιχα. Από την άλλη, το Vue έχει μικρότερη υποστήριξη από την κοινότητα αλλά αυξάνεται συνεχώς.

3.2.1 Απόδοση

Το Angular χρησιμοποιεί το **πραγματικό DOM (actual DOM)**, όπου κάθε φορά που υπάρχει μία αλλαγή στο UI, τότε ενημερώνεται ολόκληρο το δέντρο των στοιχείων HTML. Από την άλλη, η React και το Vue χρησιμοποιούν **εικονικό DOM (virtual DOM)**. Το εικονικό DOM είναι ταχύτερο και πιο αποδοτικό από το κανονικό DOM, διότι συγκρίνει και ενημερώνει μόνο τις αλλαγές στο συγκεκριμένο κόμβο και όχι ολόκληρο το δέντρο.

Οι εφαρμογές που αναπτύσσονται με Angular είναι μεγαλύτερες σε μέγεθος από τις Vue ή React εφαρμογές. Επιπρόσθετα, οι Vue εφαρμογές είναι λίγο μικρότερες από τις React εφαρμογές. Γενικά, το μέγεθος μιας εφαρμογής επηρεάζει την **απόδοση εκκίνησης (startup performance)**. Επομένως, παρατηρούμε ότι οι Vue και οι React εφαρμογές προσφέρουν καλύτερη απόδοση εκκίνησης από τις Angular εφαρμογές. Εκτός από την απόδοση εκκίνησης, υπάρχει και η **απόδοση χρόνου εκτέλεσης (runtime performance)** που είναι εξαιρετική και

στα τρία. Ακόμα, το Angular δεσμεύει περισσότερη **μνήμη** με αποτέλεσμα να είναι λίγο πιο αργό από το Vue και την React [21].

3.2.2 Καμπύλη μάθησης

Για την εκμάθηση των τριών, προαπαιτείται η γνώση των βασικών γλωσσών HTML, CSS και JavaScript. Τόσο το Angular όσο και το Vue παρέχουν πλήρεις λεπτομερείς οδηγούς για την εκμάθηση τους. Η React παρέχει οδηγό εκμάθησης που είναι ελλιπής και ανενημέρωτος.

Το **Angular** έχει απότομη καμπύλη μάθησης επειδή είναι ένα πλήρες framework που περιλαμβάνει περισσότερες λειτουργίες και έννοιες σε σύγκριση με την React και το Vue. Χρησιμοποιεί πρότυπα που βασίζονται σε HTML και τη γλώσσα TypeScript. Προκειμένου να μάθουν οι προγραμματιστές Angular, θα πρέπει να μάθουν τις λειτουργίες, έννοιες και την TypeScript. Οπότε, η εκμάθηση της TypeScript προσθέτει ένα βαθμό δυσκολίας. Η **React** με τη σειρά της, έχει πιο εύκολη καμπύλη μάθησης από το Angular διότι όπως έχουμε αναφέρει είναι βιβλιοθήκη και περιλαμβάνει λιγότερες λειτουργίες. Η React χρησιμοποιεί JavaScript και τη σύνταξη JSX. Επομένως, ο μέσος προγραμματιστής θα πρέπει να μάθει τις λειτουργίες της και τη σύνταξη JSX που πολλοί τη θεωρούν ευκολότερη από την TypeScript. Από την άλλη, το **Vue** είναι ευκολότερο στην εκμάθηση από την React και το Angular. Ο λόγος είναι ότι χρησιμοποιεί απλώς JavaScript και μία σύνταξη προτύπου που βασίζεται σε HTML. Όπως τα άλλα, περιέχει τις λειτουργίες του, οι οποίες είναι λιγότερες από τις λειτουργίες του Angular αλλά περισσότερες από τις λειτουργίες της React. Ακόμα, η React και το Vue μπορούν να χρησιμοποιήσουν TypeScript προαιρετικά.

3.2.3 Δημοτικότητα

Σε μία δημοσκόπηση που πραγματοποιήθηκε στον ιστότοπο **Stack Overflow** για το 2022, το πιο χρησιμοποιούμενο front-end framework ήταν η React, η οποία ψηφίστηκε από το 42.62% των προγραμματιστών που έλαβαν μέρος. Δεύτερο ήταν το Angular με ποσοστό 20.39% και τρίτο ήταν το Vue με 18.82% [22].

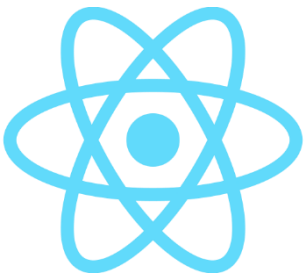
Στον ιστότοπο **GitHub**, το αποθετήριο (repository) του Vue έχει 198.000 αστέρια (GitHub stars), ενώ τα αποθετήρια των React και Angular έχουν 192.000 και 82.800 αστέρια

αντίστοιχα. Εκτός από τα GitHub stars, υπάρχει και το σήμα "Used By" σε κάθε αποθετήριο του GitHub. Το σήμα "Used By" δείχνει τα αποθετήρια του GitHub που εξαρτώνται από το συγκεκριμένο αποθετήριο. Το αποθετήριο της React εμφανίζει 10.8 εκατομμύρια αποθετήρια που εξαρτιούνται από το αποθετήριο της. Το αποθετήριο του Vue εμφανίζει 2.7 εκατομμύρια αποθετήρια, ενώ το αποθετήριο του Angular εμφανίζει 2.4 εκατομμύρια.

Ο ιστότοπος **npm trends** υπολογίζει και συγκρίνεις τις λήψεις των npm πακέτων. Έτσι, σύμφωνα με τον ιστότοπο npm trends, το πακέτο npm (npm package) της React έχει ληφθεί περισσότερες φορές και με μεγάλη διαφορά τον τελευταίο χρόνο σε σύγκριση με τα πακέτα των Vue και Angular. Δεύτερο σε λήψεις είναι το πακέτο του Vue και τελευταίο του Angular με μικρή διαφορά [23].

Ακόμα, να αναφερθεί ότι ένα σημαντικό κριτήριο επιλογής είναι η ζήτηση που έχει το κάθε front-end framework στην αγορά εργασίας. Γενικά, υπάρχουν πολλές διαθέσιμες θέσεις εργασίας. Συγκεκριμένα, αναζητώντας στην ιστοσελίδα **indeed.com** για θέσεις εργασίας σχετικές με React, εμφανίζονται 57.691 ανοικτές θέσεις εργασίας παγκοσμίως. Ενώ αναζητώντας για θέσεις εργασίας σχετικές με Angular και Vue, εμφανίζονται 32.140 θέσεις εργασίας και 8.729 αντίστοιχα. Έτσι, παρατηρούμε ότι η React έχει την περισσότερη ζήτηση στην αγορά εργασίας αυτή την περίοδο.

3.3 React



Εικόνα 3.1 Λογότυπο React

Η **React** είναι μία δημοφιλής βιβλιοθήκη JavaScript που χρησιμοποιείται για τη δημιουργία διεπαφών χρήστη (user interfaces) [19]. Αναφέρεται και ως front-end JavaScript framework. Χρησιμοποιείται για την ανάπτυξη εφαρμογών μιας σελίδας (single-page applications), οι οποίες είναι ταχύτερες από τις εφαρμογές πολλαπλών σελίδων, διότι δεν αποστέλλονται αιτήματα στον διακομιστή (server) για την αλλαγή σελίδας. Βέβαια, με τη βοήθεια μίας βιβλιοθήκης δρομολόγησης που ονομάζεται React Router, υπάρχει η δυνατότητα να δημιουργηθεί η ψευδαίσθηση ότι υπάρχουν πολλαπλές σελίδες (multi-pages) σε μια εφαρμογή μιας σελίδας. Γενικά, για την ανάπτυξη πλήρης σύνθετων εφαρμογών απαιτεί τη χρήση πρόσθετων βιβλιοθηκών [24].

Η React ακολουθεί τον δηλωτικό προγραμματισμό (declarative programming). Οι προγραμματιστές ορίζουν τις επιθυμητές καταστάσεις, δηλαδή περιγράφουν το «τι πρέπει να γίνει» και όχι βήμα - βήμα το «πώς θα γίνει» όπως στον προστακτικό προγραμματισμό (imperative programming) [25].

Η React δημιουργήθηκε από τον Jordan Walke, έναν μηχανικό λογισμικού που εργάζεται στην εταιρεία Meta (πρώην Facebook). Πρώτη φορά χρησιμοποιήθηκε στο newsfeed της πλατφόρμας κοινωνικής δικτύωσης Facebook το 2011 και κυκλοφόρησε το 2013 ως ελεύθερο λογισμικό ανοιχτού κώδικα. Εκτός από το Facebook, σήμερα χρησιμοποιείται από πολλές δημοφιλείς ιστοσελίδες και εφαρμογές. Μερικές από αυτές είναι το Instagram, το Netflix, το Airbnb και το BBC.

Είναι σημαντικό να αναφερθεί ότι για να μάθει κάποιος τη γλώσσα React, χρειάζεται να γνωρίζει τη γλώσσα JavaScript και την HTML. Θα πρέπει να έχει εξοικειωθεί με τις συναρτήσεις, τα αντικείμενα, τους πίνακες και τις κλάσεις της JavaScript. Επίσης, θα πρέπει να γνωρίζει τη νέα σύνταξη και τα χαρακτηριστικά της έκδοσης ES6 (ECMAScript 2015) [26].

3.3.1 React Components

Η React δημιουργεί σύνθετες διεπαφές χρήστη (UIs), χρησιμοποιώντας τα δομικά στοιχεία που ονομάζονται **components**. Τα components είναι μικρά επαναχρησιμοποιήσιμα κομμάτια κώδικα που λειτουργούν μεμονωμένα, και συγχωνεύονται στο τέλος για τη δημιουργία της διεπαφής χρήστη [19]. Η ύπαρξη επαναχρησιμοποιήσιμων components μας βοηθά να αποφύγουμε την επανάληψη και μπορούμε να κάνουμε αλλαγές στον κώδικα μας ευκολότερα. Κάθε component διαχειρίζεται την κατάσταση (state) του και μπορεί να περιλαμβάνει άλλα

components. Στην ουσία, ένα component αποτελείται από κώδικα JSX και επιστρέφει ένα αποτέλεσμα στην οθόνη.

Το κάθε component μπορεί να πάρει οποιοδήποτε όνομα και συνιστάται να είναι ένα ξεχωριστό αρχείο με επέκταση **.js**. Τόσο το όνομα του component όσο και του αρχείου πρέπει να αρχίζουν με κεφαλαίο χαρακτήρα. Στην περίπτωση που ένα όνομα αποτελείται από συνδεδεμένες λέξεις, τότε πρέπει η κάθε λέξη να ξεκινάει με κεφαλαίο χαρακτήρα [27]. Προκειμένου να χρησιμοποιηθεί ένα component, πρέπει να γίνει η εξαγωγή του στο αρχείο του με την εντολή **export default NameOfComponent;**. Ύστερα, η εισαγωγή του στα άλλα αρχεία-components που θα χρησιμοποιηθεί με την εντολή **import NameOfComponent from "./NameOfComponent";**, όπου το τέλος της εντολής προσδιορίζει τη διαδρομή (path) που βρίσκεται το αρχείο.

Υπάρχουν δύο τρόποι για να δημιουργήσουμε components:

1. **Class Components.** Για να δημιουργήσουμε ένα class component, γράφουμε μια κλάση της JavaScript (ES6 class). Η κλάση πρέπει να περιλαμβάνει τη δήλωση **“extends React.Component”** για να κληρονομήσει τα χαρακτηριστικά των components της React όπως τη διαχείριση κατάστασης (state management) και τις μεθόδους κύκλου ζωής (lifecycle methods) [28]. Ακόμα, απαιτείται να περιέχει τη μέθοδο **render()** ώστε να επιστρέψει το τελικό αποτέλεσμα του component στην οθόνη μας.

```
import React from "react";

class Example extends React.Component {

  render() {
    let heading = "This is a Class Component";

    return (
      <div>
        <h1>{heading}</h1>
        <p>Hello World</p>
      </div>
    );
  }
}

export default Example;
```

2. **Function Components.** Μετά την έκδοση React 16.8 όπου προστέθηκαν τα React Hooks, έγινε ο πιο δημοφιλής τρόπος δημιουργίας components. Γενικά, προτείνεται η χρήση των function components μαζί με τα React Hooks. Για να δημιουργήσουμε ένα function component απλώς γράφουμε μια JavaScript συνάρτηση (function). Η συνάρτηση μπορεί να γραφτεί και ως arrow function (συνάρτηση βέλους) της ES6, για παράδειγμα **const NameOfFunction = ( ) => { }**. Τα function components είναι πιο κατανοητά από τα class components και ακόμα γράφουμε λιγότερο κώδικα.

```
import React from "react";

function Example() {

  let heading = "This is a Function Component";

  return (
    <div>
      <h1>{heading}</h1>
      <p>Hello World</p>
    </div>
  );
}

export default Example;
```

3.3.2 JSX

Το **JSX** είναι μία επέκταση σύνταξης της γλώσσας JavaScript που χρησιμοποιείται στην React για να περιγράψουμε την εμφάνιση μιας διεπαφής χρήστη (user interface). Είναι ένας συνδυασμός κώδικα JavaScript και HTML. Το πρόγραμμα περιήγησης δεν «καταλαβαίνει» το JSX και για αυτό μετατρέπεται σε JavaScript κώδικα από τον μεταγλωττιστή Babel στο τέλος. Η React δεν επιβάλλει τη χρήση του JSX αλλά προτείνεται επειδή είναι πιο ευανάγνωστο και μας διευκολύνει στη δημιουργία εφαρμογών αφού μοιάζει με την HTML. Στο JSX μπορούμε να χρησιμοποιήσουμε εκφράσεις (expressions) JavaScript μέσα σε «άγκιστρα» (curly braces),

για παράδειγμα **{user.lastName}** [29]. Τέλος, μπορούμε να επιστρέψουμε μόνο ένα γονέα-στοιχείο από τα components, δηλαδή σε περίπτωση που υπάρχουν περισσότερα από ένα στοιχεία στον JSX κώδικα, τότε πρέπει να περιέχονται σε ένα στοιχείο ανώτερου επιπέδου.

```
const name1 = "Nick Stavropoulos";
const name2 = "Alex Kiamos";
const element = <h1>Hello, {`${name1} and ${name2}`}</h1>;

return (
  <div>
    <h1>{element}</h1>
  </div>
);
```

3.3.3 State

Οι καταστάσεις είναι τα δεδομένα που αλλάζουν με την πάροδο του χρόνου. Τα components έχουν τη δικιά τους **κατάσταση (state)**, η οποία είναι ιδιωτική (private) και χρησιμοποιείται για την αποθήκευση των τιμών των ιδιοτήτων. Συγκεκριμένα, τα class components έχουν ενσωματωμένη κατάσταση ενώ τα function components δεν έχουν. Όμως, με τη βοήθεια των React Hooks τους προσφέρεται αυτή η δυνατότητα. Η αλλαγή μίας κατάστασης προκαλείται είτε από ενέργειες των χρηστών είτε από αποκρίσεις δικτύου. Κάθε φορά που αλλάζει μέσα σε ένα component, τότε το component αποδίδεται εκ νέου (re-render) επιστρέφοντας νέα αποτελέσματα στην οθόνη. Οι καταστάσεις δεν πρέπει να τροποποιούνται απευθείας αλλά χρησιμοποιώντας τη μέθοδο **setState()** στα class components και το hook **useState()** στα function components. Επίσης, οι ενημερώσεις των καταστάσεων είναι ασύγχρονες [30].

3.3.4 Props

Τα **Props** σημαίνουν properties (ιδιότητες) και χρησιμοποιούνται από τα components προκειμένου να μεταβιβάσουν δεδομένα μεταξύ τους. Ουσιαστικά, ο γονέας-component στέλνει δεδομένα στο παιδί-component. Το παιδί-component δεν μπορεί να τροποποιήσει τα props που δέχεται, δηλαδή είναι μόνο για ανάγνωση (read-only) και μόνο ο γονέας-component

μπορεί να αλλάξει τις τιμές των props. Στην πράξη, τα props μεταβιβάζονται σαν ιδιότητες HTML από τον γονέα-component και μετά το παιδί-component δέχεται ως όρισμα ένα αντικείμενο props [31].

Animal.js

```
import React from "react";
import Dog from "./Dog";

function Animal() {
  return (
    <>
      <Dog name="Thor" legs="4" />
    </>
  );
}

export default Animal;
```

Dog.js

```
import React from "react";

function Dog(props) {
  return (
    <>
      <h1>Hello, i am {props.name}</h1>
      <p>I have {props.legs} legs</p>
    </>
  );
}

export default Dog;
```

3.3.5 React Hooks

Τα **React Hooks** είναι ειδικές συναρτήσεις που προστέθηκαν στην έκδοση 16.8 της React. Όπως αναφέρθηκε σε προηγούμενη υποενότητα, χρησιμοποιούνται μόνο στα function components και τα βοηθάνε με τη διαχείριση της κατάστασης (state management) επειδή δεν έχουν δικιά τους κατάσταση (state). Επίσης, τα βοηθάνε και με άλλες δυνατότητες της React όπως τον κύκλο ζωής (lifecycle) και τις αναφορές (refs) [32]. Για τη χρήση των hooks πρέπει

πρώτα να εισαχθούν με την εντολή **import** (JS modules) από τη βιβλιοθήκη **"react"**. Τα hooks έχουν μερικούς κανόνες που πρέπει να τηρούνται. Συγκεκριμένα, πρέπει να καλούνται στην αρχή των function components. Δεν πρέπει να καλούνται μέσα σε βρόχους (loops), συνθήκες (conditions), εμφωλευμένες συναρτήσεις (nested functions) ή κανονικές JavaScript συναρτήσεις. Εκτός από τα έτοιμα hooks της React, μπορούμε να δημιουργήσουμε τα δικά μας custom hooks.

Μερικά hooks που χρησιμοποιούνται στις React εφαρμογές είναι τα εξής: **useState**, **useEffect**, **useRef**, **useMemo**. Το **useState** χρησιμοποιείται για τη διατήρηση και ενημέρωση των καταστάσεων. Η κλήση του useState επιστρέφει έναν πίνακα που περιέχει δύο μεταβλητές, όπου η πρώτη είναι η τρέχουσα τιμή της κατάστασης (current state value) και η δεύτερη μία συνάρτηση για την ενημέρωση της. Ακόμα, δέχεται ως όρισμα μία αρχική τιμή. Ένα παράδειγμα κλήσης είναι το **const [name, setName] = useState(" ");**. Το **useEffect** χρησιμοποιείται για την εκτέλεση παρενεργειών (side effects) όπως για παράδειγμα την ανάκτηση δεδομένων από μια βάση δεδομένων. Το **useMemo** χρησιμοποιείται όταν δεν θέλουμε να εκτελείται μια JS συνάρτηση συνέχεια, παρά μόνο όταν αλλάζει μια συγκεκριμένη τιμή και αυτό έχει ως αποτέλεσμα να βελτιώνεται η απόδοση. Το **useRef** χρησιμοποιείται για τη δημιουργία αναφοράς απευθείας με ένα στοιχείο του πραγματικού DOM. Συγκεκριμένα, επιτρέπει την απόκτηση πρόσβασης σε έναν DOM κόμβο (DOM node). Πολλές φορές χρησιμοποιείται με τις ιδιότητες **.current** και **.value** για την απόκτηση της εισαγόμενης τιμής ενός στοιχείου φόρμας, για παράδειγμα **input1.current.value**, όπου το input1 είναι το όνομα της αναφοράς. Επίσης, χρησιμοποιείται με την ιδιότητα **.current** και τη μέθοδο **.focus()** για την εστίαση του χρήστη σε ένα λανθασμένο πεδίο, για παράδειγμα **input1.current.focus()** [33]. Τέλος, ένα component μπορεί να έχει πολλαπλά useState, useEffect, useMemo και useRef.

3.3.6 Virtual DOM

Η React δημιουργεί ένα **εικονικό DOM** (Virtual DOM) στη μνήμη, το οποίο είναι μία εικονική αναπαράσταση του πραγματικού DOM. Το εικονικό DOM έχει τις ίδιες δυνατότητες με το πραγματικό DOM με τη διαφορά ότι δεν μπορεί να αλλάξει απευθείας την εμφάνιση του UI. Κάθε φορά που υπάρχει μία αλλαγή στην εφαρμογή, τότε ενημερώνεται το εικονικό DOM. Μετά την ενημέρωση, η React συγκρίνει το εικονικό DOM με ένα στιγμιότυπο του εικονικού

DOM που τραβήχτηκε ακριβώς πριν γίνει η ενημέρωση και υπολογίζει τις διαφορές τους. Έπειτα, ενημερώνει μόνο τα αντίστοιχα αντικείμενα στο πραγματικό DOM. Αυτό βελτιώνει την απόδοση αφού δεν ενημερώνει όλα τα αντικείμενα παρά μόνο αυτά που έχουν αλλαχθεί [34].

3.3.7 Conditional Rendering

Στη React, έχουμε τη δυνατότητα να εμφανίσουμε στην οθόνη ή να αποκρύψουμε διαφορετικά components, στοιχεία (elements) ανάλογα με τη συνθήκη (condition) και το state [35]. Αυτό ονομάζεται **Conditional Rendering** και μερικές χρήσεις του είναι για εμφανίσεις μηνυμάτων σφάλματος, για εμφανίσεις εξωτερικών δεδομένων από ένα API και για εναλλαγή της λειτουργικότητας μιας εφαρμογής [36].

Το Conditional Rendering μπορεί να επιτευχθεί με διάφορους τρόπους, όπως τη δομή if-else, τον τριαδικό τελεστή (? :) καθώς και το λογικό τελεστή AND (&&). Οι τρόποι αυτοί συντάσσονται και λειτουργούν όπως στην JavaScript. Ο τριαδικός τελεστής και ο λογικός τελεστής AND πρέπει να περιέχονται μέσα σε curly braces **{ }**.

Η δομή **if-else** (if (συνθήκη) {αληθής} else {ψευδής}) περιέχει τη συνθήκη μετά τη λέξη-κλειδί if και πριν τη λέξη-κλειδί else. Στην περίπτωση που η συνθήκη είναι αληθής (true), τότε θα εκτελεστεί ο κώδικας JSX ή το component που βρίσκεται στο if, αλλιώς αν είναι ψευδής (false) θα εκτελεστεί το else τμήμα. Παράδειγμα:

```
import React, { useState } from "react";
import ReactDOM from "react-dom/client";

function Passed() {
  return <h1>You passed</h1>;
}

function NotPassed() {
  return <h1>You didn't pass</h1>;
}

function Exam() {
  const [hasPassed, setHasPassed] = useState(true);
  if (hasPassed){
    return <Passed/>;
  } else {
    return <NotPassed/>;
```

```
  }
}

const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(<Exam/>);
```

Ο **τριαδικός τελεστής** (συνθήκη **?** αληθής **:** ψευδής) είναι εναλλακτικός τρόπος της δήλωσης if-else και πιο σύντομος. Αρχικά, ορίζεται η συνθήκη ακολουθούμενη από ερωτηματικό (?). Στην περίπτωση που ικανοποιείται η συνθήκη, τότε εκτελείται το τμήμα κώδικα JSX ή το component που βρίσκεται μετά το ερωτηματικό και πριν την άνω και κάτω τελεία (:). Διαφορετικά, θα εκτελεστεί το τμήμα που βρίσκεται μετά την άνω και κάτω τελεία. Ακόμα, ο τρόπος αυτός χρησιμοποιείται για την εφαρμογή κλάσεων δυναμικά σε στοιχεία, δηλαδή θα εφαρμοστούν μόνο αν ισχύει η συνθήκη. Παράδειγμα:

```
import React, { useState } from "react";
import ReactDOM from "react-dom/client";
import "./index.css";

function Passed() {
  return <h1>You passed</h1>;
}

function NotPassed() {
  return <h1>You didn't pass</h1>;
}

function Exam() {
  const [hasPassed, setHasPassed] = useState(true);
  return (
    <div>
      <h1 className={`${hasPassed ? 'green' : 'red'}`}>Exam</h1>
      {hasPassed ? <Passed/> : <NotPassed/>}
    </div>
  );
}

const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(<Exam/>);
```

Ο **λογικός τελεστής AND** (συνθήκη **&&** έκφραση) είναι επίσης ένας σύντομος τρόπος. Πρώτα ορίζεται η συνθήκη, η οποία ακολουθείται από τον τελεστή && και μετά η έκφραση.

Στην πραγματικότητα, η έκφραση είναι το τμήμα κώδικα JSX ή το component που θα εμφανιστεί και εξαρτάται από το αν η συνθήκη είναι αληθής. Στην περίπτωση που η συνθήκη είναι ψευδής, τότε η React θα το παρακάμψει. Παράδειγμα:

```
import React, { useState } from "react";
import ReactDOM from "react-dom/client";

function Passed() {
  return <h1>You passed</h1>;
}

function NotPassed() {
  return <h1>You didn't pass</h1>;
}

function Exam() {
  const [hasPassed, setHasPassed] = useState(true);
  return (
    <div>
      {hasPassed && <Passed/>}
      {!hasPassed && <NotPassed/>}
    </div>
  );
}

const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(<Exam/>);
```

Κεφάλαιο 4 – React Φόρμες

Οι φόρμες αποτελούν σημαντικό μέρος των εφαρμογών ιστού. Χρησιμοποιούνται για τη συλλογή δεδομένων των χρηστών και τους προσφέρουν τη δυνατότητα αλληλεπίδρασης με την εφαρμογή. Οι χρήστες συναντούν τις φόρμες στις online αγορές τους, στις φόρμες επικοινωνίας, στις εγγραφές και στις συνδέσεις τους.

Οι **φόρμες στην React** αναπτύσσονται με παρόμοιο τρόπο όπως στην HTML. Γενικά, τα HTML στοιχεία φόρμας (HTML form elements) λειτουργούν διαφορετικά από τα άλλα DOM στοιχεία στην React επειδή διατηρούν εσωτερική κατάσταση. Για να αναπτύξουμε μία φόρμα στην React, αρχικά δημιουργούμε ένα component. Έπειτα, εντός του component χρησιμοποιούμε την HTML ετικέτα **<form>...</form>**, η οποία περιέχει τα διάφορα στοιχεία φόρμας, για παράδειγμα input types (πεδία εισαγωγής), labels (ετικέτες), buttons (κουμπιά), selects (αναπτυσσόμενες λίστες), textareas (πεδία κειμένου πολλαπλών γραμμών) [37].

```
const FormExample = () => {
  return (
    <form>
      <div>
        <label htmlFor="name">Όνομα:</label>
        <input type="text" id="name"/>
      </div>
      <div>
        <label htmlFor="email">Email:</label>
        <input type="email" id="email"/>
      </div>
      <button type="submit">Υποβολή</button>
    </form>
  );
};

export default FormExample;
```

4.1 Διαφορές μεταξύ React και HTML

Αν και αναπτύσσονται με παρόμοιο τρόπο όπως στην HTML, υπάρχουν μερικές διαφορές στη σύνταξη και στη λειτουργία. Ειδικότερα, τα properties (ιδιότητες), τα attributes

(χαρακτηριστικά) και τα events (γεγονότα) γράφονται με camelCase σύνταξη στην React, ενώ στην HTML γράφονται με πεζά ή κεφαλαία. Μερικές διαφορές είναι οι εξής:

- Η HTML χρησιμοποιεί το χαρακτηριστικό **for** στην ετικέτα **<label>** για τη σύνδεση με το id ενός input. Από την άλλη, η React χρησιμοποιεί το **htmlFor**, επειδή το for είναι δεσμευμένη λέξη στην JavaScript. Παράδειγμα:

  HTML: `<label for="name">Όνομα:</label>`

  React: `<label htmlFor="name">Όνομα:</label>`

- Για τον ορισμό των κλάσεων CSS στην HTML, χρησιμοποιείται το χαρακτηριστικό **class**. Αντιθέτως, στην React χρησιμοποιείται το **className.** Παράδειγμα:

  HTML: `<input type="text" id="name" class="red"/>`

  React: `<input type="text" id="name" className="red"/>`

- Η HTML χρησιμοποιεί το χαρακτηριστικό **selected** σε μία επιλογή (option) για να την ορίσει ως προεπιλεγμένη σε μια αναπτυσσόμενη λίστα (select). Από την άλλη, η React χρησιμοποιεί το χαρακτηριστικό **value** αντί για το selected, το οποίο προστίθεται στην ετικέτα **<select>**. Ακόμα, χρησιμοποιείται το useState hook για τον ορισμό της προεπιλεγμένης τιμής και έπειτα ενσωματώνεται η μεταβλητή κατάστασης (state variable) του useState στο χαρακτηριστικό value. Παράδειγμα:

  HTML:

```
<form>
  <select>
    <option value="React" selected>React</option>
    <option value="Angular">Angular</option>
    <option value="Vue">Vue</option>
  </select>
</form>
```

  React:

```
import { useState } from "react";
const FormExample = () => {
 const [selectValue, setSelectValue] = useState("React");
 const handleSelect = (event) => {
   setSelectValue(event.target.value);
 };
```

```
  return (
    <form>
      <select value={selectValue} onChange={handleSelect}>
        <option value="React">React</option>
        <option value="Angular">Angular</option>
        <option value="Vue">Vue</option>
      </select>
    </form>
  );
};
export default FormExample;
```

- Το γεγονός **onChange** της React συμπεριφέρεται διαφορετικά από το onchange της HTML. Συγκεκριμένα στην React ενεργοποιείται σε κάθε αλλαγή τιμής πεδίου, δηλαδή σε κάθε πάτημα πλήκτρου. Ενώ στην HTML ενεργοποιείται όταν αφαιρεθεί η εστίαση από το πεδίο ή στο πάτημα του πλήκτρου "Enter".

4.2 Διαχείριση των δεδομένων μιας φόρμας

Μετά τη δημιουργία των στοιχείων μιας φόρμας, ένα σημαντικό ζήτημα είναι η διαχείριση των δεδομένων της. Στην React υπάρχουν δύο τρόποι διαχείρισης των δεδομένων μιας φόρμας. Ο πρώτος τρόπος είναι τα controlled components και ο δεύτερος τα uncontrolled components [38].

4.2.1 Controlled Components/Inputs

Τα δεδομένα μιας ολόκληρης φόρμας ή ενός μεμονωμένου controlled input διαχειρίζονται από τα components και αποθηκεύονται σε καταστάσεις (states). Ονομάζονται **controlled (ελεγχόμενα)** επειδή χειρίζονται από την ίδια την React. Γενικά, η ομάδα της React συνιστά τη χρήση των controlled components για την υλοποίηση φορμών.

Για να διαχειριστούμε τα δεδομένα μιας φόρμας, αρχικά ορίζουμε **states** για την αποθήκευση και την ενημέρωση των τιμών των στοιχείων χρησιμοποιώντας το **useState** hook στα function components, αλλιώς τη μέθοδο **React.setState()** στα class components.

```
const [name, setName] = useState("");
```

Έπειτα, προσθέτουμε ένα οποιοδήποτε **event** σε κάθε στοιχείο, για παράδειγμα το **onChange** και ορίζουμε συναρτήσεις (functions) ως **event handlers** που θα συνδεθούν στα αντίστοιχα events. Όταν πυροδοτηθεί ένα event, για παράδειγμα όταν αλλάξει η τιμή σε ένα πεδίο, τότε θα εκτελεστεί ο αντίστοιχος event handler. Σε κάθε event handler ενσωματώνουμε ως όρισμα το αντικείμενο event και εντός καλούμε τη συνάρτηση ενημέρωσης της κατάστασης (state updater function) του useState για να ενημερώσουμε τη μεταβλητή της κατάστασης (state variable) με την τιμή που εισήγαγε ο χρήστης. Για την απόκτηση της τρέχουσας τιμής ενός στοιχείου, ενσωματώνουμε το **event.target.value** ως όρισμα στη συνάρτηση ενημέρωσης της κατάστασης.

```
const nameHandler = (event) => {
    setName(event.target.value);
  };

<input type="text" onChange={nameHandler}/>
```

Ακόμα, προκειμένου να μετατραπεί το κάθε στοιχείο της φόρμας σε ελεγχόμενο, πρέπει να προσθέσουμε στο καθένα το χαρακτηριστικό **value** και να του ενσωματώσουμε την αντίστοιχη μεταβλητή της κατάστασης του useState. Στην περίπτωση που το στοιχείο είναι κουμπί επιλογής (radio button) ή πλαίσιο ελέγχου (checkbox), του προσθέτουμε το χαρακτηριστικό **checked**.

```
<input type="text" value={name} onChange={nameHandler}/>
```

Τέλος, προσθέτουμε το event **onSubmit** στο στοιχείο **<form>** και ορίζουμε έναν event handler που θα συνδεθεί σε αυτό για τη διαχείριση της υποβολής της φόρμας. Ο event handler δέχεται ως όρισμα το αντικείμενο event και εντός του καλούμε τη μέθοδο **event.preventDefault()**. Η μέθοδος αυτή χρησιμοποιείται για να αποτραπεί η προεπιλεγμένη συμπεριφορά του προγράμματος περιήγησης, η οποία θα προκαλούσε την επαναφόρτωση της σελίδας κατά την υποβολή της φόρμας. Έτσι, έχουμε τη δυνατότητα να πραγματοποιήσουμε διάφορες ενέργειες, όπως την επικύρωση των δεδομένων ή την αποστολή αιτήματος σε ένα API.

```
const submitHandler = (event) => {
    event.preventDefault();
    ...
  };
```

```
<form onSubmit={submitHandler}>
...
</form>
```

Παράδειγμα διαχείρισης των δεδομένων μιας φόρμας με Controlled Component:

```
import { useState } from "react";

const FormExample = () => {
  const [name, setName] = useState("");
  const [email, setEmail] = useState("");

  const nameHandler = (event) => {
    setName(event.target.value);
  };

  const emailHandler = (event) => {
    setEmail(event.target.value);
  };

  const submitHandler = (event) => {
    event.preventDefault();
    console.log("Controlled Component");
    console.log(name);
    console.log(email);
  };

  return (
    <form onSubmit={submitHandler}>
      <div>
        <label htmlFor="name">Όνομα:</label>
        <input type="text" id="name" value={name} onChange={nameHandler}/>
      </div>
      <div>
        <label htmlFor="email">Email:</label>
        <input type="email" id="email" value={email} onChange={emailHandler}/>
      </div>
      <button type="submit">Υποβολή</button>
    </form>
  );
};

export default FormExample;
```

Όπως αναφέρθηκε σε προηγούμενο κεφάλαιο, οι ενημερώσεις των καταστάσεων αποδίδουν εκ νέου (re-render) το component στην οθόνη με αποτέλεσμα να προκαλούνται άμεσα αλλαγές στη διεπαφή χρήστη. Αυτό σημαίνει ότι οι καταστάσεις και η διεπαφή χρήστη είναι πάντα συγχρονισμένες.

Τα controlled components χρησιμοποιούνται σε σύνθετες φόρμες, διότι παρέχουν πολλές δυνατότητες όπως:

- Επικύρωση των δεδομένων και εμφάνιση μηνυμάτων σφάλματος κατά την υποβολή της φόρμας, δηλαδή στο πάτημα του κουμπιού Submit.
- Επικύρωση των δεδομένων της φόρμας σε πραγματικό χρόνο, δηλαδή άμεση επικύρωση πεδίου και εμφάνιση μηνυμάτων σφάλματος σε κάθε πάτημα πλήκτρου.
- Δημιουργία δυναμικών πεδίων.
- Επιβολή συγκεκριμένης μορφής στην εισαγόμενη τιμή σε ένα πεδίο, για παράδειγμα η μορφοποίηση της εισαγόμενης τιμής σε ένα πεδίο για πιστωτική κάρτα.
- Τα δεδομένα είναι διαθέσιμα οποιαδήποτε χρονική στιγμή.

Από την άλλη πλευρά, απαιτούν περισσότερο και επαναλαμβανόμενο κώδικα με ελάχιστη παραλλαγή (boilerplate code). Ακόμα, οι αποδόσεις εκ νέου του component πραγματοποιούνται γρήγορα αλλά κοστίζουν σε απόδοση. Σε μικρομεσαίες φόρμες δεν γίνονται αντιληπτές από τους χρήστες, ενώ σε μεγάλες με πολλαπλά πεδία οι χρήστες μπορεί να παρατηρήσουν καθυστέρηση στην πληκτρολόγηση εντός των πεδίων.

4.2.2 Uncontrolled Components/Inputs

Τα δεδομένα μιας ολόκληρης φόρμας ή ενός μεμονωμένου uncontrolled input διαχειρίζονται από το DOM, όπως και στην HTML. Ονομάζονται **uncontrolled (μη ελεγχόμενα)** επειδή δεν ελέγχονται από το state της React και συνήθως χρησιμοποιούνται σε απλές φόρμες. Να αναφερθεί ότι το πεδίο επιλογής αρχείου **<input type="file">** είναι πάντα uncontrolled.

Για να διαχειριστούμε τα δεδομένα μιας φόρμας, αρχικά ορίζουμε **αναφορές (refs)** χρησιμοποιώντας το **useRef** hook στα function components, αλλιώς τη μέθοδο **React.createRef()** στα class components. Έπειτα, προσθέτουμε το χαρακτηριστικό **ref** σε κάθε στοιχείο της φόρμας και του συνδέουμε το αντίστοιχο δημιουργημένο ref για την απόκτηση πρόσβασης στον αντίστοιχο DOM κόμβο (DOM node). Ακόμα, για τον ορισμό μίας αρχικής

τιμής σε ένα στοιχείο, χρησιμοποιείται το χαρακτηριστικό **defaultValue**. Στην περίπτωση που το στοιχείο είναι radio button ή checkbox, τότε χρησιμοποιείται το χαρακτηριστικό **defaultChecked**.

```
const nameRef = useRef();

<input type="text" ref={nameRef}/>
```

Τέλος, προσθέτουμε το event **onSubmit** στο στοιχείο **<form>** και ορίζουμε έναν **event handler** που θα ενσωματωθεί σε αυτό για τη διαχείριση της υποβολής. Ο event handler υλοποιείται με παρόμοιο τρόπο όπως και στα controlled. Η μόνη διαφορά είναι ότι εντός του χρησιμοποιούμε το όνομα του ref μαζί με τις ιδιότητες **.current** και **.value** για την απόκτηση της τιμής του αντίστοιχου DOM στοιχείου.

```
const submitHandler = (event) => {
    event.preventDefault();
    console.log(nameRef.current.value);
  };

<form onSubmit={submitHandler}>

...

</form>
```

Παράδειγμα διαχείρισης των δεδομένων μιας φόρμας με Uncontrolled Component:

```
import { useRef } from "react";

const FormExample = () => {

  const nameRef = useRef();
  const emailRef = useRef();

  const submitHandler = (event) => {
    event.preventDefault();
    console.log("Uncontrolled Component");
    console.log(nameRef.current.value);
    console.log(emailRef.current.value);
  };

  return (
    <form onSubmit={submitHandler}>
      <div>
        <label htmlFor="name">Όνομα:</label>
```

```
        <input type="text" id="name" ref={nameRef}/>
      </div>
      <div>
        <label htmlFor="email">Email:</label>
        <input type="email" id="email" ref={emailRef}/>
      </div>
      <button type="submit">Υποβολή</button>
    </form>
  );
};

export default FormExample;
```

Στα uncontrolled components, δεν χρειάζεται να προσθέσουμε events και να ορίσουμε event handlers για κάθε στοιχείο. Αυτό έχει ως αποτέλεσμα να απαιτούν λιγότερο κώδικα. Γενικά, παρέχουν λιγότερες δυνατότητες από τα controlled επειδή οι αλλαγές των τιμών δεν προκαλούν την απόδοση εκ νέου του component. Ειδικότερα, παρέχουν τη δυνατότητα απόκτησης των δεδομένων, της επικύρωσης τους αλλά και την εμφάνιση μηνυμάτων σφάλματος μόνο στην υποβολή της φόρμας.

4.3 Επικύρωση φόρμας από την πλευρά του πελάτη

Η **επικύρωση φόρμας από την πλευρά του πελάτη (client-side form validation)** είναι ένα κρίσιμο μέρος της ανάπτυξης φορμών και πραγματοποιείται στο πρόγραμμα περιήγησης. Είναι η διαδικασία η οποία διασφαλίζει ότι τα δεδομένα που έχουν εισάγει οι χρήστες σε μία φόρμα είναι στη σωστή μορφή και εντός των περιορισμών, πριν την αποστολή τους σε έναν διακομιστή ή API. Παρέχει μηνύματα σφάλματος στην εισαγωγή μη έγκυρων δεδομένων, με αποτέλεσμα να καθοδηγεί τους χρήστες στην ορθή συμπλήρωση και να τους προσφέρει καλύτερη εμπειρία. Ωστόσο, η επικύρωση από την πλευρά του πελάτη (client-side validation) δεν παρέχει ασφάλεια επειδή είναι εύκολο να παρακαμφθεί. Επομένως, για καλύτερη ασφάλεια θα πρέπει να πραγματοποιείται και επικύρωση από την πλευρά του διακομιστή (server-side validation) [39].

Στην React, η επικύρωση φόρμας από την πλευρά του πελάτη μπορεί να πραγματοποιηθεί χρησιμοποιώντας τους εξής τρόπους:

- Ενσωματωμένη επικύρωση φόρμας HTML (Built-in HTML form validation)
- Προσαρμοσμένη επικύρωση φόρμας (Custom form validation)

4.3.1 Ενσωματωμένη επικύρωση φόρμας HTML

Η **ενσωματωμένη επικύρωση HTML** παρέχει βασική επικύρωση και εκτελείται αυτόματα από το πρόγραμμα περιήγησης. Μπορεί να χρησιμοποιηθεί στα controlled αλλά και στα uncontrolled components. Για την υλοποίηση της χρησιμοποιούνται τα χαρακτηριστικά επικύρωσης της HTML σε στοιχεία της φόρμας. Μερικά από αυτά είναι το required, type, min, max, minLength, maxLength και pattern.

Το χαρακτηριστικό **required** καθορίζει ότι ένα πεδίο εισαγωγής απαιτείται να συμπληρωθεί πριν την υποβολή της φόρμας.

```
<input type="text" value={name} onChange={nameHandler} required/>
```

Το χαρακτηριστικό **type** καθορίζει τον τύπο των δεδομένων που πρέπει να εισαχθεί σε ένα πεδίο, για παράδειγμα email, αριθμός.

```
<input type="email" value={email} onChange={emailHandler}/>
```

Τα χαρακτηριστικά **minLength** και **maxLength** καθορίζουν το ελάχιστο και το μέγιστο μήκος χαρακτήρων αντίστοιχα σε ένα πεδίο.

```
<input type="text" value={name} onChange={nameHandler} minLength="3"
maxLength="15"/>
```

Τα χαρακτηριστικά **min** και **max** καθορίζουν την ελάχιστη και τη μέγιστη τιμή αντίστοιχα που δέχεται ένα αριθμητικό πεδίο.

```
<input type="number" value={number} onChange={numberHandler} min="1"
max="10"/>
```

Το χαρακτηριστικό **pattern** καθορίζει μία έκφραση με την οποία πρέπει να συμμορφώνεται η τιμή που έχει εισαχθεί σε ένα πεδίο. Στο παρακάτω παράδειγμα, το πεδίο κωδικός απαιτεί να περιέχει ακριβώς τέσσερα ψηφία.

```
<input type="password" value={pin} onChange={pinHandler} pattern="[0-9]{4}"/>
```

Από την άλλη πλευρά, η ενσωματωμένη επικύρωση HTML δεν επαρκεί για την υλοποίηση μίας πλήρους λειτουργικής επικύρωσης επειδή δεν μπορεί να εκτελέσει περίπλοκους ελέγχους και να ικανοποιήσει σύνθετες απαιτήσεις. Επιπλέον, δεν προσφέρει τη δυνατότητα

προσαρμογής του στυλ και των μηνυμάτων σφάλματος. Γενικά, τα μηνύματα σφάλματος που παρέχει είναι ασαφή και δεν υποστηρίζεται από μερικά προγράμματα περιήγησης.

4.3.2 Προσαρμοσμένη επικύρωση φόρμας

Λόγω των μειονεκτημάτων της ενσωματωμένης επικύρωσης HTML, προτιμάται η δημιουργία και η χρήση προσαρμοσμένης επικύρωσης. Η **προσαρμοσμένη επικύρωση (custom validation)** υλοποιείται χρησιμοποιώντας JavaScript και είναι πλήρως προσαρμόσιμη. Προσφέρει τη δυνατότητα εκτέλεσης σύνθετων ελέγχων και αυτό έχει ως αποτέλεσμα να μπορούν να ικανοποιηθούν πολύπλοκες απαιτήσεις.

Γενικά η επικύρωση των δεδομένων και η εμφάνιση των μηνυμάτων σφάλματος μπορεί να πραγματοποιηθεί κατά την υποβολή της φόρμας, κατά την αφαίρεση της εστίασης ενός πεδίου και ακόμα σε πραγματικό χρόνο.

4.3.2.1 Επικύρωση κατά την υποβολή της φόρμας

Τα δεδομένα επικυρώνονται και τα μηνύματα σφάλματος εμφανίζονται **κατά την υποβολή της φόρμας**, δηλαδή στο πάτημα του κουμπιού της Υποβολής (Submit). Όπως αναφέρθηκε σε προηγούμενη υποενότητα, μπορεί να υλοποιηθεί στα controlled αλλά και στα uncontrolled components.

Οι χρήστες συμπληρώνουν τα πεδία της φόρμας και δεν ενημερώνονται άμεσα αν τα δεδομένα που έχουν εισάγει είναι έγκυρα. Αντιθέτως ενημερώνονται κατά την υποβολή της. Αυτό έχει ως αποτέλεσμα να αποφεύγονται οι περιττές προειδοποιήσεις, αλλά το μειονέκτημα είναι ότι τα μηνύματα σφάλματος εμφανίζονται αργά. Ακόμα, στην περίπτωση που οι χρήστες έχουν εισάγει μη έγκυρες τιμές, τότε πρέπει να επιστρέψουν στα αντίστοιχα πεδία για να τα διορθώσουν, πράγμα που είναι χρονοβόρο.

Για να επικυρώσουμε κατά την υποβολή της φόρμας, εκτός από τα ήδη ορισμένα states για την αποθήκευση και την ενημέρωση των τιμών των πεδίων, ορίζουμε **δύο states** για κάθε πεδίο. Το πρώτο είναι για το αν είναι έγκυρη (valid) η εισαγόμενη τιμή και το δεύτερο για το αν ο χρήστης έχει "αγγίξει" (touched) το πεδίο. Τα states αυτά τα αρχικοποιούμε με τιμή false επειδή αρχικά οι εισαγωγές δεν είναι έγκυρες και ο χρήστης δεν έχει "αγγίξει" τα πεδία.

```
const [name, setName] = useState("");
```

```
const [nameIsValid, setNameIsValid] = useState(false);
const [nameTouched, setNameTouched] = useState(false);
```

Έπειτα για κάθε πεδίο δημιουργούμε **μία μεταβλητή**, η οποία αντιπροσωπεύει τη μη εγκυρότητα του αντίστοιχου πεδίου και ισούται με μία λογική πράξη AND (&&) μεταξύ του valid και touched state. Για παράδειγμα, **const nameInputIsInvalid = !nameIsValid && nameTouched**. Επίσης, χρησιμοποιούμε **Conditional Rendering** εντός της ετικέτας **<form>...</form>** για την εμφάνιση των μηνυμάτων σφάλματος και του προσθέτουμε την αντίστοιχη μεταβλητή ως συνθήκη. Αν το τελικό αποτέλεσμα της είναι true, τότε θα εμφανίζεται το αντίστοιχο μήνυμα σφάλματος στην οθόνη.

```
const nameInputIsInvalid = !nameIsValid && nameTouched;

return (
  <form onSubmit={submitHandler}>
    ...
    {nameInputIsInvalid && <p>Εισάγετε το πεδίο Όνομα</p>}
  </form>
);
```

Στη συνέχεια εντός του event handler της υποβολής της φόρμας, ενημερώνουμε το κάθε touched state με την τιμή true. Αυτές οι ενημερώσεις πραγματοποιούνται ακόμη κι αν ο χρήστης δεν έχει συμπληρώσει τα πεδία. Επειδή όταν υποβάλει τη φόρμα, σημαίνει ότι επιβεβαιώνει την ολοκλήρωση της. Μετά την ενημέρωση των touched, πραγματοποιούμε ελέγχους **if** για την επικύρωση των εισαγόμενων τιμών. Αν μία τιμή είναι μη έγκυρη, τότε ενημερώνουμε το αντίστοιχο valid state με την τιμή false ώστε να εμφανιστεί το αντίστοιχο μήνυμα σφάλματος. Ακόμα, χρησιμοποιούμε την εντολή **return** για να διακόψουμε την εκτέλεση του event handler και για να μην πραγματοποιηθεί η υποβολή της φόρμας. Διαφορετικά αν μία τιμή είναι έγκυρη, τότε ενημερώνουμε το αντίστοιχο valid state με την τιμή true. Τέλος, η υποβολή της φόρμας ολοκληρώνεται επιτυχώς όταν όλες οι τιμές είναι έγκυρες.

```
const submitHandler = (event) => {
  event.preventDefault();

  setNameTouched(true);

  if (name.trim() === "") {
    setNameIsValid(false);
    return;
```

```
  } else {
    setNameIsValid(true);
  }

  console.log(name);
};
```

Παράδειγμα επικύρωσης κατά την υποβολή της φόρμας:

```
import { useState } from "react";

const FormExample = () => {
  const [name, setName] = useState("");
  const [nameIsValid, setNameIsValid] = useState(false);
  const [nameTouched, setNameTouched] = useState(false);

  const [email, setEmail] = useState("");
  const [emailIsValid, setEmailIsValid] = useState(false);
  const [emailTouched, setEmailTouched] = useState(false);

  const emailRegex =
    /^(([^<>()[\]\\.,;:\s@"]+(\.[^<>()[\]\\.,;:\s@"]+)*)|(".+"))@((\[[0-
9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}])|(([a-zA-Z\-0-9]+\.)+[a-zA-
Z]{2,}))$/;

  const nameInputIsInvalid = !nameIsValid && nameTouched;
  const emailInputIsInvalid = !emailIsValid && emailTouched;

  const nameHandler = (event) => {
    setName(event.target.value);
  };

  const emailHandler = (event) => {
    setEmail(event.target.value);
  };

  const submitHandler = (event) => {
    event.preventDefault();

    setNameTouched(true);
    setEmailTouched(true);

    if (name.trim() === "" && !email.match(emailRegex)) {
      setNameIsValid(false);
      setEmailIsValid(false);
      return;
    } else {
      setNameIsValid(true);
```

```
      setEmailIsValid(true);
    }

    if (name.trim() === "") {
      setNameIsValid(false);
      return;
    }

    if (!email.match(emailRegex)) {
      setEmailIsValid(false);
      return;
    }

    console.log("Controlled Component");
    console.log(name);
    console.log(email);
  };

  return (
    <form noValidate onSubmit={submitHandler}>
      <div>
        <label htmlFor="name">Όνομα:</label>
        <input type="text" id="name" value={name} onChange={nameHandler}/>
      </div>
      <div>
        <label htmlFor="email">Email:</label>
        <input type="email" id="email" value={email} onChange={emailHandler}/>
      </div>
      <button type="submit">Υποβολή</button>
      {nameInputIsInvalid && <p>Εισάγετε το πεδίο Όνομα</p>}
      {emailInputIsInvalid && <p>Εισάγετε το πεδίο Email και να είναι
έγκυρο</p>}
    </form>
  );
};

export default FormExample;
```

4.3.2.2 Επικύρωση κατά την αφαίρεση της εστίασης από ένα πεδίο

Η επικύρωση της εισαγόμενης τιμής και η εμφάνιση του μηνύματος σφάλματος πραγματοποιείται **όταν αφαιρεθεί η εστίαση από ένα πεδίο**, δηλαδή στο event **onBlur**. Με λίγα λόγια όταν ο χρήστης βρίσκεται εντός του πεδίου και κάνει κλικ εκτός του ή πατήσει το

πλήκτρο "Tab". Η επικύρωση πρέπει να πραγματοποιηθεί σε κάθε πεδίο μεμονωμένα διότι δεν επικυρώνεται η φόρμα ως σύνολο.

Ο χρήστης ενημερώνεται για το αν έχει εισάγει μία μη έγκυρη τιμή μόλις ολοκληρώσει το πεδίο. Αυτό έχει ως αποτέλεσμα να αποφεύγονται οι περιττές προειδοποιήσεις και να μην χρειάζεται να περιμένει τη συνολική υποβολή ώστε να πληροφορηθεί για τη διόρθωση ενός πεδίου. Ωστόσο, το μειονέκτημα είναι ότι στην περίπτωση που έχει ήδη εισάγει μία μη έγκυρη τιμή και επιστρέψει να τη διορθώσει, τότε πάλι δεν ενημερώνεται άμεσα αν είναι έγκυρη έως ότου ολοκληρώσει το πεδίο. Επίσης να σημειωθεί ότι αυτή η επικύρωση δεν αρκεί μόνη της. Ο λόγος είναι ότι ο χρήστης έχει τη δυνατότητα να υποβάλει τη φόρμα ακόμη κι όταν έχει συμπληρώσει μη έγκυρες τιμές και τα μηνύματα σφάλματος εμφανίζονται. Επομένως πρέπει πάντα να συνδυάζεται με την επικύρωση κατά την υποβολή.

Η επικύρωση κατά την αφαίρεση της εστίασης από ένα πεδίο υλοποιείται με παρόμοιο τρόπο όπως η επικύρωση κατά την υποβολή. Αρχικά εφαρμόζουμε ακριβώς τα ίδια βήματα, δηλαδή ορίζουμε για κάθε πεδίο **δύο states** και **μία μεταβλητή** που αντιπροσωπεύει τη μη εγκυρότητα του αντίστοιχου πεδίου. Επιπλέον χρησιμοποιούμε **Conditional Rendering** εντός της ετικέτας **<form>...</form>** για την εμφάνιση των μηνυμάτων σφάλματος.

```
const [name, setName] = useState("");
const [nameIsValid, setNameIsValid] = useState(false);
const [nameTouched, setNameTouched] = useState(false);


const nameInputIsInvalid = !nameIsValid && nameTouched;

return (
  <form onSubmit={submitHandler}>
    ...
    {nameInputIsInvalid && <p>Εισάγετε το πεδίο Όνομα</p>}
  </form>
);
```

Ωστόσο υπάρχουν μερικές διαφορές. Ειδικότερα, προσθέτουμε το event **onBlur** σε κάθε πεδίο και ορίζουμε **event handlers** που θα συνδεθούν στα αντίστοιχα events. Εντός του κάθε onBlur event handler, ενημερώνουμε το αντίστοιχο touched state με την τιμή true επειδή ο χρήστης "άγγιξε" το αντίστοιχο πεδίο. Έπειτα, πραγματοποιούμε έλεγχο **if** για την επικύρωση της εισαγόμενη τιμής. Στην περίπτωση που η τιμή είναι μη έγκυρη, τότε ενημερώνουμε το αντίστοιχο valid state με την τιμή false ώστε να εμφανιστεί το μήνυμα σφάλματος. Αντιθέτως

αν είναι έγκυρη, τότε ενημερώνουμε το valid state με την τιμή true και δεν εμφανίζεται το μήνυμα.

```
const nameBlurHandler = () => {
  setNameTouched(true);
  if (name.trim() === "") {
    setNameIsValid(false);
  } else {
    setNameIsValid(true);
  }
};


<input type="text" id="name" value={name} onChange={nameHandler}
onBlur={nameBlurHandler}/>
```

Παράδειγμα επικύρωσης κατά την αφαίρεση της εστίασης από ένα πεδίο:

```
import { useState } from "react";

const FormExample = () => {
  const [name, setName] = useState("");
  const [nameIsValid, setNameIsValid] = useState(false);
  const [nameTouched, setNameTouched] = useState(false);

  const [email, setEmail] = useState("");
  const [emailIsValid, setEmailIsValid] = useState(false);
  const [emailTouched, setEmailTouched] = useState(false);

  const emailRegex =
    /^(([^<>()[\]\\.,;:\s@"]+(\.[^<>()[\]\\.,;:\s@"]+)*)|(".+"))@((\[[0-
9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}])|(([a-zA-Z\-0-9]+\.)+[a-zA-
Z]{2,}))$/;

  const nameInputIsInvalid = !nameIsValid && nameTouched;
  const emailInputIsInvalid = !emailIsValid && emailTouched;

  const nameHandler = (event) => {
    setName(event.target.value);
  };

  const emailHandler = (event) => {
    setEmail(event.target.value);
  };

  const nameBlurHandler = () => {
    setNameTouched(true);
    if (name.trim() === "") {
```

```
      setNameIsValid(false);
    } else {
      setNameIsValid(true);
    }
  };

  const emailBlurHandler = () => {
    setEmailTouched(true);
    if (!email.match(emailRegex)) {
      setEmailIsValid(false);
    } else {
      setEmailIsValid(true);
    }
  };

  const submitHandler = (event) => {
    event.preventDefault();

    setNameTouched(true);
    setEmailTouched(true);

    console.log("Controlled Component");
    console.log(name);
    console.log(email);
  };

  return (
    <form noValidate onSubmit={submitHandler}>
      <div>
        <label htmlFor="name">Όνομα:</label>
        <input type="text" id="name" value={name} onChange={nameHandler}
onBlur={nameBlurHandler}/>
      </div>
      <div>
        <label htmlFor="email">Email:</label>
        <input type="email" id="email" value={email} onChange={emailHandler}
onBlur={emailBlurHandler}/>
      </div>
      <button type="submit">Υποβολή</button>
      {nameInputIsInvalid && <p>Εισάγετε το πεδίο Όνομα</p>}
      {emailInputIsInvalid && <p>Εισάγετε το πεδίο Email και να είναι
έγκυρο</p>}
    </form>
  );
};

export default FormExample;
```

4.3.2.3 Επικύρωση σε πραγματικό χρόνο

Η επικύρωση της εισαγόμενης τιμής και η εμφάνιση του μηνύματος σφάλματος πραγματοποιείται **σε πραγματικό χρόνο (real-time)**, δηλαδή στο event **onChange**. Με λίγα λόγια σε κάθε πάτημα πλήκτρου ή σε κάθε αλλαγή τιμής που κάνει ο χρήστης σε ένα πεδίο. Όπως αναφέρθηκε σε προηγούμενη υποενότητα, μπορεί να υλοποιηθεί μόνο στα controlled components. Ακόμα, η επικύρωση πρέπει να δημιουργηθεί σε κάθε πεδίο ξεχωριστά διότι δεν επικυρώνεται η φόρμα ως σύνολο.

Ο χρήστης ενημερώνεται άμεσα στον κάθε χαρακτήρα που εισάγει για τον αν η τιμή είναι έγκυρη ή μη. Έτσι σε περίπτωση που η τιμή είναι μη έγκυρη, τότε θα τη διορθώσει άμεσα. Από την άλλη πλευρά, το αρνητικό είναι ότι προειδοποιείται στον πρώτο χαρακτήρα πριν καν αυτός έχει την ευκαιρία να εισάγει μία έγκυρη τιμή. Επίσης αξίζει να αναφερθεί ότι η επικύρωση αυτή δεν αρκεί μόνη της, επειδή ο χρήστης έχει ακόμα τη δυνατότητα να υποβάλει τη φόρμα με μη έγκυρες τιμές. Οπότε πρέπει πάντα να συνδυάζεται με την επικύρωση κατά την υποβολή.

Για να επικυρώσουμε σε πραγματικό χρόνο, αρχικά ορίζουμε για κάθε πεδίο **ένα touched state** με αρχική τιμή false και **μία μεταβλητή** που αντιπροσωπεύει τη μη εγκυρότητα του πεδίου όπως στις προηγούμενες επικυρώσεις. Ακόμα, ορίζουμε για κάθε πεδίο **μία δεύτερη μεταβλητή**, η οποία αντιπροσωπεύει το αν είναι έγκυρη (valid) η εισαγόμενη τιμή και ισούται με το αποτέλεσμα του αντίστοιχου ελέγχου επικύρωσης της.

```
const [name, setName] = useState("");
const [nameTouched, setNameTouched] = useState(false);


const nameIsValid = name.trim() !== "";
const nameInputIsInvalid = !nameIsValid && nameTouched;
```

Όπως και στις προηγούμενες επικυρώσεις, χρησιμοποιούμε **Conditional Rendering** εντός της ετικέτας **<form>...</form>** για την εμφάνιση των μηνυμάτων σφάλματος.

```
return (
  <form onSubmit={submitHandler}>
    ...
    {nameInputIsInvalid && <p>Εισάγετε το πεδίο Όνομα</p>}
  </form>
);
```

Τέλος, προσθέτουμε το event **onChange** σε κάθε πεδίο και ορίζουμε **event handlers** που θα συνδεθούν στα αντίστοιχα events. Εντός του κάθε onChange event handler, ενημερώνουμε το αντίστοιχο state με την τιμή που εισήγαγε ο χρήστης και ακόμα ενημερώνουμε το αντίστοιχο touched state με την τιμή true επειδή ο χρήστης "άγγιξε" το πεδίο. Όπως έχει αναφερθεί πολλές φορές, οι ενημερώσεις των states προκαλούν την εκτέλεση και την απόδοση εκ νέου του component. Έτσι σε κάθε αλλαγή τιμής ελέγχονται οι δύο αντίστοιχες μεταβλητές με αποτέλεσμα να πραγματοποιείται η επικύρωση της αντίστοιχης τιμής. Αν η τιμή είναι μη έγκυρη, τότε εμφανίζεται το αντίστοιχο μήνυμα σφάλματος.

```
const nameHandler = (event) => {
  setNameTouched(true);
  setName(event.target.value);
};


<input type="text" id="name" value={name} onChange={nameHandler}/>
```

Παράδειγμα επικύρωσης σε πραγματικό χρόνο:

```
import { useState } from "react";

const FormExample = () => {
  const [name, setName] = useState("");
  const [nameTouched, setNameTouched] = useState(false);

  const [email, setEmail] = useState("");
  const [emailTouched, setEmailTouched] = useState(false);

  const emailRegex =
    /^(([^<>()[\]\\.,;:\s@"]+(\.[^<>()[\]\\.,;:\s@"]+)*)|(".+"))@((\[[0-
9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}])|(([a-zA-Z\-0-9]+\.)+[a-zA-
Z]{2,}))$/;

  const nameIsValid = name.trim() !== "";
  const nameInputIsInvalid = !nameIsValid && nameTouched;

  const emailIsValid = email.match(emailRegex);
  const emailInputIsInvalid = !emailIsValid && emailTouched;

  const nameHandler = (event) => {
    setNameTouched(true);
    setName(event.target.value);
  };
```

```
  const emailHandler = (event) => {
    setEmailTouched(true);
    setEmail(event.target.value);
  };

  const submitHandler = (event) => {
    event.preventDefault();

    setNameTouched(true);
    setEmailTouched(true);

    console.log("Controlled Component");
    console.log(name);
    console.log(email);
  };

  return (
    <form noValidate onSubmit={submitHandler}>
      <div>
        <label htmlFor="name">Όνομα:</label>
        <input type="text" id="name" value={name} onChange={nameHandler}/>
      </div>
      <div>
        <label htmlFor="email">Email:</label>
        <input type="email" id="email" value={email} onChange={emailHandler}/>
      </div>
      <button type="submit">Υποβολή</button>
      {nameInputIsInvalid && <p>Εισάγετε το πεδίο Όνομα</p>}
      {emailInputIsInvalid && <p>Εισάγετε το πεδίο Email και να είναι
έγκυρο</p>}
    </form>
  );
};

export default FormExample;
```

4.4 Βιβλιοθήκες φόρμας

Εκτός από την ανάπτυξη των φορμών με «καθαρή» React, οι προγραμματιστές μπορούν να χρησιμοποιήσουν και εξωτερικές βιβλιοθήκες φόρμας (external form libraries). Μερικές δημοφιλείς βιβλιοθήκες ανοιχτού κώδικα είναι οι Formik, React Hook Form και React Final Form. Οι βιβλιοθήκες αυτές βοηθούν με τη διαχείριση των δεδομένων, την επικύρωση των

δεδομένων αλλά και τη διαχείριση της υποβολής της φόρμας. Γενικά, προτείνεται η χρήση τους για τη δημιουργία σύνθετων φορμών διότι εξοικονομούν χρόνο και βελτιώνουν την απόδοση. Ακόμα, απαιτούν λιγότερο και μη επαναλαμβανόμενο κώδικα. Από την άλλη πλευρά, δεν χρειάζεται η χρήση τους για τη δημιουργία απλών φορμών με περιορισμένη λειτουργικότητα [40].

Κεφάλαιο 5 – Frameworks, βιβλιοθήκες, πλατφόρμες και εργαλεία

5.1 Bootstrap

Εικόνα 5.1 Λογότυπο Bootstrap

Το **Bootstrap** είναι ένα δημοφιλές front-end framework, το οποίο είναι ελεύθερο λογισμικό ανοιχτού κώδικα και χρησιμοποιείται για την ανάπτυξη responsive ιστοσελίδων και εφαρμογών ιστού, οι οποίες προσαρμόζονται σε όλες τις συσκευές (smartphones, tablets, laptops, desktops). Αναφέρεται και ως CSS framework. Επίσης, ακολουθεί την προσέγγιση mobile-first, δηλαδή δίνει προτεραιότητα πρώτα στα κινητά [41] [42].

Το Bootstrap προσφέρει πρότυπα σχεδίασης βασισμένα σε HTML, CSS και προαιρετικά JavaScript για φόρμες, κουμπιά, εικόνες, τυπογραφία, πίνακες, πλοήγηση και διάφορα στοιχεία διεπαφής (interface components). Με το Bootstrap έχουμε τη δυνατότητα να μορφοποιήσουμε γρήγορα και εύκολα τις ιστοσελίδες και τις εφαρμογές ιστού, έχοντας απλώς βασικές γνώσεις HTML και CSS. Γενικά, το Bootstrap προσφέρει προκαθορισμένες κλάσεις CSS, με τις οποίες μπορούμε να μορφοποιήσουμε διάφορα στοιχεία. Για τη χρήση μιας προκαθορισμένης κλάσης, το μόνο που χρειάζεται είναι να προστεθεί το όνομα της στο χαρακτηριστικό class του αντίστοιχου στοιχείου.

Ένας από τους κύριους λόγους που πολλοί προγραμματιστές προσθέτουν το Bootstrap στα project τους είναι για να χρησιμοποιήσουν το **Σύστημα Πλέγματος (Grid System)**. Το Grid System προσφέρει στοίχιση περιεχομένου και διάταξη (layout) που προσαρμόζεται ανάλογα με το μέγεθος της οθόνης της συσκευής. Το Grid System του Bootstrap είναι ευέλικτο και χρησιμοποιεί μία σειρά από container, σειρές (rows) και στήλες (columns). Οι στήλες που υποστηρίζει είναι μέχρι δώδεκα [43].

Τα **breakpoints** στο Bootstrap είναι πλάτη (widths) τα οποία προσαρμόζονται και καθορίζουν τη συμπεριφορά των διατάξεων σε διάφορες συσκευές ή viewports (θύρες προβολών). Το Bootstrap 5 παρέχει 6 προεπιλεγμένα breakpoints [44].

Breakpoint	Class infix	Διαστάσεις
X-Small	*None*	<576px
Small	sm	≥576px
Medium	md	≥768px
Large	lg	≥992px
Extra large	xl	≥1200px
Extra extra large	xxl	≥1400px

Εικόνα 5.2 Τα breakpoints του Bootstrap

Το **container** είναι θεμελιώδες στοιχείο διάταξης στο Bootstrap και απαιτείται κατά τη χρήση του Grid System. Χρησιμοποιείται για να ορίσει τα περιθώρια του περιεχομένου και πολλές φορές για να «κεντράρει» το περιεχόμενο. Ουσιαστικά, το περιεχόμενο περιλαμβάνεται εντός του container [45].

Το Bootstrap παρέχει τρεις κλάσεις containers:

1. **Κλάση .container**. Είναι η προεπιλεγμένη κλάση και παρέχει ένα responsive container που το μέγιστο πλάτος του αλλάζει σε κάθε breakpoint. Το πλάτος του container αλλάζει σε 100% στο extra small breakpoint.

```
<div class="container">
<!-- Περιεχόμενο -->
</div>
```

2. **Κλάση .container-fluid**. Παρέχει ένα container που το πλάτος του είναι 100% σε κάθε breakpoint.

```
<div class="container-fluid">
<!-- Περιεχόμενο -->
</div>
```

3. **Κλάση .container-{breakpoint}.** Παρέχει ένα container που το πλάτος του είναι 100% μέχρι το συγκεκριμένο breakpoint. Για παράδειγμα, στην κλάση .container-md το πλάτος θα είναι 100% μέχρι το medium breakpoint. Έπειτα, το πλάτος θα προσαρμόζεται στα επόμενα breakpoints, δηλαδή στα lg, xl, και xxl breakpoints.

```
<div class="container-md">
<!-- Περιεχόμενο -->
</div>
```

Το Bootstrap δημιουργήθηκε το 2010 από τον Mark Otto και τον Jacob Thornton στο Twitter και κυκλοφόρησε δημόσια το 2011 ως framework ανοιχτού κώδικα. Το 2012 κυκλοφόρησε το Bootstrap 2, δίνοντας την επιλογή της χρήσης του responsive σχεδιασμού. Το Bootstrap 3 κυκλοφόρησε το 2013, έχοντας τον responsive σχεδιασμό από προεπιλογή και ακολούθησε την προσέγγιση mobile-first. Το Bootstrap 4 ανακοινώθηκε το 2014, ωστόσο κυκλοφόρησε επίσημα το 2018 και άρχισε να χρησιμοποιεί το μοντέλο διάταξης CSS flexbox στο Grid System. Μέχρι και σήμερα, η τελευταία μεγάλη έκδοση είναι το Bootstrap 5, η οποία κυκλοφόρησε επίσημα το 2021. Μία σημαντική αλλαγή στο Bootstrap 5 ήταν η απόρριψη του jQuery για την «κανονική» JavaScript (Vanilla JavaScript) [42] [46].

Η **προσθήκη του Bootstrap** στα έργα γίνεται πολύ εύκολα και με πολλούς τρόπους. Οι πιο συχνοί τρόποι είναι οι εξής [47]:

- Χρήση του Bootstrap CDN.
- Εγκατάσταση μέσω του διαχειριστή πακέτων npm ή του διαχειριστή πακέτων yarn.
- Λήψη από την επίσημη ιστοσελίδα **getbootstrap.com**.

5.2 Material UI

Εικόνα 5.3 Λογότυπο Material UI

Το **Material UI** (MUI) είναι μία βιβλιοθήκη που προσφέρει έτοιμα προς χρήση React components για τη δημιουργία διεπαφών χρήστη (user interfaces). Ακολουθεί την mobile-first προσέγγιση και έχει βασιστεί στο σύστημα σχεδιασμού Material Design της Google [48]. Η χρήση του MUI εξοικονομεί χρόνο αφού δεν χρειάζεται να δημιουργηθούν components από την αρχή. Διαθέτει πρότυπα (templates), εικονίδια, θέματα (themes), διάφορα στυλ, τα οποία μπορούν να προσαρμοστούν από τους προγραμματιστές, ώστε να ταιριάζουν στις προσωπικές προτιμήσεις τους. Επίσης, το MUI επιτρέπει και την εξατομίκευση (customization) του στυλ των components.

Το Material UI παρέχει μία ευρεία γκάμα components, όπως components εισόδων (inputs), components εμφάνισης δεδομένων (data display), components πλοήγησης (navigation), components διάταξης (layout) και components ανατροφοδότησης (feedback). Τέλος, πρέπει να προσθέσουμε τη βιβλιοθήκη στην React εφαρμογή μας και μετά να εισάγουμε (import) τα MUI components στα components που θα τα χρησιμοποιήσουμε.

Η **προσθήκη του MUI** στις React εφαρμογές γίνεται με τους εξής τρόπους:

- Εγκατάσταση μέσω του διαχειριστή πακέτων npm ή του διαχειριστή πακέτων yarn.
- Χρήση CDN.

5.3 Σύστημα Σχεδιασμού Govgr

Το **Σύστημα Σχεδιασμού Govgr** δημιουργήθηκε για να βοηθήσει τους σχεδιαστές και τους προγραμματιστές να δημιουργήσουν εύχρηστες ψηφιακές δημόσιες υπηρεσίες του GOV.GR για την αποτελεσματική εξυπηρέτηση των πολιτών [49]. Το Σύστημα Σχεδιασμού παρέχει ένα Πρότυπο Εξυπηρέτησης (Service Standard) που αποτελείται από αρχές, οι οποίες πρέπει να τηρούνται από τους προγραμματιστές και τους σχεδιαστές ώστε να δημιουργήσουν φιλικές ψηφιακές υπηρεσίες GOV.GR. Οι αρχές αυτές είναι οι εξής:

1. Κατανοήστε τους χρήστες και τις ανάγκες τους.
2. Λύστε ένα ολόκληρο πρόβλημα για τους χρήστες.
3. Παρέχετε μια κοινή εμπειρία σε όλα τα κανάλια.
4. Κάντε την υπηρεσία απλή στη χρήση.

5. Βεβαιωθείτε ότι όλοι μπορούν να χρησιμοποιήσουν την υπηρεσία.
6. Να έχετε μια διεπιστημονική ομάδα.
7. Χρησιμοποιήστε ευέλικτους τρόπους εργασίας.
8. Να υπάρχει επανάληψη και βελτίωση συχνά.
9. Δημιουργήστε μια ασφαλή υπηρεσία που προστατεύει το απόρρητο των χρηστών.
10. Καθορίστε πώς φαίνεται η επιτυχία και δημοσιεύστε δεδομένα απόδοσης.
11. Επιλέξτε τα σωστά εργαλεία και την τεχνολογία.
12. Ο νέος πηγαίος κώδικας να είναι ανοιχτού κώδικα.
13. Χρησιμοποιήστε και συνεισφέρετε σε ανοιχτά πρότυπα, κοινά στοιχεία (components) και μοτίβα (patterns).
14. Λειτουργήστε μια αξιόπιστη υπηρεσία.

Η επίσημη βιβλιοθήκη CSS του συστήματος σχεδίασης GOV.GR ονομάζεται **Digigov CSS** και χρησιμοποιείται για την ανάπτυξη responsive εφαρμογών ή υπηρεσιών με στυλ του GOV.GR. Το Digigov CSS ακολουθεί την mobile-first προσέγγιση και έχει κατασκευαστεί χρησιμοποιώντας το CSS framework Tailwind CSS. Ακόμα, είναι ελαφρύ και εύκολο στη χρήση. Η **προσθήκη του Digigov CSS** σε ένα έργο πραγματοποιείται εύκολα με πολλούς τρόπους. Οι τρόποι αυτοί είναι οι εξής:

- Χρήση CDN του jsdelivr.net, δηλαδή κάνοντας αντιγραφή και επικόλληση των διατιθέμενων **<link>** εντός του **<head>** σε ένα αρχείο.
- Εγκατάσταση μέσω του διαχειριστή πακέτων npm ή του διαχειριστή πακέτων yarn.
- Λήψη των μεταγλωττισμένων CSS αρχείων από την ιστοσελίδα **guide.services.gov.gr**.

Χρησιμοποιώντας τη βιβλιοθήκη Digigov CSS και ακολουθώντας το Σύστημα Σχεδιασμού, μπορούμε να μορφοποιήσουμε τις υπηρεσίες με στυλ του GOV.GR. Η εφαρμογή στυλ πραγματοποιείται με χρήση των προκαθορισμένων κλάσεων του Digigov CSS. Το Digigov CSS παρέχει ένα προκαθορισμένο στυλ, δηλαδή διαθέτει συγκεκριμένα χρώματα, διατάξεις, εικόνες και τυπογραφία. Σε περίπτωση που θέλουμε να εφαρμόσουμε ένα δικό μας προσαρμοσμένο στυλ, θα πρέπει να ακολουθούμε τους κανόνες του GOV.GR.

Το Σύστημα Σχεδιασμού Govgr προσφέρει παραδείγματα κώδικα, τα οποία μπορούμε να χρησιμοποιήσουμε για να δημιουργήσουμε διάφορα επαναχρησιμοποιήσιμα components, για

παράδειγμα components για φόρμες, κουμπιά, πίνακες, λίστες δεδομένων, πλοήγηση, επικεφαλίδα, υποσέλιδο. Έτσι, εξοικονομούμε χρόνο και αποφεύγουμε την επανάληψη του ίδιου κώδικα. Περιλαμβάνει patterns (μοτίβα), τα οποία μπορούν να χρησιμοποιηθούν για να βοηθήσουν τους χρήστες με την ολοκλήρωση εργασιών της υπηρεσίας, όπως τη συμπλήρωση φορμών και τη δημιουργία λογαριασμών. Παρέχει κώδικα για τύπους σελίδων όπως αρχική σελίδα, σελίδα σφαλμάτων, σελίδα ερωτήσεων, σελίδα διαχειριστή. Γενικά, παρέχει πληροφορίες για τη χρήση του στυλ, των παραδειγμάτων κώδικα, των τύπων σελίδων καθώς και για τη λειτουργία τους. Τέλος, περιλαμβάνει οδηγίες σχετικά με την προσβασιμότητα (accessibility), ώστε οι υπηρεσίες να είναι προσβάσιμες και να μπορούν να χρησιμοποιηθούν από όλους τους χρήστες, συμπεριλαμβανομένου των ατόμων με ειδικές ανάγκες ή των ατόμων σε διαφορετικές περιστάσεις.

5.4 React Router

Το **React Router** είναι μια βιβλιοθήκη δρομολόγησης (routing library) που χρησιμοποιείται στη βιβλιοθήκη React για τη δημιουργία εφαρμογών πολλαπλών σελίδων (multi-pages applications) [50]. Στην πραγματικότητα δημιουργεί την ψευδαίσθηση ότι υπάρχουν πολλαπλές σελίδες (multi-pages) σε μία εφαρμογή μιας σελίδας, επειδή όπως έχουμε αναφέρει η React δημιουργεί εφαρμογές μίας σελίδας (single-page applications).

Το React Router βοηθάει την React διαθέτοντας τα δικά του React components και hooks. Επιτρέπει την πλοήγηση (navigation) μεταξύ των components σε μια εφαρμογή React, δημιουργεί διευθύνσεις URL, παρέχει τη δυνατότητα αλλαγής της διεύθυνσης URL του προγράμματος περιήγησης και συγχρονίζει τη διεπαφή χρήστη με τη διεύθυνση URL. Ουσιαστικά, η πλοήγηση στα διάφορα components προκαλεί την εναλλαγή των προβολών (views) στην οθόνη [51].

Το React Router μπορεί να εγκατασταθεί στις React εφαρμογές μέσω του διαχειριστή πακέτων npm ή του διαχειριστή πακέτων yarn. Συγκεκριμένα, πρέπει να εγκατασταθεί το react-router-dom package (πακέτο). Επίσης, τα components ή hooks που παρέχει το React Router, θα πρέπει να εισαχθούν με την εντολή import στα αντίστοιχα React components, προκειμένου να χρησιμοποιηθούν [52].

Τα βασικά components του React Router είναι [53]**:**

- **<BrowserRouter>**. Χρησιμοποιείται ως γονέας-component και περικλείονται μέσα του όλα τα components που υπάρχουν σε μια εφαρμογή. Χάρη σε αυτό, λειτουργούν τα components και hooks του React Router. Ακόμα, συγχρονίζει τη διεπαφή χρήστη με τις διευθύνσεις URL.
- **<Routes>**. Είναι παιδί-component του BrowserRouter και περικλείονται μέσα του όλα τα Route components, δηλαδή οι διαδρομές.
- **<Route>**. Είναι παιδί-component του Routes και δημιουργεί μια διαδρομή. Έχει τις ιδιότητες **path** και **element**, για παράδειγμα **<Route path="…" element="…"/>**, όπου το path είναι μοτίβο (pattern) και στην ιδιότητα element ενσωματώνεται ένα component. Σε περίπτωση που η ιδιότητα path του Route ταιριάξει με την τρέχουσα διεύθυνση URL, τότε το component που βρίσκεται στην ιδιότητα element θα εμφανιστεί στην οθόνη.
- **<Link>.** Χρησιμοποιείται για την πλοήγηση στην εφαρμογή και περιέχει κάποιο στοιχείο, για παράδειγμα ένα button (κουμπί). Έχει την ιδιότητα **to**, η οποία ορίζει το μονοπάτι που θα γίνει η μετάβαση, σε περίπτωση που πατηθεί το στοιχείο που περιλαμβάνει μέσα το Link, για παράδειγμα **<Link to="…">…</Link>**.

App.js

```
import { BrowserRouter, Routes, Route } from "react-router-dom";
import Home from "./Home";
import History from "./History";
import "./App.css";

const App = () => {
  return (
    <BrowserRouter>
      <h1>React Router Example</h1>
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/history" element={<History />} />
      </Routes>
    </BrowserRouter>
  );
};

export default App;
```

History.js

```
import { Link } from "react-router-dom";

const History = () => {
  return (
    <div>
      <Link to="/">
        <h1>Go to Home</h1>
      </Link>
    </div>
  );
};

export default History;
```

5.5 npm

Εικόνα 5.4 Λογότυπο npm

Το **npm** είναι ένας JavaScript διαχειριστής πακέτων (package manager) για το Node.js. Είναι το μεγαλύτερο μητρώο λογισμικού (software registry) στον κόσμο και μπορεί να χρησιμοποιηθεί δωρεάν. Εφευρέθηκε από τον Isaac Z. Schlueter και διατηρείται από τον npm οργανισμό (npm Inc.) [54]. Οι προγραμματιστές χρησιμοποιούν το npm για τον διαμοιρασμό πακέτων λογισμικού και ακόμα πολλοί οργανισμοί το χρησιμοποιούν για τη διαχείριση της ανάπτυξης τους ιδιωτικά αλλά και δημόσια. Η δημοσίευση δημόσιων πακέτων είναι δωρεάν, ενώ των ιδιωτικών πακέτων είναι επί πληρωμή [55].

Το npm περιέχει μια **διεπαφή γραμμής εντολών (Command-Line Interface (CLI))**, η οποία χρησιμοποιείται από τους προγραμματιστές για την αλληλεπίδραση με το npm. Ειδικότερα, οι προγραμματιστές ορίζουν εντολές στη γραμμή εντολών για να κατεβάσουν, να εγκαταστήσουν και να ενημερώσουν πακέτα. Εκτός από το CLI, διαθέτει το **μητρώο npm (registry npm)**, το οποίο είναι μία CouchDB βάση δεδομένων με πακέτα JavaScript, όπου το κάθε πακέτο

απαρτίζεται από λογισμικό και μεταδεδομένα (metadata). Τα διάφορα πακέτα είναι διαθέσιμα για προβολή στην επίσημη ιστοσελίδα **npmjs.com**.

Το npm εγκαθιστά και διαχειρίζεται όλες τις εξαρτήσεις (dependencies) ενός project. Οι εξαρτήσεις ορίζονται σε ένα αρχείο που ονομάζεται **package.json** και το περιεχόμενο του είναι σε μορφή JSON. Γενικά, δεν μπορεί να χρησιμοποιηθεί αν δεν υπάρχει εγκαταστημένο το Node.js στον υπολογιστή μας. Συγκεκριμένα, το npm συμπεριλαμβάνεται στην εγκατάσταση του Node.js.

5.6 Visual Studio Code

Εικόνα 5.5 Λογότυπο Visual Studio Code

Το **Visual Studio Code** ή **VS Code** είναι ένα ελαφρύ πρόγραμμα επεξεργασίας πηγαίου κώδικα και διατίθεται δωρεάν για λήψη από τον επίσημο ιστότοπο **code.visualstudio.com** για τα λειτουργικά συστήματα Windows, macOS και Linux [56]. Δημιουργήθηκε από την Microsoft το 2015 και χρησιμοποιείται για τη συγγραφή και επεξεργασία κώδικα σε διάφορες γλώσσες προγραμματισμού όπως JavaScript, Python, Java. Γενικά, παρέχει υποστήριξη για τις πιο βασικές γλώσσες προγραμματισμού. Βέβαια, μπορεί να υποστηρίξει και άλλες πρόσθετες γλώσσες με την εγκατάσταση επεκτάσεων [57].

Το VS Code προσφέρει πολλές δυνατότητες. Συγκεκριμένα, προσφέρει υποστήριξη για αποσφαλμάτωση (debugging), αποσπάσματα (snippets) και ανακατασκευή κώδικα (code refactoring). Περιέχει ένα εργαλείο που ονομάζεται IntelliSense, το οποίο παρέχει έξυπνη συμπλήρωση κώδικα (intelligent code completion) και επισήμανση σύνταξης (syntax highlighting). Επιτρέπει στους προγραμματιστές να ανοίξουν περισσότερα από ένα project ταυτόχρονα. Επίσης, έχει ενσωματωμένο το σύστημα ελέγχου εκδόσεων Git και terminal.

Το VS Code είναι επεκτάσιμο και προσαρμόσιμο. Οι προγραμματιστές έχουν τη δυνατότητα να διαμορφώσουν το περιβάλλον του VS Code ανάλογα με τις προτιμήσεις τους. Ειδικότερα, μπορούν να αλλάξουν το θέμα και τις συντομεύσεις πληκτρολογίου. Τέλος, μπορούν να εγκαταστήσουν επεκτάσεις για πρόσθετη λειτουργικότητα και προτιμήσεις.

5.7 Git

Εικόνα 5.6 Λογότυπο Git

Το **Git** είναι ένα δωρεάν δημοφιλές κατανεμημένο σύστημα ελέγχου εκδόσεων (Version Control System) που χρησιμοποιείται για την αποθήκευση και διαχείριση έργων τοπικά για την ανάπτυξη λογισμικού. Είναι γρήγορο και ικανό να διαχειρίζεται μεγάλα έργα. Αναπτύχθηκε το 2005 από την κοινότητα ανάπτυξης του Linux και κυρίως από τον Linus Torvalds [58].

Για τη χρήση του Git απαιτείται η εγκατάσταση του τοπικά. Ακόμα, ο φάκελος του project που θέλουμε να το χρησιμοποιήσουμε, θα πρέπει να αρχικοποιηθεί ως Git repository (Git αποθετήριο) με την εντολή **git init**. Να σημειωθεί ότι οι Git εντολές ορίζονται μέσω της γραμμής εντολών.

Το Git προσφέρει τη δυνατότητα στους προγραμματιστές να παρακολουθούν τις αλλαγές σε αρχεία και να τις καταγράφουν. Γενικά, οι αλλαγές που καταγράφονται από τους προγραμματιστές ονομάζονται commits. Βέβαια, πριν γίνει ένα commit, ο προγραμματιστής πρέπει να αναφέρει στο Git ότι ένα αρχείο ή ένα σύνολο αρχείων είναι τροποποιημένα και έτοιμα για commit. Η αναφορά αυτή πραγματοποιείται με την εντολή **git add name** για ένα αρχείο, όπου το name προσδιορίζει το όνομα του αρχείου ή με την εντολή **git add --all** για το σύνολο αρχείων. Μετά την αναφορά, πραγματοποιούμε το commit με την εντολή **git commit –m "message"**, όπου το message είναι ένα μήνυμα που προσδιορίζει την αλλαγή. Το Git διατηρεί ένα αρχείο με το ιστορικό των καταγεγραμμένων commits και κάποιες πληροφορίες,

όπως ποιος πραγματοποίησε ένα commit καθώς και την ημερομηνία που πραγματοποιήθηκε. Επίσης, υπάρχει η δυνατότητα μετάβασης των αρχείων σε προηγούμενες εκδόσεις, δηλαδή η μετάβαση σε προηγούμενο commit.

Ένα έργο μπορεί να αποθηκευτεί και απομακρυσμένα σε έναν online πάροχο που φιλοξενεί Git αποθετήρια όπως είναι το GitLab ή το GitHub. Αρχικά, χρειάζεται να συνδεθεί το τοπικό αποθετήριο (local repository) με το απομακρυσμένο αποθετήριο (remote repository) που έχει δημιουργηθεί στον αντίστοιχο πάροχο φιλοξενίας αποθετηρίων. Η σύνδεση πραγματοποιείται με την εντολή **git remote add origin URL**, όπου το URL προσδιορίζει τη διεύθυνση του απομακρυσμένου αποθετηρίου. Έπειτα, για την προώθηση των commits στο απομακρυσμένο αποθετήριο, χρησιμοποιείται η εντολή **git push origin**. Ακόμα, υπάρχει η δυνατότητα λήψης (pull) των τελευταίων αλλαγών ώστε να παραμείνουν ενημερωμένοι όλοι οι προγραμματιστές που δουλεύουν στο ίδιο έργο, ορίζοντας την εντολή **git pull**.

Το Git επιτρέπει τη δημιουργία διακλάδωσης (branch). Οι διακλαδώσεις είναι χρήσιμες όταν θέλουμε να προσθέσουμε στο έργο μας μια καινούρια δυνατότητα ή να διορθώσουμε ένα σφάλμα χωρίς να τροποποιήσουμε τον κύριο κώδικα του έργου, δηλαδή τη κύρια διακλάδωση (master branch). Τέλος, όταν ολοκληρώσουμε τις αλλαγές, τότε συγχωνεύουμε (merge) τη δημιουργημένη διακλάδωση με την κύρια διακλάδωση.

5.8 GitLab

Εικόνα 5.7 Λογότυπο GitLab

Το **GitLab** είναι μια διαδικτυακή πλατφόρμα DevOps που βασίζεται στο σύστημα Git και φιλοξενεί δωρεάν δημόσια ή ιδιωτικά απομακρυσμένα αποθετήρια [59]. Με λίγα λόγια, το GitLab χρησιμοποιεί το Git και τις δυνατότητες του. Εκτός από τις δυνατότητες του Git, το GitLab προσφέρει επιπλέον δυνατότητες για να βελτιώσει την ανάπτυξη λογισμικού. Διευκολύνει τη συνεργασία μεταξύ πολλών προγραμματιστών που δουλεύουν στο ίδιο έργο. Ειδικότερα, τους συντονίζει και συγχωνεύει τον πηγαίο κώδικα των αρχείων. Τα

απομακρυσμένα αποθετήρια είναι διαθέσιμα διαδικτυακά για προβολή από όλους τους προγραμματιστές και διαχειρίσιμα από αυτούς που έχουν δικαιώματα.

Οι προγραμματιστές έχουν τη δυνατότητα να δημιουργούν issues (ζητήματα), τα οποία χρησιμοποιούνται για πολλούς σκοπούς όπως για την ανταλλαγή ιδεών, την επίλυση προβλημάτων, την οργάνωση της ροής εργασίας και την παρακολούθηση των εργασιών. Επίσης, χρησιμοποιούνται για ερωτήσεις, αιτήματα υποστήριξης και αναφορές σφαλμάτων. Τέλος, υπάρχει η δυνατότητα τροποποίησης και ολοκλήρωσης των υπαρχόντων ζητημάτων [60].

5.9 JSON

Το **JSON (JavaScript Object Notation)** είναι ένα απλό πρότυπο ανταλλαγής δεδομένων, που χρησιμοποιείται από όλες τις διάσημες γλώσσες προγραμματισμού. Είναι ελαφρύ και ανεξάρτητο από γλώσσες προγραμματισμού. Οι άνθρωποι μπορούν να το αναγνώσουν και να το συντάξουν εύκολα. Ακόμα, οι μηχανές μπορούν να το αναλύσουν και να το δημιουργήσουν [61]. Συνήθως, χρησιμοποιείται για την αποθήκευση καθώς και τη μεταφορά δεδομένων μεταξύ εφαρμογής ιστού (web application) και διακομιστή (server).Το JSON ανακαλύφθηκε από τον Douglas Crockford στις αρχές του 2000. Ο ίδιος ισχυρίζεται ότι δεν εφηύρε το JSON αλλά το βρήκε, το ονόμασε και περιέγραψε τη χρησιμότητα του [13].

Η **σύνταξη του JSON** βασίζεται σε ένα υποσύνολο της σύνταξης της JavaScript. Ένα αρχείο JSON αποθηκεύεται με επέκταση **.json** και τα δεδομένα του για να είναι έγκυρα πρέπει να είναι σε δύο δομές δεδομένων:

- Μία συλλογή από ζεύγη ονόματος/τιμής, όπου περικλείεται από curly braces **{ }**. Για παράδειγμα, αυτό μοιάζει όπως ένα αντικείμενο στην JavaScript.
- Μία ταξινομημένη λίστα τιμών, όπου περικλείεται από square brackets **[ ]**. Για παράδειγμα, αυτό μοιάζει όπως ένας πίνακας στην JavaScript.

Τα δεδομένα ορίζονται ως ζεύγη ονόματος/τιμής. Ένα ζεύγος αποτελείται από το όνομα και μία τιμή, που διαχωρίζονται μεταξύ τους με άνω και κάτω τελεία (**:**). Το όνομα είναι τύπου string και περικλείεται από διπλά «αυτάκια». Από την άλλη, η τιμή μπορεί να είναι μία από

τους ακόλουθους τύπους δεδομένων: string, number, boolean, object, array, null. Επίσης, τα ζεύγη διαχωρίζονται μεταξύ τους με κόμμα (,).

```
{
    "firstName" : "Unknown",
    "lastName" : "Programmer",
    "legs" : 2,
    "programmingLanguages" : [
        "JavaScript",
        "C++",
        "Python"
    ]
}
```

5.10 JSON Schema

Το **JSON Schema** είναι μία προδιαγραφή που καθορίζει τη δομή και σημασιολογία των αντικειμένων JSON. Είναι το ίδιο αντικείμενο JSON, το οποίο προσφέρει σαφή τεκμηρίωση και είναι ευανάγνωστο τόσο από τους ανθρώπους όσο και από τις μηχανές. Περιγράφει τα δεδομένα ενός αρχείου JSON και καθορίζει τις ιδιότητες του [62] [63]. Συγκεκριμένα, παρέχει περιορισμούς, για παράδειγμα αν απαιτείται (required) μία λέξη-κλειδί. Ακόμα, ορίζει τύπους για τις τιμές των λέξεων-κλειδιών, για παράδειγμα η τιμή είναι τύπου πίνακα ("type" : "array").

Το JSON Schema προσφέρει συντακτική και σημασιολογική επικύρωση για τα δεδομένα των αρχείων JSON, επειδή τα δεδομένα ενός JSON δεν παρέχουν σημασιολογική επικύρωση. Η επικύρωση μπορεί να πραγματοποιηθεί από διαδικτυακούς αυτόματους validators, απλώς υποβάλλοντας το αρχείο JSON Schema και το αρχείο JSON που θα επικυρωθεί. Επίσης, μπορεί να χρησιμοποιηθεί για την επικύρωση των δεδομένων που έχουν υποβληθεί από έναν πελάτη.

Ένα βασικό JSON Schema περιέχει στην αρχή τις ακόλουθες λέξεις-κλειδιά: "$schema", "$id", "title", "description", "type" και έπειτα καθορίζονται οι ιδιότητες [64].

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$id": "https://example.com/product.schema.json",
  "title": "JSON Schema Example",
```

```
  "description": "A programmer",
  "type": "object",
  "properties": {
    "firstName": {
      "description": "First Name of Programmer",
      "type": "string"
    },
    "lastName": {
      "type": "string"
    },
    "legs": {
      "type": "integer"
    }
  },
  "required": ["firstName", "lastName"]
}
```

5.11 REST

Το **REST (REpresentational State Transfer)** είναι ένα αρχιτεκτονικό στυλ που ορίζει ένα σύνολο από περιορισμούς και χρησιμοποιείται στην ανάπτυξη διαδικτυακών υπηρεσιών. Παρουσιάστηκε από τον Roy Fielding στη διδακτορική του διατριβή το 2000 [65].

Οι RESTful εφαρμογές χρησιμοποιούν συνήθως τις HTTP μεθόδους για την επικοινωνία μεταξύ πελάτη-διακομιστή (client-server), οι οποίες HTTP μέθοδοι αντιστοιχούν στις λειτουργίες CRUD (Create, Read, Update, Delete) και οι πιο συνηθισμένες είναι οι POST, GET, PUT, DELETE. Για παράδειγμα, η POST μέθοδος δημιουργεί έναν καινούριο πόρο, η GET διαβάζει έναν πόρο ή μια συλλογή πόρων, η PUT ενημερώνει έναν υπάρχοντα συγκεκριμένο πόρο και τέλος η DELETE διαγράφει έναν υπάρχοντα συγκεκριμένο πόρο [66].

Οι πόροι (resources) μαζί με τις αναπαραστάσεις πόρων (representations of resources) είναι από τις πιο βασικές έννοιες στην REST αρχιτεκτονική. Ένας **πόρος** είναι οποιαδήποτε πληροφορία που μπορεί να ονομαστεί σε ένα REST σύστημα, για παράδειγμα μια εικόνα, ένα βιβλίο, μία συλλογή χρηστών. Γενικά, κάθε πόρος αντιμετωπίζεται ως περιεχόμενο, παρέχεται από τον διακομιστή και προσδιορίζεται μέσω μοναδικών URI. Ακόμα, η ονομασία των πόρων ορίζεται ως ουσιαστικό αντί για ρήμα, για παράδειγμα το URI test.com/posts είναι σωστό και όχι το test.com/getPosts. Η **αναπαράσταση ενός πόρου** είναι η κατάσταση που βρίσκεται ο πόρος μια συγκεκριμένη χρονική στιγμή και αποστέλλεται μεταξύ πελάτη-διακομιστή. Επίσης,

είναι συνήθως σε μορφή JSON ή XML και αποτελείται από δεδομένα, μεταδεδομένα και μερικές φορές σχεσιακούς συνδέσμους.

Ένα API για να θεωρείται **REST API** ή **RESTful API**, πρέπει να τηρεί τους ακόλουθους περιορισμούς [67]:

- **Uniform Interface**. Η ομοιόμορφη διεπαφή είναι ο κύριος περιορισμός που απαιτείται για τον σχεδιασμό ενός RESTful συστήματος και ορίζει τη διεπαφή μεταξύ πελάτη-διακομιστή. Μία ομοιόμορφη διεπαφή μπορεί να επιτευχθεί τηρώντας τους ακόλουθους περιορισμούς:
    - **Αναγνώριση πόρων σε αιτήματα**. Οι πόροι προσδιορίζονται μοναδικοί σε αιτήματα μέσω URI και οι έννοιες μεταξύ πόρων και αναπαραστάσεων των πόρων διαχωρίζονται.
    - **Χειρισμός πόρων μέσω αναπαραστάσεων**. Σε περίπτωση που οι πελάτες διαθέτουν ένα αναγνωριστικό πόρου μαζί με μεταδεδομένα, μπορούν να τροποποιήσουν ή να διαγράψουν τον πόρο στον διακομιστή, εφόσον έχουν το δικαίωμα.
    - **Αυτοπεριγραφικά μηνύματα**. Κάθε μήνυμα περιέχει αρκετές πληροφορίες ώστε να περιγράψει τον τρόπο επεξεργασίας του στον πελάτη.
    - **Τα υπερμέσα ως μηχανισμός κατάστασης της εφαρμογής (HATEOAS)**. Οι πελάτες βρισκόμενοι στην αρχική σελίδα, μπορούν να πλοηγηθούν και να ανακαλύψουν δυναμικά τους διαθέσιμους πόρους μέσω υπερσυνδέσμων. Οι υπερσύνδεσμοι διατίθενται από τον διακομιστή και περιλαμβάνονται στην απάντηση.
- **Stateless**. Κάθε αίτημα (request) που υποβάλλει ο πελάτης στον διακομιστή είναι ξεχωριστό από τα υπόλοιπα και πρέπει να περιλαμβάνει όλες τις απαραίτητες πληροφορίες. Με λίγα λόγια, οι πληροφορίες δεν αποθηκεύονται στον διακομιστή και έτσι βελτιώνεται η απόδοση.
- **Cacheable**. Ο πελάτης έχει τη δυνατότητα να αποθηκεύει προσωρινά τις απαντήσεις (responses) στην κρυφή μνήμη. Βέβαια, οι απαντήσεις πρέπει να ορίζουν τον εαυτό τους ως προσωρινά αποθηκευμένες ή μη προσωρινά αποθηκευμένες. Έτσι, αποφεύγονται μερικές περιττές αλληλεπιδράσεις μεταξύ πελάτη-διακομιστή με

αποτέλεσμα να βελτιώνεται η απόδοση (performance) και η επεκτασιμότητα (scalability).

- **Client-Server**. Ο πελάτης με τον διακομιστή διαχωρίζονται μεταξύ τους και ο καθένας ασχολείται με τη δική του δουλειά. Για παράδειγμα, ο πελάτης ασχολείται με τη διεπαφή χρήστη ενώ ο διακομιστής με την αποθήκευση των δεδομένων. Έτσι, ο πελάτης και ο διακομιστής μπορούν να αναπτύσσονται και να επεκτείνονται ανεξάρτητα με την προϋπόθεση όμως ότι δεν έχει υποστεί τροποποίηση η διεπαφή.
- **Layered System**. Είναι ένα πολυεπίπεδο σύστημα, το οποίο αποτελείται από πολλά ιεραρχικά επίπεδα και κάθε στοιχείο μπορεί να αλληλεπιδράσει μόνο με το πάνω ή κάτω επίπεδο. Έτσι, το πολυεπίπεδο σύστημα προσφέρει καλύτερη ασφάλεια και επεκτασιμότητα.
- **Code on demand** (προαιρετικό). Ο διακομιστής έχει τη δυνατότητα να επεκτείνει τη λειτουργικότητα του πελάτη, μεταφέροντας εκτελέσιμο κώδικα, για παράδειγμα σενάρια JavaScript.

5.12 mockAPI

Το **mockAPI** είναι ένα εργαλείο που επιτρέπει τη δημιουργία εικονικών APIs και χρησιμοποιείται για εκμάθηση και δοκιμές. Παρέχει απλή μοντελοποίηση δεδομένων, η οποία περιλαμβάνει τη γρήγορη δημιουργία πόρων και τον καθορισμό των σχέσεων τους. Υποστηρίζει την παραγωγή δεδομένων χειροκίνητα ή αυτόματα, χρησιμοποιώντας τη βιβλιοθήκη faker.js [68] [69]. Γενικά, είναι ένα χρήσιμο εργαλείο καθώς επιτρέπει στους front-end προγραμματιστές να ολοκληρώσουν τη δουλειά τους, χωρίς να περιμένουν την ανάπτυξη του πραγματικού API από τους back-end προγραμματιστές και αυτό έχει ως αποτέλεσμα να αυξάνεται ο ρυθμός ανάπτυξης ενός έργου.

Οι προγραμματιστές μπορούν εύκολα να εκτελέσουν λειτουργίες CRUD στο mockAPI μέσω της RESTful διεπαφής και να προσαρμόσουν τις απαντήσεις που αποστέλλονται πίσω, δηλαδή το JSON αρχείο. Επίσης, μπορούν να προσθέσουν παραμέτρους ερωτήματος (query parameters) στα αιτήματα GET για σελιδοποίηση, ταξινόμηση, φιλτράρισμα και αναζήτηση.

Το mockAPI είναι ιδανικό για ομαδική εργασία, καθώς επιτρέπει στους προγραμματιστές να προσθέσουν τους συνεργάτες τους σε ένα έργο. Έπειτα, οι συνεργάτες έχουν τη δυνατότητα να δημιουργήσουν, να ενημερώσουν και να διαγράψουν τους πόρους στο έργο. Ακόμα, ένα έργο μπορεί να διαμοιραστεί για κλωνοποίηση μέσω ενός συνδέσμου.

Το mockAPI διατίθεται δωρεάν ή επί πληρωμή. Η δωρεάν έκδοση επιτρέπει τη δημιουργία μόνο ενός έργου και μέχρι τέσσερις πόρους. Από την άλλη, η έκδοση επί πληρωμή επιτρέπει τη δημιουργία είκοσι έργων και πενήντα πόρων ανά έργο.

5.13 Axios

Το **Axios** είναι μία απλή δημοφιλής βιβλιοθήκη που χρησιμοποιείται για τη δημιουργία και αποστολή HTTP αιτημάτων (HTTP requests) προς ένα API ή έναν εξυπηρετητή (server). Συγκεκριμένα, είναι ένας HTTP πελάτης που βασίζεται στο promise χαρακτηριστικό της ES6 (promise-based HTTP client), ο οποίος εκτελείται στο πρόγραμμα περιήγησης και στο node.js [70]. Δεδομένου ότι βασίζεται στο promise, οι προγραμματιστές μπορούν να δημιουργήσουν τα αιτήματα χρησιμοποιώντας τη σύνταξη async-await, η οποία προσφέρει πιο ευανάγνωστο ασύγχρονο κώδικα [71]. Το Axios χρησιμοποιεί τα XMLHttpRequests για τη δημιουργία των HTTP αιτημάτων στο πρόγραμμα περιήγησης, ενώ στο node.js χρησιμοποιεί το module http του node.js. Έχει επηρεαστεί από την υπηρεσία $http του AngularJS αλλά χρησιμοποιείται και σε εφαρμογές που έχουν δημιουργηθεί από άλλα JS frameworks, για παράδειγμα React εφαρμογές, Vue εφαρμογές. Γενικά, η δημιουργία των αιτημάτων πραγματοποιείται με λιγότερο κώδικα σε αντίθεση με εναλλακτικά, όπως για παράδειγμα το fetch API και ακόμα διευκολύνει την εκτέλεση των CRUD λειτουργιών.

Το Axios διαθέτει ένα σύνολο χαρακτηριστικών. Συγκεκριμένα, μετατρέπει αυτόματα τα JSON δεδομένα, επιτρέπει την ακύρωση των αιτημάτων, επιτρέπει την αποστολή πολλαπλών ταυτόχρονων αιτημάτων, προσφέρει προστασία στην πλευρά του πελάτη έναντι πλαστογράφησης αιτημάτων μεταξύ ιστοτόπων (CSRF) και παρέχει τη δυνατότητα μετασχηματισμού των δεδομένων αιτήματος (request) ή απάντησης (response). Επίσης, παρέχει καλό χειρισμό σφαλμάτων και μπορεί να εμφανίσει σφάλματα εμβέλειας 400 και 500.

Το Axios παρέχει μεθόδους που χρησιμοποιούνται για την εκτέλεση των HTTP αιτημάτων. Οι πιο ευρέως χρησιμοποιούμενες μέθοδοι είναι:

- **axios.post( )** . Εκτελεί ένα POST αίτημα και χρησιμοποιείται για τη δημιουργία ενός νέου πόρου.
- **axios.get( )** . Εκτελεί ένα GET αίτημα και χρησιμοποιείται για την ανάγνωση ενός πόρου ή μιας συλλογής πόρων.
- **axios.put( )** . Εκτελεί ένα PUT αίτημα και χρησιμοποιείται για την ενημέρωση ενός υπάρχοντα πόρου.
- **axios.delete( )** . Εκτελεί ένα DELETE αίτημα και χρησιμοποιείται για τη διαγραφή ενός υπάρχοντα πόρου.

Το Axios μπορεί να εγκατασταθεί σε μία εφαρμογή μέσω του διαχειριστή πακέτων npm ή yarn ή bower. Διαφορετικά, μπορεί να εγκατασταθεί με χρήση CDN. Τέλος, προκειμένου να χρησιμοποιηθεί σε ένα αρχείο, πρέπει πρώτα να εισαχθεί με την εντολή **import axios from 'axios';**.

5.14 Figma

Το **Figma** είναι μία διαδικτυακή πλατφόρμα σχεδιασμού που βασίζεται στο cloud και χρησιμοποιείται για τον σχεδιασμό διεπαφών χρήστη (UI) και εμπειρίας χρήστη (UX) [72]. Παρέχει στους σχεδιαστές και προγραμματιστές μία ποικιλία εργαλείων σχεδιασμού με τα οποία έχουν τη δυνατότητα να σχεδιάσουν wireframes και mockups ιστοσελίδων ή εφαρμογών. Ακόμα, μπορούν να δημιουργήσουν πρωτότυπα (prototypes) και να ελέγξουν τη λειτουργία τους πριν ξεκινήσουν την υλοποίηση του έργου τους.

Το Figma περιέχει έναν διαδικτυακό πίνακα που ονομάζεται FigJam, ο οποίος είναι ιδανικός για ομαδικές συναντήσεις, προτάσεις ιδεών, σχεδιασμό πλάνων και διαγραμμάτων. Ένα από τα πιο σημαντικά πλεονεκτήματα του Figma είναι η συνεργασία. Πιο συγκεκριμένα, οι ομάδες των χρηστών μπορούν να συνεργάζονται και να σχεδιάζουν μαζί στην ίδια σελίδα σε πραγματικό χρόνο. Έτσι, αυξάνεται η αποτελεσματικότητα, εξοικονομείται χρόνος και υπάρχει συνέπεια. Η κοινότητα του Figma διαθέτει ένα ευρύ φάσμα από έτοιμα αρχεία, components, πρότυπα (templates), widgets και πρόσθετα (plugins), τα οποία μπορούν να εκμεταλλευτούν οι χρήστες ώστε να επιταχύνουν τον σχεδιασμό των έργων ή των

διαγραμμάτων τους. Επίσης, ο καθένας μπορεί να δημιουργήσει τα δικά του και να τα προμηθεύσει στην κοινότητα.

Το Figma μπορεί να χρησιμοποιηθεί από το πρόγραμμα περιήγησης αλλά και από εφαρμογές που είναι διαθέσιμες για λήψη από την επίσημη ιστοσελίδα. Για την ακρίβεια υπάρχει εφαρμογή για desktop και εφαρμογή για κινητό. Τέλος, διαθέτει δωρεάν έκδοση αλλά και διάφορες επί πληρωμή που προσφέρουν περισσότερες δυνατότητες.

Κεφάλαιο 6 – Ανάπτυξη Εφαρμογής Ιστού

Στο κεφάλαιο αυτό πραγματοποιείται η ανάλυση, ο σχεδιασμός και η παρουσίαση της εφαρμογής ιστού. Επίσης αναφέρονται οι γλώσσες και οι τεχνολογίες που χρησιμοποιήθηκαν. Συγκεκριμένα, αναπτύξαμε το front-end κομμάτι της εφαρμογής, δηλαδή τη διεπαφή χρήστη που βλέπουν και αλληλεπιδρούν οι χρήστες.

6.1 Ανάλυση Εφαρμογής

6.1.1 Περιγραφή Οντότητας Εκπαιδευτή Ενηλίκων

Ένας **εκπαιδευτής ενηλίκων** συνεργάζεται με παρόχους κατάρτισης της μη τυπικής εκπαίδευσης όπως ΙΕΚ (Ινστιτούτο Επαγγελματικής Κατάρτισης), ΚΔΒΜ (Κέντρα Δια Βίου Μάθησης), ΚΕΔΙΒΙΜ (Κέντρο Επιμόρφωσης και Δια Βίου Μάθησης) και εκπαιδεύει τους καταρτιζόμενους στο πλαίσιο ενός προγράμματος κατάρτισης στο οποίο αυτοί συμμετέχουν.

Ο εκπαιδευτής μπορεί να διαθέτει Πιστοποίηση Εκπαιδευτικής Επάρκειας από τον ΕΟΠΠΕΠ (Εθνικός Οργανισμός Πιστοποίησης Προσόντων & Επαγγελματικού Προσανατολισμού), η οποία πιστοποιεί ότι διαθέτει τις απαραίτητες γνώσεις, δεξιότητες και ικανότητες, προκειμένου να διδάξει. Κατά την πιστοποίηση ο ΕΟΠΠΕΠ αποδίδει στον εκπαιδευτή Κωδικούς ΣΤΕΠ (ταξινομικό σύστημα επαγγελμάτων της Ελληνικής Στατιστικής Αρχής) που αντιστοιχούν στην ειδικότητα/εξειδίκευση του. Επιπλέον, ο εκπαιδευτής μπορεί να είναι εγγεγραμμένος σε ένα ή περισσότερα μητρώα εκπαιδευτών, ανεξάρτητα από το αν διαθέτει πιστοποίηση από τον ΕΟΠΠΕΠ.

Η βασική οντότητα της εφαρμογής είναι ο εκπαιδευτής ενηλίκων. Η **οντότητα εκπαιδευτής ενηλίκων** περιγράφεται με τα εξής στοιχεία:

- **Κωδικός (code)**. Δηλώνει τον μοναδικό κωδικό που έχει ο εκπαιδευτής στην εφαρμογή.
- **Όνομα (name)**
    - **Όνομα (first)**. Δηλώνει το όνομα του εκπαιδευτή.
    - **Επώνυμο (last)**. Δηλώνει το επώνυμο του εκπαιδευτή.

    - **Πατρώνυμο (father)**. Δηλώνει το όνομα του πατέρα του εκπαιδευτή.
- **Επάγγελμα (occupation)**. Δηλώνει το ασκούμενο επάγγελμα του εκπαιδευτή.
- **Στοιχεία Επικοινωνίας (contact)**
    - **Διεύθυνση (address)**
        - **Οδός (street).** Δηλώνει την οδό της διεύθυνσης που διαμένει ο εκπαιδευτής.
        - **Αριθμός (number)**. Δηλώνει τον αριθμό της οδού που διαμένει ο εκπαιδευτής.
        - **Δήμος (municipality)**. Δηλώνει το δήμο που διαμένει ο εκπαιδευτής.
            - **Κωδικός (code)**
            - **Όνομα (name)**
    - **Τηλέφωνο (phone)**. Δηλώνει το τηλέφωνο επικοινωνίας του εκπαιδευτή.
    - **Email (email)**. Δηλώνει τη διεύθυνση ηλεκτρονικού ταχυδρομείου του εκπαιδευτή.
- **Στοιχεία Πιστοποίησης ΕΟΠΠΕΠ (certification)**
    - **Κωδικός Πιστοποίησης (code)**. Δηλώνει τον κωδικό της πιστοποίησης ΕΟΠΠΕΠ του εκπαιδευτή.
    - **Κωδικοί ΣΤΕΠ (step) [ ]**. Δηλώνουν τους κωδικούς ΣΤΕΠ που έχει αποδώσει ο ΕΟΠΠΕΠ στον εκπαιδευτή, οι οποίοι αντιστοιχούν στην ειδικότητα/εξειδίκευση του.
- **Μητρώα (registries) [ ]**. Δηλώνουν τα μητρώα εκπαιδευτών στα οποία είναι εγγεγραμμένος ο εκπαιδευτής.
    - **Μητρώο (registry)**
        - **Όνομα Μητρώου (name)**. Δηλώνει το αντίστοιχο όνομα του μητρώου εκπαιδευτών στο οποίο είναι εγγεγραμμένος ο εκπαιδευτής.
    - **ΑΜ / Κωδικός εκπαιδευτή στο Μητρώο (code)**. Δηλώνει τον αντίστοιχο αριθμό μητρώου/κωδικό του εκπαιδευτή στο μητρώο εκπαιδευτών που είναι εγγεγραμμένος.

6.1.2 Απαιτήσεις εφαρμογής

Η εφαρμογή απευθύνεται σε μία κατηγορία χρηστών, η οποία είναι οι διαχειριστές. Οι απαιτήσεις της εφαρμογής διακρίνονται σε δύο κατηγορίες, στις λειτουργικές και στις μη λειτουργικές. Οι λειτουργικές απαιτήσεις δηλώνουν τις λειτουργικές δυνατότητες ή υπηρεσίες που πρέπει να παρέχει η εφαρμογή στους διαχειριστές καθώς και τη συμπεριφορά της σε διάφορες καταστάσεις. Οι μη λειτουργικές απαιτήσεις ορίζουν τις ιδιότητες και τους περιορισμούς που πρέπει να τηρεί η εφαρμογή.

Οι **λειτουργικές απαιτήσεις** της εφαρμογής είναι οι εξής:

- Οι διαχειριστές πρέπει να έχουν τη δυνατότητα να εισάγουν νέους εκπαιδευτές.
- Η εφαρμογή πρέπει να προβάλλει τη λίστα των εισαχθέντων εκπαιδευτών ταξινομημένη από τον πιο παλιό στον πιο πρόσφατο.
- Οι διαχειριστές πρέπει να μπορούν να προβάλλουν τα συνολικά στοιχεία των εκπαιδευτών.
- Οι διαχειριστές πρέπει να έχουν τη δυνατότητα να τροποποιούν τα στοιχεία των εκπαιδευτών.
- Η εφαρμογή πρέπει να επικυρώνει τα δεδομένα που εισάγουν οι διαχειριστές στα πεδία της φόρμας και να εμφανίζει μηνύματα σφάλματος στην εισαγωγή μη έγκυρων δεδομένων σε πραγματικό χρόνο.
- Η εφαρμογή πρέπει να επικυρώνει τα δεδομένα που εισάγουν οι διαχειριστές στα πεδία της φόρμας και να εμφανίζει μηνύματα σφάλματος στην εισαγωγή μη έγκυρων δεδομένων κατά την υποβολή.
- Η εφαρμογή πρέπει να αποθηκεύει τα στοιχεία των εκπαιδευτών που εισάγουν ή τροποποιούν οι διαχειριστές μόνο αν είναι έγκυρα.
- Οι διαχειριστές πρέπει να μπορούν να αναζητούν συγκεκριμένους εκπαιδευτές με βάση το Όνομα ή το Επώνυμο ή το Ονοματεπώνυμο ή τον Κωδικό ή τον Κωδικό ΕΟΠΠΕΠ ή τον Κωδικό ΣΤΕΠ ή τον Κωδικό Μητρώου.
- Οι διαχειριστές πρέπει να έχουν τη δυνατότητα να βρίσκουν συγκεκριμένους εκπαιδευτές, αναζητώντας και επιλέγοντας ένα συγκεκριμένο δήμο.
- Οι διαχειριστές πρέπει να μπορούν να ταξινομούν τη λίστα των εκπαιδευτών κατά Κωδικό Αύξουσα ή Κωδικό Φθίνουσα ή Επώνυμο Αύξουσα ή Επώνυμο Φθίνουσα.

- Οι διαχειριστές πρέπει να μπορούν να επιλέγουν και να βρίσκουν τους πιστοποιημένους ή τους μη πιστοποιημένους εκπαιδευτές.
- Οι διαχειριστές πρέπει να έχουν τη δυνατότητα να εκκαθαρίζουν τα φίλτρα που έχουν εφαρμόσει.
- Η εφαρμογή πρέπει να προβάλλει σελιδοποιημένη τη λίστα των εισαχθέντων εκπαιδευτών.
- Οι διαχειριστές πρέπει να μπορούν να περιηγηθούν και να προβάλλουν όλες τις σελίδες της λίστας των εισαχθέντων εκπαιδευτών μέσω της σελιδοποίησης.
- Η εφαρμογή πρέπει να εμφανίζει breadcrumbs για να πληροφορεί τους διαχειριστές σε ποιο σημείο βρίσκονται.
- Οι διαχειριστές πρέπει να έχουν τη δυνατότητα να πλοηγηθούν στις προηγούμενες σελίδες μέσω breadcrumbs.
- Η εφαρμογή πρέπει να εμφανίζει HTTP μηνύματα σφάλματος όταν τα HTTP αιτήματα αποτυγχάνουν.

Οι **μη λειτουργικές απαιτήσεις** της εφαρμογής είναι οι εξής:

- Η εφαρμογή πρέπει να είναι συνεχώς διαθέσιμη και λειτουργική με ελάχιστο χρόνο διακοπής λειτουργίας.
- Η εφαρμογή πρέπει να είναι προσβάσιμη από διαχειριστές που έχουν εκπαιδευτεί.
- Η εφαρμογή πρέπει να αποκρίνεται άμεσα στις ενέργειες των διαχειριστών και στον αυξημένο όγκο αιτημάτων.
- Η εφαρμογή πρέπει να υποστηρίζει πολλούς διαχειριστές ταυτόχρονα.
- Η εφαρμογή πρέπει να είναι εύκολη στη χρήση, στην εκμάθηση και στην πλοήγηση.
- Η εφαρμογή πρέπει να είναι φιλική προς τους διαχειριστές.
- Η εφαρμογή πρέπει να είναι ικανή για μελλοντικές αναπτύξεις και επεκτάσεις.
- Η εφαρμογή πρέπει να είναι ικανή για διορθώσεις ελαττωμάτων.
- Η εφαρμογή πρέπει να προσαρμόζεται δυναμικά σε διαφορετικά μεγέθη οθονών και συσκευών όπως desktops, laptops, tablets και κινητά τηλέφωνα.
- Η εφαρμογή πρέπει να μπορεί να χρησιμοποιείται στα προγράμματα περιήγησης Google Chrome, Mozilla Firefox, Microsoft Edge, Opera και Safari.

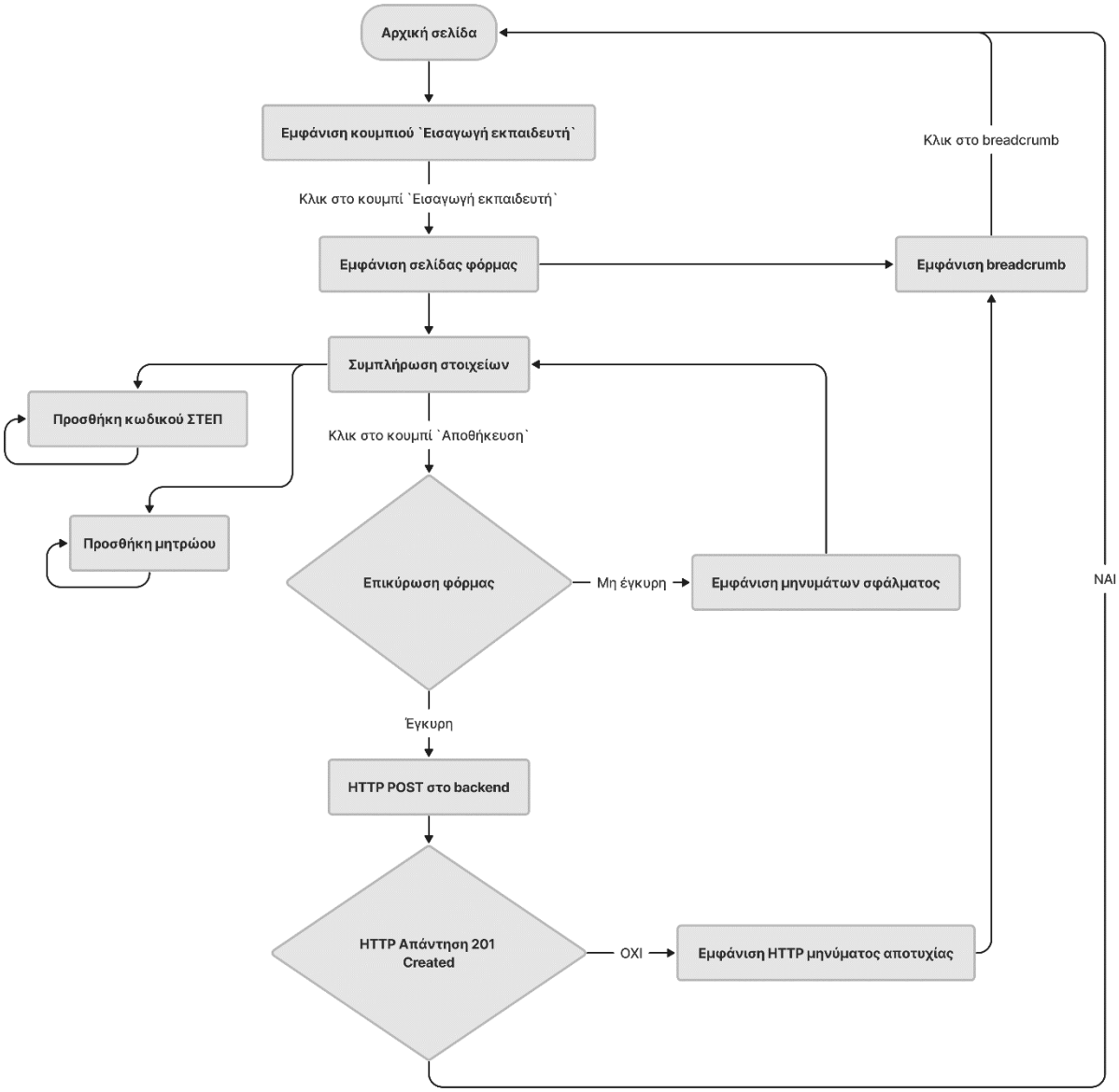
6.2 Σχεδιασμός Εφαρμογής

Το επόμενο στάδιο στην ανάπτυξη της εφαρμογής είναι ο σχεδιασμός της λαμβάνοντας υπόψη τις απαιτήσεις που καθορίστηκαν προηγουμένως στο στάδιο της ανάλυσης. Έτσι, σχεδιάστηκαν αρχικά τα **διαγράμματα ροής εργασίας (workflow diagrams)** και έπειτα τα **mockups** για τις σελίδες της διεπαφής χρήστη της εφαρμογής. Ο σχεδιασμός τους πραγματοποιήθηκε στο εργαλείο σχεδιασμού Figma που αναφέρθηκε στο προηγούμενο κεφάλαιο [72]. Ακόμα σχεδιάστηκε η αποθήκευση των δεδομένων.

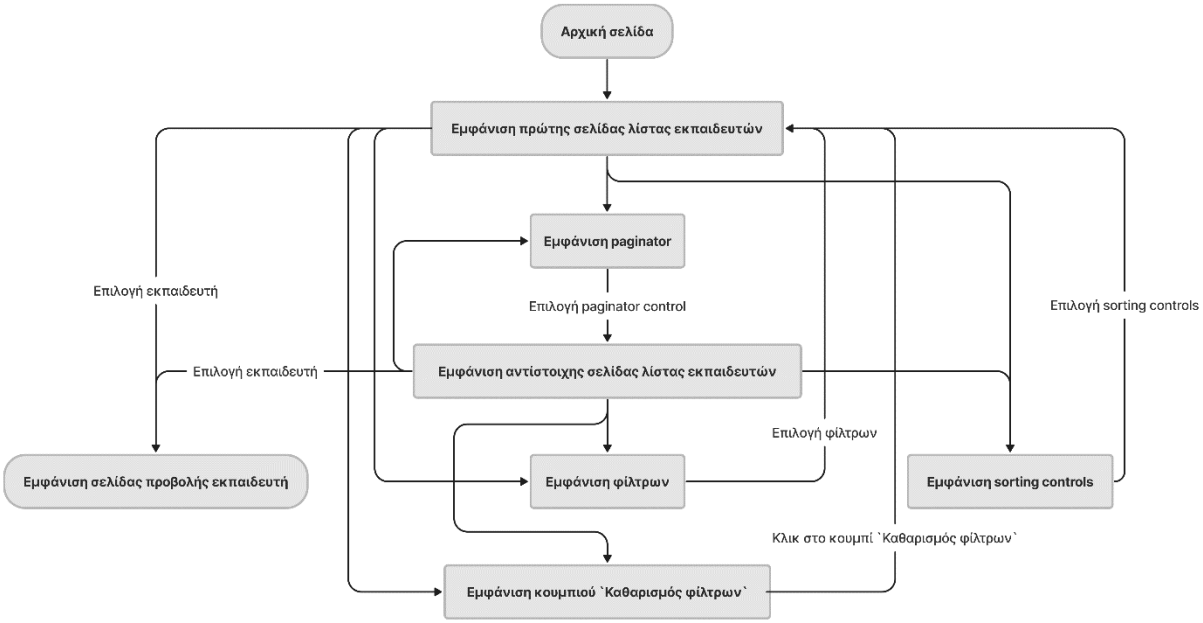
6.2.1 Διαγράμματα ροής εργασίας

Το διάγραμμα ροής εργασίας είναι μία οπτική αναπαράσταση της ροής μιας διαδικασίας. Συγκεκριμένα απεικονίζει τη σειρά των βημάτων που πρέπει να ακολουθήσουν οι προγραμματιστές, προκειμένου να υλοποιήσουν τις διαδικασίες της εφαρμογής τους. Επιπλέον, διευκολύνει τη δουλειά τους και τους βοηθάει να κατανοήσουν τους ρόλους και τις αρμοδιότητες τους. Τέλος, αυξάνει την παραγωγικότητα, την αποτελεσματικότητα, εξοικονομεί κόστος και χρόνο [73].

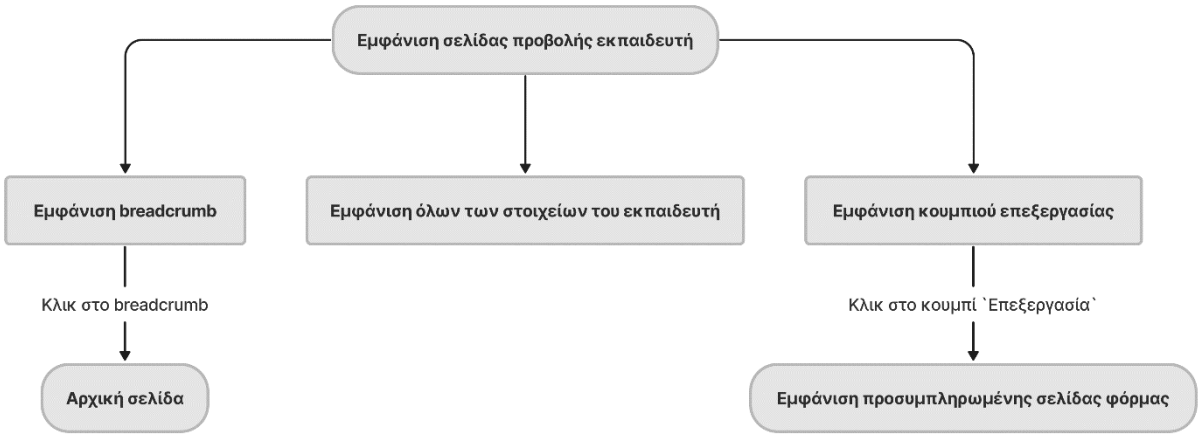
Τα διαγράμματα ροής εργασίας που σχεδιάστηκαν για τις σελίδες των λειτουργιών της εισαγωγής εκπαιδευτή, προβολής της λίστας των εκπαιδευτών, προβολής και επεξεργασίας των στοιχείων του εκπαιδευτή απεικονίζονται παρακάτω στις εικόνες 6.1, 6.2, 6.3 και 6.4.



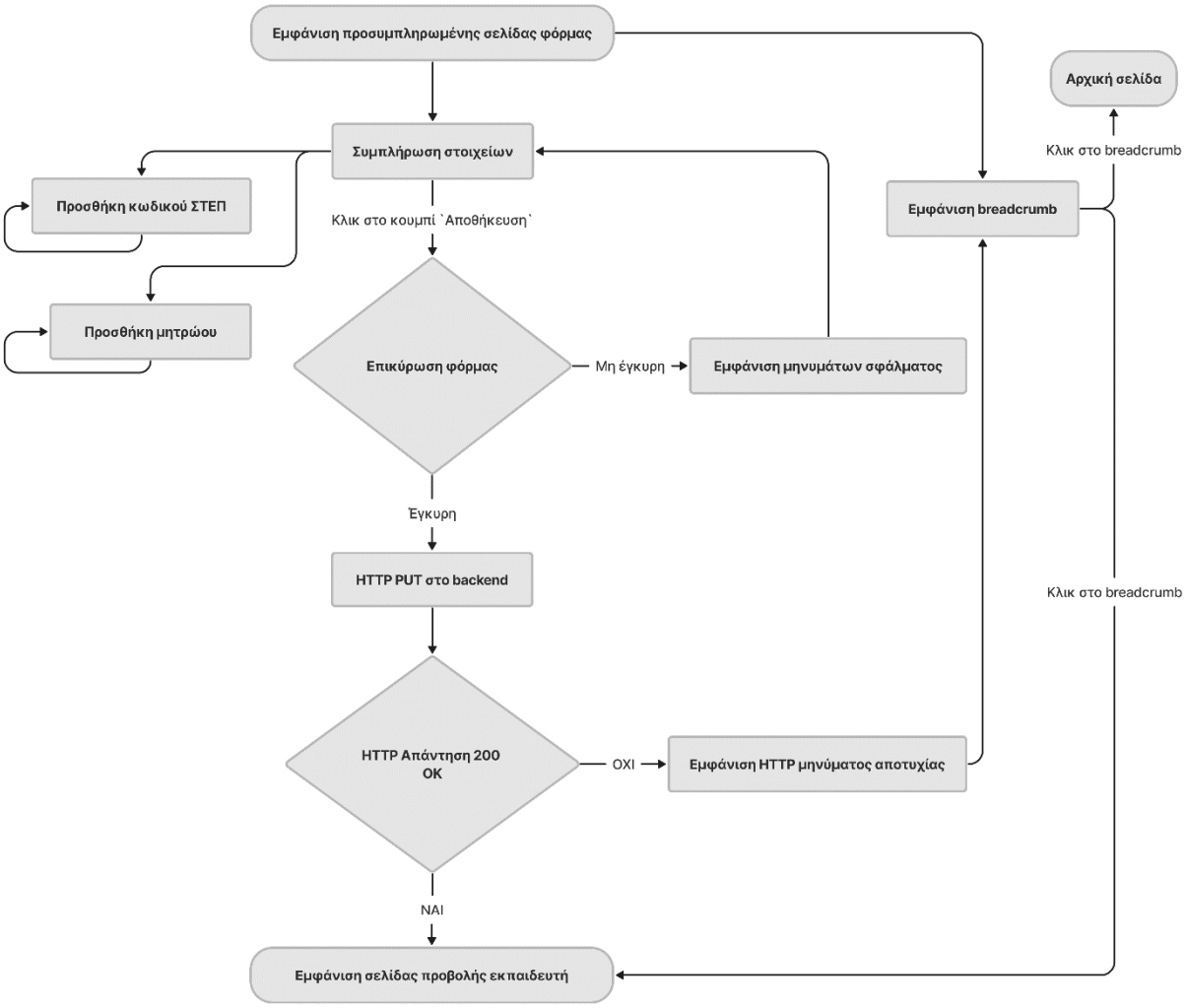
Εικόνα 6.1 Διάγραμμα ροής εργασίας για τη λειτουργία της εισαγωγής εκπαιδευτή



Εικόνα 6.2 Διάγραμμα ροής εργασίας για τη λειτουργία της προβολής της λίστας των εκπαιδευτών



Εικόνα 6.3 Διάγραμμα ροής εργασίας για τη λειτουργία της προβολής των στοιχείων του εκπαιδευτή

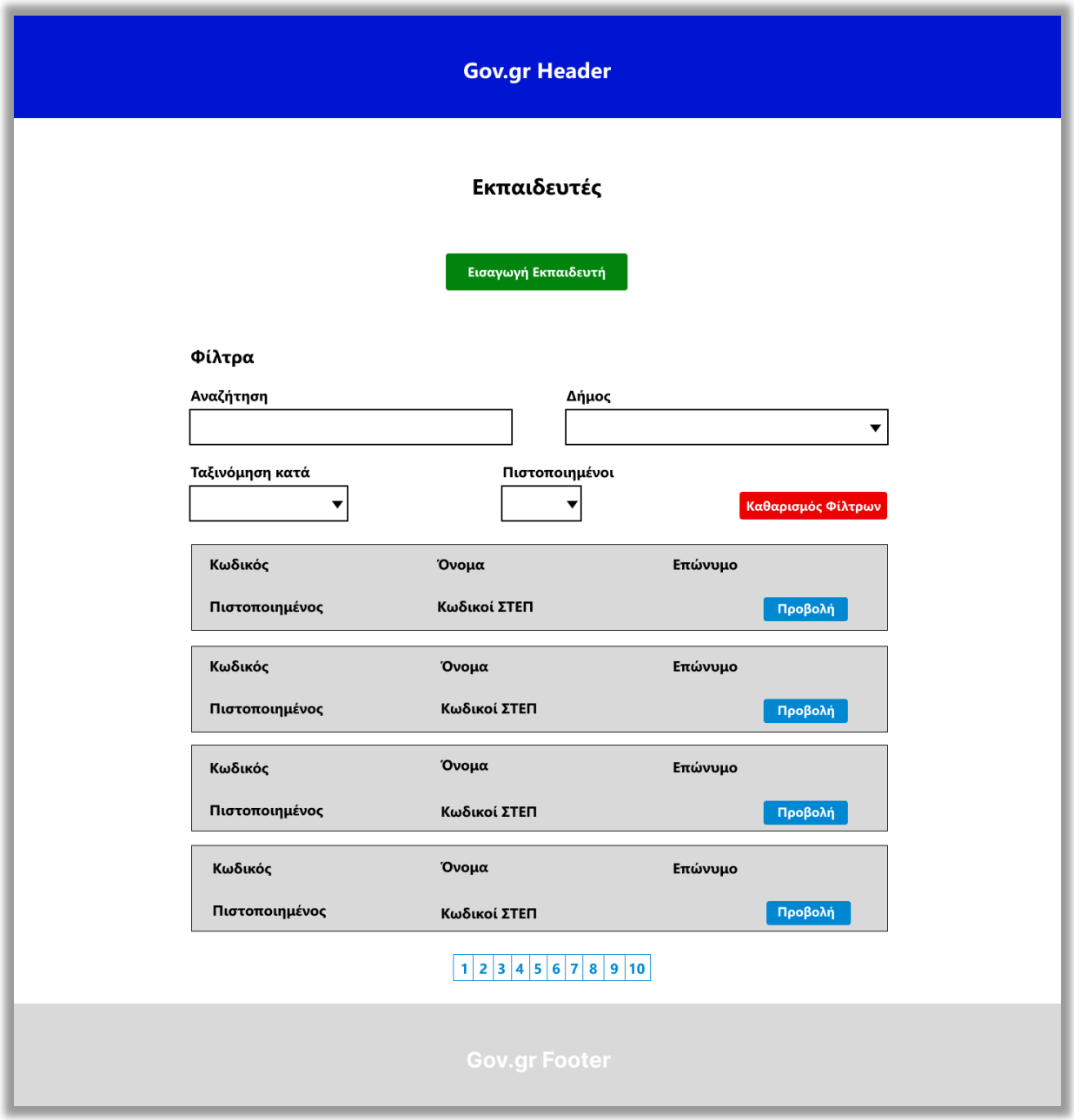


Εικόνα 6.4 Διάγραμμα ροής εργασίας για τη λειτουργία της επεξεργασίας των στοιχείων του εκπαιδευτή

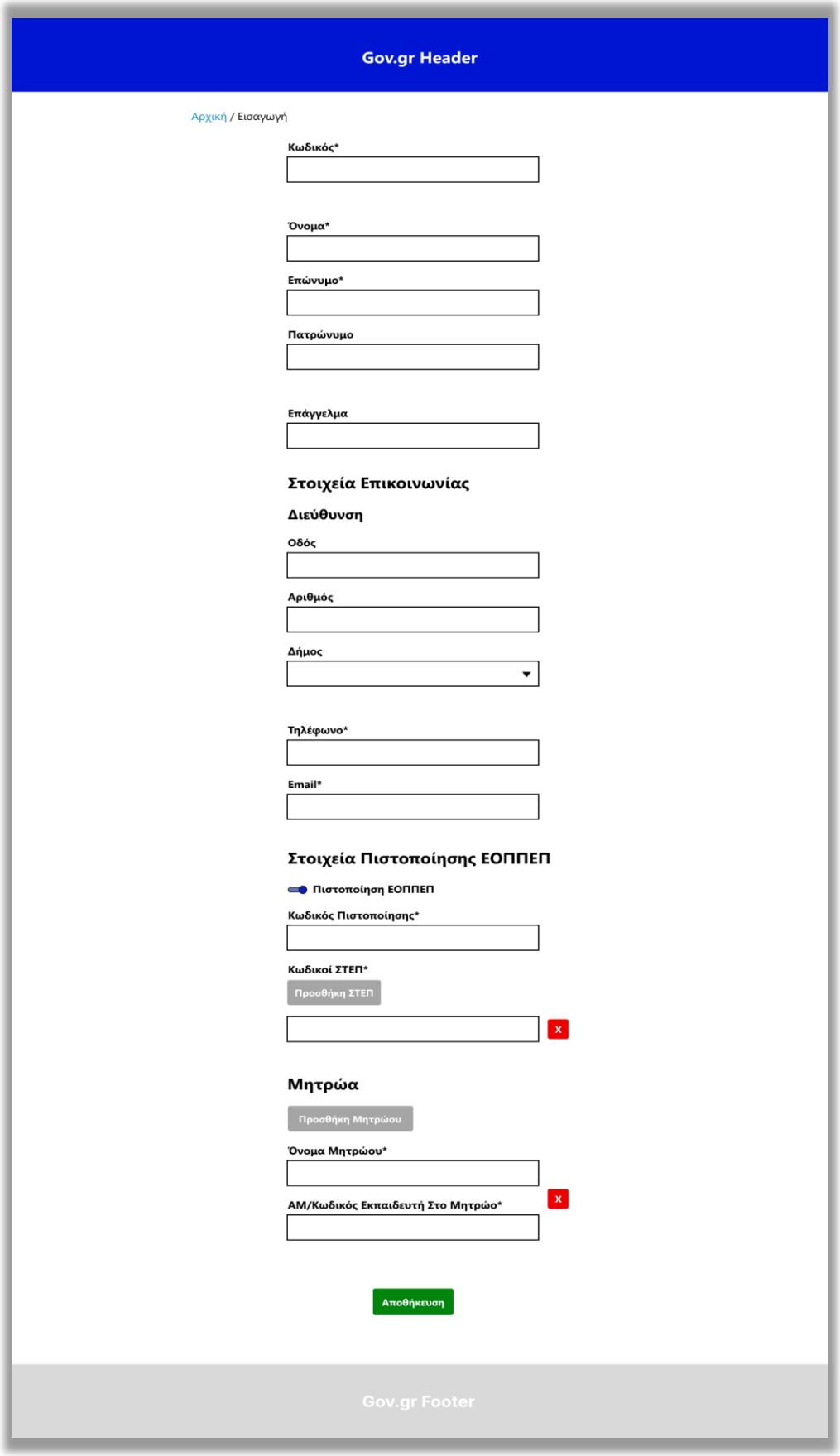
6.2.2 Mockups

Τα mockups είναι σχέδια που παρουσιάζουν το πως θα φαίνονται οι σελίδες της ιστοσελίδας ή εφαρμογής. Για την ακρίβεια δείχνουν τη δομή, τη διάταξη, την τυπογραφία, τα χρώματα, τα στοιχεία της καθώς και τη θέση τους στη σελίδα. Ο σχεδιασμός των mockups προσφέρει καλύτερη συνεργασία μεταξύ των προγραμματιστών και πελατών, αφού θα συμφωνήσουν στην εμφάνιση πριν προχωρήσουν στο στάδιο της υλοποίησης της εφαρμογής. Αυτό έχει ως αποτέλεσμα να αυξάνεται η αποτελεσματικότητα, να εντοπίζονται σφάλματα, να εξοικονομείται χρόνος και κόστος [74].

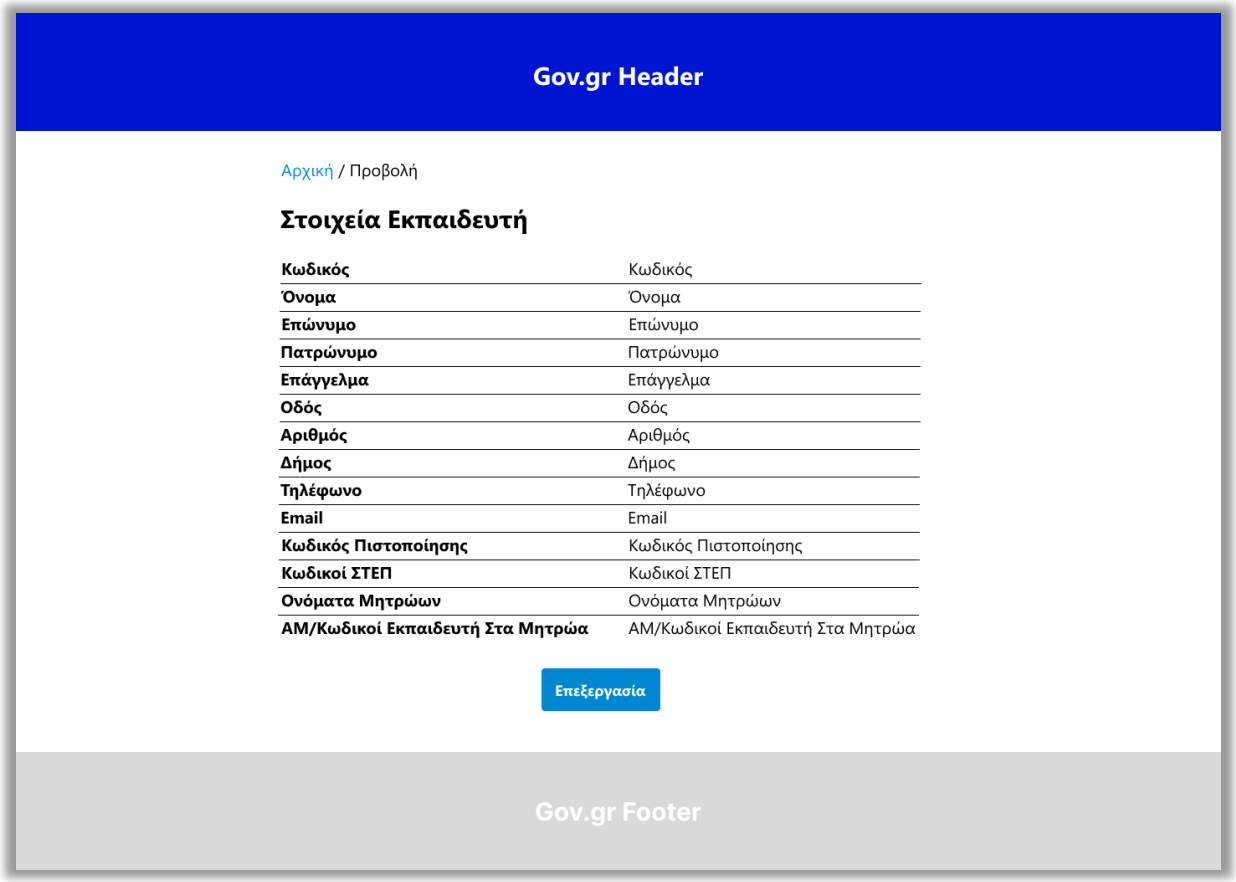
Στις εικόνες 6.5, 6.6, 6.7 παρουσιάζονται τα mockups που σχεδιάστηκαν για desktop για τις σελίδες των λειτουργιών της προβολής της λίστας των εκπαιδευτών, εισαγωγής εκπαιδευτή, προβολής και επεξεργασίας των στοιχείων του εκπαιδευτή. Στην εικόνα 6.8 απεικονίζεται ο σχεδιασμός των mockups για κινητά τηλέφωνα. Να σημειωθεί ότι δεν έγινε ξεχωριστή σχεδίαση για τη σελίδα της επεξεργασίας των στοιχείων του εκπαιδευτή, επειδή είναι παρόμοια με τη σελίδα της εισαγωγής εκπαιδευτή. Η διαφορά είναι ότι στη σελίδα της επεξεργασίας θα εμφανίζονται προσυμπληρωμένα τα πεδία της φόρμας.



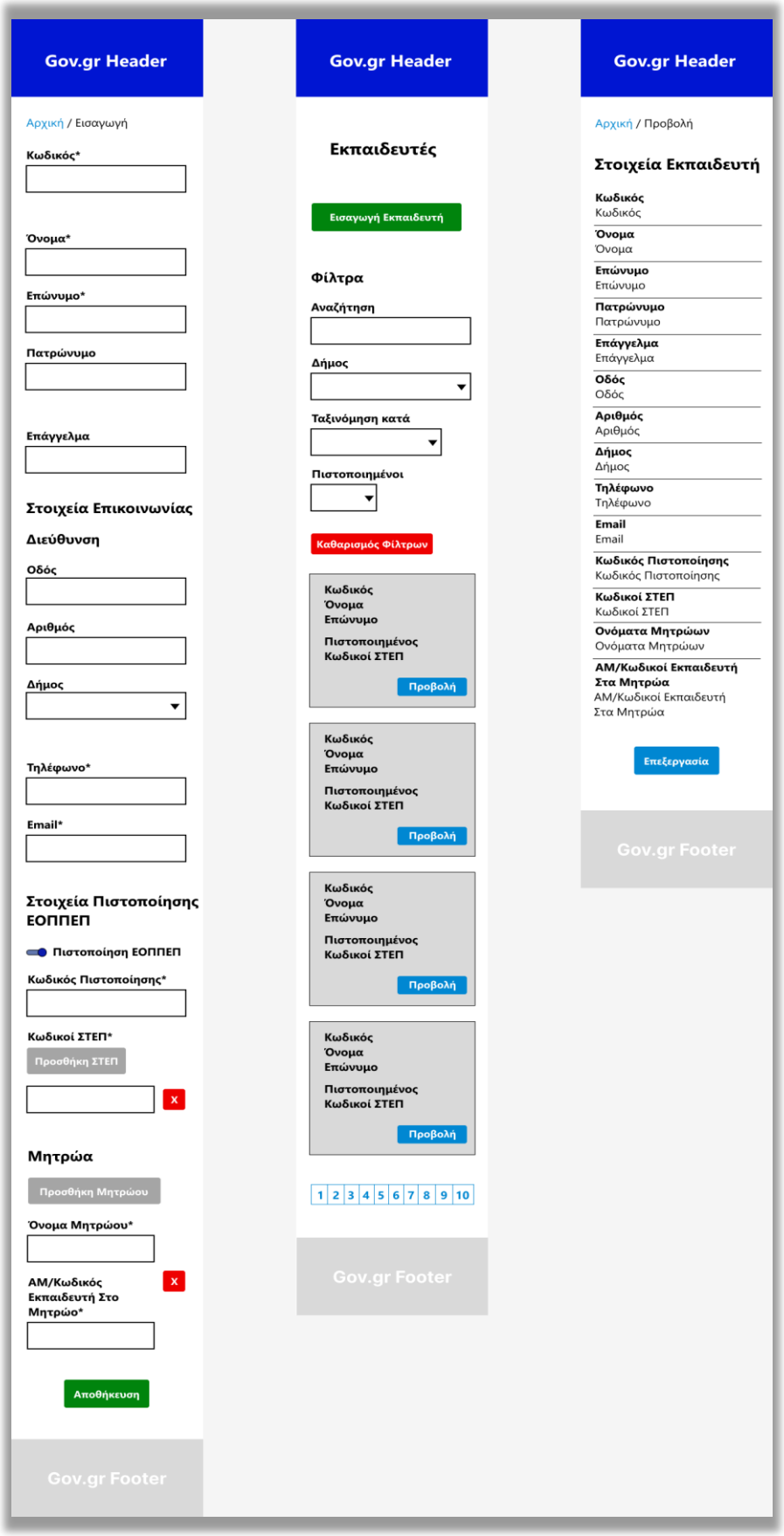
Εικόνα 6.5 Mockup για την αρχική σελίδα / σελίδα της προβολής της λίστας των εκπαιδευτών



Εικόνα 6.6 Mockup για τη σελίδα της εισαγωγής εκπαιδευτή



Εικόνα 6.7 Mockup για τη σελίδα της προβολής των στοιχείων του εκπαιδευτή



Εικόνα 6.8 Mockups των σελίδων για κινητά τηλέφωνα

6.2.3 Αποθήκευση δεδομένων

Τα δεδομένα της φόρμας θα αποθηκεύονται προσωρινά με τη μορφή JSON στο εργαλείο mockAPI. Όπως αναφέρθηκε σε προηγούμενο κεφάλαιο, το mockAPI είναι ένα εργαλείο δημιουργίας εικονικών APIs που χρησιμοποιείται για δοκιμές και περιέχει προσωρινή αποθήκευση των δεδομένων. Σε μία ολοκληρωμένη εφαρμογή, δηλαδή σε μία full stack, τα δεδομένα θα αποθηκεύονται ως JSON μορφή σε μία μη σχεσιακή βάση δεδομένων (NoSQL), όπως είναι η MongoDB. Τα αντικείμενα JSON που θα αποθηκεύονται θα πρέπει να επικυρώνονται επιτυχώς από το ακόλουθο JSON Schema:

```
{
  "$id": "https://voucher.gov.gr/trainer.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "description": "An vocational training trainer",
  "type": "object",
  "required": ["code", "name", "contact"],
  "properties": {
    "code": {
      "type": "string"
    },
    "name": {
      "type": "object",
      "required": ["first", "last"],
      "properties": {
        "first": {
          "type": "string"
        },
        "last": {
          "type": "string"
        },
        "father": {
          "type": "string"
        }
      }
    },
    "occupation": {
      "type": "string"
    },
    "contact": {
      "type": "object",
      "required": ["phone", "email"],
      "properties": {
        "address": {
          "type": "object",
```

```
      "required": ["municipality"],
      "properties": {
        "street": {
          "type": "string"
        },
        "number": {
          "type": "string"
        },
        "municipality": {
          "type": "object",
          "required": ["code"],
          "properties": {
            "code": {
              "type": "string"
            },
            "name": {
              "type": "string"
            }
          }
        }
      }
    },
    "phone": {
      "type": "string",
      "minLength": 10,
      "maxLength": 10
    },
    "email": {
      "type": "string",
      "format": "email"
    }
  }
},
"certification": {
  "type": "object",
  "required": ["code", "step"],
  "properties": {
    "code": {
      "type": "string"
    },
    "step": {
      "type": "array",
      "items": {
        "type": "string"
      }
    }
  }
},
```

```
    "registries": {
      "type": "array",
      "items": {
        "type": "object",
        "required": ["registry", "code"],
        "properties": {
          "registry": {
            "type": "object",
            "required": ["name"],
            "properties": {
              "name": {
                "type": "string"
              }
            }
          },
          "code": {
            "type": "string"
          }
        }
      }
    }
  }
}
```

Όπως αναφέρθηκε σε προηγούμενο κεφάλαιο, το JSON Schema προσδιορίζει τη δομή, τις ιδιότητες και τους περιορισμούς των αντικειμένων JSON. Επιπροσθέτως επικυρώνει συντακτικά και σημασιολογικά τα δεδομένα τους.

Παράδειγμα JSON αντικειμένου που επικυρώνεται επιτυχώς από το παραπάνω JSON Schema:

```
{
  "code": "EB38886",
  "name": {
    "first": "ΓΕΩΡΓΙΟΣ",
    "last": "ΦΟΡΤΑΚΗΣ",
    "father": "ΑΘΑΝΑΣΙΟΣ"
  },
  "occupation": "ΕΞΩΤΕΡΙΚΟΣ ΣΥΝΕΡΓΑΤΗΣ ΤΕΧΝΟΛΟΓΙΚΟΥ ΕΚΠΑΙΔΕΥΤΙΚΟΥ ΙΔΡΥΜΑΤΟΣ
ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ",
  "contact": {
    "address": {
      "street": "ΑΓ. ΔΙΟΝΥΣΙΟΥ",
      "number": "15Β",
      "municipality": {
        "code": "9134",
        "name": "ΠΑΤΡΕΩΝ"
      }
    },
```

```
    "phone": "2610786534",
    "email": "geo@gmail.com"
  },
  "certification": {
    "code": "EB38886",
    "step": ["2299", "3222", "3431", "4115", "4190", "4232"]
  },
  "registries": [
    {
      "registry": {
        "name": "Μητρώο εκπαιδευτών ΚΕΔΙΒΙΜ ΠΑΠΕΛ"
      },
      "code": "243"
    },
    {
      "registry": {
        "name": "Μητρώο εκπαιδευτών ΚΕΔΙΒΙΜ ΠΑΝΕΠΙΣΤΗΜΙΟΥ ΠΑΤΡΩΝ"
      },
      "code": "68"
    }
  ]
}
```


6.3 Γλώσσες και τεχνολογίες που χρησιμοποιήθηκαν

Η εφαρμογή ιστού υλοποιήθηκε χρησιμοποιώντας πολλές γλώσσες και τεχνολογίες ανάπτυξης web εφαρμογών. Συγκεκριμένα, η front-end JavaScript βιβλιοθήκη React χρησιμοποιήθηκε ως βασική για τη δημιουργία της διεπαφής χρήστη (UI) και επιλέχθηκε λόγω της απόδοσης, της ευκολίας εκμάθησης και της μεγάλης ζήτησης στην αγορά εργασίας.

Η γλώσσα σήμανσης HTML5 χρησιμοποιήθηκε για τον καθορισμό της δομής του περιεχομένου, και η γλώσσα προγραμματισμού JavaScript για την προσθήκη δυναμικής συμπεριφοράς και διαδραστικότητας. Στην πραγματικότητα έγινε χρήση του JSX που είναι ο συνδυασμός των δύο. Η γλώσσα CSS3, το CSS framework Bootstrap και το σύστημα σχεδιασμού GOV.GR χρησιμοποιήθηκαν για τον καθορισμό του στυλ, της διάταξης και για να προσαρμόζεται δυναμικά η εφαρμογή σε διαφορετικές οθόνες και συσκευές. Ειδικότερα αξιοποιήθηκαν οι προκαθορισμένες κλάσεις CSS και τα πρότυπα σχεδίασης που παρέχουν. Ακόμα ενσωματώθηκαν έτοιμα προς χρήση components από τη βιβλιοθήκη Material UI.

Η βιβλιοθήκη δρομολόγησης React Router ήταν υπεύθυνη για τη διαχείριση της δρομολόγησης και της πλοήγησης, ενώ ο διαχειριστής πακέτων npm ήταν υπεύθυνος για τη

διαχείριση των πακέτων και των εξαρτήσεων. Το Visual Studio Code χρησιμοποιήθηκε για τη συγγραφή και επεξεργασία του κώδικα, ενώ το Git και το GitLab για τον έλεγχο των εκδόσεων του κώδικα και τη συνεργασία. Η εφαρμογή τηρεί το σύστημα σχεδιασμού GOV.GR και ακολουθεί το αρχιτεκτονικό στυλ REST. Έτσι, χρησιμοποιήθηκε η βιβλιοθήκη Axios για τη δημιουργία και αποστολή HTTP αιτημάτων προς το API ή διακομιστή, και το πρότυπο JSON για την ανταλλαγή των δεδομένων. Επιπλέον, το εργαλείο mockAPI χρησιμοποιήθηκε για τη δημιουργία ενός εικονικού API και την προσωρινή αποθήκευση των δεδομένων. Τέλος, όλες οι γλώσσες και οι τεχνολογίες που χρησιμοποιήθηκαν αναλύονται εκτενέστερα στα κεφάλαια 2, 3, 4 και 5.

6.4 Παρουσίαση Εφαρμογής

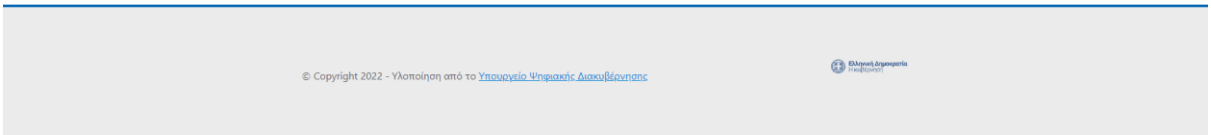
Στο υποκεφάλαιο αυτό παρουσιάζεται η εφαρμογή ιστού διαχείρισης των εκπαιδευτών ενηλίκων που αναπτύχθηκε στα πλαίσια της παρούσας πτυχιακής εργασίας. Αρχικά πραγματοποιείται η παρουσίαση της σε επιτραπέζιους υπολογιστές (desktops) και έπειτα σε κινητά τηλέφωνα (mobile phones). Προς το παρόν δεν απαιτείται κάποια σύνδεση (login) ή εγγραφή (sign-up) από τους διαχειριστές (χρήστες) στην εφαρμογή, ωστόσο σε μελλοντική επέκταση θα απαιτείται.

6.4.1 Κεφαλίδα και Υποσέλιδο

Η κεφαλίδα (header) της εφαρμογής έχει δημιουργηθεί από το αρχείο-component **Header.js** και εμφανίζεται στο πάνω μέρος της κάθε σελίδας. Περιλαμβάνει το λογότυπο του GOV.GR, το οποίο όταν το πατήσει ο διαχειριστής θα μεταβεί στην αρχική σελίδα. Η κεφαλίδα της εφαρμογής παρουσιάζεται στην εικόνα 6.9.

Εικόνα 6.9 Η κεφαλίδα της εφαρμογής

Το υποσέλιδο (footer) της εφαρμογής έχει δημιουργηθεί από το αρχείο-component **Footer.js** και εμφανίζεται στο κάτω μέρος της κάθε σελίδας. Το υποσέλιδο της εφαρμογής απεικονίζεται στην εικόνα 6.10.



Εικόνα 6.10 Το υποσέλιδο της εφαρμογής

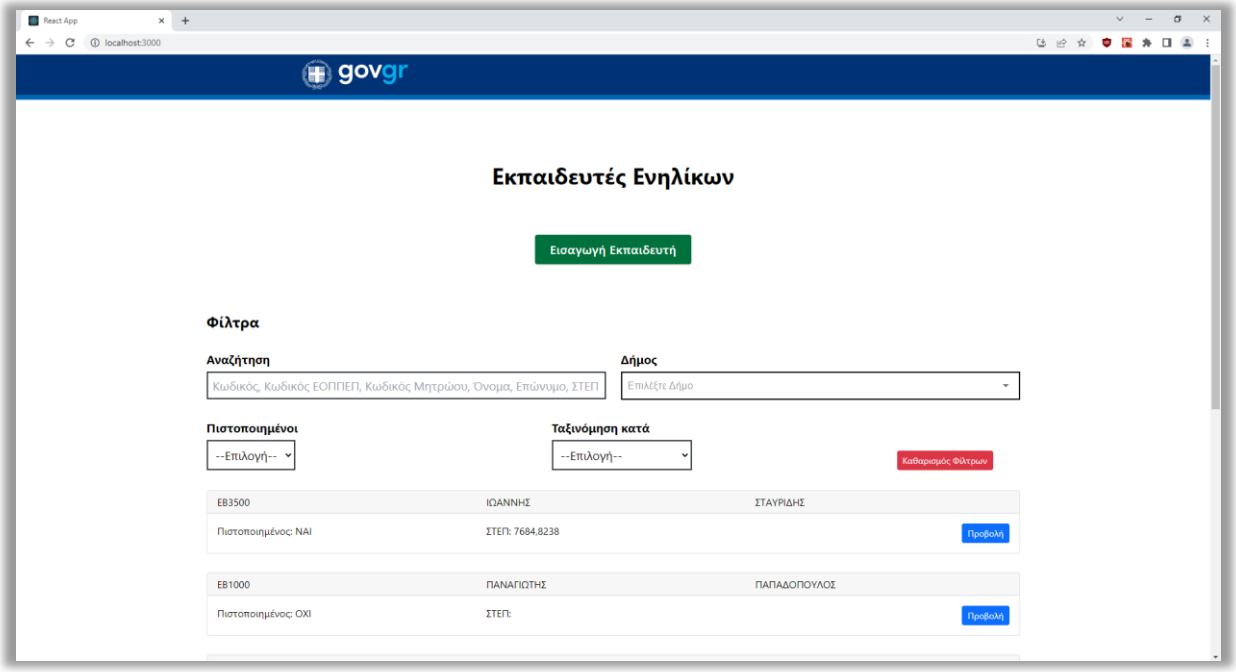
6.4.2 Αρχική σελίδα

Η αρχική σελίδα έχει δημιουργηθεί από τον συνδυασμό των αρχείων-components **Home.js** και **Read.js**. Στην αρχική σελίδα, κάτω από την κεφαλίδα εμφανίζεται ο τίτλος "Εκπαιδευτές Ενηλίκων" που βοηθάει τους χρήστες (διαχειριστές) να κατανοήσουν ποια ψηφιακή υπηρεσία χρησιμοποιούν. Κάτω από τον τίτλο βρίσκεται το πράσινο κουμπί "Εισαγωγή Εκπαιδευτή", το οποίο όταν το πατήσει ο διαχειριστής θα ανακατευθυνθεί στη σελίδα της λειτουργίας της εισαγωγής εκπαιδευτή. Στο υπόλοιπο της αρχικής σελίδας προβάλλονται τα φίλτρα που θα αναλυθούν παρακάτω και η λίστα με τους εισαχθέντες εκπαιδευτές ενηλίκων.

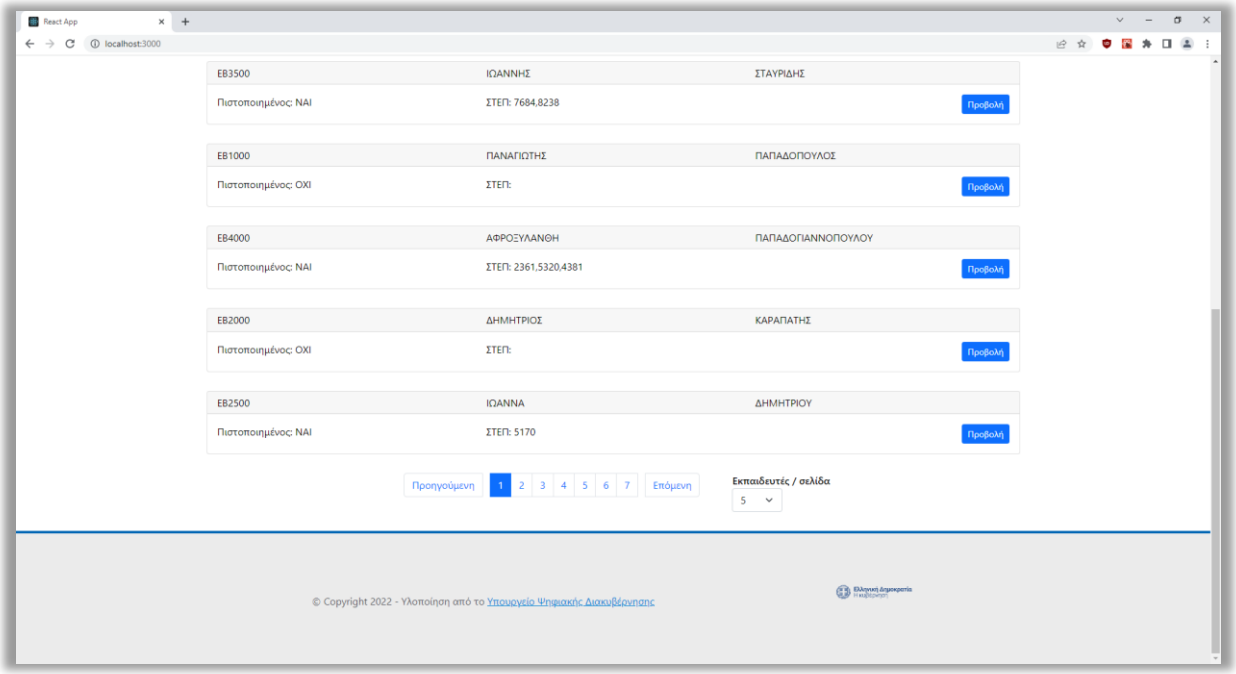
6.4.2.1 Λίστα εκπαιδευτών ενηλίκων

Η λίστα εμφανίζεται ταξινομημένη από τον πιο παλιό στον πιο πρόσφατο και χωρίζεται σε σελίδες, όπου σε κάθε σελίδα εμφανίζονται 5 εκπαιδευτές από προεπιλογή. Σε κάθε εισαχθέντα εκπαιδευτή ενηλίκων, στην πρώτη γραμμή αναγράφονται τα στοιχεία Κωδικός, Όνομα, Επώνυμο και στη δεύτερη αν είναι πιστοποιημένος από τον ΕΟΠΠΕΠ, οι Κωδικοί ΣΤΕΠ που του έχουν αποδοθεί και το μπλε κουμπί "Προβολή". Πατώντας το κουμπί "Προβολή" σε έναν εκπαιδευτή, ο διαχειριστής μεταβαίνει στη σελίδα της προβολής των συνολικών στοιχείων του.

Κάτω από τη λίστα βρίσκεται η σελιδοποίηση της (pagination) και η αναπτυσσόμενη λίστα "Εκπαιδευτές / σελίδα". Μέσω της σελιδοποίησης, ο διαχειριστής μπορεί να περιηγηθεί και να προβάλλει όλους τους εκπαιδευτές. Στις εικόνες 6.11 και 6.12 παρουσιάζεται η συνολική αρχική σελίδα της εφαρμογής.



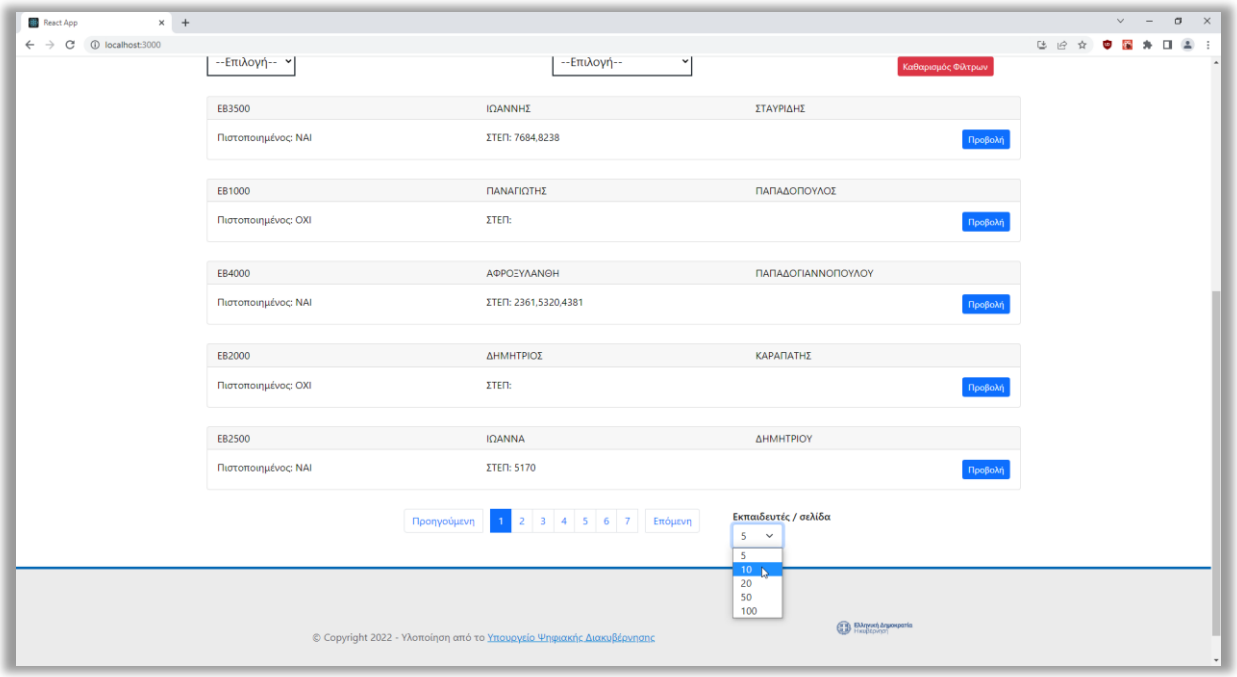
Εικόνα 6.11 Η αρχική σελίδα της εφαρμογής



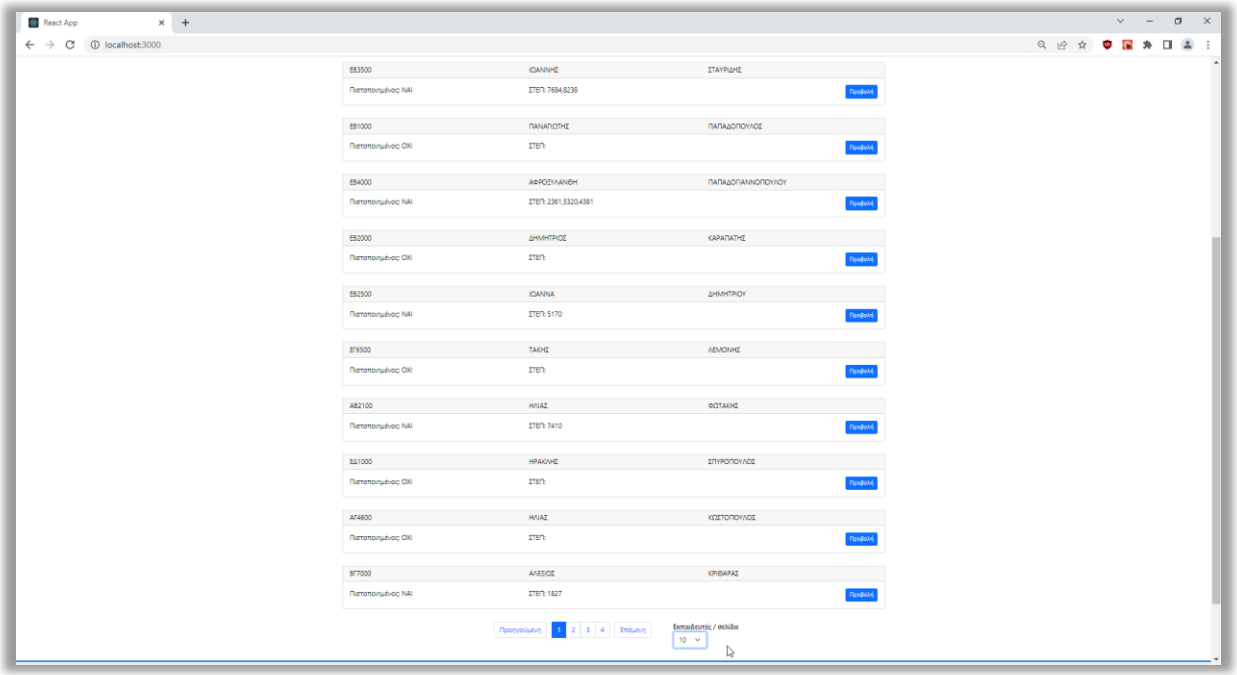
Εικόνα 6.12 Η αρχική σελίδα της εφαρμογής (συνέχεια)

Ο διαχειριστής έχει τη δυνατότητα να αλλάξει το πλήθος των εκπαιδευτών που θα εμφανίζεται σε κάθε σελίδα, επιλέγοντας διαφορετικό αριθμό από την αναπτυσσόμενη λίστα "Εκπαιδευτές

/ σελίδα". Συγκεκριμένα, μπορεί να επιλέξει 5 ή 10 ή 20 ή 50 ή 100 εκπαιδευτές ανά σελίδα. Στις εικόνες 6.13 και 6.14 απεικονίζεται ένα παράδειγμα επιλογής για την εμφάνιση 10 εκπαιδευτών ανά σελίδα.



Εικόνα 6.13 Επιλογή 10 εκπαιδευτών ανά σελίδα

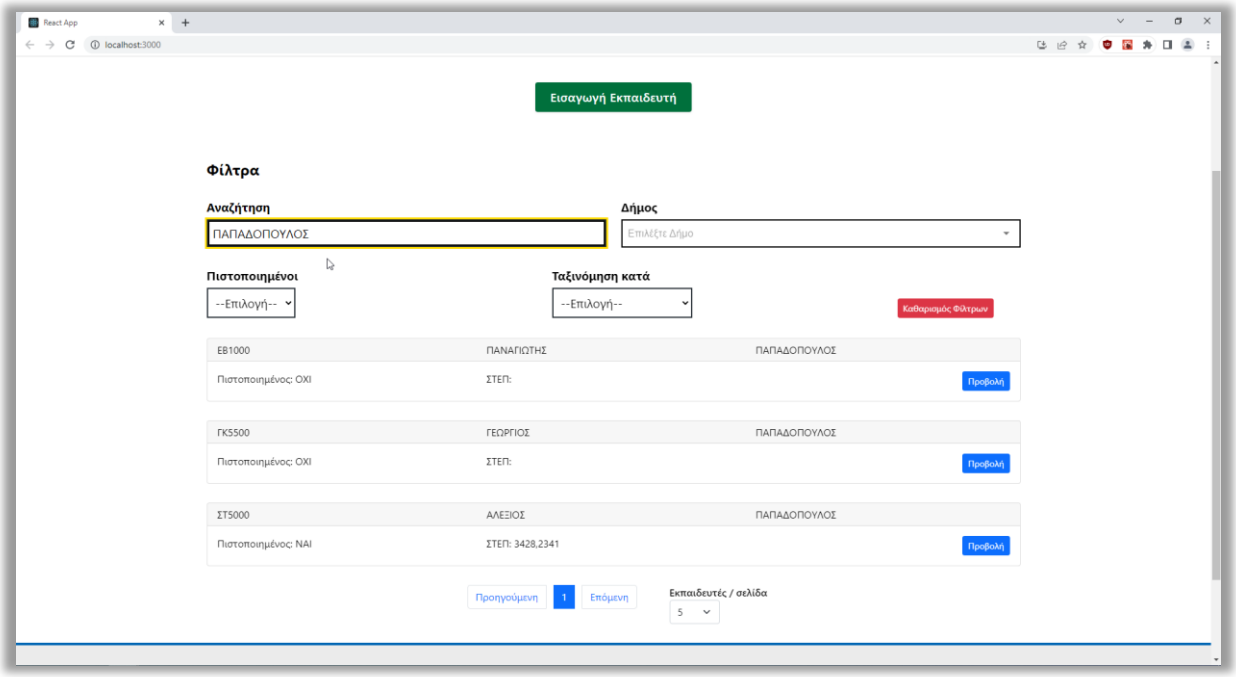


Εικόνα 6.14 Αποτέλεσμα επιλογής 10 εκπαιδευτών ανά σελίδα

6.4.2.2 Φίλτρα λίστας

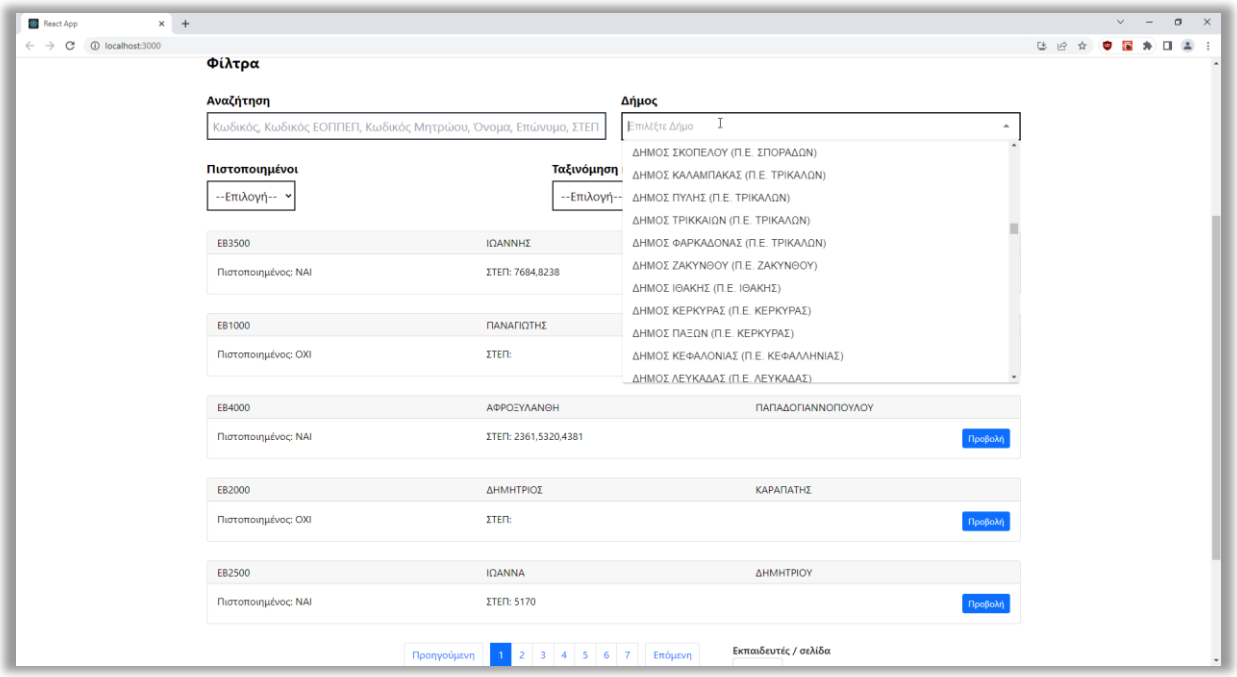
Όπως αναφέρθηκε προηγουμένως, πάνω από τη λίστα προβάλλονται τα φίλτρα της, όπου στην πάνω σειρά βρίσκονται το πεδίο "Αναζήτηση" και η αναπτυσσόμενη λίστα "Δήμος". Στην κάτω σειρά βρίσκονται η αναπτυσσόμενη λίστα "Πιστοποιημένοι", η αναπτυσσόμενη λίστα "Ταξινόμηση κατά" και το κόκκινο κουμπί "Καθαρισμός Φίλτρων".

Στο πεδίο "Αναζήτηση" ο διαχειριστής έχει τη δυνατότητα να αναζητήσει συγκεκριμένους εκπαιδευτές με βάση τον Κωδικό ή τον Κωδικό ΕΟΠΠΕΠ ή τον Κωδικό Μητρώου ή το Όνομα ή το Επώνυμο ή το Ονοματεπώνυμο ή τον Κωδικό ΣΤΕΠ. Στην εικόνα 6.15 προβάλλεται ένα παράδειγμα αναζήτησης με βάση το επώνυμο.

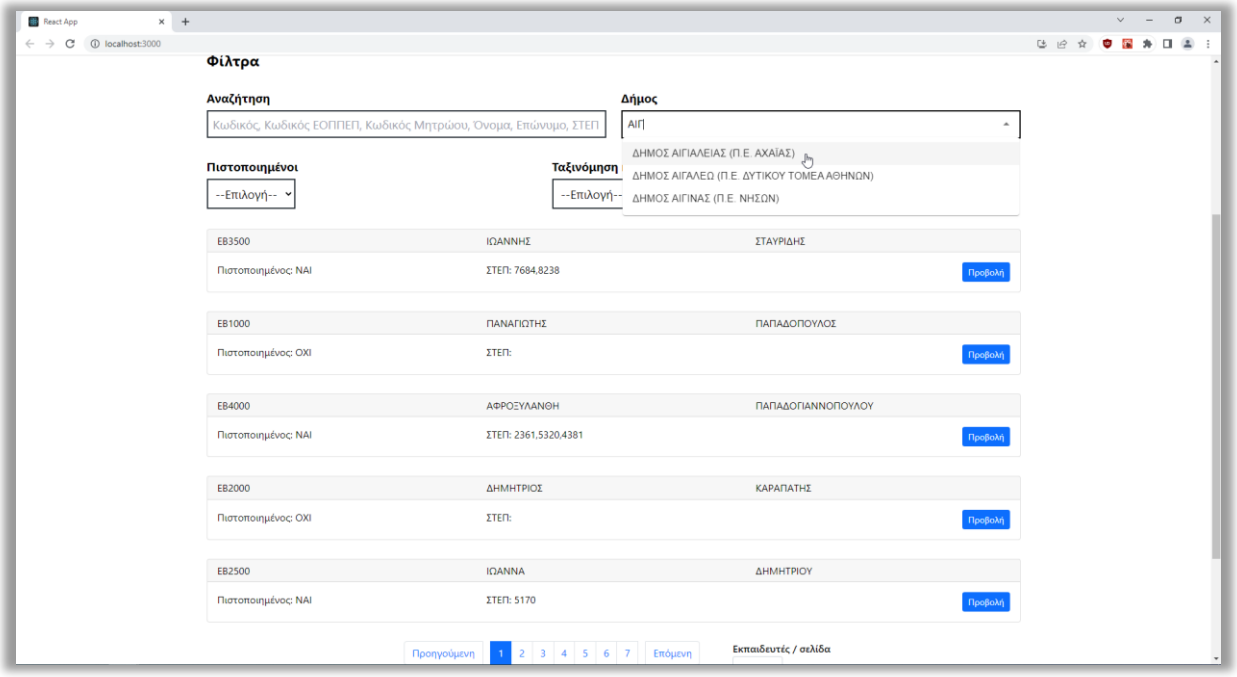


Εικόνα 6.15 Αναζήτηση εκπαιδευτών με βάση το επώνυμο

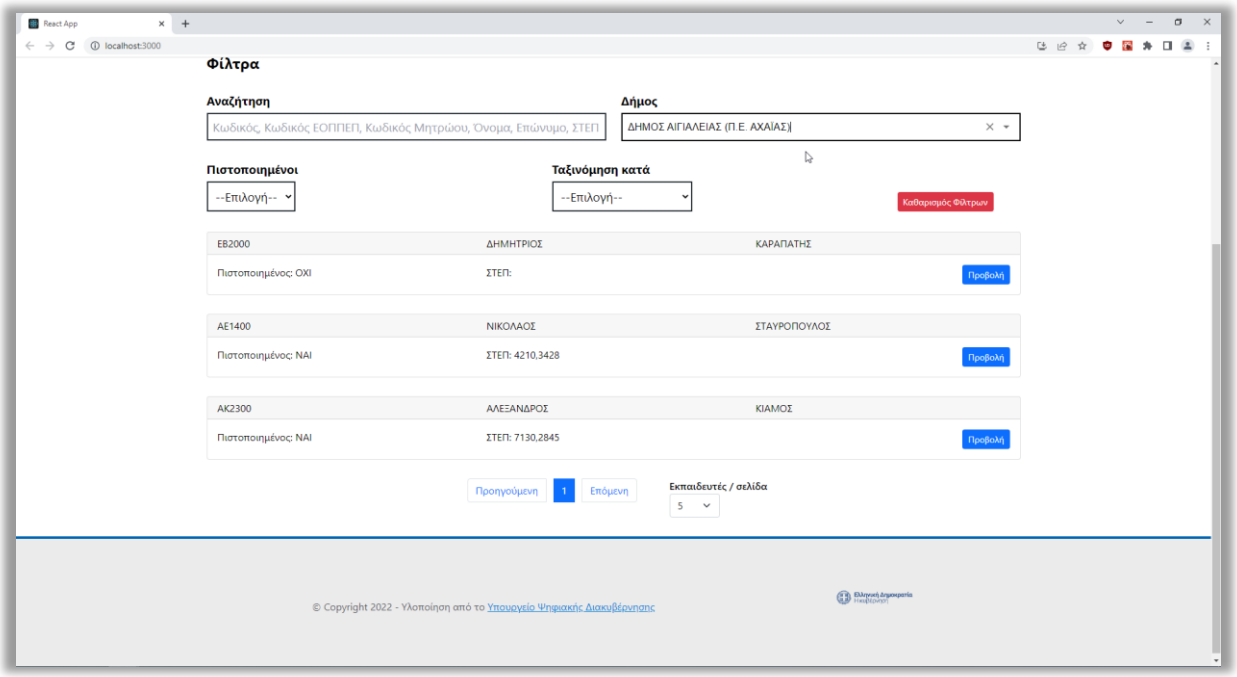
Η αναπτυσσόμενη λίστα "Δήμος" περιέχει όλους τους δήμους της Ελλάδας, όπου σε αυτήν ο διαχειριστής μπορεί να αναζητήσει και να επιλέξει ένα συγκεκριμένο δήμο, προκειμένου να βρει τους εκπαιδευτές που διαμένουν σε αυτόν. Στην εικόνα 6.16 εμφανίζονται οι δήμοι της αναπτυσσόμενης λίστας "Δήμος". Στις εικόνες 6.17 και 6.18 παρουσιάζεται ένα παράδειγμα αναζήτησης και επιλογής ενός δήμου.



Εικόνα 6.16 Εμφάνιση των δήμων της αναπτυσσόμενης λίστας "Δήμος"

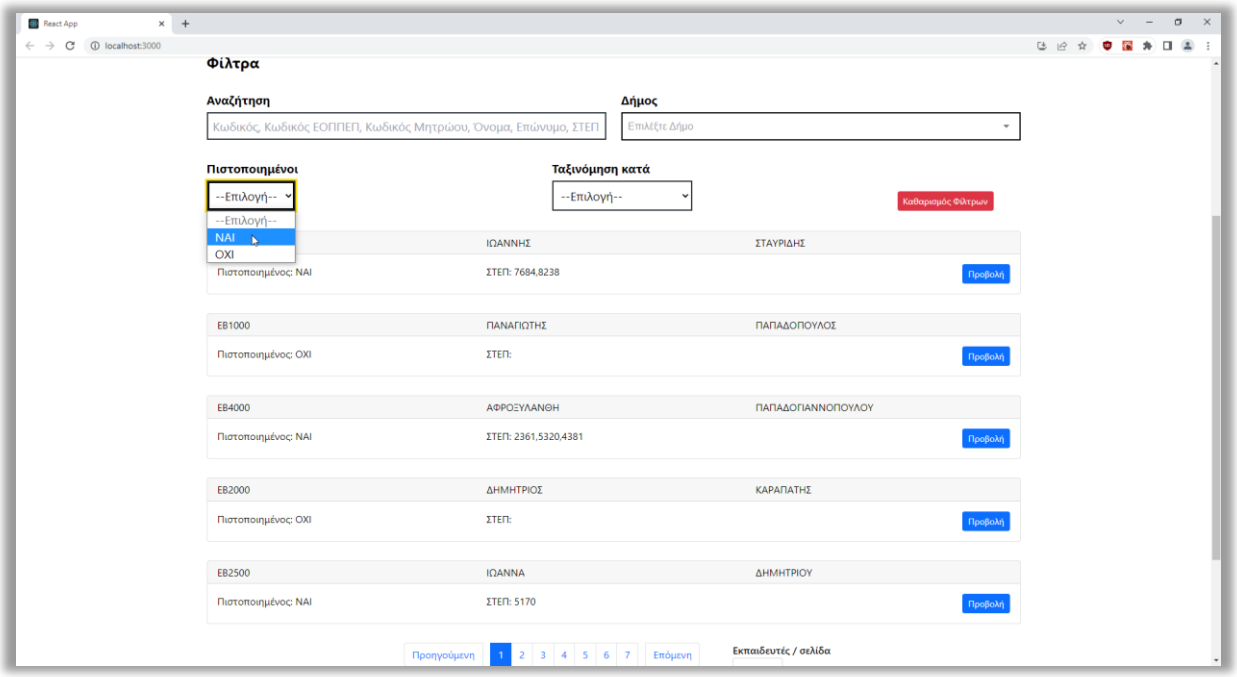


Εικόνα 6.17 Αναζήτηση και επιλογή ενός δήμου από την αναπτυσσόμενη λίστα "Δήμος"

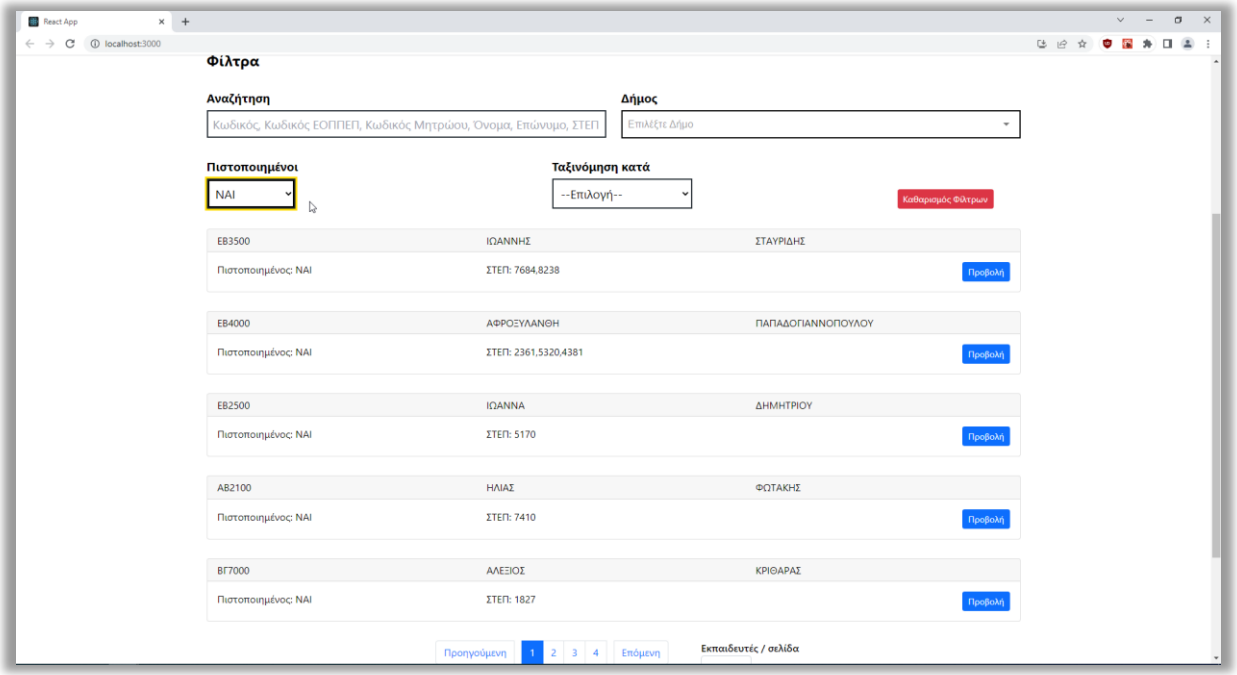


Εικόνα 6.18 Αποτέλεσμα αναζήτησης και επιλογής ενός δήμου από την αναπτυσσόμενη λίστα "Δήμος"

Η αναπτυσσόμενη λίστα "Πιστοποιημένοι" περιλαμβάνει τις επιλογές "ΝΑΙ" και "ΟΧΙ". Όταν ο διαχειριστής επιλέγει το "ΝΑΙ", τότε εμφανίζονται οι πιστοποιημένοι εκπαιδευτές. Ενώ όταν επιλέγει το "ΟΧΙ", τότε εμφανίζονται οι μη πιστοποιημένοι. Στις εικόνες 6.19 και 6.20 απεικονίζεται ένα παράδειγμα της επιλογής "ΝΑΙ" για την εμφάνιση των πιστοποιημένων.



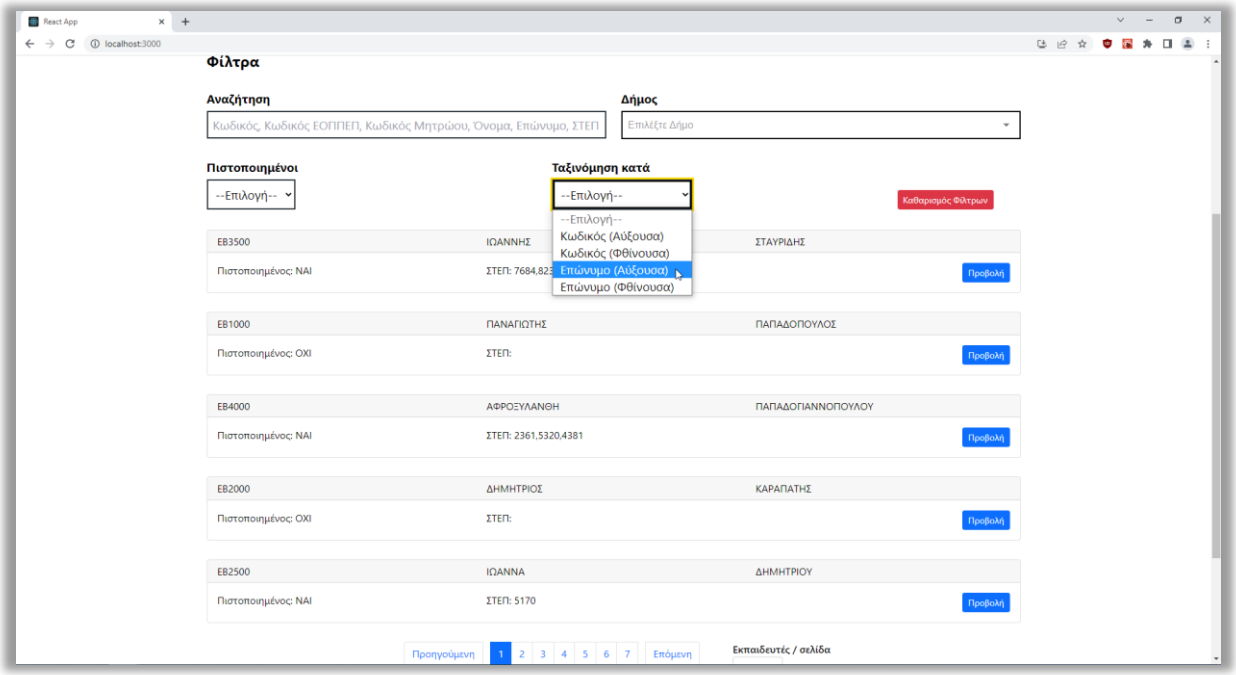
Εικόνα 6.19 Επιλογή εμφάνισης των πιστοποιημένων εκπαιδευτών



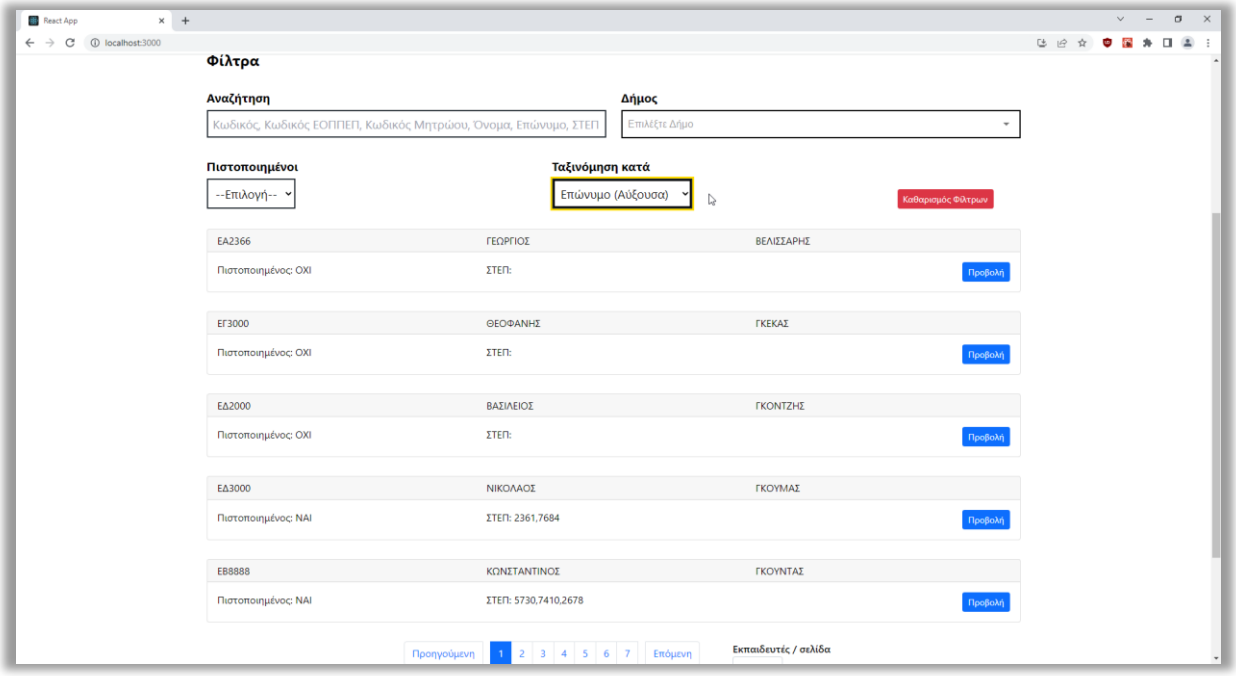
Εικόνα 6.20 Αποτέλεσμα επιλογής των πιστοποιημένων εκπαιδευτών

Μέσω της αναπτυσσόμενης λίστας "Ταξινόμηση κατά" ο διαχειριστής μπορεί να ταξινομήσει τη λίστα των εκπαιδευτών κατά Κωδικό (Αύξουσα) ή Κωδικό (Φθίνουσα) ή Επώνυμο

(Αύξουσα) ή Επώνυμο (Φθίνουσα). Στις εικόνες 6.21 και 6.22 προβάλλεται ένα παράδειγμα ταξινόμησης της λίστας κατά επώνυμο (αύξουσα).



Εικόνα 6.21 Επιλογή ταξινόμησης της λίστας κατά επώνυμο (αύξουσα)



Εικόνα 6.22 Αποτέλεσμα της επιλογής ταξινόμησης της λίστας κατά επώνυμο (αύξουσα)

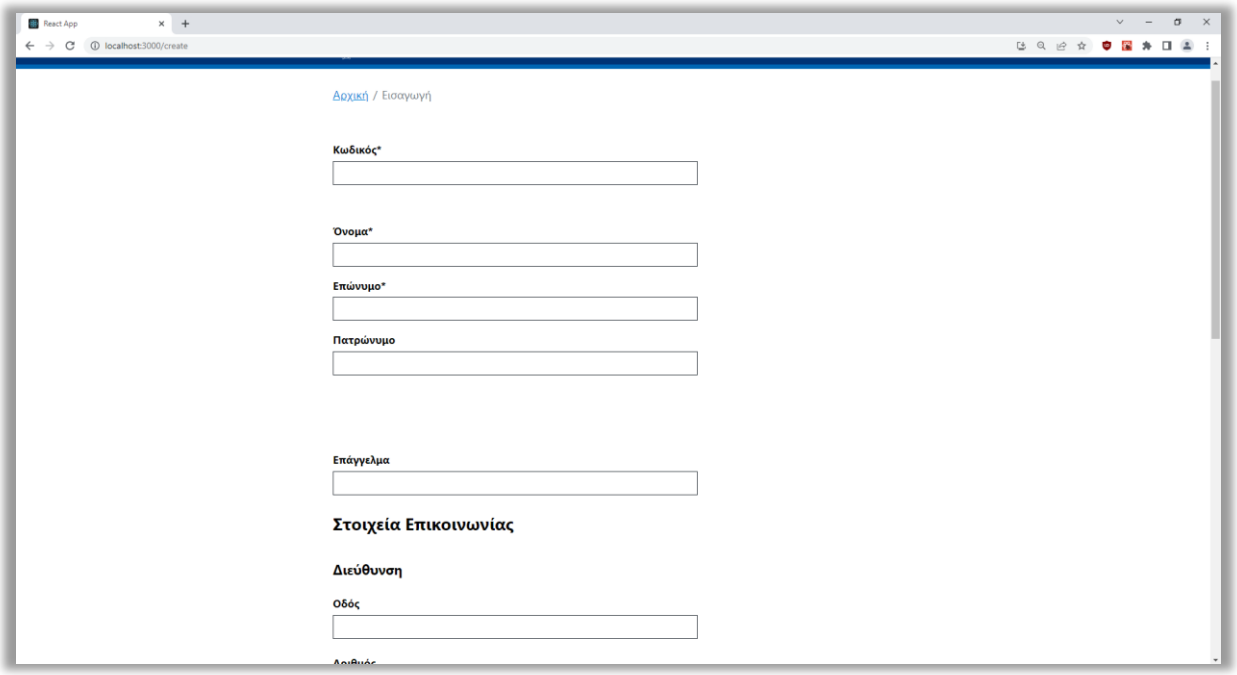
Ακόμα να σημειωθεί ότι ο διαχειριστής μπορεί να χρησιμοποιήσει τα φίλτρα συνδυαστικά. Τέλος, έχει τη δυνατότητα να εκκαθαρίσει τα εφαρμοσμένα φίλτρα πατώντας το κουμπί "Καθαρισμός Φίλτρων".

6.4.3 Σελίδα Εισαγωγής Εκπαιδευτή

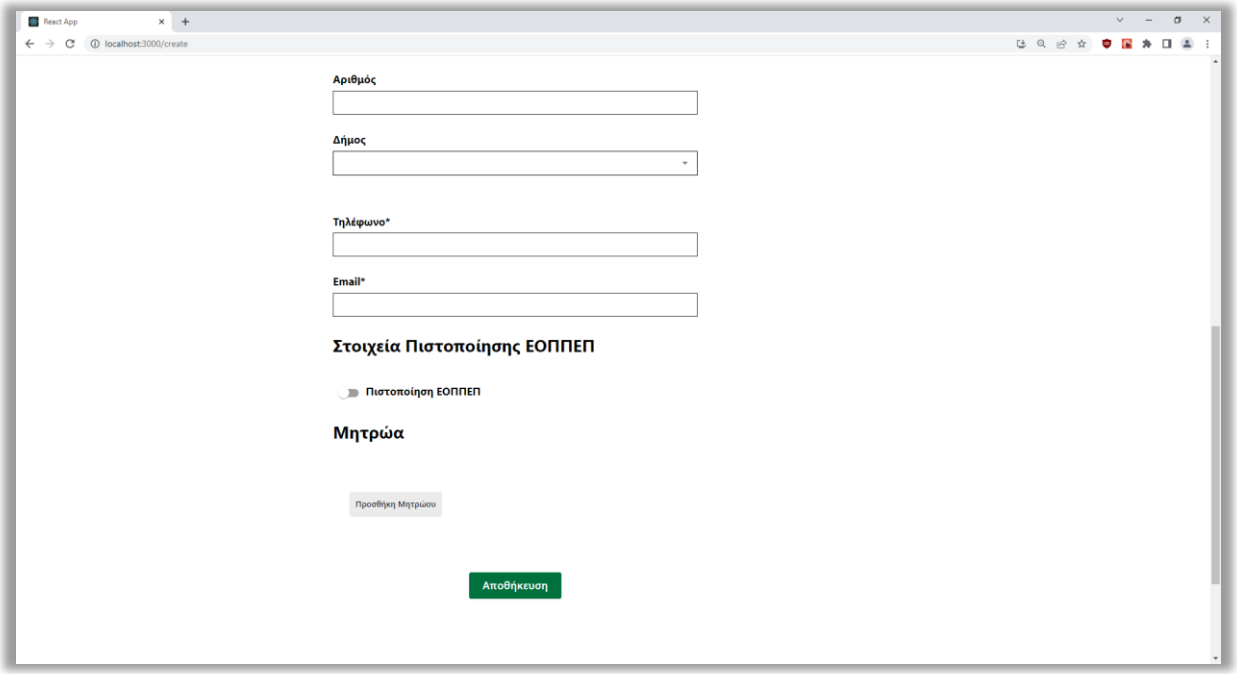
Όπως αναφέρθηκε στην προηγούμενη υποενότητα, ο διαχειριστής ανακατευθύνεται στη σελίδα της εισαγωγής εκπαιδευτή από την αρχική σελίδα, πατώντας το κουμπί "Εισαγωγή Εκπαιδευτή". Η σελίδα αυτή έχει δημιουργηθεί από το αρχείο-component **Form.js** και δίνει τη δυνατότητα στον διαχειριστή να εισάγει έναν εκπαιδευτή ενηλίκων.

Σε αυτή τη σελίδα, κάτω από την κεφαλίδα εμφανίζεται το breadcrumb "Αρχική / Εισαγωγή" που ενημερώνει τον διαχειριστή σε ποια σελίδα βρίσκεται. Επιπλέον πατώντας το σύνδεσμο "Αρχική" στο breadcrumb, πλοηγείται πίσω στην αρχική σελίδα. Κάτω από το breadcrumb προβάλλεται η φόρμα με τα πεδία που καλείται να συμπληρώσει ο διαχειριστής.

Στη φόρμα από πάνω προς τα κάτω προβάλλονται τα πεδία "Κωδικός", "Όνομα", "Επώνυμο", "Πατρώνυμο" και "Επάγγελμα". Παρακάτω εμφανίζονται οι επικεφαλίδες "Στοιχεία Επικοινωνίας" και "Διεύθυνση" που βοηθούν τον διαχειριστή να καταλάβει τι στοιχεία πρέπει να συμπληρώσει στα πεδία που ακολουθούν. Αυτά τα πεδία είναι τα "Οδός", "Αριθμός", "Δήμος", τα οποία αντιστοιχούν στη διεύθυνση και τα πεδία "Τηλέφωνο", "Email". Να σημειωθεί ότι το πεδίο "Δήμος" είναι μία αναπτυσσόμενη λίστα που περιέχει όλους τους δήμους της Ελλάδας, όπως το πεδίο δήμος στην αρχική σελίδα, ωστόσο η διαφορά τους είναι ότι εδώ ο διαχειριστής δηλώνει το δήμο που διαμένει ο εκπαιδευτής. Έπειτα, εμφανίζεται η επικεφαλίδα "Στοιχεία Πιστοποίησης ΕΟΠΠΕΠ" ακολουθούμενη από τον απενεργοποιημένο διακόπτη (switch) "Πιστοποίηση ΕΟΠΠΕΠ" και από κάτω η επικεφαλίδα "Μητρώα" ακολουθούμενη από το γκρι κουμπί "Προσθήκη Μητρώου". Στο τέλος της φόρμας βρίσκεται το πράσινο κουμπί της υποβολής "Αποθήκευση". Στις εικόνες 6.23 και 6.24 παρουσιάζεται η συνολική σελίδα της εισαγωγής εκπαιδευτή που προβάλλει ο διαχειριστής αρχικά.



Εικόνα 6.23 Η σελίδα της εισαγωγής εκπαιδευτή που προβάλλει ο διαχειριστής αρχικά



Εικόνα 6.24 Η σελίδα της εισαγωγής εκπαιδευτή που προβάλλει ο διαχειριστής αρχικά (συνέχεια)

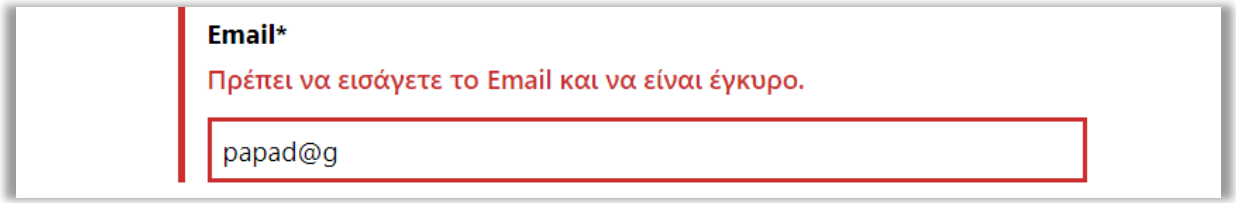
Σε κάθε πεδίο της φόρμας εφαρμόζεται επικύρωση ώστε να διασφαλίζεται ότι τα δεδομένα που εισάγει ο διαχειριστής είναι εντός των περιορισμών. Τα πεδία "Κωδικός", "Όνομα" και

"Επώνυμο" είναι υποχρεωτικά προς συμπλήρωση, ενώ τα πεδία "Πατρώνυμο" και "Επάγγελμα" είναι προαιρετικά. Τα δεδομένα που δέχονται τα πεδία πρέπει να είναι αλφαριθμητικού τύπου.

Τα πεδία "Οδός", "Αριθμός" και "Δήμος" είναι προαιρετικά προς συμπλήρωση, ωστόσο στην περίπτωση που ο διαχειριστής συμπληρώσει το πεδίο "Αριθμός", τότε θα πρέπει να συμπληρώσει υποχρεωτικά και το πεδίο "Οδός". Επιπλέον, στην περίπτωση που συμπληρώσει το πεδίο "Οδός", τότε θα πρέπει να συμπληρώσει υποχρεωτικά και το πεδίο "Δήμος". Τα πεδία δέχονται αλφαριθμητικού τύπου δεδομένα.

Το πεδίο "Τηλέφωνο" πρέπει να συμπληρωθεί υποχρεωτικά και τα δεδομένα που εισάγονται πρέπει να είναι ακριβώς 10 ψηφία. Το πεδίο "Email" είναι υποχρεωτικό προς συμπλήρωση και οι τιμές που εισάγονται πρέπει να αντιστοιχούν σε έγκυρα email, για παράδειγμα example@gmail.com.

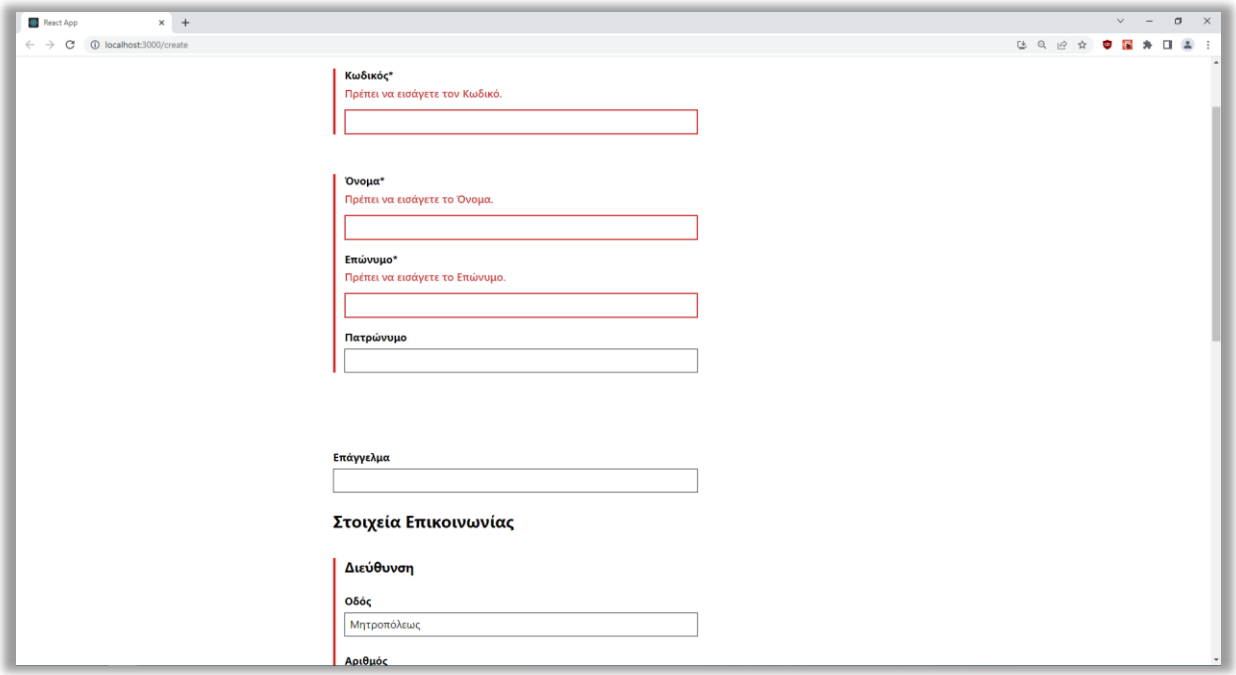
Ειδικότερα, η επικύρωση των δεδομένων πραγματοποιείται κατά την υποβολή και σε πραγματικό χρόνο (real-time), δηλαδή όταν ο διαχειριστής πατάει το κουμπί "Αποθήκευση" και σε κάθε αλλαγή τιμής που κάνει σε ένα πεδίο. Συνεπώς, όταν ο διαχειριστής εισάγει μη έγκυρα δεδομένα σε κάποιο πεδίο, τότε αμέσως εμφανίζεται κατάλληλο μήνυμα σφάλματος με κόκκινο χρώμα από πάνω του, το περίγραμμα του γίνεται κόκκινο και αριστερά του εμφανίζεται μία κόκκινη κάθετη μπάρα. Αυτά εξαφανίζονται αμέσως όταν εισάγει έγκυρα δεδομένα. Στις εικόνες 6.25 και 6.26 προβάλλεται ένα παράδειγμα επικύρωσης σε πραγματικό χρόνο.



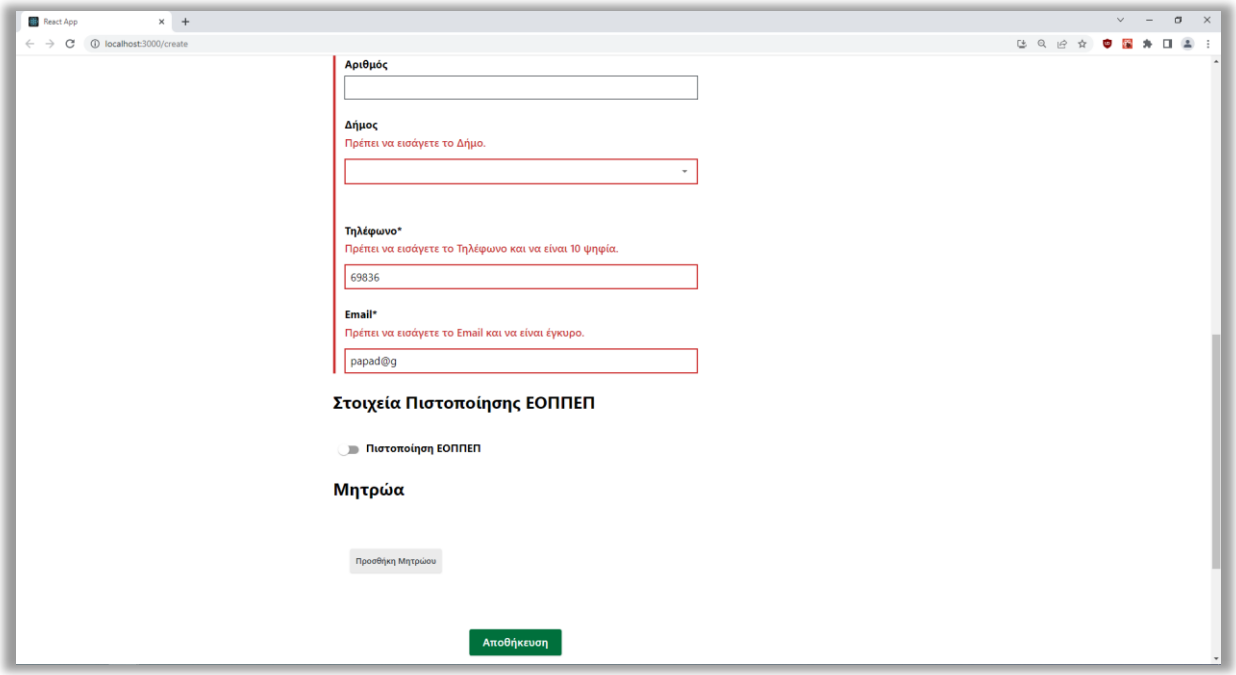
Εικόνα 6.25 Επικύρωση του πεδίου "Email" σε πραγματικό χρόνο (μη έγκυρο πεδίο)

Εικόνα 6.26 Επικύρωση του πεδίου "Email" σε πραγματικό χρόνο (έγκυρο πεδίο)

Αν υποβάλλει (αποθηκεύει) τη φόρμα με μη έγκυρα δεδομένα, τότε πάλι εμφανίζονται τα μηνύματα σφάλματος και οι ενδείξεις στα πεδία που δεν είναι έγκυρα. Ακόμα γίνεται εστίαση (focus) στο πρώτο μη έγκυρο πεδίο. Στις εικόνες 6.27 και 6.28 απεικονίζεται η επικύρωση κατά την υποβολή της φόρμας.



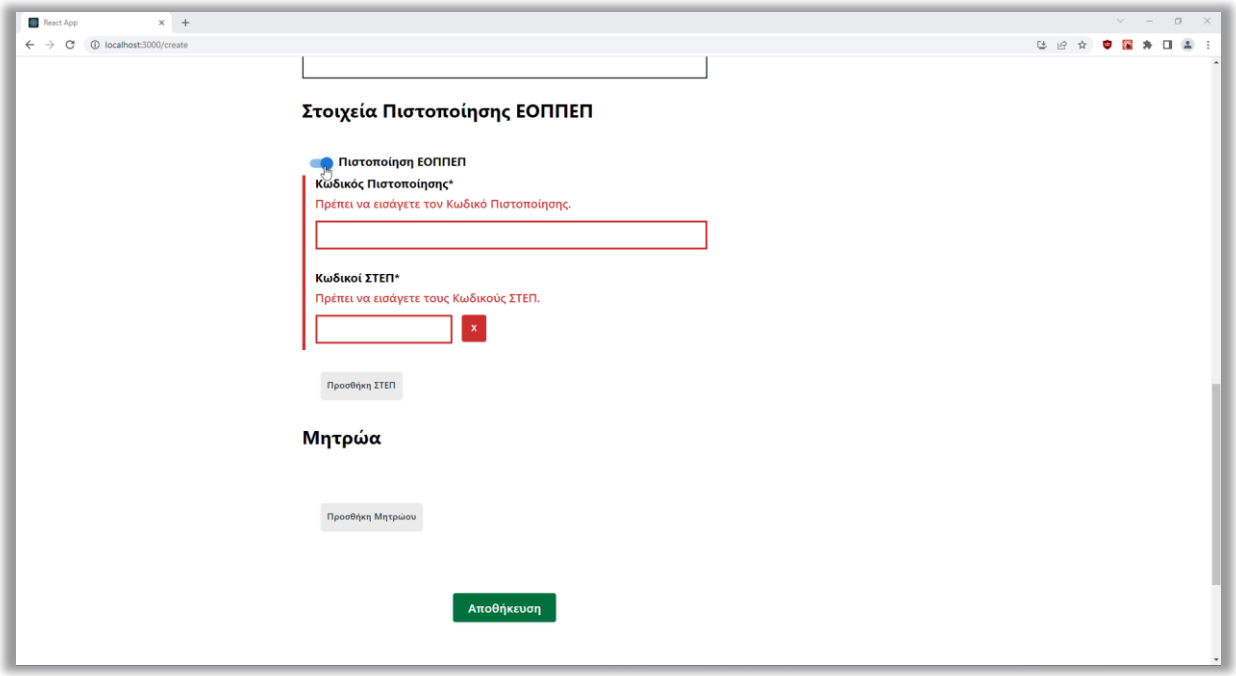
Εικόνα 6.27 Επικύρωση κατά την υποβολή της φόρμας



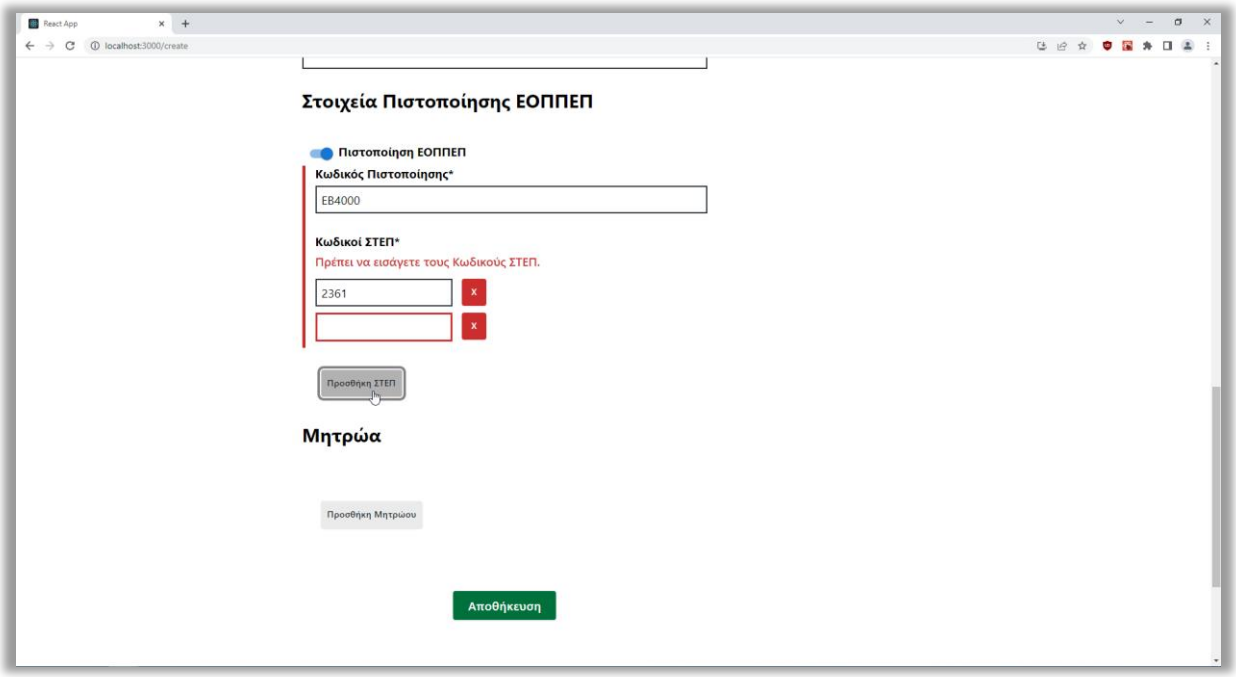
Εικόνα 6.28 Επικύρωση κατά την υποβολή της φόρμας (συνέχεια)

Αν ο διαχειριστής εισάγει έναν εκπαιδευτή ενηλίκων χωρίς πιστοποίηση ΕΟΠΠΕΠ, τότε αφήνει απενεργοποιημένο το διακόπτη (switch) "Πιστοποίηση ΕΟΠΠΕΠ". Διαφορετικά αν εισάγει έναν εκπαιδευτή ενηλίκων που διαθέτει πιστοποίηση ΕΟΠΠΕΠ, τότε ενεργοποιεί το διακόπτη και εμφανίζονται από πάνω προς τα κάτω το πεδίο "Κωδικός Πιστοποίησης", το δυναμικό πεδίο "Κωδικοί ΣΤΕΠ" και το γκρι κουμπί "Προσθήκη ΣΤΕΠ". Πατώντας το κουμπί "Προσθήκη ΣΤΕΠ", ο διαχειριστής έχει τη δυνατότητα να προσθέσει ένα νέο πεδίο στους κωδικούς ΣΤΕΠ. Ακόμα, μπορεί να αφαιρέσει ένα πεδίο από τους κωδικούς ΣΤΕΠ πατώντας το κόκκινο κουμπί "X", το οποίο βρίσκεται δίπλα σε κάθε πεδίο.

Το πεδίο "Κωδικός Πιστοποίησης" και όλα τα πεδία "Κωδικοί ΣΤΕΠ" δέχονται αλφαριθμητικού τύπου δεδομένα και είναι υποχρεωτικά προς συμπλήρωση μόνο όταν ο διακόπτης είναι ενεργοποιημένος. Στα πεδία αυτά προβάλλονται από την αρχή μηνύματα σφάλματος και ενδείξεις για να καταλάβει ο διαχειριστής ότι πρέπει να τα συμπληρώσει υποχρεωτικά. Τα μηνύματα σφάλματος και οι ενδείξεις εξαφανίζονται αμέσως όταν εισάγει έγκυρα δεδομένα όπως και στα προηγούμενα πεδία της φόρμας. Στην εικόνα 6.29 απεικονίζονται τα πεδία της πιστοποίησης μετά την ενεργοποίηση του διακόπτη "Πιστοποίηση ΕΟΠΠΕΠ". Στην εικόνα 6.30 προβάλλεται η προσθήκη ενός νέου πεδίου στους κωδικούς ΣΤΕΠ.



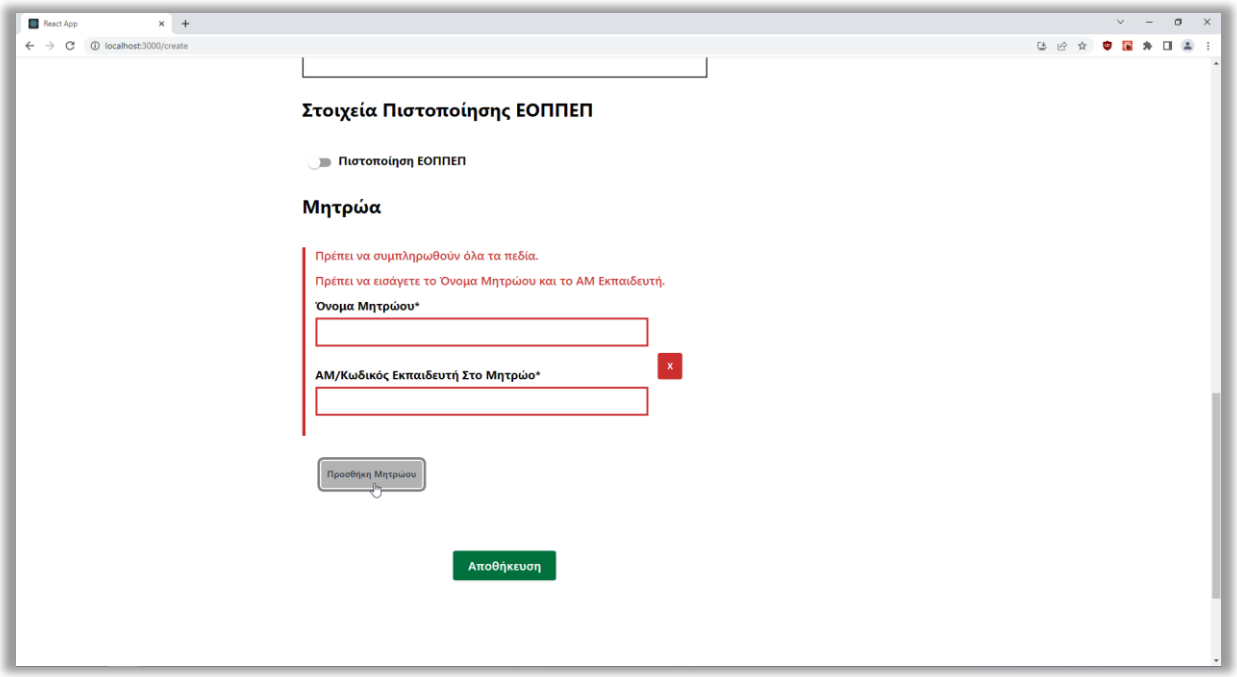
Εικόνα 6.29 Εμφάνιση των πεδίων της πιστοποίησης μετά την ενεργοποίηση του διακόπτη "Πιστοποίηση ΕΟΠΠΕΠ"



Εικόνα 6.30 Προσθήκη ενός νέου πεδίου στους κωδικούς ΣΤΕΠ

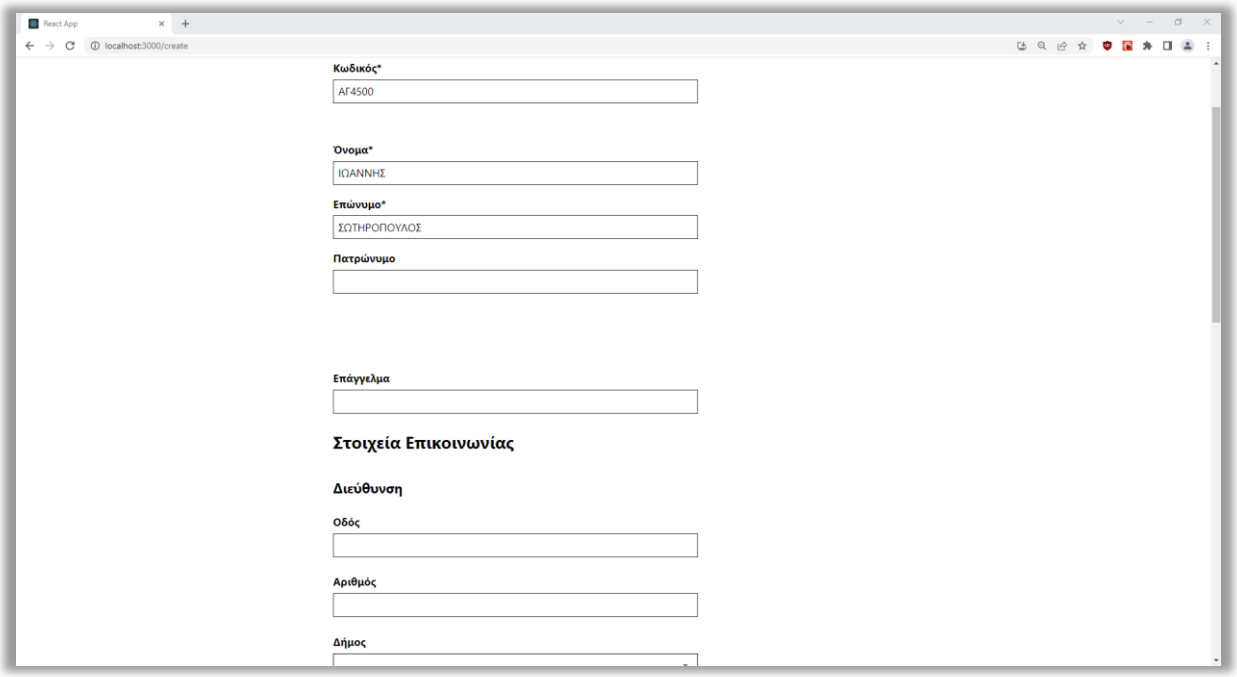
Στην περίπτωση που ο διαχειριστής εισάγει έναν εκπαιδευτή ενηλίκων, ο οποίος είναι εγγεγραμμένος σε ένα ή περισσότερα μητρώα εκπαιδευτών, τότε έχει τη δυνατότητα να τα προσθέσει πατώντας το γκρι κουμπί "Προσθήκη Μητρώου". Έτσι, κάθε φορά που πατάει το κουμπί "Προσθήκη Μητρώου", τότε θα προστίθενται δύο νέα πεδία. Το πρώτο πεδίο είναι το "Όνομα Μητρώου" και το δεύτερο είναι το "ΑΜ/Κωδικός Εκπαιδευτή Στο Μητρώο". Ακόμα μπορεί να τα αφαιρέσει πατώντας το κόκκινο κουμπί "Χ" που βρίσκεται δίπλα τους. Από την άλλη αν εισάγει έναν εκπαιδευτή ενηλίκων που δεν είναι εγγεγραμμένος σε κάποιο μητρώο, τότε δεν πατάει το κουμπί "Προσθήκη Μητρώου".

Τα πεδία "Όνομα Μητρώου" και "ΑΜ/Κωδικός Εκπαιδευτή Στο Μητρώο" δέχονται αλφαριθμητικού τύπου δεδομένα και είναι υποχρεωτικά προς συμπλήρωση μόνο αν υπάρχουν. Τα πεδία αυτά είναι δυναμικά, επομένως θα πρέπει να συμπληρωθούν υποχρεωτικά όλα τα πεδία των μητρώων που θα υπάρχουν. Όπως στα πεδία της πιστοποίησης, στα πεδία αυτά εμφανίζονται από την αρχή μηνύματα σφάλματος και ενδείξεις και εξαφανίζονται αμέσως όταν ο διαχειριστής εισάγει έγκυρα δεδομένα. Στην εικόνα 6.31 φαίνεται η προσθήκη ενός μητρώου στα "Μητρώα".

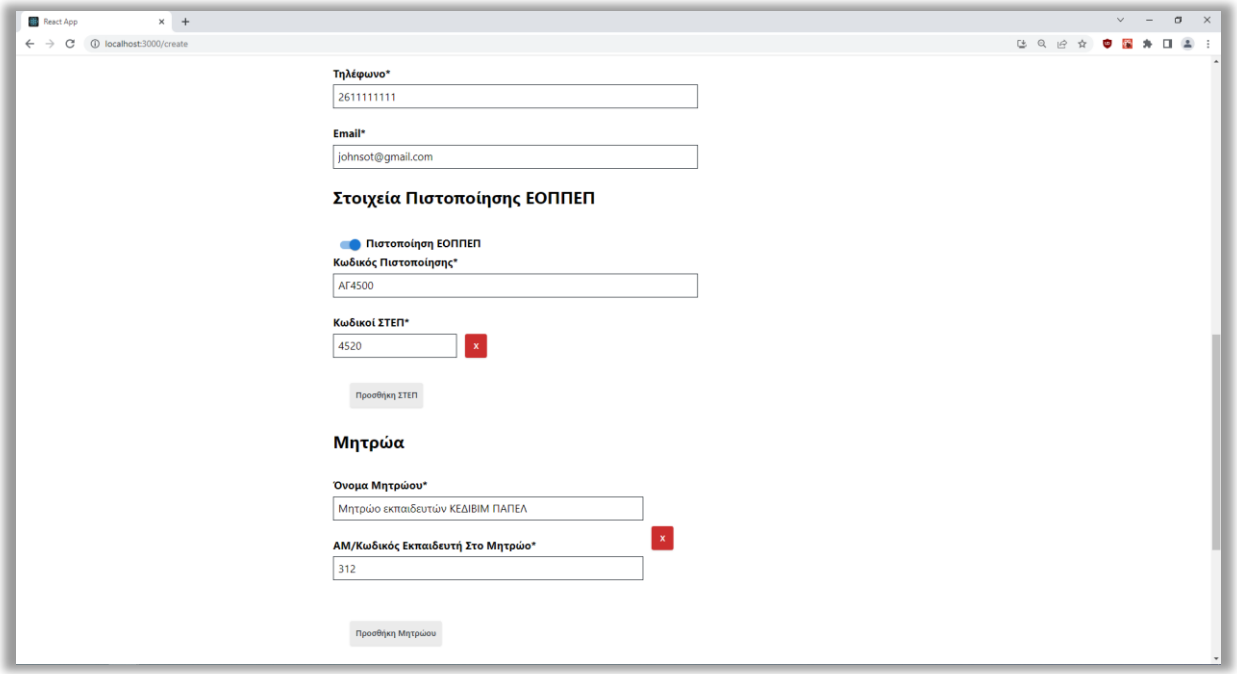


Εικόνα 6.31 Προσθήκη ενός μητρώου στα "Μητρώα"

Τέλος, τα στοιχεία του εκπαιδευτή αποθηκεύονται (υποβάλλονται) επιτυχώς όταν ο διαχειριστής συμπληρώσει τα πεδία με έγκυρα δεδομένα και πατήσει το κουμπί "Αποθήκευση". Μετά την εισαγωγή ανακατευθύνεται αυτόματα στην αρχική σελίδα. Στις εικόνες 6.32 και 6.33 παρουσιάζεται ένα παράδειγμα με έγκυρα δεδομένα στα πεδία της φόρμας.



Εικόνα 6.32 Η σελίδα της εισαγωγής εκπαιδευτή με έγκυρα δεδομένα



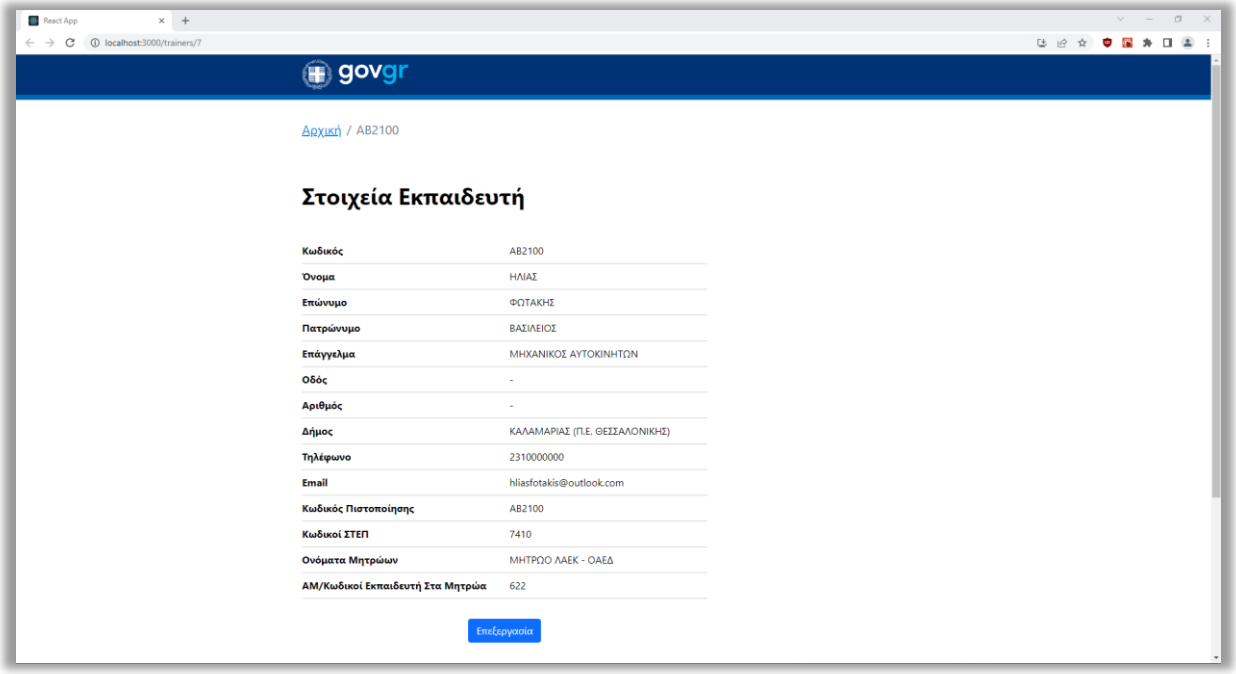
Εικόνα 6.33 Η σελίδα της εισαγωγής εκπαιδευτή με έγκυρα δεδομένα (συνέχεια)

6.4.4 Σελίδα Προβολής Στοιχείων Εκπαιδευτή

Όπως αναφέρθηκε σε προηγούμενη υποενότητα, ο διαχειριστής μεταβαίνει στη σελίδα της προβολής των στοιχείων του εκπαιδευτή από την αρχική σελίδα, πατώντας το κουμπί "Προβολή" σε έναν εκπαιδευτή στη λίστα. Η σελίδα αυτή έχει δημιουργηθεί από το αρχείο-component **Trainer.js** και παρέχει τη δυνατότητα στον διαχειριστή να προβάλλει τα συνολικά στοιχεία ενός εκπαιδευτή ενηλίκων.

Στη συγκεκριμένη σελίδα, κάτω από την κεφαλίδα βρίσκεται το breadcrumb "Αρχική / Κωδικός", όπου ο κωδικός είναι διαφορετικός και προσδιορίζει τον εκπαιδευτή που προβάλλει ο διαχειριστής. Ακόμα, πατώντας το σύνδεσμο "Αρχική" στο breadcrumb, μεταβαίνει πίσω στην αρχική σελίδα.

Κάτω από το breadcrumb προβάλλεται η επικεφαλίδα "Στοιχεία Εκπαιδευτή", η οποία πληροφορεί τον διαχειριστή σε ποια σελίδα βρίσκεται. Έπειτα, κάτω από την επικεφαλίδα εμφανίζονται τα συνολικά στοιχεία του εκπαιδευτή. Συγκεκριμένα, εμφανίζονται από πάνω προς τα κάτω τα στοιχεία "Κωδικός", "Όνομα", "Επώνυμο", "Πατρώνυμο", "Επάγγελμα", "Οδός", "Αριθμός", "Δήμος", "Τηλέφωνο", "Email", "Κωδικός Πιστοποίησης", "Κωδικοί ΣΤΕΠ", "Ονόματα Μητρώων" και "ΑΜ/Κωδικοί Εκπαιδευτή Στα Μητρώα". Στην περίπτωση που ο διαχειριστής δεν είχε συμπληρώσει κάποιο στοιχείο κατά την εισαγωγή, τότε στη θέση του στοιχείου εμφανίζεται μία παύλα (-). Τέλος, κάτω από τα στοιχεία βρίσκεται το μπλε κουμπί "Επεξεργασία", το οποίο όταν το πατήσει ο διαχειριστής θα πλοηγηθεί στη σελίδα της επεξεργασίας των στοιχείων του εκπαιδευτή. Στην εικόνα 6.34 απεικονίζεται η σελίδα της προβολής των στοιχείων ενός εκπαιδευτή.



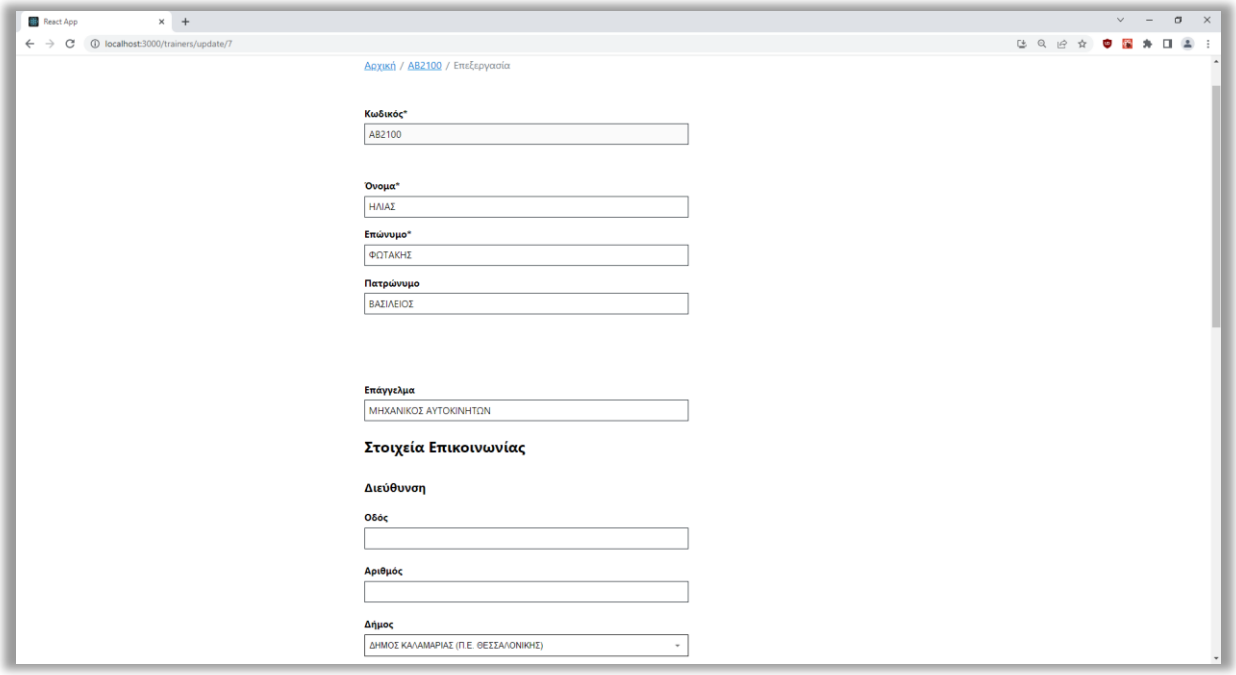
Εικόνα 6.34 Η σελίδα της προβολής των στοιχείων ενός εκπαιδευτή

6.4.5 Σελίδα Επεξεργασίας Στοιχείων Εκπαιδευτή

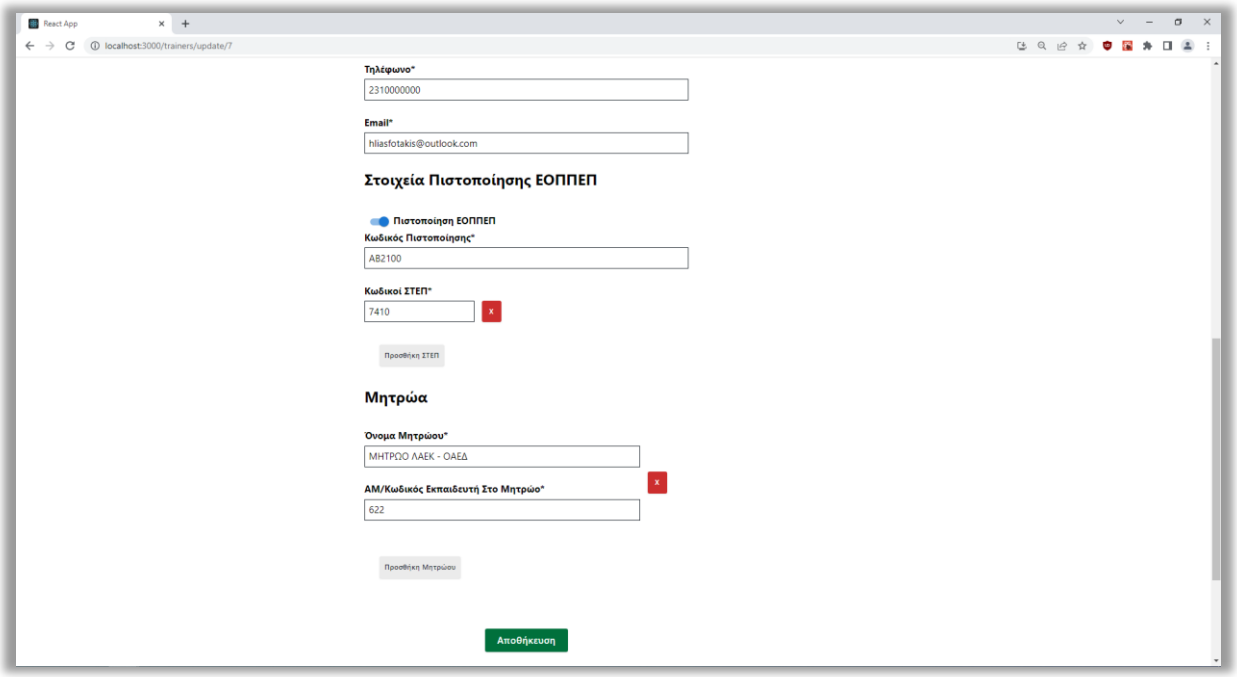
Όπως αναφέρθηκε στην προηγούμενη υποενότητα, ο διαχειριστής μεταβαίνει στη σελίδα της επεξεργασίας των στοιχείων του εκπαιδευτή από τη σελίδα της προβολής των στοιχείων του, πατώντας το κουμπί "Επεξεργασία". Αυτή η σελίδα έχει δημιουργηθεί από το αρχείο-component **Update.js** και προσφέρει τη δυνατότητα στον διαχειριστή να επεξεργαστεί τα στοιχεία ενός εκπαιδευτή ενηλίκων.

Η συγκεκριμένη σελίδα είναι παρόμοια με τη σελίδα της εισαγωγής εκπαιδευτή με μερικές διαφορές. Ειδικότερα, κάτω από την κεφαλίδα βρίσκεται το breadcrumb "Αρχική / Κωδικός / Επεξεργασία", όπου ο κωδικός προσδιορίζει τον εκπαιδευτή που επεξεργάζεται ο διαχειριστής και το "Επεξεργασία" τον ενημερώνει σε ποια σελίδα βρίσκεται. Πατώντας το σύνδεσμο κωδικός στο breadcrumb, ο διαχειριστής πλοηγείται πίσω στη σελίδα της προβολής των στοιχείων του εκπαιδευτή, ενώ πατώντας το σύνδεσμο "Αρχική" πλοηγείται στην αρχική σελίδα. Κάτω από το breadcrumb εμφανίζεται η ίδια φόρμα με τα ίδια πεδία, όπως στη σελίδα της εισαγωγής εκπαιδευτή, ωστόσο η διαφορά είναι ότι εδώ τα πεδία εμφανίζονται προσυμπληρωμένα με τα στοιχεία του εκπαιδευτή.

Ο διαχειριστής μπορεί να επεξεργαστεί τα στοιχεία αλλάζοντας τις τιμές στα αντίστοιχα πεδία, ωστόσο δεν μπορεί να αλλάξει τον κωδικό του εκπαιδευτή, επειδή το πεδίο "Κωδικός" είναι απενεργοποιημένο. Σε κάθε πεδίο της φόρμας εφαρμόζεται η ίδια επικύρωση, όπως στη φόρμα της σελίδας της εισαγωγής εκπαιδευτή. Τέλος, η επεξεργασία ολοκληρώνεται επιτυχώς όταν ο διαχειριστής συμπληρώσει με έγκυρα δεδομένα τα πεδία και πατήσει το πράσινο κουμπί "Αποθήκευση". Μετά την επιτυχή επεξεργασία μεταβαίνει αυτόματα στη σελίδα της προβολής των στοιχείων του εκπαιδευτή. Στις εικόνες 6.35 και 6.36 παρουσιάζεται η σελίδα της επεξεργασίας των στοιχείων ενός εκπαιδευτή.



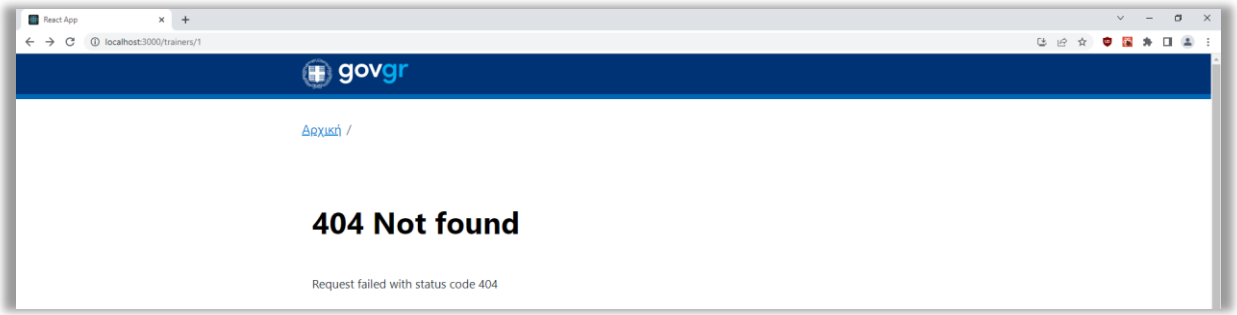
Εικόνα 6.35 Η σελίδα της επεξεργασίας των στοιχείων ενός εκπαιδευτή



Εικόνα 6.36 Η σελίδα της επεξεργασίας των στοιχείων ενός εκπαιδευτή (συνέχεια)

6.4.6 Σελίδα Σφαλμάτων

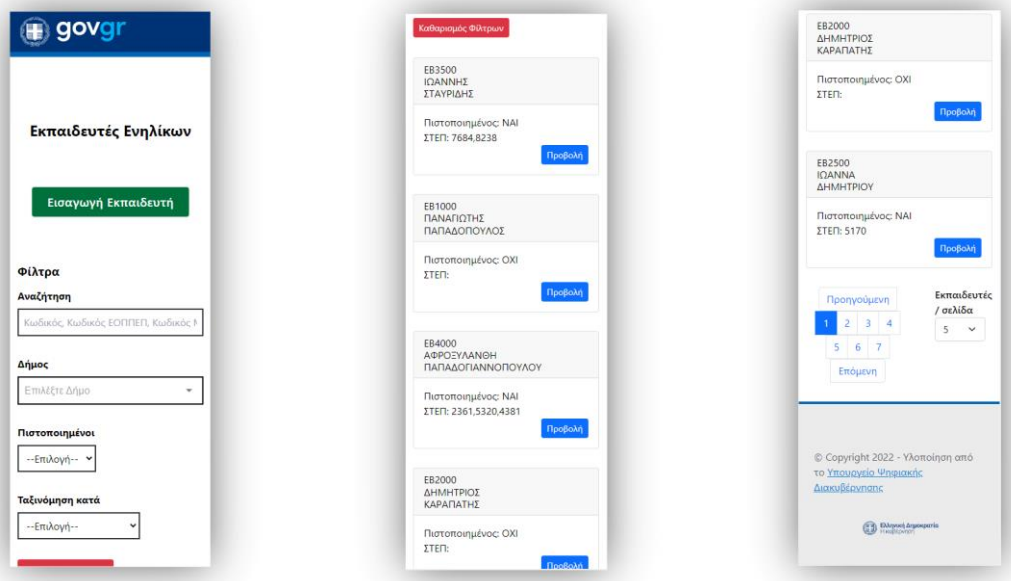
Η σελίδα σφαλμάτων έχει δημιουργηθεί από το αρχείο-component **ErrorPage.js** και εμφανίζεται όταν ένα HTTP αίτημα αποτυγχάνει. Για την ακρίβεια, σε αυτή τη σελίδα υπάρχει αρχικά το breadcrumb, όπου μέσω αυτού ο διαχειριστής μπορεί να μεταβεί στην αρχική σελίδα. Στη συνέχεια, εμφανίζεται μία επικεφαλίδα με το κατάλληλο HTTP μήνυμα σφάλματος και από κάτω μία σύντομη περιγραφή του. Αυτό βοηθάει τον διαχειριστή να κατανοήσει ότι υπάρχει πρόβλημα με την υπηρεσία. Στην εικόνα 6.37 προβάλλεται η σελίδα σφαλμάτων που εμφανίζει το κατάλληλο HTTP μήνυμα σφάλματος.



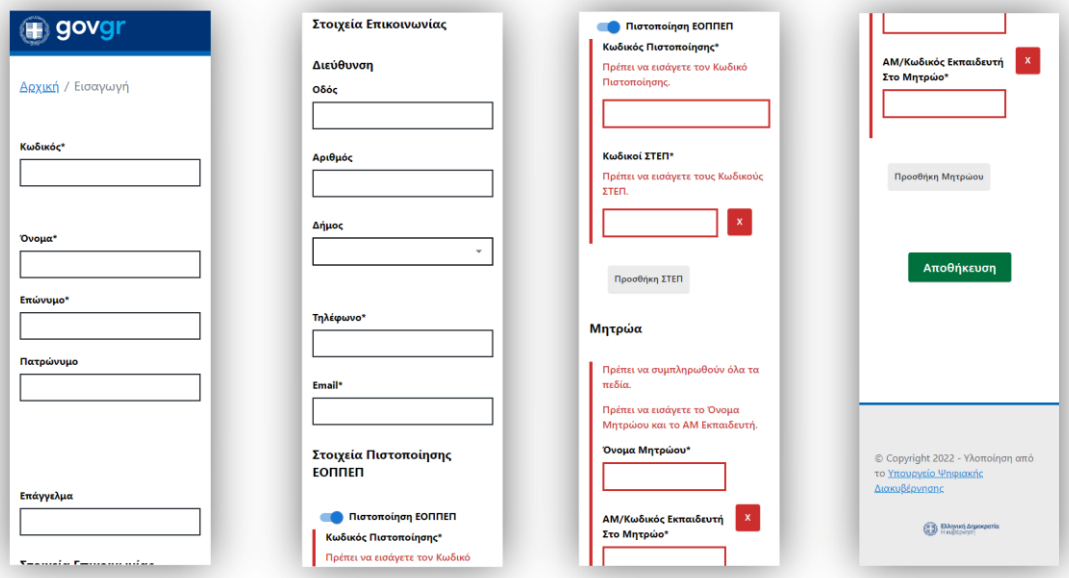
Εικόνα 6.37 Η σελίδα σφαλμάτων

6.4.7 Η εφαρμογή σε κινητά τηλέφωνα

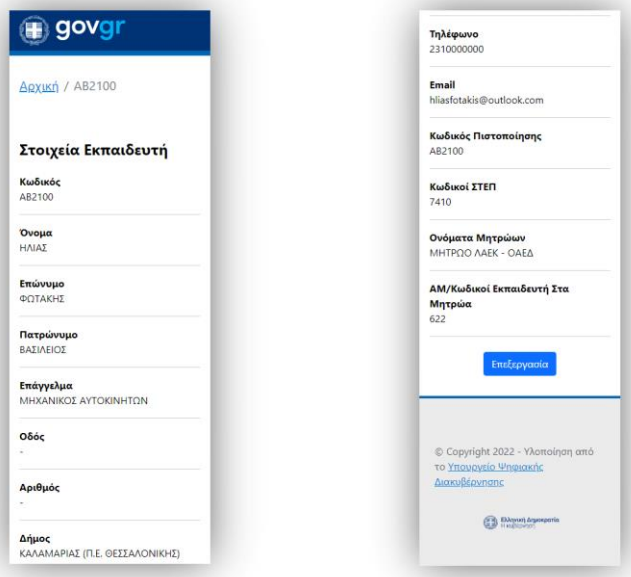
Η εφαρμογή προσαρμόζεται δυναμικά σε διαφορετικές οθόνες και συσκευές (desktops, laptops, tablets, κινητά). Στις παρακάτω εικόνες προβάλλονται οι σελίδες της εφαρμογής σε κινητά τηλέφωνα.



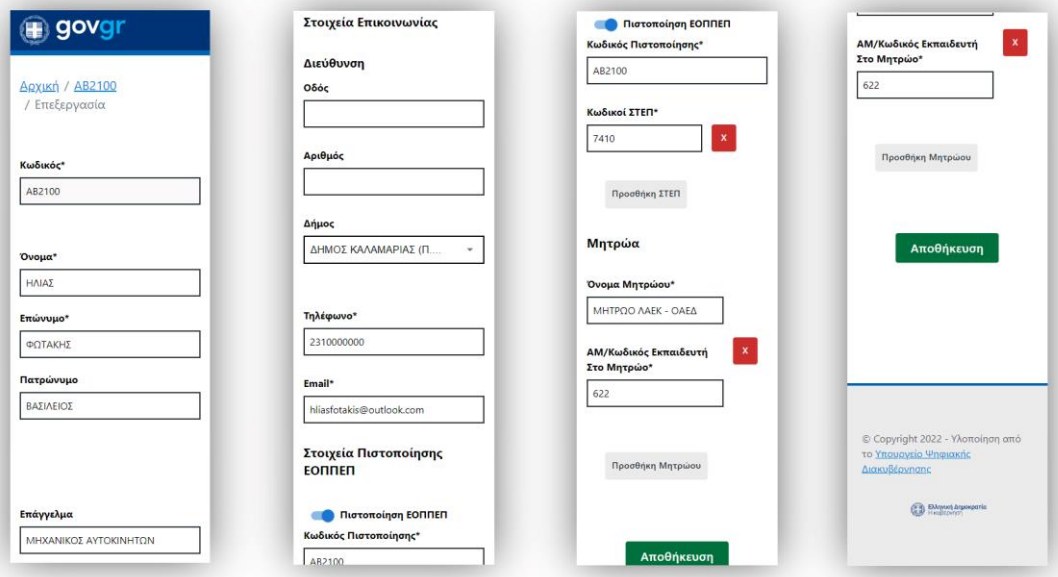
Εικόνα 6.38 Η αρχική σελίδα σε κινητά



Εικόνα 6.39 Η σελίδα της εισαγωγής εκπαιδευτή σε κινητά



Εικόνα 6.40 Η σελίδα της προβολής των στοιχείων ενός εκπαιδευτή σε κινητά



Εικόνα 6.41 Η σελίδα της επεξεργασίας των στοιχείων ενός εκπαιδευτή σε κινητά

Κεφάλαιο 7 – Επίλογος

7.1 Συμπεράσματα

Σκοπός της παρούσας πτυχιακής εργασίας ήταν η εκμάθηση της front-end JavaScript βιβλιοθήκης React και η ανάπτυξη μίας front-end εφαρμογής ιστού. Στόχος της εφαρμογής ήταν να διευκολύνει τους διαχειριστές με τη διαχείριση των εκπαιδευτών ενηλίκων.

Συμπερασματικά, η εφαρμογή ιστού ικανοποιεί τους αρχικούς στόχους και επομένως μπορεί να διευκολύνει και να βελτιώσει τη διαχείριση των εκπαιδευτών ενηλίκων. Δεν είναι μία ολοκληρωμένη εφαρμογή, διότι αναπτύξαμε μόνο το front-end κομμάτι, δηλαδή τη διεπαφή χρήστη. Αν και τηρεί τις αρχές του Συστήματος Σχεδιασμού GOV.GR, θα πρέπει να προστεθεί αυθεντικοποίηση χρήστη και ακόμα να αναπτυχθεί το back-end, ώστε να ενσωματωθεί στις ψηφιακές υπηρεσίες του GOV.GR. Γενικά, με την ίδια λογική μπορούν να αναπτυχθούν εφαρμογές διαχείρισης άλλων οντοτήτων, για παράδειγμα ιατρών, ποδοσφαιριστών. Αυτό που θα αλλάζει θα είναι η οντότητα, τα στοιχεία της, τα πεδία των φορμών και η επικύρωση τους.

Η React με τις λειτουργίες της διευκόλυναν και επιτάχυναν την ανάπτυξη της εφαρμογής ιστού. Βέβαια, λόγω του ότι δεν είναι ένα ολοκληρωμένο framework, ήταν απαραίτητο να μελετηθούν και να χρησιμοποιηθούν και άλλες σύγχρονες τεχνολογίες ανάπτυξης εφαρμογών ιστού. Όλες ανταποκρίθηκαν πλήρως στις απαιτήσεις της εφαρμογής και ο συνδυασμός τους απέφερε το επιθυμητό λειτουργικό αποτέλεσμα.

Η υλοποίηση της πτυχιακής εργασίας ήταν απαιτητική σε όλα τα στάδια της. Αρχικά, έπρεπε να επιλεχθούν και να μελετηθούν πρωτόγνωρες για εμάς τεχνολογίες, γεγονός που αύξησε τον βαθμό δυσκολίας. Επίσης, προέκυψαν διάφορα προβλήματα και σφάλματα (bugs) στην ανάπτυξη της εφαρμογής που αύξησαν τον χρόνο ολοκλήρωσης της. Ωστόσο, αυτά επιλύθηκαν μετά από πολλές αποτυχημένες προσπάθειες. Να σημειωθεί πως υπό κανονικές συνθήκες χρήσης της είναι πιθανόν να εμφανιστούν κι άλλα σφάλματα, τα οποία δεν έχουν εντοπιστεί μέχρι στιγμής. Μέσω της συγκεκριμένης πτυχιακής εργασίας αποκτήθηκαν γνώσεις, δεξιότητες και πρακτική εμπειρία στην ανάπτυξη front-end εφαρμογών ιστού με JS frameworks καθώς και στην ανάπτυξη React φορμών. Επιπλέον, αποκτήθηκε γνώση στον τρόπο ανάπτυξης μίας ψηφιακής υπηρεσίας του GOV.GR.

7.2 Μελλοντικές επεκτάσεις

Μία σημαντική μη λειτουργική απαίτηση της εφαρμογής ιστού ήταν η ικανότητα ενσωμάτωσης μελλοντικών επεκτάσεων. Έτσι, η εφαρμογή σχεδιάστηκε με τρόπο που την καθιστά ευέλικτη σε μελλοντικές επεκτάσεις, καθώς οι ανάγκες των χρηστών αλλάζουν με την πάροδο του χρόνου. Μερικές πιθανές μελλοντικές επεκτάσεις είναι οι ακόλουθες:

- Αυθεντικοποίηση χρήστη: Η αυθεντικοποίηση των διαχειριστών θα διασφαλίσει ότι μόνο εξουσιοδοτημένα άτομα μπορούν να έχουν πρόσβαση στις λειτουργίες και να διαχειρίζονται τα δεδομένα των εκπαιδευτών ενηλίκων.
- Λειτουργία διαγραφής: Μέχρι στιγμής, οι διαχειριστές μπορούν να διαγράψουν έναν εκπαιδευτή χειροκίνητα μέσω του εργαλείου mockAPI. Με τη λειτουργία της διαγραφής, οι διαχειριστές θα μπορούν να διαγράψουν οριστικά έναν εκπαιδευτή απευθείας από την εφαρμογή. Αυτό μπορεί να επιτευχθεί προσθέτοντας ένα κόκκινο κουμπί "Διαγραφή" στη σελίδα της προβολής των στοιχείων του εκπαιδευτή.
- Εξαγωγή των συνολικών εκπαιδευτών σε Excel: Στην αρχική σελίδα, κάτω από τη λίστα θα μπορούσε να υλοποιηθεί ένα γκρι κουμπί "Εξαγωγή σε Excel", το οποίο θα επιτρέπει στους διαχειριστές να εξάγουν τους συνολικούς εισαχθέντες της λίστας σε αρχείο Excel.
- Εξαγωγή των στοιχείων του εκπαιδευτή σε PDF: Στη σελίδα της προβολής των στοιχείων του εκπαιδευτή θα μπορούσε να προστεθεί ένα γκρι κουμπί "Εξαγωγή σε PDF". Μέσω αυτού, οι διαχειριστές θα είχαν τη δυνατότητα να αποθηκεύσουν τα συνολικά στοιχεία ενός συγκεκριμένου εκπαιδευτή σε μορφή PDF.
- Διαφορετικές κατηγορίες και δικαιώματα χρηστών: Στο παρόν στάδιο η εφαρμογή έχει μία κατηγορία χρηστών, τους διαχειριστές, οι οποίοι έχουν πλήρη δικαιώματα. Εκτός από αυτούς, θα μπορούσε να υπάρχουν και οι απλοί διαχειριστές που θα είχαν περιορισμένα δικαιώματα, για παράδειγμα θα μπορούσαν μόνο να αναζητούν εκπαιδευτές και να προβάλλουν τα στοιχεία τους.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] "Introduction to HTML," [Online]. Available: https://www.w3schools.com/html/html_intro.asp. [Accessed 16 March 2022].

[2] "HTML - Βικιπαίδεια," [Online]. Available: https://el.wikipedia.org/wiki/HTML. [Accessed 16 March 2022].

[3] Δ. Γαβαλάς, Β. Κασαπάκης και Θ. Χατζηδημήτρης, Κινητές Τεχνολογίες, Εκδόσεις Νέων Τεχνολογιών, 2015.

[4] "CSS Introduction," [Online]. Available: https://www.w3schools.com/css/css_intro.asp. [Accessed 26 March 2022].

[5] "History Of CSS - The Crazy Programmer," [Online]. Available: https://www.thecrazyprogrammer.com/2021/11/history-of-css.html. [Accessed 28 March 2022].

[6] "CSS Syntax," [Online]. Available: https://www.w3schools.com/css/css_syntax.asp. [Accessed 26 March 2022].

[7] "CSS Selectors," [Online]. Available: https://www.w3schools.com/css/css_selectors.asp. [Accessed 28 March 2022].

[8] "How To Add CSS," [Online]. Available: https://www.w3schools.com/css/css_howto.asp. [Accessed 28 March 2022].

[9] "About JavaScript - JavaScript | MDN," [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript. [Accessed 31 March 2022].

[10] "JavaScript Statements," [Online]. Available: https://www.w3schools.com/js/js_statements.asp. [Accessed 2 April 2022].

[11] "JavaScript Where To," [Online]. Available: https://www.w3schools.com/js/js_whereto.asp. [Accessed 3 April 2022].

[12] "JavaScript - Βικιπαίδεια," [Online]. Available: https://el.wikipedia.org/wiki/JavaScript. [Accessed 4 April 2022].

[13] A. Rauschmayer, Speaking JavaScript, O'Reilly Media, 2014.

[14] "Introduction to client-side frameworks - Learn web development | MDN," [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/Introduction. [Accessed 23 June 2022].

[15] "What is a JavaScript Framework?," [Online]. Available: https://linuxhint.com/what-is-a-javascript-framework/. [Accessed 24 June 2022].

[16] "Angular vs React vs Vue - Maximilian Schwarzmüller," [Online]. Available: https://academind.com/tutorials/angular-vs-react-vs-vue-my-thoughts. [Accessed 20 July 2022].

[17] "Angular vs. React vs. Vue.js - choosing a JavaScript framework for your project," [Online]. Available: https://relevant.software/blog/angular-vs-react-vs-vue-js-choosing-a-javascript-framework-for-your-project/. [Accessed 19 July 2022].

[18] "Angular," [Online]. Available: https://angular.io/. [Accessed 18 July 2022].

[19] "React - A JavaScript library for building user interfaces," [Online]. Available: https://reactjs.org/. [Accessed 15 April 2022].

[20] "Vue.js - The Progressive JavaScript Framework | Vue.js," [Online]. Available: https://vuejs.org/. [Accessed 18 July 2022].

[21] "Angular vs. React vs. Vue.js: Comparing performance | LogRocket Blog," [Online]. Available: https://blog.logrocket.com/angular-vs-react-vs-vue-js-comparing-performance/. [Accessed 20 July 2022].

[22] "Stack Overflow Developer Survey 2022," [Online]. Available: https://survey.stackoverflow.co/2022/#section-most-popular-technologies-web-frameworks-and-technologies. [Accessed 5 July 2022].

[23] "@angular/core vs react vs vue | npm trends," [Online]. Available: https://npmtrends.com/@angular/core-vs-react-vs-vue. [Accessed 5 July 2022].

[24] "React (JavaScript library) - Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/React_(JavaScript_library). [Accessed 15 April 2022].

[25] A. Banks and E. Porcello, Learning React: Modern Patterns for Developing React Apps, 2nd ed., O'Reilly Media, 2020.

[26] "Tutorial: Intro to React - React," [Online]. Available: https://reactjs.org/tutorial/tutorial.html. [Accessed 15 April 2022].

[27] "JSX In Depth - React," [Online]. Available: https://reactjs.org/docs/jsx-in-depth.html. [Accessed 19 September 2022].

[28] R. Wieruch, The Road to React (2021 Edition).

[29] "Introducing JSX - React," [Online]. Available: https://reactjs.org/docs/introducing-jsx.html. [Accessed 18 April 2022].

[30] "State and Lifecycle - React," [Online]. Available: https://reactjs.org/docs/state-and-lifecycle.html. [Accessed 2 October 2022].

[31] "React Props," [Online]. Available: https://www.w3schools.com/react/react_props.asp. [Accessed 22 April 2022].

[32] "Hooks at a Glance - React," [Online]. Available: https://reactjs.org/docs/hooks-overview.html. [Accessed 26 April 2022].

[33] "Hooks API Reference - React," [Online]. Available: https://reactjs.org/docs/hooks-reference.html. [Accessed 26 April 2022].

[34] "React: The Virtual DOM | Codecademy," [Online]. Available: https://www.codecademy.com/article/react-virtual-dom. [Accessed 23 April 2022].

[35] "Conditional Rendering - React," [Online]. Available: https://reactjs.org/docs/conditional-rendering.html. [Accessed 7 October 2022].

[36] "3 Ways to Implement Conditional Rendering in React," [Online]. Available: https://www.knowledgehut.com/blog/web-development/ways-to-implement-conditional-rendering-in-react. [Accessed 8 October 2022].

[37] "Using Forms in React," [Online]. Available: https://daveceddia.com/react-forms/. [Accessed 19 October 2022].

[38] "Controlled and uncontrolled form inputs in React don't have to be complicated - Gosha Arinich," [Online]. Available: https://goshacmd.com/controlled-vs-uncontrolled-inputs-react/. [Accessed 24 October 2022].

[39] "Client-side form validation - Learn web development | MDN," [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn/Forms/Form_validation. [Accessed 1 November 2022].

[40] "Choosing the best React form library for your project," [Online]. Available: https://retool.com/blog/choosing-a-react-form-library/. [Accessed 9 November 2022].

[41] "Introduction - Bootstrap v5.1," [Online]. Available: https://getbootstrap.com/docs/5.1/getting-started/introduction/. [Accessed 1 May 2022].

[42] "Bootstrap (front-end framework) - Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Bootstrap_(front-end_framework). [Accessed 1 May 2022].

[43] "Grid system - Bootstrap v5.1," [Online]. Available: https://getbootstrap.com/docs/5.1/layout/grid/. [Accessed 1 May 2022].

[44] "Breakpoints - Bootstrap v5.1," [Online]. Available: https://getbootstrap.com/docs/5.1/layout/breakpoints/. [Accessed 1 May 2022].

[45] "Containers - Bootstrap v5.1," [Online]. Available: https://getbootstrap.com/docs/5.1/layout/containers/. [Accessed 2 May 2022].

[46] "About - Boostrap v5.1," [Online]. Available: https://getbootstrap.com/docs/5.1/about/overview/. [Accessed 2 May 2022].

[47] "Download - Bootstrap v5.1," [Online]. Available: https://getbootstrap.com/docs/5.1/getting-started/download/. [Accessed 2 May 2022].

[48] "MUI: The React component library you always wanted," [Online]. Available: https://mui.com/. [Accessed 3 May 2022].

[49] "Design your service with the look and feel of GOV.GR | Design your service with the look and feel of GOV.GR," [Online]. Available: https://guide.services.gov.gr/. [Accessed 7 June 2022].

[50] "React Router | Tutorial," [Online]. Available: https://reactrouter.com/docs/en/v6/getting-started/tutorial. [Accessed 4 May 2022].

[51] "ReactJS | Router - GeeksForGeeks," [Online]. Available: https://www.geeksforgeeks.org/reactjs-router/. [Accessed 4 May 2022].

[52] "React Router | Installation," [Online]. Available: https://reactrouter.com/docs/en/v6/getting-started/installation. [Accessed 5 May 2022].

[53] "React Router | Main Concepts," [Online]. Available: https://reactrouter.com/docs/en/v6/getting-started/concepts. [Accessed 5 May 2022].

[54] "npm (software) - Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Npm_(software). [Accessed 9 May 2022].

[55] "About npm | npm Docs," [Online]. Available: https://docs.npmjs.com/about-npm. [Accessed 8 May 2022].

[56] "Visual Studio Code - Code Editing. Redefined," [Online]. Available: https://code.visualstudio.com/. [Accessed 10 May 2022].

[57] "Visual Studio Code - Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Visual_Studio_Code. [Accessed 10 May 2022].

[58] S. Chacon and B. Straub, Pro Git, 2nd ed., Apress, 2014.

[59] "The One DevOps Platform | GitLab," [Online]. Available: https://about.gitlab.com/. [Accessed 13 May 2022].

[60] "Issues | GitLab," [Online]. Available: https://docs.gitlab.com/ee/user/project/issues/. [Accessed 13 May 2022].

[61] "JSON," [Online]. Available: https://www.json.org/json-en.html. [Accessed 16 May 2022].

[62] "JSON Schema | The home of JSON Schema," [Online]. Available: https://json-schema.org/. [Accessed 17 May 2022].

[63] "JSON Schema," [Online]. Available: https://restfulapi.net/json-schema/. [Accessed 17 May 2022].

[64] "Getting Started Step-By-Step | JSON Schema," [Online]. Available: https://json-schema.org/learn/getting-started-step-by-step.html. [Accessed 17 May 2022].

[65] "What is REST - REST API Tutorial," [Online]. Available: https://restfulapi.net/. [Accessed 22 May 2022].

[66] "RESTful Services Quick Tips," [Online]. Available: https://www.restapitutorial.com/lessons/restquicktips.html. [Accessed 23 May 2022].

[67] "What Is REST?," [Online]. Available: https://www.restapitutorial.com/lessons/whatisrest.html. [Accessed 22 May 2022].

[68] "mockAPI," [Online]. Available: https://mockapi.io/. [Accessed 27 May 2022].

[69] "mockAPI Docs," [Online]. Available: https://mockapi.io/docs. [Accessed 27 May 2022].

[70] "Getting Started | Axios Docs," [Online]. Available: https://axios-http.com/docs/intro. [Accessed 1 June 2022].

[71] "How To Use Axios With React: The Definitive Guide (2021)," [Online]. Available: https://www.freecodecamp.org/news/how-to-use-axios-with-react/. [Accessed 1 June 2022].

[72] "Figma: the collaborative interface design tool.," [Online]. Available: https://www.figma.com/. [Accessed 22 December 2022].

[73] "What Are Workflow Diagrams? | IBM," [Online]. Available: https://www.ibm.com/cloud/blog/workflow-diagrams. [Accessed 16 February 2023].

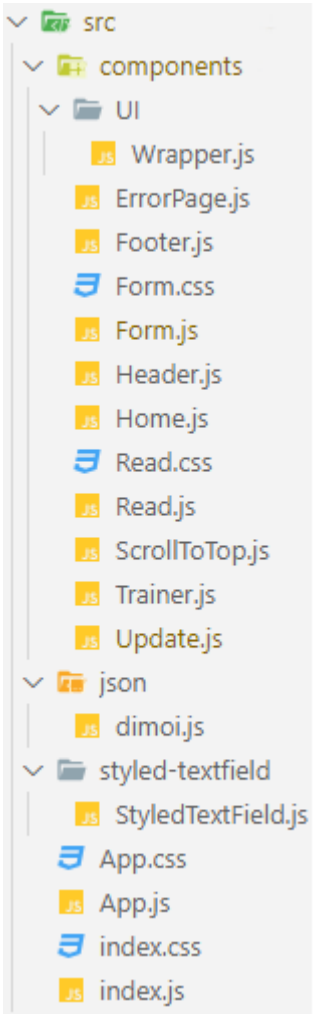
[74] "Τι είναι το Mock Up," [Online]. Available: https://thedesignstore.gr/post.asp?blog_Articles_id=31633. [Accessed 18 February 2023].

ΑΚΡΩΝΥΜΙΑ - ΣΥΝΤΟΜΟΓΡΑΦΙΕΣ

HTML:	HyperText Markup Language
SVG:	Scalable Vector Graphics
CSS:	Cascading Style Sheets
JS:	JavaScript
DOM:	Document Object Model
URL:	Uniform Resource Locator
W3C:	World Wide Web Consortium
ES:	ECMAScript
UI:	User Interface
HTTP:	HyperText Transfer Protocol
CLI:	Command-Line Interface
API:	Application Programming Interface
CDN:	Content Delivery Network
MUI:	Material UI
JSON:	JavaScript Object Notation
REST:	REpresentational State Transfer
CRUD:	Create, Read, Update, and Delete
URI:	Uniform Resource Identifier
XML:	Extensible Markup Language
HATEOAS:	Hypermedia as the Engine of Application State
CSRF:	Cross-Site Request Forgery
UX:	User Experience

ΠΑΡΑΡΤΗΜΑ

Στο παράρτημα παρατίθεται και επεξηγείται ο συνολικός κώδικας της εφαρμογής. Στην εικόνα Α1 απεικονίζεται η δομή των δημιουργημένων αρχείων της εφαρμογής.



Εικόνα Α1 Η δομή των δημιουργημένων αρχείων

Για να αποφύγουμε την επανάληψη να αναφερθεί ότι στην αρχή των περισσότερων αρχείων-components που έχουμε δημιουργήσει, εισάγουμε τις απαραίτητες εξαρτήσεις από τις βιβλιοθήκες, τα απαραίτητα components και styles από τα αντίστοιχα αρχεία, προκειμένου να τα χρησιμοποιήσουμε στα αρχεία. Αυτές τις εισαγωγές τις πραγματοποιούμε με τη δήλωση **import** που είναι JavaScript module. Επιπλέον, στο τέλος του κάθε αρχείου-component

εξάγουμε το component με τη δήλωση **export default**, η οποία είναι και αυτή JavaScript module, ώστε να το χρησιμοποιήσουμε στα αρχεία που χρειάζεται.

Τέλος, στα HTML στοιχεία εντός των περισσότερων components χρησιμοποιούμε προκαθορισμένες κλάσεις CSS του Digigov CSS και του Bootstrap για στυλ του GOV.GR, για τη διάταξη και για το responsiveness. Ακόμα, χρησιμοποιούμε δικές μας custom κλάσεις CSS για περαιτέρω μορφοποίηση.

Αρχείο index.js

Το αρχείο **index.js** είναι το πρώτο που εκτελείται κατά την εκτέλεση της εφαρμογής. Εντός του αρχείου καλούμε τη μέθοδο **ReactDOM.render( )** με δύο ορίσματα, όπου το πρώτο είναι το root component **<App />** που θέλουμε να αποδώσουμε και το δεύτερο είναι το DOM στοιχείο που θέλουμε να αποδώσουμε την εφαρμογή. Ουσιαστικά, αυτό το κομμάτι κώδικα αποδίδει την εφαρμογή στην ιστοσελίδα.

```
1. import React from 'react';
2. import ReactDOM from 'react-dom';
3. import './index.css';
4. import App from './App';
5.
6. ReactDOM.render(
7.   <React.StrictMode>
8.     <App />
9.   </React.StrictMode>,
10.  document.getElementById('root')
11.);
```

Αρχείο-component App.js

Το συγκεκριμένο είναι το root component της εφαρμογής και αποδίδεται απευθείας στην οθόνη. Γενικά περιέχει όλα τα components που έχουμε δημιουργήσει. Εντός του αρχείου δημιουργούμε το function component "App" ορίζοντας τη JavaScript συνάρτηση **function App( ) { return ( ); }**. Μέσα στο **return** τοποθετούμε ως γονέα-component το **<Router>** της βιβλιοθήκης React Router για τη διαχείριση της δρομολόγησης και της πλοήγησης. Πιο συγκεκριμένα, όπως έχει αναφερθεί πάλι χρησιμοποιούμε το **<Router>** για να λειτουργήσουν

όλα τα components και hooks του React Router που χρησιμοποιούμε και για τον συγχρονισμό της διεπαφής χρήστη με τις διευθύνσεις URL.

Εντός του **<Router>** χρησιμοποιούμε το component **<Routes>** της βιβλιοθήκης React Router, όπου μέσα του περικλείουμε τις διαδρομές της εφαρμογής, τις οποίες τις ορίζουμε χρησιμοποιώντας το component **<Route>**. Σε κάθε **<Route>** προσθέτουμε τις ιδιότητες **exact path** και **element**, όπου στο **path** ενσωματώνουμε το μοτίβο διαδρομής και στο **element** ενσωματώνουμε το component που θα εμφανίζεται στην οθόνη όταν το μοτίβο διαδρομής ταιριάξει με την τρέχουσα διεύθυνση URL. Για παράδειγμα, αν η τρέχουσα διεύθυνση URL είναι "http://localhost:3000/create", τότε θα εμφανιστεί το component **<Form />** δηλαδή η σελίδα της εισαγωγής εκπαιδευτή, επειδή του έχουμε ενσωματώσει το **path="/create"**. Στη γραμμή 20 του κώδικα έχουμε προσθέσει στο **path** του **<Route>** το δυναμικό τμήμα **:id**, ώστε να εμφανίζεται η σελίδα της προβολής των στοιχείων του αντίστοιχου εκπαιδευτή. Για παράδειγμα, αν η τρέχουσα διεύθυνση είναι "http://localhost:3000/trainers/3", τότε θα εμφανιστεί η σελίδα της προβολής των στοιχείων του εκπαιδευτή για τον εκπαιδευτή που έχει **id:3** στο mockAPI (fake API). Στη γραμμή 21 πραγματοποιούμε το ίδιο για τη σελίδα της επεξεργασίας των στοιχείων του αντίστοιχου εκπαιδευτή.

Στη συνέχεια, πάνω από το **<Routes>** κάνουμε χρήση των components **<ScrollToTop />** και **<Header />**. Το **<ScrollToTop />** το χρησιμοποιούμε για να μετακινείται αυτόματα η κάθετη μπάρα κύλισης στην κορυφή όταν γίνεται αλλαγή σελίδας. Το **<Header />** είναι η κεφαλίδα της εφαρμογής και εμφανίζεται στο πάνω μέρος της κάθε σελίδας. Τέλος, κάτω από το **<Routes>** χρησιμοποιούμε το component **<Footer />**, το οποίο είναι το υποσέλιδο της εφαρμογής και εμφανίζεται στο κάτω μέρος της κάθε σελίδας.

```
1. import React from "react";
2. import { BrowserRouter as Router, Route, Routes } from "react-router-dom";
3. import ScrollToTop from "./components/ScrollToTop";
4. import Header from "./components/Header";
5. import Home from "./components/Home";
6. import Form from "./components/Form";
7. import Trainer from "./components/Trainer";
8. import Update from "./components/Update";
9. import Footer from "./components/Footer";
10.
11.function App() {
12.
13.  return (
14.    <Router>
15.      <ScrollToTop />
```

```
      <Header />
      <Routes>
        <Route exact path="/" element={<Home />} />
        <Route exact path="/create" element={<Form />} />
        <Route exact path="/trainers/:id" element={<Trainer />} />
        <Route exact path="/trainers/update/:id" element={<Update />} />
      </Routes>
      <Footer />
    </Router>
  );
}

export default App;
```

Αρχείο-component ScrollToTop.js

Στο αρχείο αυτό δημιουργούμε και εξάγουμε το function component "ScrollToTop", ορίζοντας το **export default function ScrollToTop( ) { }**. Μέσα στο component δημιουργούμε τη μεταβλητή **const { pathname }** και της αποθηκεύουμε το **useLocation( )** hook του React Router. Το **useLocation( )** επιστρέφει το αντικείμενο location που αντιπροσωπεύει το τρέχον URL. Στη συνέχεια χρησιμοποιούμε το **useEffect( )** hook με δύο ορίσματα, όπου το πρώτο είναι μία JavaScript συνάρτηση, ενώ το δεύτερο είναι ο πίνακας των εξαρτήσεων του. Στη JavaScript συνάρτηση καλούμε το **window.scrollTo(0, 0)**, ενώ στον πίνακα των εξαρτήσεων προσθέτουμε τη μεταβλητή **pathname**. Το **window.scrollTo(0, 0)** μετακινεί την κάθετη μπάρα κύλισης στην κορυφή της σελίδας και θα εκτελείται κάθε φορά που αλλάζει η τιμή του **pathname**, δηλαδή όταν αλλάζει η σελίδα και το URL. Τέλος επιστρέφουμε **null**, διότι δεν θέλουμε το component να αποδίδει περιεχόμενο στην οθόνη.

```
import { useEffect } from "react";
import { useLocation } from "react-router-dom";

export default function ScrollToTop() {
  const { pathname } = useLocation();

  useEffect(() => {
    window.scrollTo(0, 0);
  }, [pathname]);

  return null;
}
```

Αρχείο-component Header.js

Στο συγκεκριμένο αρχείο δημιουργούμε την κεφαλίδα της εφαρμογής. Εντός του δημιουργούμε το function component "Header" ορίζοντας τη JavaScript συνάρτηση βέλους (arrow function) **const Header = ( ) => { return ( ); }**. Μέσα στο **return** αξιοποιούμε το σημασιολογικό στοιχείο **<header>** για να καθορίσουμε την κεφαλίδα και εντός του χρησιμοποιούμε την ετικέτα **<img/>** με τις ιδιότητες της για να προσθέσουμε το λογότυπο του GOV.GR. Την ετικέτα **<img/>** την περικλείουμε εντός του component **<Link>** του React Router για να λειτουργεί το λογότυπο ως σύνδεσμος πλοήγησης. Στο **<Link>** προσθέτουμε την ιδιότητα **to** με την τιμή "/", προκειμένου να γίνεται μετάβαση στην αρχική σελίδα.

```
1. import { Link } from "react-router-dom";
2.
3. const Header = () => {
4.   return (
5.     <header className="govgr-header" role="banner" data-module="govgr-
   header">
6.       <div className="govgr-header__container">
7.         <Link to="/">
8.           <img
9.             className="govgr-header-logo"
10.            src="https://www.gov.gr/gov_gr_logo.svg"
11.            alt="govgr logo"
12.          />
13.        </Link>
14.      </div>
15.    </header>
16.  );
17.};
18.
19.export default Header;
```

Αρχείο-component Footer.js

Στο αρχείο αυτό κατασκευάζουμε το υποσέλιδο της εφαρμογής. Στον κώδικα ορίζουμε τη JavaScript συνάρτηση βέλους (arrow function) **const Footer = ( ) => { return ( ); }** και

δημιουργούμε το function component "Footer". Στο εσωτερικό του **return** κάνουμε χρήση του σημασιολογικού στοιχείου <**footer**> για να καθορίσουμε ότι αυτό είναι το υποσέλιδο και εντός του χρησιμοποιούμε διάφορα στοιχεία <**div**>.

Στις γραμμές 8 έως 22 εισάγουμε με την ετικέτα <**p**> μια παράγραφο για τα πνευματικά δικαιώματα και εντός της χρησιμοποιούμε την ετικέτα <**a**> με τις ιδιότητες της για να μετατρέψουμε σε υπερσύνδεσμο ένα τμήμα της παραγράφου, δηλαδή το "Υπουργείο Ψηφιακής Διακυβέρνησης". Στην ετικέτα <**a**> έχουμε προσθέσει την ιδιότητα **href="https://mindigital.gr/"**, ώστε όταν γίνεται κλικ στον υπερσύνδεσμο να μεταβαίνει στη διεύθυνση "https://mindigital.gr/". Ακόμα έχουμε προσθέσει την ιδιότητα **target="_blank"** για να ανοίγει σε καινούρια καρτέλα. Στη γραμμή 26 προσθέτουμε το λογότυπο της Ελληνικής κυβέρνησης με την ετικέτα <**img/**>.

```
1. const Footer = () => {
2.   return (
3.     <footer className="govgr-footer" role="contentinfo">
4.       <div className="govgr-width-container">
5.         <div className="govgr-footer__meta">
6.           <div className="govgr-footer__meta-item govgr-footer__meta-item--
   grow">
7.             <div className="govgr-footer__content">
8.               <p className="govgr-footer__licence-description">
9.                 © Copyright 2022 - Υλοποίηση από το{" "}
10.                <a
11.                  href="https://mindigital.gr/"
12.                  target="_blank"
13.                  rel="noreferrer noopener"
14.                  className="govgr-link"
15.                >
16.                  {" "}
17.                  Υπουργείο Ψηφιακής Διακυβέρνησης{" "}
18.                  <span className="govgr-visually-hidden">
19.                    (ανοίγει σε καινούρια καρτέλα)
20.                  </span>
21.                </a>
22.              </p>
23.            </div>
24.          </div>
25.          <div className="govgr-footer__meta-item">
26.            <img src="https://government.gov.gr/wp-
   content/uploads/2017/09/kyvernisi-logo-big.png" alt="kyvernisi-logo"
   className="govgr-footer__government-logo" />
27.          </div>
28.        </div>
```

```
29.       </div>
30.     </footer>
31.   );
32. };
33.
34. export default Footer;
```

Αρχείο-component Home.js

Σε αυτό το αρχείο δημιουργούμε την αρχική σελίδα της εφαρμογής. Εντός του δημιουργούμε το function component "Home" με τη JavaScript arrow function **const Home = ( ) => { return ( ); }**. Στο **return** κομμάτι χρησιμοποιούμε το HTML στοιχείο ** ** ως γονέα και του ενσωματώνουμε την προκαθορισμένη κλάση **"container-sm"** του Bootstrap, η οποία είναι ένα container που μας βοηθάει με τη διάταξη και το responsiveness της αρχικής σελίδας. Το πλάτος του container είναι 100% μέχρι το small breakpoint και προσαρμόζεται στα υπόλοιπα breakpoints (md, lg, xl, xxl).

Μέσα στο container περικλείουμε το περιεχόμενο της αρχικής σελίδας. Ειδικότερα, στη γραμμή 8 ορίζουμε την επικεφαλίδα "Εκπαιδευτές Ενηλίκων" με την ετικέτα **<h1>**. Στις γραμμές 10 έως 12 δημιουργούμε το κουμπί "Εισαγωγή Εκπαιδευτή" με χρήση της ετικέτας **<button>** και την περιλαμβάνουμε εντός του component **<Link>**. Στο **<Link>** προσθέτουμε την ιδιότητα **to="/create"**, ώστε με το πάτημα του κουμπιού να γίνεται πλοήγηση στη σελίδα της εισαγωγής εκπαιδευτή. Στη γραμμή 14 χρησιμοποιούμε το component **<Read />** για να αποδώσουμε στην οθόνη το υπόλοιπο περιεχόμενο της αρχικής σελίδας.

```
1. import { Link } from "react-router-dom";
2. import Read from "./Read";
3. import "./Read.css";
4.
5. const Home = () => {
6.   return (
7.     <div className="container-sm">
8.       <h1 className="govgr-heading-l gap-top text-center">Εκπαιδευτές
   Ενηλίκων</h1>
9.       <div className="div-create space-top">
10.        <Link to="/create">
11.          <button className="govgr-btn govgr-btn-primary btn-
   create">Εισαγωγή Εκπαιδευτή</button>
```

```
12.         </Link>
13.       </div>
14.       <Read />
15.     </div>
16.   );
17.};
18.
19.export default Home;
```

Αρχείο-component Read.js

Στο συγκεκριμένο αρχείο δημιουργούμε το υπόλοιπο περιεχόμενο της αρχικής σελίδας, δηλαδή τη λίστα με τους εισαχθέντες εκπαιδευτές, τα φίλτρα της και τη σελιδοποίηση της. Αρχικά, δημιουργούμε το function component "Read" με τη JavaScript arrow function **const Read = ( ) => { return ( ); }**. Όπως αναφέρθηκε προηγουμένως το component αυτό το αξιοποιούμε εντός του component "Home".

Στις γραμμές 12 έως 20 ορίζουμε states (καταστάσεις) για την αποθήκευση και την ενημέρωση των δεδομένων καλώντας το useState hook της React. Πιο αναλυτικά, στη γραμμή 12 καλούμε το **useState( )** και αυτό επιστρέφει έναν πίνακα με δύο στοιχεία, τα οποία τα αποθηκεύουμε σε ξεχωριστές μεταβλητές χρησιμοποιώντας τη JavaScript σύνταξη array destructuring **const [trainers, setTrainers]**. Επιπλέον ενσωματώνουμε έναν άδειο πίνακα **([ ])** για αρχική τιμή. Η μεταβλητή **trainers** είναι η τρέχουσα τιμή της κατάστασης, ενώ το **setTrainers** είναι η συνάρτηση για την ενημέρωση της. Αυτό το state το χρησιμοποιούμε για την αποθήκευση της λίστας των εκπαιδευτών.

Στις γραμμές 13 έως 20 ορίζουμε με παρόμοιο τρόπο και τα υπόλοιπα states, ωστόσο χρησιμοποιούμε διαφορετικά ονόματα μεταβλητών και διαφορετικές αρχικές τιμές. Το **const [error, setError] = useState(null);** είναι για την αποθήκευση των HTTP σφαλμάτων. Το **const [searchTerm, setSearchTerm] = useState("");** είναι για την αποθήκευση της τιμής που εισάγει ο χρήστης στο πεδίο (input) "Αναζήτηση". Το **const [filterCert, setFilterCert] = useState("");** είναι για την αποθήκευση της τιμής που επιλέγει ο χρήστης στην αναπτυσσόμενη λίστα (select) "Πιστοποιημένοι". Το **const [filterMunicipality, setFilterMunicipality] = useState(null);** είναι για την αποθήκευση της τιμής που επιλέγει ο χρήστης στην αναπτυσσόμενη λίστα (select) "Δήμος". Το **const [sortTerm, setSortTerm] =**

useState(""); είναι για την αποθήκευση της τιμής που επιλέγει ο χρήστης στην αναπτυσσόμενη λίστα (Autocomplete) "Ταξινόμηση κατά". Το **const [currentPage, setCurrentPage] = useState(1);** είναι για την αποθήκευση και ενημέρωση του τρέχοντα αριθμού σελίδας της σελιδοποίησης (pagination). Το **const [totalTrainers, setTotalTrainers] = useState(0);** είναι για την αποθήκευση του συνολικού αριθμού των φιλτραρισμένων εκπαιδευτών. Το **const [trainersPerPage, setTrainersPerPage] = useState(5);** είναι για την αποθήκευση της τιμής που επιλέγει ο χρήστης στην αναπτυσσόμενη λίστα (select) "Εκπαιδευτές / σελίδα" της σελιδοποίησης.

Στις γραμμές 22 έως 32 κάνουμε χρήση του **useEffect( )** hook της React για την ανάκτηση των εκπαιδευτών από το mockAPI. Στο **useEffect( )** ενσωματώνουμε δύο ορίσματα, όπου το πρώτο όρισμα είναι μια callback συνάρτηση και το δεύτερο είναι ο πίνακας με τις εξαρτήσεις.

Μέσα στη συνάρτηση εκτελούμε ένα HTTP GET αίτημα προς το mockAPI αξιοποιώντας τη μέθοδο **get( )** της βιβλιοθήκης Axios και στο **get( )** προσθέτουμε το endpoint `https://61e53aed595afe00176e5423.mockapi.io/trainers` που είναι αποθηκευμένοι οι εκπαιδευτές. Αν το αίτημα ολοκληρωθεί επιτυχώς, τότε εκτελείται ο κώδικας της callback συνάρτησης της **then( )** μεθόδου, όπου καλούμε τη συνάρτηση ενημέρωσης (state updater function) **setTrainers( )** με όρισμα το αντικείμενο **response** με την ιδιότητα **.data** για να αποθηκεύσουμε τους εκπαιδευτές στη μεταβλητή κατάστασης (state variable) **trainers**. Ακόμα καλούμε το **setError( )** και του ενσωματώνουμε την τιμή **null**. Διαφορετικά αν το αίτημα αποτύχει, τότε εκτελείται το **catch** τμήμα, στο οποίο καλούμε το **setError( )** με όρισμα το αντικείμενο **error** για να αποθηκεύσουμε το HTTP σφάλμα στη μεταβλητή κατάστασης **error**. Να επισημανθεί ότι η συνάρτηση του **useEffect( )** εκτελείται μία φορά όταν το component τοποθετείται αρχικά στο DOM, δηλαδή όταν αποδίδεται για πρώτη φορά στην οθόνη. Ο λόγος που γίνεται αυτό είναι επειδή έχουμε ενσωματώσει ως δεύτερο όρισμα έναν άδειο πίνακα εξαρτήσεων.

Στις γραμμές 34 έως 38 υπολογίζουμε τους αριθμούς της σελιδοποίησης της λίστας των εκπαιδευτών. Στις γραμμές 40 έως 103 δηλώνουμε τη μεταβλητή **trainersData** και της αποθηκεύουμε το React hook **useMemo( )**. Στο **useMemo( )** περνάμε δύο παραμέτρους, όπου η πρώτη παράμετρος είναι μία callback συνάρτηση και η δεύτερη είναι ο πίνακας των εξαρτήσεων. Το **useMemo( )** απομνημονεύει την υπολογισμένη τιμή μεταξύ των re-renders και υπολογίζει εκ νέου την τιμή μόνο αν αλλάξει κάποια εξάρτηση. Αυτό έχει ως αποτέλεσμα να βελτιστοποιείται η απόδοση αφού αποφεύγονται οι περιττοί επανυπολογισμοί.

Σε γενικές γραμμές στη συνάρτηση του **useMemo( )** φιλτράρουμε και ταξινομούμε τη λίστα των εκπαιδευτών βάσει των επιλεγμένων φίλτρων και των επιλογών ταξινόμησης. Επιπλέον, υπολογίζουμε τους εκπαιδευτές που θα εμφανίζονται στην τρέχουσα σελίδα. Ειδικότερα, στη γραμμή 41 δηλώνουμε την τοπική μεταβλητή **computedTrainers** και της αποθηκεύουμε τη μεταβλητή **trainers** που είναι ο πίνακας με όλους τους εκπαιδευτές.

Στις γραμμές 43 έως 53 δημιουργούμε μία δήλωση **if** με συνθήκη τη μεταβλητή κατάστασης **searchTerm**, όπου ελέγχουμε την εισαγόμενη τιμή στο πεδίο "Αναζήτηση". Η συνθήκη θα είναι true αν υπάρχει ένας ή περισσότεροι χαρακτήρες στο πεδίο "Αναζήτηση". Αν η συνθήκη είναι αληθής, τότε φιλτράρουμε τον πίνακα **computedTrainers** με τη μέθοδο **filter( )**. Στο **filter( )** ενσωματώνουμε ως παράμετρο μία callback συνάρτηση, στην οποία περνάμε ως όρισμα το **trainer** και εντός της ελέγχουμε με τη μέθοδο **includes( )** αν μερικές ιδιότητες (properties) του κάθε αντικειμένου (object) **trainer** περιέχουν την τιμή που έχει εισαχθεί στο πεδίο. Για την ακρίβεια στη γραμμή 46 ελέγχουμε αν η ιδιότητα **code** (Κωδικός) του **trainer** περιέχει την εισαγόμενη τιμή στο πεδίο. Στη γραμμή 47 ελέγχουμε αν την περιέχει το **trainer.certification.code** (Κωδικός ΕΟΠΠΕΠ).

Στη γραμμή 48 ελέγχουμε αν την περιέχει κάποιος Κωδικός στα Μητρώα, αξιοποιώντας τη μέθοδο **some( )** στον πίνακα **trainer.registries**. Στη μέθοδο **some( )** περνάμε ως παράμετρο μια callback συνάρτηση, στην οποία περνάμε ως όρισμα το **registry** και εντός της πραγματοποιούμε τον έλεγχο.

Στη γραμμή 49 χρησιμοποιούμε πάλι τη μέθοδο **some( )** αλλά στον πίνακα **trainer.certification.step** για να ελέγξουμε αν κάποιος από τους Κωδικούς ΣΤΕΠ περιέχει την εισαγόμενη τιμή.

Στις γραμμές 50 και 51 ελέγχουμε αν το όνομα ή το επώνυμο περιέχει την τιμή που έχει εισαχθεί. Η σύνταξη **`${trainer.name.first} ${trainer.name.last}`** που χρησιμοποιούμε ονομάζεται template literals και μας επιτρέπει να τα ελέγχουμε και συνδυαστικά (Ονοματεπώνυμο).

Όλες τις παραπάνω εκφράσεις τις ενώνουμε με τον λογικό τελεστή OR **(||)**. Επιπλέον μετατρέπουμε με τη μέθοδο **toLowerCase( )** σε πεζούς χαρακτήρες τις τιμές των ιδιοτήτων και την εισαγόμενη τιμή, προκειμένου να πραγματοποιηθούν οι έλεγχοι σωστά. Τέλος μόλις ολοκληρωθούν οι έλεγχοι στον κάθε εκπαιδευτή, ο πίνακας **computedTrainers** θα περιέχει μόνο τους εκπαιδευτές που ικανοποιούν κάποια συνθήκη.

Στις γραμμές 55 έως 60 κατασκευάζουμε μία δήλωση **if** με συνθήκη τη μεταβλητή **filterMunicipality** για να ελέγξουμε την τιμή που έχει επιλεχθεί στην αναπτυσσόμενη λίστα "Δήμος". Αν έχει επιλεχθεί κάποιος δήμος, τότε φιλτράρουμε τον πίνακα **computedTrainers** με τη μέθοδο **filter( )** όπως στην προηγούμενη **if**. Η διαφορά είναι ότι στην callback συνάρτηση μετατρέπουμε αρχικά σε JSON string τον επιλεγμένο δήμο (**filterMunicipality**) και τον καταχωρημένο δήμο του εκπαιδευτή (**trainer.contact.address.municipality**) με τη μέθοδο **JSON.stringify( )** και έπειτα τα συγκρίνουμε με τον τελεστή ===. Έτσι, αν οι δύο τιμές είναι ίσες και ίδιου τύπου, τότε ο εκπαιδευτής θα περιέχεται στον πίνακα **computedTrainers**.

Στις γραμμές 62 έως 66 ελέγχουμε αν η τιμή της μεταβλητής **filterCert** είναι ίση και ίδιου τύπου με την τιμή "ΝΑΙ", δηλαδή αν έχει επιλεχθεί το "ΝΑΙ" στην αναπτυσσόμενη λίστα "Πιστοποιημένοι". Αν η τιμή της είναι ίση με το "ΝΑΙ", τότε φιλτράρουμε πάλι με τη μέθοδο **filter( )** τον πίνακα **computedTrainers**. Μέσα στην callback συνάρτηση του **filter( )** ελέγχουμε αν η τιμή της ιδιότητας **certification.code** (Κωδικός ΕΟΠΠΕΠ) του κάθε **trainer** είναι διαφορετική του κενού. Επομένως στον πίνακα **computedTrainers** θα περιέχονται μόνο οι εκπαιδευτές που ικανοποιούν τη συνθήκη, με άλλα λόγια οι πιστοποιημένοι εκπαιδευτές.

Στις γραμμές 68 έως 72 ελέγχουμε αν η τιμή της μεταβλητής **filterCert** είναι ίση με το "ΟΧΙ", δηλαδή την περίπτωση που έχει επιλεχθεί το "ΟΧΙ". Αν ισχύει, τότε πραγματοποιούμε παρόμοια βήματα όπως στην προηγούμενη περίπτωση του "ΝΑΙ". Η διαφορά είναι ότι στην callback συνάρτηση αυτή τη φορά ελέγχουμε αν η τιμή της ιδιότητας **certification.code** είναι ίση και ίδιου τύπου με την κενή τιμή. Αυτό έχει ως αποτέλεσμα να περιέχονται στον πίνακα **computedTrainers** μόνο οι μη πιστοποιημένοι εκπαιδευτές.

Στις γραμμές 74 έως 94 πραγματοποιούμε διάφορους ελέγχους **if … else if … else** στην μεταβλητή **sortTerm** που αποθηκεύουμε την επιλογή ταξινόμησης που επιλέγεται στην αναπτυσσόμενη λίστα "Ταξινόμηση κατά".

Στις γραμμές 74 έως 77 ελέγχουμε αν η τιμή της μεταβλητής **sortTerm** ισούται με την τιμή "Κωδικός (Αύξουσα)". Αν ισούται, τότε ταξινομούμε τον πίνακα **computedTrainers** με τη μέθοδο **sort( )** και της προσθέτουμε μία callback συνάρτηση ως παράμετρο. Στην callback συνάρτηση ενσωματώνουμε δύο ορίσματα και εντός της συγκρίνουμε την ιδιότητα **code** των δύο ορισμάτων με τη μέθοδο **localeCompare( )**. Με την ολοκλήρωση του **sort( )**, οι εκπαιδευτές στον πίνακα **computedTrainers** θα είναι ταξινομημένοι κατά Κωδικό αύξουσα.

Στους ελέγχους **else if** των γραμμών 78 έως 89 ελέγχουμε αν η τιμή της μεταβλητής **sortTerm** ισούται με άλλες τιμές, όπως "Κωδικός (Φθίνουσα)", "Επώνυμο (Αύξουσα)", "Επώνυμο (Φθίνουσα)". Αν ισούται με κάποια, τότε ταξινομούμε τον πίνακα πάλι με τη μέθοδο **sort( )**, ωστόσο εντός της κάθε callback χρησιμοποιούμε διαφορετικά τη μέθοδο **localeCompare( )**.

Στις γραμμές 90 έως 94 ορίζουμε το **else** τμήμα, το οποίο θα εκτελεστεί αν το **sortTerm** δεν ισούται με καμία από τις προηγούμενες τιμές. Μέσα στο **else** χρησιμοποιούμε πάλι τη μέθοδο **sort( )** στον πίνακα **computedTrainers** και εντός της callback συγκρίνουμε την ιδιότητα **id** των δύο ορισμάτων. Αυτό έχει ως αποτέλεσμα να εφαρμόζεται η προεπιλεγμένη ταξινόμηση, δηλαδή να ταξινομούνται οι εκπαιδευτές στον πίνακα σε αύξουσα σειρά με βάση το **id**.

Στη γραμμή 96 καλούμε τη συνάρτηση **setTotalTrainers( )** με όρισμα τον πίνακα **computedTrainers** με την ιδιότητα **length** για να αποθηκεύσουμε στη μεταβλητή κατάστασης **totalTrainers** τον συνολικό αριθμό των φιλτραρισμένων εκπαιδευτών.

Στις γραμμές 99 έως 102 επιστρέφουμε τους εκπαιδευτές που θα εμφανίζονται στην τρέχουσα σελίδα και αυτό το υλοποιούμε αξιοποιώντας τη μέθοδο **slice( )** στον πίνακα **computedTrainers**. Η μέθοδος **slice( )** "κόβει" ένα τμήμα εκπαιδευτών από τον πίνακα **computedTrainers** με βάση τους δείκτες που προσθέτουμε ως ορίσματα. Το πρώτο όρισμα **(currentPage - 1) * trainersPerPage** αντιπροσωπεύει το αρχικό index (δείκτη), ενώ το δεύτερο **(currentPage - 1) * trainersPerPage + trainersPerPage** το τελικό index.

Στη γραμμή 103 ενσωματώνουμε τη δεύτερη παράμετρο του **useMemo( )** που είναι ο πίνακας με τις εξαρτήσεις. Στον πίνακα περνάμε ως εξαρτήσεις τις αντίστοιχες μεταβλητές κατάστασης. Αν αλλάξει η τιμή σε κάποια, τότε εκτελείται ξανά το **useMemo( )**.

Στις γραμμές 105 έως 143 ορίζουμε συναρτήσεις ως event handlers. Πιο συγκεκριμένα, στις γραμμές 105 έως 110 ορίζουμε τον event handler **const prevPaginate = ( ) => { };**, τον οποίο τον συνδέουμε στο **onClick** event του κουμπιού (button) "Προηγούμενη". Έτσι, αυτός ο event handler εκτελείται όταν γίνεται κλικ στο κουμπί "Προηγούμενη" της σελιδοποίησης. Εντός του δημιουργούμε μια δήλωση **if**, στην οποία προσθέτουμε ως συνθήκη το **currentPage > 1** για να ελέγξουμε αν η τρέχουσα σελίδα είναι μεγαλύτερη του 1. Αν ισχύει η συνθήκη, τότε καλούμε τη συνάρτηση ενημέρωσης **setCurrentPage( )** με όρισμα το **currentPage – 1** και αυτό έχει ως αποτέλεσμα να γίνεται πλοήγηση στην προηγούμενη σελίδα. Στη συνέχεια, καλούμε το **window.scrollTo(0, 0)** για την μετακίνηση της κάθετης μπάρας κύλισης στην κορυφή της σελίδας.

Στις γραμμές 112 έως 117 ορίζουμε τον event handler **const nextPaginate = ( ) => { };** και τον συνδέουμε στο **onClick** του κουμπιού "Επόμενη" της σελιδοποίησης. Ο συγκεκριμένος event handler εκτελείται όταν πατιέται το κουμπί "Επόμενη". Μέσα σε αυτόν ελέγχουμε αν η τιμή της μεταβλητής **currentPage** είναι μικρότερη από το αποτέλεσμα της διαίρεσης **totalTrainers / trainersPerPage**, δηλαδή αν η τρέχουσα σελίδα είναι μικρότερη από την τελευταία. Αν είναι μικρότερη, τότε καλούμε το **setCurrentPage( )** με όρισμα **currentPage + 1** για να πραγματοποιηθεί μετάβαση στην επόμενη σελίδα. Τέλος καλούμε το **window.scrollTo(0, 0)**.

Στις γραμμές 119 έως 124 δημιουργούμε τον event handler **const paginate = (pageNumber) => { }**, τον οποίο σε αντίθεση με τους προηγούμενους event handlers τον συνδέουμε στο **onClick** σε κάθε αριθμό (κουμπί) της σελιδοποίησης. Επομένως αυτός εκτελείται όταν γίνει κλικ σε κάποιον αριθμό στη σελιδοποίηση. Στον event handler περνάμε ως όρισμα το **pageNumber** που αντιπροσωπεύει τον αριθμό σελίδας που πατήθηκε και εντός του καλούμε το **setCurrentPage(pageNumber)** για να γίνει πλοήγηση στη συγκεκριμένη σελίδα. Έπειτα, ελέγχουμε αν η τρέχουσα σελίδα είναι διαφορετική από τον αριθμό σελίδας που πατήθηκε. Αν είναι διαφορετική, τότε καλούμε το **window.scrollTo(0, 0)**.

Στις γραμμές 126 έως 130 κατασκευάζουμε τον event handler **const showTrainersPerPage = (e) => { }** και του ενσωματώνουμε ως όρισμα το αντικείμενο **event**. Τον συγκεκριμένο event handler τον συνδέουμε στο **onChange** event της αναπτυσσόμενης λίστας (select) "Εκπαιδευτές / σελίδα" και εκτελείται όταν αλλάζει η επιλεγμένη τιμή. Αρχικά, μέσα του καλούμε το **window.scrollTo(0, 0)** και στη συνέχεια τη συνάρτηση **setTrainersPerPage( )** για να ενημερώσουμε τη μεταβλητή **trainersPerPage** με την τιμή που επιλέχθηκε. Στη συνάρτηση **setTrainersPerPage( )** ενσωματώνουμε το **e.target.value** ως όρισμα για να αποκτήσουμε την τρέχουσα τιμή. Επιπλέον χρησιμοποιούμε τη μέθοδο **parseInt( )** στο **e.target.value**, ώστε να μετατρέψουμε την τρέχουσα τιμή σε ακέραιο αριθμό. Έτσι, με την κλήση του **setTrainersPerPage(parseInt(e.target.value))** θα αλλάξει το πλήθος των εκπαιδευτών που θα εμφανίζονται ανά σελίδα. Τέλος, καλούμε το **setCurrentPage(1)** για να γίνει μετάβαση στην πρώτη σελίδα.

Στις γραμμές 132 έως 135 ορίζουμε τον event handler **const municipalityChangeHandler = (newInputValue) => { }** και τον συνδέουμε στο **onChange** του Autocomplete component "Δήμος". Ο συγκεκριμένος event handler εκτελείται όταν επιλέγεται νέα τιμή ή όταν εκαθαρίζεται στην αναπτυσσόμενη λίστα "Δήμος" και εντός του καλούμε το

setFilterMunicipality(newInputValue) για να αποθηκεύσουμε την τιμή στη μεταβλητή **filterMunicipality**. Ακόμα καλούμε το **setCurrentPage(1)**.

Στις γραμμές 137 έως 143 δημιουργούμε τον event handler **const clearFilter = ( ) => { }**, τον οποίο τον συνδέουμε στο κουμπί "Καθαρισμός Φίλτρων" και επομένως εκτελείται όταν γίνεται κλικ στο συγκεκριμένο κουμπί. Εντός του καλούμε τις συναρτήσεις **setSearchTerm("")**, **setFilterCert("")**, **setFilterMunicipality(null)**, **setSortTerm("")** και **setCurrentPage(1)** με αποτέλεσμα να εκκαθαρίζουμε τα φίλτρα και να γίνεται μετάβαση στην πρώτη σελίδα.

Στις γραμμές 145 έως 283 το **return** κομμάτι είναι αυτό που εμφανίζεται στην οθόνη, δηλαδή η λίστα με τους εισαχθέντες εκπαιδευτές, τα φίλτρα και η σελιδοποίηση της. Μέσα στο **return** στη γραμμή 148 χρησιμοποιούμε conditional rendering και το επιτυγχάνουμε με τη χρήση του λογικού τελεστή AND **(&&**). Στην έκφραση **{error && <ErrorPage error={error}/>}** η μεταβλητή κατάστασης **error** είναι η συνθήκη και ελέγχουμε αν υπάρχει HTTP σφάλμα. Αν είναι αληθής, τότε εκτελείται το component "ErrorPage" και εμφανίζεται η σελίδα σφαλμάτων. Ακόμα, στο component "ErrorPage" προσθέτουμε το prop **error** και του ενσωματώνουμε τη μεταβλητή **error** για να μεταβιβάσουμε το HTTP σφάλμα.

Στη γραμμή 150 χρησιμοποιούμε πάλι conditional rendering και ελέγχουμε αν η συνθήκη **!error** είναι αληθής, δηλαδή αν δεν υπάρχει HTTP σφάλμα. Αν δεν υπάρχει, τότε εμφανίζεται το περιεχόμενο της αρχικής σελίδας.

Στις γραμμές 152 έως 200 ορίζουμε ένα στοιχείο ** ** και του ενσωματώνουμε την προκαθορισμένη κλάση **"row"** του Bootstrap για να τοποθετήσουμε τα στοιχεία που περιλαμβάνει στην ίδια γραμμή. Εντός του ορίζουμε δύο ένθετα ** **, όπου στο πρώτο δημιουργούμε το πεδίο "Αναζήτηση" και στο δεύτερο την αναπτυσσόμενη λίστα "Δήμος". Ειδικότερα, μέσα στο πρώτο ** ** δημιουργούμε την ετικέτα "Αναζήτηση" χρησιμοποιώντας το στοιχείο **<label>**. Στο στοιχείο **<label>** προσθέτουμε το χαρακτηριστικό **htmlFor** με την τιμή **"search"** για να το συνδέσουμε με το **<input>** που έχει **id="search"**. Στη συνέχεια κατασκευάζουμε με το στοιχείο **<input>** το πεδίο "Αναζήτηση" και του προσθέτουμε τα χαρακτηριστικά **className, id, name, type, value, placeholder, autoComplete** και **onChange**. Στο **className** ενσωματώνουμε δύο κλάσεις του Digigov CSS για να ακολουθήσουμε το στυλ του GOV.GR. Το **id** καθορίζει το μοναδικό αναγνωριστικό του πεδίου και του τοποθετούμε τη μοναδική τιμή **"search"**. Στο **type** προσθέτουμε το **"text"** επειδή θέλουμε το **<input>** να είναι πεδίο εισαγωγής κειμένου. Στο **value** ενσωματώνουμε τη μεταβλητή κατάστασης **searchTerm** για να μετατρέψουμε το στοιχείο σε ελεγχόμενο

(controlled). Στο χαρακτηριστικό **placeholder** εισάγουμε το **"Κωδικός, Κωδικός ΕΟΠΠΕΠ, Κωδικός Μητρώου, Όνομα, Επώνυμο, ΣΤΕΠ"** για να υποδείξουμε στους χρήστες ότι μπορούν να αναζητήσουν εκπαιδευτές με βάση τα συγκεκριμένα στοιχεία. Το **autoComplete** το θέτουμε σε **"off"** για να απενεργοποιήσουμε την αυτόματη συμπλήρωση. Στο **onChange** ενσωματώνουμε μία callback συνάρτηση ως event handler, η οποία εκτελείται όταν αλλάζει η τιμή στο πεδίο. Στην callback περνάμε ως όρισμα το **event** και εντός της καλούμε τη συνάρτηση **setSearchTerm( )** με όρισμα το **e.target.value** για να αποθηκεύσουμε την εισαγόμενη τιμή στη μεταβλητή **searchTerm**. Επιπλέον καλούμε το **setCurrentPage(1)**.

Εντός του δεύτερου ** ** χρησιμοποιούμε πάλι το **<label>** για να δημιουργήσουμε την ετικέτα "Δήμος" και του προσθέτουμε το **htmlFor="municipality"** για να το συνδέσουμε με το στοιχείο που έχει **id="municipality"**. Έπειτα, δημιουργούμε την αναπτυσσόμενη λίστα "Δήμος" χρησιμοποιώντας τα components **<Autocomplete>**, **<Box>** και **<StyledTextField>** της βιβλιοθήκης Material UI. Στο **<Autocomplete>** χρησιμοποιούμε τα χαρακτηριστικά **id**, **value** και **onChange** όπως στο προηγούμενο πεδίο, ωστόσο ενσωματώνουμε σε αυτά διαφορετικές τιμές. Ακόμα προσθέτουμε τα χαρακτηριστικά **options**, **autoHighlight**, **getOptionLabel**, **renderOption** και **renderInput**. Στο **options** ενσωματώνουμε τη μεταβλητή **dimoi** για να φορτώσουμε τους δήμους ως επιλογές. Η μεταβλητή **dimoi** είναι ένας πίνακας (array) που περιέχει αντικείμενα (objects), όπου το κάθε αντικείμενο είναι ένας δήμος. Το **autoHighlight** επισημαίνει αυτόματα την πρώτη επιλογή της αναπτυσσόμενης λίστας. Στο **getOptionLabel** ορίζουμε τον τρόπο που θα εμφανίζεται η επιλεγμένη τιμή στο πεδίο, ενώ στο **renderOption** κάνουμε χρήση του component **<Box>** και προσαρμόζουμε την εμφάνιση της κάθε επιλογής. Στη συνέχεια στο **renderInput** προσαρμόζουμε την εμφάνιση του πεδίου. Ειδικότερα ενσωματώνουμε μια callback συνάρτηση, στην οποία περνάμε ως παράμετρο το αντικείμενο **params** και εντός αξιοποιούμε το styled component **<StyledTextField>**. Στο **<StyledTextField>** χρησιμοποιούμε το **{...params}**, όπου η σύνταξη **…** ονομάζεται spread operator και μας επιτρέπει να αντιγράψουμε στο component τις ιδιότητες του **params**. Επιπλέον προσθέτουμε τα χαρακτηριστικά **inputProps**, **sx** και **variant** για περαιτέρω εφαρμογή ιδιοτήτων και στυλ.

Στις γραμμές 203 έως 225 χρησιμοποιούμε το ** … ** όπως στις γραμμές 152 έως 200. Αυτή τη φορά εντός του ορίζουμε τρία ** **, στα οποία δημιουργούμε την αναπτυσσόμενη λίστα "Πιστοποιημένοι", την αναπτυσσόμενη λίστα "Ταξινόμηση κατά" και το κουμπί "Καθαρισμός Φίλτρων". Πιο συγκεκριμένα, μέσα στο πρώτο ** ** κατασκευάζουμε την ετικέτα "Πιστοποιημένοι" με το **<label>** και έπειτα την αναπτυσσόμενη

λίστα "Πιστοποιημένοι" με το στοιχείο **<select>**. Στο **<select>** προσθέτουμε τα χαρακτηριστικά **className**, **id**, **name**, **value** και **onChange** όπως στο input "Αναζήτηση", ωστόσο τους εκχωρούμε μοναδικές τιμές. Μέσα στο **<select>** ορίζουμε τρεις επιλογές χρησιμοποιώντας το στοιχείο **<option>**, όπου στο πρώτο χρησιμοποιούμε το χαρακτηριστικό **disabled**, ώστε να είναι απενεργοποιημένη η επιλογή του και να λειτουργεί ως placeholder. Τα άλλα δύο **<option>** αντιπροσωπεύουν τις επιλογές "ΝΑΙ" και "ΟΧΙ" στην αναπτυσσόμενη λίστα και προσθέτουμε σε αυτά τα χαρακτηριστικά **value="NAI"** και **value="OXI"** αντίστοιχα.

Εντός του δεύτερου ** ** χρησιμοποιούμε τα ίδια στοιχεία και χαρακτηριστικά με διαφορετικές τιμές για να δημιουργήσουμε την ετικέτα και την αναπτυσσόμενη λίστα "Ταξινόμηση κατά".

Στο εσωτερικό του τρίτου ** ** κατασκευάζουμε με το **<button>** το κουμπί "Καθαρισμός Φίλτρων" και προσθέτουμε τα χαρακτηριστικά **type**, **className** και το event **onClick**. Στο **type** εκχωρούμε την τιμή **"button"** για να καθορίσουμε τον τύπο του κουμπιού, ενώ στο **onClick** ενσωματώνουμε τον event handler **clearFilter**.

Στις γραμμές 226 έως 254 εμφανίζουμε τη λίστα με τους εισαχθέντες εκπαιδευτές αξιοποιώντας τη JavaScript μέθοδο **map( )** στη μεταβλητή **trainersData**. Στη **map( )** ενσωματώνουμε μία callback συνάρτηση ως παράμετρο, στην οποία τοποθετούμε ως όρισμα το **trainer** και εντός της επιστρέφουμε με το **return( )** μία κάρτα (card) του Bootstrap για τον κάθε εκπαιδευτή. Να επισημανθεί ότι η συνάρτηση εκτελείται για τον κάθε εκπαιδευτή που περιέχεται στον πίνακα **trainersData**. Μέσα στο **return( )** ορίζουμε ένα στοιχείο ** ** ως γονέα και του προσθέτουμε τα χαρακτηριστικά **key={trainer.id}** και **className="card margin-bottom"**. Το **key={trainer.id}** χρειάζεται για να γνωρίζει η React σε ποιον εκπαιδευτή του πίνακα αντιστοιχεί το κάθε card και να γίνονται σωστές ενημερώσεις στο DOM.

Στις γραμμές 229 έως 235 χρησιμοποιούμε το στοιχείο **<h5 className="card-header">** για να μορφοποιήσουμε το πάνω μέρος της κάρτας. Στη συνέχεια, μέσα του χρησιμοποιούμε το ** ** για να εμφανίζονται τα στοιχεία που περιέχει στην ίδια γραμμή. Εντός του ορίζουμε τρία **** και τους τοποθετούμε τις εκφράσεις **{`${trainer.code}`}**, **{`${trainer.name.first}`}** και **{`${trainer.name.last}`}** αντίστοιχα για να εμφανίσουμε δυναμικά τον Κωδικό, το Όνομα και το Επώνυμο.

Στις γραμμές 236 έως 251 ορίζουμε το ** ** και στο εσωτερικό του κάνουμε χρήση του ** ** για να προσαρμόσουμε το κάτω μέρος της κάρτας. Έπειτα, αξιοποιούμε το ** ** όπως στο πάνω μέρος της κάρτας και εντός του ορίζουμε δύο **** και ένα ** **.

Στο πρώτο **** προσθέτουμε την έκφραση **{`Πιστοποιημένος: ${trainer.certification.code !== "" ? "ΝΑΙ" : "ΟΧΙ"}`}**, όπου κάνουμε χρήση της σύνταξης template literals, ώστε να χρησιμοποιήσουμε τον τριαδικό τελεστή **(? :)** και να επιτύχουμε conditional rendering. Στην έκφραση ελέγχουμε αν η ιδιότητα **certification.code** του **trainer** είναι διαφορετική του κενού, δηλαδή αν ο εκπαιδευτής διαθέτει πιστοποίηση. Αν είναι αληθής, τότε προβάλλεται το "Πιστοποιημένος: ΝΑΙ", αλλιώς προβάλλεται το "Πιστοποιημένος: ΟΧΙ".

Στο δεύτερο **** τοποθετούμε την έκφραση **{`ΣΤΕΠ: ${trainer.certification.step}`}** για να εμφανίσουμε δυναμικά τους Κωδικούς ΣΤΕΠ.

Τέλος, στο εσωτερικό του ** ** κατασκευάζουμε το κουμπί "Προβολή" με το **<button type="button" className="btn btn-primary btn-sm">** και το περικλείουμε στο **<Link>** component του React Router. Στο **<Link>** ενσωματώνουμε το χαρακτηριστικό **to** με την τιμή **{`trainers/${trainer.id}`}**, στην οποία τιμή χρησιμοποιούμε template literals και εισάγουμε δυναμικά την τιμή του **${trainer.id}**, ώστε με το πάτημα του κουμπιού να γίνεται μετάβαση στη σελίδα της προβολής των στοιχείων του αντίστοιχου εκπαιδευτή.

Στις γραμμές 256 έως 278 αξιοποιούμε το σημασιολογικό στοιχείο **<nav>** για να καθορίσουμε ότι αυτή είναι η σελιδοποίηση της λίστας. Μέσα του χρησιμοποιούμε το **<ul>** στοιχείο ως γονέα και εντός του ορίζουμε **<li>** στοιχεία για να δημιουργήσουμε τη σελιδοποίηση. Ειδικότερα, στο **<li>** της γραμμής 258 κατασκευάζουμε το κουμπί "Προηγούμενη" με το **<button>** και του προσθέτουμε το event **onClick**. Στο **onClick** ενσωματώνουμε μια callback συνάρτηση, όπου εντός της καλούμε τον event handler **prevPaginate( )**. Στις γραμμές 259 έως 265 προβάλλουμε τους αριθμούς της σελιδοποίησης χρησιμοποιώντας τη μέθοδο **map( )** στον πίνακα **pageNumbers**. Στη μέθοδο ενσωματώνουμε μία callback συνάρτηση, η οποία εκτελείται για τον κάθε αριθμό της σελίδας που υπάρχει στον πίνακα **pageNumbers**. Στη συνάρτηση τοποθετούμε το **number** ως όρισμα και εντός της ορίζουμε ένα στοιχείο **<li>**, στο οποίο προσθέτουμε το **key={number}** και το **className={`page-item ${currentPage === number ? 'active' : ''}`}**. Στο **className** χρησιμοποιούμε τον τριαδικό τελεστή, όπου ελέγχουμε αν η τρέχουσα σελίδα είναι ίση και ίδιου τύπου με το **number**. Αν είναι, τότε

εφαρμόζεται δυναμικά η κλάση **'active'** και επισημαίνεται ο αριθμός ως επιλεγμένος. Έπειτα, στο εσωτερικό του **<li>** κατασκευάζουμε τον αριθμό (κουμπί) με το **<button>** και του τοποθετούμε το event **onClick**. Στο **onClick** εισάγουμε μία callback, όπου εντός της καλούμε τον event handler **paginate( )** με όρισμα το **number**. Μέσα στο **<button>** περικλείουμε την έκφραση **{number}** για να εμφανίζουμε δυναμικά τον αντίστοιχο αριθμό. Στο τελευταίο **<li>** της γραμμής 266 δημιουργούμε το κουμπί "Επόμενη", όπως το κουμπί "Προηγούμενη". Η διαφορά είναι ότι στη συνάρτηση του **onClick** καλούμε τον event handler **nextPaginate( )**.

Τέλος, στις γραμμές 268 έως 277 υλοποιούμε την αναπτυσσόμενη λίστα "Εκπαιδευτές / σελίδα" με παρόμοιο τρόπο, όπως τις αναπτυσσόμενες λίστες "Πιστοποιημένοι" και "Ταξινόμηση κατά". Οι διαφορές είναι ότι εκχωρούμε μοναδικές τιμές στα χαρακτηριστικά και ότι ορίζουμε διαφορετικές επιλογές.

```
1. import React, { useState, useEffect, useMemo } from "react";
2. import { Link } from "react-router-dom";
3. import axios from "axios";
4. import ErrorPage from "./ErrorPage";
5. import Box from '@mui/material/Box';
6. import Autocomplete from '@mui/material/Autocomplete';
7. import StyledTextField from "../styled-textfield/StyledTextField";
8. import "./Read.css";
9. import dimoi from "../json/dimoi";
10.
11.const Read = () => {
12.   const [trainers, setTrainers] = useState([]);
13.   const [error, setError] = useState(null);
14.   const [searchTerm, setSearchTerm] = useState("");
15.   const [filterCert, setFilterCert] = useState("");
16.   const [filterMunicipality, setFilterMunicipality] = useState(null);
17.   const [sortTerm, setSortTerm] = useState("");
18.   const [currentPage, setCurrentPage] = useState(1);
19.   const [totalTrainers, setTotalTrainers] = useState(0);
20.   const [trainersPerPage, setTrainersPerPage] = useState(5);
21.
22.   useEffect(() => {
23.     axios
24.       .get(`https://61e53aed595afe00176e5423.mockapi.io/trainers`)
25.       .then((response) => {
26.         setTrainers(response.data);
27.         setError(null);
28.       })
29.       .catch((error) => {
30.         setError(error);
31.       });
```

```
32.   }, []);
33.
34.   const pageNumbers = [];
35.
36.   for (let i = 1; i <= Math.ceil(totalTrainers / trainersPerPage); i++) {
37.     pageNumbers.push(i);
38.   }
39.
40.   const trainersData = useMemo(() => {
41.     let computedTrainers = trainers;
42.
43.     if (searchTerm) {
44.       computedTrainers = computedTrainers.filter(
45.         (trainer) =>
46.           trainer.code.toLowerCase().includes(searchTerm.toLowerCase()) ||
47.           trainer.certification.code.toLowerCase().includes(searchTerm.toLo
    werCase()) ||
48.           trainer.registries.some((registry) =>
    registry.code.toLowerCase().includes(searchTerm.toLowerCase())) ||
49.           trainer.certification.step.some((step)
    =>step.toLowerCase().includes(searchTerm.toLowerCase())) ||
50.           `${trainer.name.first}
    ${trainer.name.last}`.toLowerCase().includes(searchTerm.toLowerCase()) ||
51.           `${trainer.name.last}
    ${trainer.name.first}`.toLowerCase().includes(searchTerm.toLowerCase())
52.       );
53.     }
54.
55.     if (filterMunicipality) {
56.       computedTrainers = computedTrainers.filter(
57.         (trainer) =>
58.           JSON.stringify(filterMunicipality) ===
    JSON.stringify(trainer.contact.address.municipality)
59.       );
60.     }
61.
62.     if (filterCert === "NAI") {
63.       computedTrainers = computedTrainers.filter(
64.         (trainer) => filterCert === "NAI" && trainer.certification.code !==
    ""
65.       );
66.     }
67.
68.     if (filterCert === "OXI") {
69.       computedTrainers = computedTrainers.filter(
70.         (trainer) => filterCert === "OXI" && trainer.certification.code ===
    ""
71.       );
```

```
72.     }
73.
74.     if (sortTerm === "Κωδικός (Αύξουσα)") {
75.       computedTrainers = computedTrainers.sort(
76.         (a, b) => a.code.localeCompare(b.code)
77.        );
78.     } else if (sortTerm === "Κωδικός (Φθίνουσα)") {
79.       computedTrainers = computedTrainers.sort(
80.         (a, b) => b.code.localeCompare(a.code)
81.        );
82.     } else if (sortTerm === "Επώνυμο (Αύξουσα)") {
83.       computedTrainers = computedTrainers.sort(
84.         (a, b) => a.name.last.localeCompare(b.name.last)
85.        );
86.     } else if (sortTerm === "Επώνυμο (Φθίνουσα)") {
87.       computedTrainers = computedTrainers.sort(
88.         (a, b) => b.name.last.localeCompare(a.name.last)
89.        );
90.     } else {
91.       computedTrainers = computedTrainers.sort(
92.         (a, b) => a.id - b.id
93.        );
94.     }
95.
96.     setTotalTrainers(computedTrainers.length);
97.
98.     //Current Page slice
99.     return computedTrainers.slice(
100.         (currentPage - 1) * trainersPerPage,
101.         (currentPage - 1) * trainersPerPage + trainersPerPage
102.       );
103.     }, [trainers, currentPage, searchTerm, filterMunicipality,
    filterCert, sortTerm, trainersPerPage]);
104.
105.     const prevPaginate = () => {
106.       if (currentPage > 1) {
107.         setCurrentPage(currentPage - 1);
108.         window.scrollTo(0, 0);
109.       }
110.     };
111.
112.     const nextPaginate = () => {
113.       if (currentPage < (totalTrainers / trainersPerPage)) {
114.         setCurrentPage(currentPage + 1);
115.         window.scrollTo(0, 0);
116.       }
117.     };
118.
```

```
119.     const paginate = (pageNumber) => {
120.       setCurrentPage(pageNumber);
121.       if (currentPage !== pageNumber) {
122.         window.scrollTo(0, 0);
123.       }
124.     }
125.
126.     const showTrainersPerPage = (e) => {
127.       window.scrollTo(0, 0);
128.       setTrainersPerPage(parseInt(e.target.value));
129.       setCurrentPage(1);
130.     }
131.
132.     const municipalityChangeHandler = (newInputValue) => {
133.       setFilterMunicipality(newInputValue);
134.       setCurrentPage(1);
135.     }
136.
137.     const clearFilter = () => {
138.       setSearchTerm("");
139.       setFilterCert("");
140.       setFilterMunicipality(null);
141.       setSortTerm("");
142.       setCurrentPage(1);
143.     }
144.
145.     return (
146.       <div className="space-top">
147.
148.         {error && <ErrorPage error={error}/>}
149.
150.         {!error && <><h3 className="govgr-heading-s">Φίλτρα</h3>
151.
152.         <div className="row">
153.           <div className="govgr-form-group col-sm">
154.             <label className="govgr-label govgr-!-font-weight-bold"
    htmlFor="search">Αναζήτηση</label>
155.
156.             <input
157.               className="govgr-input govgr-input govgr-!-width-full"
158.               id="search"
159.               name="search"
160.               type="text"
161.               value={searchTerm}
162.               placeholder="Κωδικός, Κωδικός ΕΟΠΠΕΠ, Κωδικός Μητρώου,
    Όνομα, Επώνυμο, ΣΤΕΠ"
163.               autoComplete="off"
164.               onChange={(e) => {
```

```
165.                 setSearchTerm(e.target.value);
166.                 setCurrentPage(1);
167.               }}
168.             />
169.           </div>
170.
171.           <div className="govgr-form-group col-sm">
172.             <label className="govgr-label govgr-!-font-weight-bold"
    htmlFor="municipality">Δήμος</label>
173.             <Autocomplete
174.               id="municipality"
175.               value={filterMunicipality}
176.               options={dimoi}
177.               autoHighlight
178.               getOptionLabel={(option) => `ΔΗΜΟΣ ${option.name}`}
179.               renderOption={(props, option) => (
180.                 <Box component="li" {...props}>
181.                   ΔΗΜΟΣ {option.name}
182.                 </Box>
183.               )}
184.               renderInput={(params) => (
185.                 <StyledTextField
186.                   {...params}
187.                   inputProps={{
188.                     ...params.inputProps,
189.                     autoComplete: "off",
190.                     placeholder: "Επιλέξτε Δήμο", // disable autocomplete
    and autofill,
191.                     style: {padding: 0,},
192.                   }}
193.                   sx={{border: "2px solid black"}}
194.                   variant="outlined"
195.                 />
196.               )}
197.               onChange={municipalityChangeHandler}
198.             />
199.           </div>
200.         </div>
201.
202.
203.         <div className="row">
204.           <div className="govgr-form-group col-sm-5">
205.             <label className="govgr-label govgr-!-font-weight-bold"
    htmlFor="cert-list">Πιστοποιημένοι</label>
206.             <select className="govgr-select" id="cert-list" name="cert-
    list" value={filterCert}
    onChange={(e)=>{setFilterCert(e.target.value);setCurrentPage(1);}}>
207.               <option value="" disabled>--Επιλογή--</option>
```

```
208.               <option value="NAI">NAI</option>
209.               <option value="OXI">OXI</option>
210.             </select>
211.           </div>
212.           <div className="govgr-form-group col-sm-5">
213.             <label className="govgr-label govgr-!-font-weight-bold"
   htmlFor="sort-list">Ταξινόμηση κατά</label>
214.             <select className="govgr-select" id="sort-list" name="sort-
   list" value={sortTerm}
   onChange={(e)=>{setSortTerm(e.target.value);setCurrentPage(1);}}>
215.                 <option value="" disabled>--Επιλογή--</option>
216.                 <option value="Κωδικός (Αύξουσα)">Κωδικός
   (Αύξουσα)</option>
217.                 <option value="Κωδικός (Φθίνουσα)">Κωδικός
   (Φθίνουσα)</option>
218.                 <option value="Επώνυμο (Αύξουσα)">Επώνυμο
   (Αύξουσα)</option>
219.                 <option value="Επώνυμο (Φθίνουσα)">Επώνυμο
   (Φθίνουσα)</option>
220.             </select>
221.           </div>
222.           <div className="govgr-form-group div-clear col-sm-2">
223.             <button type="button" className="btn btn-danger btn-sm"
   onClick={clearFilter}>Καθαρισμός Φίλτρων</button>
224.           </div>
225.         </div>
226.           {trainersData.map((trainer) => {
227.             return (
228.               <div key={trainer.id} className="card margin-bottom">
229.                 <h5 className="card-header">
230.                   <div className="row">
231.                     <span className="col-sm">{`${trainer.code}`}</span>
232.                     <span className="col-
   sm">{`${trainer.name.first}`}</span>
233.                     <span className="col-
   sm">{`${trainer.name.last}`}</span>
234.                   </div>
235.                 </h5>
236.                 <div className="card-body">
237.                   <div className="card-text">
238.                     <div className="row">
239.                       <span className="col-sm">{`Πιστοποιημένος: ${
240.                         trainer.certification.code !== "" ? "NAI" : "OXI"
241.                       }`}</span>
242.                       <span className="col-sm">{`ΣΤΕΠ:
   ${trainer.certification.step}`}</span>
243.                       <div className="col-sm div-show">
244.                         <Link to={`trainers/${trainer.id}`}>
```

```
245.                         <button type="button" className="btn btn-
   primary btn-sm">Προβολή</button>
246.                       </Link>
247.                     </div>
248.                   </div>
249.                 </div>
250.
251.               </div>
252.             </div>
253.           );
254.         })}
255.
256.         <nav className="margin-bottom flex-row justify-content-center">
257.           <ul className="pagination justify-content-center wrap">
258.             <li className="page-item btn-page-margin"><button
   onClick={() => prevPaginate()} className="page-
   link">Προηγούμενη</button></li>
259.             {pageNumbers.map((number) => (
260.               <li key={number} className={`page-item ${currentPage ===
   number ? 'active' : ''}`}>
261.                 <button onClick={() => paginate(number)}
   className="page-link">
262.                   {number}
263.                 </button>
264.               </li>
265.             ))}
266.             <li className="page-item btn-page-margin"><button
   onClick={() => nextPaginate()} className="page-link">Επόμενη</button></li>
267.           </ul>
268.           <div className="ml-10">
269.             <label className="text-bold text-wrap"
   htmlFor="trainersPerPage">Εκπαιδευτές / σελίδα</label>
270.               <select className="form-select w-20" id="trainerPerPage"
   aria-label="Show Trainers Per Page" value={trainersPerPage}
   onChange={showTrainersPerPage}>
271.                 <option value="5">5</option>
272.                 <option value="10">10</option>
273.                 <option value="20">20</option>
274.                 <option value="50">50</option>
275.                 <option value="100">100</option>
276.               </select>
277.           </div>
278.         </nav></>}
279.
280.
281.
282.       </div>
283.     );
```

```
284.  };
285.
286.  export default Read;
```

Αρχείο-component Form.js

Σε αυτό το αρχείο υλοποιούμε τη σελίδα της εισαγωγής εκπαιδευτή, δηλαδή τη φόρμα με τα πεδία που καλείται να συμπληρώσει ο χρήστης προκειμένου να εισάγει έναν εκπαιδευτή. Αρχικά μέσα του κατασκευάζουμε το component "Form" ορίζοντας τη JavaScript arrow function **const Form = ( ) => { return ( ); }**.

Στο εσωτερικό του component στις γραμμές 15 έως 59 ορίζουμε states με το **useState( )** hook για την αποθήκευση και την ενημέρωση των δεδομένων των πεδίων και άλλα states που αντιπροσωπεύουν αν ο χρήστης έχει "αγγίξει" τα πεδία. Ακόμα ορίζουμε αναφορές (refs) με το **useRef( )** hook, τις οποίες τις συνδέουμε στα αντίστοιχα πεδία για να αποκτήσουμε πρόσβαση στους αντίστοιχους DOM κόμβους (DOM nodes) και για να εστιάζουμε (focus) στα μη έγκυρα πεδία όταν υποβάλλεται η φόρμα. Πιο συγκεκριμένα, στη γραμμή 15 καλούμε το **useState( )** που όπως έχει αναφερθεί πάλι επιστρέφει έναν πίνακα με δύο στοιχεία, τα οποία τα αποθηκεύουμε σε ξεχωριστές μεταβλητές με χρήση της σύνταξης array destructuring **const [enteredCode, setEnteredCode]**. Το **enteredCode** είναι η μεταβλητή που περιέχει την τρέχουσα τιμή της κατάστασης, ενώ το **setEnteredCode** είναι η συνάρτηση για την ενημέρωση της. Αυτό το state το αρχικοποιούμε με κενό string **('')** και το χρησιμοποιούμε για την αποθήκευση της τιμής που εισάγει ο χρήστης στο πεδίο (input) "Κωδικός".

Στη γραμμή 16 ορίζουμε το state **const [enteredCodeTouched, setEnteredCodeTouched] = useState(false);**, το οποίο αντιπροσωπεύει αν ο χρήστης έχει "αγγίξει" το πεδίο "Κωδικός" και το χρησιμοποιούμε αργότερα για την επικύρωση του πεδίου και την εμφάνιση του μηνύματος σφάλματος. Επιπλέον το αρχικοποιούμε με την τιμή **false** επειδή ο χρήστης αρχικά δεν έχει "αγγίξει" το πεδίο.

Στη γραμμή 17 ορίζουμε την αναφορά **const codeRef = useRef();**, την οποία μετά τη συνδέουμε στο πεδίο "Κωδικός" για να αποκτήσουμε πρόσβαση στον DOM κόμβο του και για να εστιάζουμε στο πεδίο όταν είναι μη έγκυρο και υποβάλλεται η φόρμα.

Στις γραμμές 19 έως 57 δημιουργούμε με παρόμοιο τρόπο και για τον ίδιο σκοπό τα υπόλοιπα states και refs για τα υπόλοιπα πεδία. Ωστόσο χρησιμοποιούμε μοναδικά ονόματα μεταβλητών και διαφορετικές αρχικοποιήσεις. Ειδικότερα, οι γραμμές 19 έως 21 είναι για το πεδίο (input) "Όνομα", ενώ οι γραμμές 23 έως 25 είναι για το πεδίο "Επώνυμο".

Οι γραμμές 27 και 28 είναι για τα πεδία "Πατρώνυμο" και "Επάγγελμα" αντίστοιχα. Οι γραμμές 30 έως 32, 34 έως 35 και 37 έως 38 είναι για τα πεδία "Οδός", "Αριθμός" και την αναπτυσσόμενη λίστα (Autocomplete) "Δήμος" αντίστοιχα. Οι γραμμές 40 έως 42 είναι για το πεδίο "Τηλέφωνο", ενώ οι γραμμές 44 έως 46 είναι για το πεδίο "Email". Οι γραμμές 48 και 49 είναι για το πεδίο "Κωδικός Πιστοποίησης". Στη γραμμή 51 το state **const [codesteps, setCodesteps] = useState(['']);** είναι για την αποθήκευση των τιμών που εισάγονται στο δυναμικό πεδίο "Κωδικοί ΣΤΕΠ" και του ενσωματώνουμε ως αρχική τιμή έναν πίνακα με ένα κενό string **([''])** επειδή το **codesteps** είναι πίνακας που θα περιέχει τους κωδικούς ΣΤΕΠ. Στη γραμμή 53 το **const [registryValues, setRegistryValues] = useState([]);** το χρησιμοποιούμε για την αποθήκευση των τιμών που εισάγονται στα πεδία των μητρώων, δηλαδή στα δυναμικά πεδία "Όνομα Μητρώου" και "ΑΜ/Κωδικός Εκπαιδευτή Στο Μητρώο". Το state αυτό το αρχικοποιούμε με έναν άδειο πίνακα **([])**, διότι το **registryValues** είναι πίνακας που θα περιλαμβάνει objects.

Στη γραμμή 55 το **const fieldRef = useRef();** το χρησιμοποιούμε για να έχουμε πρόσβαση στους DOM κόμβους των δυναμικών πεδίων "Κωδικοί ΣΤΕΠ", "Όνομα Μητρώου" και "ΑΜ/Κωδικός Εκπαιδευτή Στο Μητρώο".

Στη γραμμή 57 το **const [showCertification, setShowCertification] = useState(false);** το αξιοποιούμε για την ενεργοποίηση ή την απενεργοποίηση του switch (διακόπτη) "Πιστοποίηση ΕΟΠΠΕΠ" και για την εμφάνιση ή την απόκρυψη των πεδίων της πιστοποίησης. Στο συγκεκριμένο state περνάμε ως αρχική τιμή το **false**, ώστε το switch να είναι αρχικά απενεργοποιημένο και να αποκρύπτονται τα πεδία της πιστοποίησης.

Στη γραμμή 59 το **const [error, setError] = useState(null);** είναι για την αποθήκευση των HTTP σφαλμάτων.

Στις γραμμές 61 έως 66 αξιοποιούμε το **useEffect( )** hook και του ενσωματώνουμε δύο ορίσματα, όπου το πρώτο είναι μία callback συνάρτηση, ενώ το δεύτερο είναι ο πίνακας με τις εξαρτήσεις. Εντός της callback δημιουργούμε μία δήλωση **if** με συνθήκη το **!showCertification**, όπου ελέγχουμε αν η τιμή της μεταβλητής κατάστασης

showCertification είναι false, δηλαδή αν είναι απενεργοποιημένο το switch (διακόπτης) "Πιστοποίηση ΕΟΠΠΕΠ" και αποκρύπτονται τα πεδία της πιστοποίησης ("Κωδικός Πιστοποίησης", "Κωδικοί ΣΤΕΠ"). Αν η συνθήκη είναι αληθής, τότε καλούμε τα **setEnteredCodecert("")** και **setCodesteps([""])** για να επαναφέρουμε τις μεταβλητές **enteredCodecert** και **codesteps** στις αρχικές τους καταστάσεις. Οι λόγοι που το πραγματοποιούμε αυτό είναι για να εκκαθαρίζουμε τις τιμές που έχουν εισαχθεί στα πεδία της πιστοποίησης και για να μην τις αποθηκεύουμε όταν ο χρήστης απενεργοποιεί τον διακόπτη. Ακόμα αν ο χρήστης ενεργοποιήσει ξανά τον διακόπτη, τότε τα πεδία θα είναι κενά. Τέλος, στον πίνακα των εξαρτήσεων προσθέτουμε τη μεταβλητή **showCertification**, ώστε το **useEffect( )** να εκτελείται κατά την αρχική τοποθέτηση του component στο DOM αλλά και κάθε φορά που αλλάζει η τιμή της μεταβλητής.

Στη γραμμή 68 δημιουργούμε τη μεταβλητή **navigate** και της αποθηκεύουμε το **useNavigate( )** hook της βιβλιοθήκης React Router. Το **useNavigate( )** επιστρέφει μία συνάρτηση που μας επιτρέπει να πλοηγηθούμε.

Στις γραμμές 70 έως 99 ορίζουμε για κάθε πεδίο δύο μεταβλητές, τις οποίες τις χρησιμοποιούμε για να επικυρώνουμε σε πραγματικό χρόνο (real-time) και κατά την υποβολή. Πιο συγκεκριμένα, στη γραμμή 70 ορίζουμε τη μεταβλητή **enteredCodeIsValid**, η οποία αντιπροσωπεύει το αν είναι έγκυρη η τιμή που έχει εισαχθεί στο πεδίο "Κωδικός". Στη μεταβλητή αποθηκεύουμε τον έλεγχο **enteredCode.trim() !==** '', όπου ελέγχουμε αν η τιμή της μεταβλητής κατάστασης **enteredCode** είναι διαφορετική του κενού string. Αν το αποτέλεσμα είναι true, τότε η εισαγόμενη τιμή είναι έγκυρη. Επίσης χρησιμοποιούμε τη μέθοδο **trim( )** για να αφαιρέσουμε τυχόν περιττά κενά που μπορούν να επηρεάσουν την επικύρωση.

Στη γραμμή 71 δημιουργούμε τη μεταβλητή **codeInputIsInvalid**, η οποία αντιπροσωπεύει τη μη εγκυρότητα του πεδίου "Κωδικός" και τη χρησιμοποιούμε για να εμφανίσουμε το μήνυμα σφάλματος καθώς και για να επισημαίνουμε το πεδίο όταν είναι μη έγκυρο. Στη μεταβλητή αποθηκεύουμε μία λογική πράξη AND (**&&**) μεταξύ του **!enteredCodeIsValid** και του **enteredCodeTouched**, όπου ελέγχουμε αν η εισαγόμενη τιμή είναι μη έγκυρη και έχει "αγγιχτεί" το πεδίο. Αν το αποτέλεσμα της πράξης είναι αληθής, τότε το πεδίο είναι μη έγκυρο και εμφανίζεται το μήνυμα σφάλματος και οι ενδείξεις.

Στις γραμμές 73 έως 77 πραγματοποιούμε παρόμοιους ελέγχους επικύρωσης για τα πεδία "Όνομα" και "Επώνυμο".

Στις γραμμές 79 έως 80 επικυρώνουμε το πεδίο "Οδός". Πιο συγκεκριμένα, στη γραμμή 79 κατασκευάζουμε τη μεταβλητή **enteredStreetIsValid**, η οποία αντιπροσωπεύει το αν είναι έγκυρο το πεδίο "Οδός". Στο **enteredStreetIsValid** αποθηκεύουμε μία πράξη, στην οποία χρησιμοποιούμε τους λογικούς τελεστές AND (**&&**) και OR **(||)**. Το αποτέλεσμα της πράξης θα είναι αληθής αν τα πεδία "Οδός" και "Αριθμός" είναι κενά και τα δύο ή αν δεν είναι κενά και τα δύο ή αν μόνο το πεδίο "Αριθμός" είναι κενό. Έτσι αν είναι αληθής, τότε το πεδίο είναι έγκυρο.

Στη γραμμή 80 ορίζουμε τη μεταβλητή **streetInputIsInvalid**, η οποία αντιπροσωπεύει τη μη εγκυρότητα του πεδίου "Οδός". Στο **streetInputIsInvalid** αποθηκεύουμε τον έλεγχο **!enteredStreetIsValid**, όπου ελέγχουμε αν το πεδίο είναι μη έγκυρο. Αν το αποτέλεσμα είναι true, τότε το πεδίο είναι μη έγκυρο και προβάλλεται το αντίστοιχο μήνυμα σφάλματος και οι αντίστοιχες ενδείξεις.

Στις γραμμές 82 έως 83 επικυρώνουμε την αναπτυσσόμενη λίστα "Δήμος" με παρόμοιο τρόπο όπως το πεδίο "Οδός". Η διαφορά είναι ότι η αναπτυσσόμενη λίστα "Δήμος" εξαρτάται από το πεδίο "Οδός", ενώ το πεδίο "Οδός" εξαρτάται από το πεδίο "Αριθμός".

Στις γραμμές 85 έως 86 επικυρώνουμε το πεδίο "Τηλέφωνο". Ειδικότερα, στη γραμμή 85 δημιουργούμε τη μεταβλητή **enteredPhoneIsValid**, η οποία αντιπροσωπεύει το αν είναι έγκυρη η τιμή που έχει εισαχθεί στο πεδίο "Τηλέφωνο". Στη μεταβλητή εκχωρούμε το **enteredPhone.match(/^[0-9]{10}$/)**, όπου χρησιμοποιούμε τη μέθοδο **match( )** με παράμετρο το regular expression **/^[0-9]{10}$/** για να ελέγξουμε αν η εισαγόμενη τιμή ταιριάζει με το regular expression. Το αποτέλεσμα είναι true όταν η εισαγόμενη τιμή είναι ακριβώς 10 ψηφία.

Στη γραμμή 86 ορίζουμε τη μεταβλητή **phoneInputIsInvalid**, η οποία αντιπροσωπεύει τη μη εγκυρότητα του πεδίου "Τηλέφωνο". Στη μεταβλητή ελέγχουμε αν η εισαγόμενη τιμή είναι μη έγκυρη και έχει "αγγιχτεί" το πεδίο (**!enteredPhoneIsValid && enteredPhoneTouched**). Αν ισχύει, τότε το πεδίο δεν είναι έγκυρο και εμφανίζεται το αντίστοιχο μήνυμα σφάλματος και οι αντίστοιχες ενδείξεις.

Στις γραμμές 88 έως 90 επικυρώνουμε το πεδίο "Email". Για την ακρίβεια, στις γραμμές 88 έως 89 κατασκευάζουμε τη μεταβλητή **enteredEmailIsValid**, η οποία αντιπροσωπεύει το αν είναι έγκυρη η εισαγόμενη τιμή στο πεδίο "Email". Στο **enteredEmailIsValid** πραγματοποιούμε τον έλεγχο **enteredEmail.trim() !== '' &&**

enteredEmail.match(/^(([^<>()[\]\\.,;:\s@"]+(\.[^<>()[\]\\.,;:\s@"]+)*)|(".+"))@((\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}])|(([a-zA-Z\-0-9]+\.)+[a-zA-Z]{2,}))$/) με τον οποίο ελέγχουμε αν η τιμή που έχει εισαχθεί στο πεδίο είναι διαφορετική του κενού string και ταιριάζει με το συγκεκριμένο regular expression. Αν ικανοποιείται ο έλεγχος, τότε η εισαγόμενη τιμή είναι ένα έγκυρο email.

Στη γραμμή 90 δημιουργούμε τη μεταβλητή **emailInputIsInvalid**, η οποία αντιπροσωπεύει τη μη εγκυρότητα του πεδίου "Email". Στο **emailInputIsInvalid** εκχωρούμε την πράξη **!enteredEmailIsValid && enteredEmailTouched** με την οποία ελέγχουμε αν η τιμή που έχει εισαχθεί είναι μη έγκυρη και το πεδίο έχει "αγγιχτεί". Αν το αποτέλεσμα της πράξης είναι αληθής, τότε το πεδίο επισημαίνεται ως μη έγκυρο με τις αντίστοιχες ενδείξεις και το αντίστοιχο μήνυμα σφάλματος.

Στις γραμμές 92 έως 93 επικυρώνουμε το πεδίο "Κωδικός Πιστοποίησης". Συγκεκριμένα, στη γραμμή 92 ορίζουμε τη μεταβλητή **enteredCodecertIsValid** που υποδεικνύει αν έχει εισαχθεί έγκυρη τιμή στο πεδίο "Κωδικός Πιστοποίησης". Στη μεταβλητή αυτή ελέγχουμε αν η εισαγόμενη τιμή είναι διαφορετική του κενού string (**enteredCodecert.trim() !==** ''). Αν ισχύει, τότε η τιμή που έχει εισαχθεί είναι έγκυρη.

Στη γραμμή 93 ορίζουμε το **codecertInputIsInvalid**, το οποίο αντιπροσωπεύει τη μη εγκυρότητα του πεδίου "Κωδικός Πιστοποίησης". Στο **codecertInputIsInvalid** αποθηκεύουμε το **!enteredCodecertIsValid** με το οποίο ελέγχουμε αν η εισαγόμενη τιμή είναι ίση με το κενό string, δηλαδή αν είναι μη έγκυρη. Σε αντίθεση με τις προηγούμενες μεταβλητές μη εγκυρότητας, σε αυτή τη μεταβλητή δεν ελέγχουμε αν έχει "αγγιχτεί" το πεδίο, διότι θέλουμε το πεδίο να είναι από την αρχή μη έγκυρο και να προβάλλεται το μήνυμα σφάλματος και οι ενδείξεις όταν ενεργοποιείται ο διακόπτης "Πιστοποίηση ΕΟΠΠΕΠ".

Στις γραμμές 95 έως 96 επικυρώνουμε το δυναμικό πεδίο "Κωδικοί ΣΤΕΠ". Ειδικότερα στη γραμμή 95 δημιουργούμε τη μεταβλητή **codestepsIsValid**, η οποία αντιπροσωπεύει το αν είναι έγκυρα όλα τα πεδία "Κωδικοί ΣΤΕΠ". Στη μεταβλητή αποθηκεύουμε την πράξη **codesteps.length >= 1 && codesteps.every(codestep => codestep.trim() !== '')**, όπου στο πρώτο μέρος της πράξης (**codesteps.length >= 1**) ελέγχουμε αν ο πίνακας **codesteps** περιέχει ένα ή περισσότερα πεδία. Στο δεύτερο μέρος της πράξης (**codesteps.every(codestep => codestep.trim() !==** '') ελέγχουμε αν η κάθε εισαγόμενη τιμή στα πεδία "Κωδικοί ΣΤΕΠ" είναι διαφορετική του κενού string. Αυτό το πραγματοποιούμε χρησιμοποιώντας στον πίνακα **codesteps** τη μέθοδο **every( )**, στην οποία περνάμε ως παράμετρο μία callback συνάρτηση.

Στην callback ενσωματώνουμε ως παράμετρο το **codestep**, το οποίο αντιπροσωπεύει το κάθε στοιχείο του πίνακα και εντός της callback πραγματοποιούμε τον έλεγχο. Αν το αποτέλεσμα της πράξης είναι true, τότε τα πεδία είναι έγκυρα.

Στη γραμμή 96 κατασκευάζουμε τη μεταβλητή **codestepInputIsInvalid**, η οποία αντιπροσωπεύει τη μη εγκυρότητα του δυναμικού πεδίου "Κωδικοί ΣΤΕΠ". Στη μεταβλητή εκχωρούμε το **!codestepsIsValid**, όπου ελέγχουμε αν είναι false η τιμή της μεταβλητής **codestepsIsValid**. Ακόμα δεν ελέγχουμε αν έχουν "αγγιχτεί" τα πεδία, επειδή θέλουμε να είναι μη έγκυρα από την αρχή και να εμφανίζονται οι ενδείξεις και το μήνυμα σφάλματος όταν ενεργοποιείται ο διακόπτης "Πιστοποίηση ΕΟΠΠΕΠ".

Στις γραμμές 98 έως 99 επικυρώνουμε τα δυναμικά πεδία "Όνομα Μητρώου" και "ΑΜ/Κωδικός Εκπαιδευτή Στο Μητρώο". Για την ακρίβεια στη γραμμή 98 ορίζουμε τη μεταβλητή **registriesIsValid** και της εκχωρούμε το **registryValues.every(registry => registry.registry.name.trim() !== '' && registry.code.trim() !== '')**, όπου αξιοποιούμε πάλι τη μέθοδο **every( )** αλλά στον πίνακα **registryValues** για να ελέγξουμε αν είναι έγκυρες όλες οι εισαγόμενες τιμές στα πεδία "Όνομα Μητρώου" και "ΑΜ/Κωδικός Εκπαιδευτή Στο Μητρώο". Στη μέθοδο **every( )** ενσωματώνουμε μία callback συνάρτηση, στην οποία περνάμε ως όρισμα το αντικείμενο **registry**. Το **registry** αντιπροσωπεύει το κάθε μητρώο στον πίνακα **registryValues**. Εντός της callback ελέγχουμε αν οι ιδιότητες **registry.name** και **code** του κάθε **registry** είναι διαφορετικές του κενού string. Αν πληρείται ο έλεγχος για το κάθε **registry**, τότε όλες οι εισαγόμενες τιμές στα δυναμικά πεδία των μητρώων είναι έγκυρες.

Στη γραμμή 99 κατασκευάζουμε τη μεταβλητή **registriesInputIsInvalid**, η οποία αντιπροσωπεύει τη μη εγκυρότητα των δυναμικών πεδίων "Όνομα Μητρώου" και "ΑΜ/Κωδικός Εκπαιδευτή Στο Μητρώο". Σε αυτή τη μεταβλητή αποθηκεύουμε τον έλεγχο **!registriesIsValid** με τον οποίο ελέγχουμε αν είναι false η τιμή της μεταβλητής **registriesIsValid**. Επιπλέον, στη μεταβλητή δεν ελέγχουμε αν ο χρήστης έχει "αγγίξει" τα πεδία, διότι όταν προστίθενται θέλουμε από την αρχή να εμφανίζονται μηνύματα σφάλματος και να επισημαίνονται ως μη έγκυρα.

Να αναφερθεί πάλι ότι οι ενημερώσεις των states προκαλούν την εκτέλεση εκ νέου του component με αποτέλεσμα να ελέγχονται οι αντίστοιχες μεταβλητές και να πραγματοποιείται επικύρωση σε πραγματικό χρόνο.

Στις γραμμές 102 έως 316 ορίζουμε συναρτήσεις ως event handlers για τη διαχείριση των δεδομένων που εισάγονται στα πεδία και των αλληλεπιδράσεων. Ειδικότερα στις γραμμές 102 έως 105 ορίζουμε τον event handler **const codeChangeHandler = (event) => { }** και του ενσωματώνουμε ως όρισμα το αντικείμενο **event**. Τον συγκεκριμένο event handler τον συνδέουμε στο **onChange** event του πεδίου (input) "Κωδικός" και εκτελείται κάθε φορά που αλλάζει η εισαγόμενη τιμή στο πεδίο. Συγκεκριμένα εκτελείται σε κάθε πάτημα πλήκτρου. Εντός του event handler καλούμε τη συνάρτηση ενημέρωσης **setEnteredCodeTouched(true)** για να ενημερώσουμε τη μεταβλητή κατάστασης **enteredCodeTouched** με την τιμή **true** επειδή ο χρήστης "άγγιξε" το πεδίο. Στη συνέχεια, καλούμε τη συνάρτηση **setEnteredCode( )** με όρισμα το **event.target.value** για να αποθηκεύσουμε την εισαγόμενη τιμή στη μεταβλητή **enteredCode**. Να αναφερθεί ότι με το **event.target.value** αποκτούμε την τρέχουσα τιμή.

Στις γραμμές 107 έως 115 δημιουργούμε με παρόμοιο τρόπο τους event handlers **firstChangeHandler** και **lastChangeHandler** και τους συνδέουμε στο **onChange** των πεδίων "Όνομα" και "Επώνυμο" αντίστοιχα. Εντός τους καλούμε τις αντίστοιχες συναρτήσεις ενημέρωσης.

Στις γραμμές 117 έως 123 κατασκευάζουμε τους event handlers **fatherChangeHandler** και **occupationChangeHandler** για τα πεδία "Πατρώνυμο" και "Επάγγελμα" αντίστοιχα. Η διαφορά με τα προηγούμενα πεδία είναι ότι σε αυτά δεν ενημερώνουμε κάποιο touched state, διότι τα πεδία είναι προαιρετικά και δεν μας ενδιαφέρει αν έχουν "αγγιχτεί".

Στις γραμμές 125 έως 133 ορίζουμε τον event handler **streetChangeHandler** για το πεδίο "Οδός" και τον **numberChangeHandler** για το πεδίο "Αριθμός". Αυτούς τους event handlers τους δημιουργούμε με παρόμοιο τρόπο όπως τους event handlers **codeChangeHandler**, **firstChangeHandler** και **lastChangeHandler**.

Στις γραμμές 135 έως 137 κατασκευάζουμε τον event handler **const municipalityChangeHandler = (newInputValue) => { }** και του περνάμε ως όρισμα το **newInputValue**. Τον συγκεκριμένο event handler τον συνδέουμε στο **onChange** της αναπτυσσόμενης λίστας (Autocomplete) "Δήμος" και εκτελείται όταν επιλέγεται νέα τιμή ή εκκαθαρίζεται. Μέσα του καλούμε το **setEnteredMunicipality( )** με όρισμα το **newInputValue** για να αποθηκεύσουμε την επιλεγμένη τιμή στη μεταβλητή κατάστασης **enteredMunicipality**. Στην πραγματικότητα αποθηκεύουμε το δήμος object που αντιστοιχεί στην επιλεγμένη τιμή.

Στις γραμμές 139 έως 148 ορίζουμε τους event handlers **phoneChangeHandler** και **emailChangeHandler** για τα πεδία "Τηλέφωνο" και "Email" αντίστοιχα. Μέσα τους πραγματοποιούμε παρόμοια βήματα όπως στους event handlers **codeChangeHandler**, **firstChangeHandler**, **lastChangeHandler**, **streetChangeHandler** και **numberChangeHandler**.

Στις γραμμές 150 έως 153 δημιουργούμε τον event handler **const showCertificationHandler = (event) => { }** και του ενσωματώνουμε ως όρισμα το αντικείμενο **event**. Αυτόν τον event handler τον συνδέουμε στο **onClick** του switch "Πιστοποίηση ΕΟΠΠΕΠ" και επομένως εκτελείται όταν γίνεται κλικ σε αυτό. Εντός του καλούμε τη μέθοδο **event.preventDefault()** για να αποτρέψουμε την προεπιλεγμένη συμπεριφορά του και έπειτα τη συνάρτηση **setShowCertification( )**. Στη συνάρτηση περνάμε ως όρισμα μία callback συνάρτηση, στην οποία περνάμε το όρισμα **prevState** που αντιπροσωπεύει την προηγούμενη τιμή του **showCertification**. Μέσα στην callback χρησιμοποιούμε το **!prevState** για να αντιστρέψουμε την boolean τιμή. Έτσι αν η προηγούμενη τιμή είναι false, τότε την μετατρέπουμε σε true και ενεργοποιείται το switch. Διαφορετικά αν είναι true, τότε την μετατρέπουμε σε false με αποτέλεσμα να απενεργοποιείται το switch.

Στις γραμμές 155 έως 157 ορίζουμε τον event handler **const codecertChangeHandler = (event) => { }** για το πεδίο "Κωδικός Πιστοποίησης". Στο εσωτερικό του αποθηκεύουμε την τιμή που έχει εισαχθεί στη μεταβλητή **enteredCodecert** καλώντας το **setEnteredCodecert(event.target.value)** και δεν ενημερώνουμε κάποιο touched state επειδή δεν μας ενδιαφέρει αν έχει "αγγιχτεί" το πεδίο.

Στις γραμμές 159 έως 163 δημιουργούμε τον event handler **const stepChangeHandler = (i, event) => { }**, τον οποίο τον συνδέουμε στο δυναμικό πεδίο "Κωδικοί ΣΤΕΠ" και του περνάμε τα ορίσματα **i** και **event**, όπου το **i** (index) αντιπροσωπεύει τον δείκτη του αντίστοιχου στοιχείου στον πίνακα. Μέσα του αρχικά δημιουργούμε τον τοπικό πίνακα **values** και του αντιγράφουμε τον πίνακα **codesteps** χρησιμοποιώντας τη σύνταξη spread operator (**…**). Στη συνέχεια αποκτούμε πρόσβαση στον συγκεκριμένο δείκτη **i** στον πίνακα **values** και του αποθηκεύουμε την εισαγόμενη τιμή (**values[i] = event.target.value**). Τέλος, καλούμε το **setCodesteps(values)** για να αποθηκεύσουμε τις εισαγόμενες τιμές στον πίνακα **codesteps**.

Στις γραμμές 165 έως 170 κατασκευάζουμε τον event handler **const addStepHandler = (event) => { }** με όρισμα το **event** και τον συνδέουμε στο **onClick** του κουμπιού "Προσθήκη ΣΤΕΠ". Ο συγκεκριμένος event handler εκτελείται όταν πατιέται το κουμπί "Προσθήκη

ΣΤΕΠ". Εντός του αποτρέπουμε την προεπιλεγμένη συμπεριφορά του χρησιμοποιώντας το **event.preventDefault()** και δημιουργούμε τον τοπικό πίνακα **values**. Στον τοπικό πίνακα αντιγράφουμε τον πίνακα **codesteps**, με άλλα λόγια τα υπάρχοντα πεδία "Κωδικοί ΣΤΕΠ" με τις εισαγόμενες τιμές τους. Έπειτα αξιοποιούμε τη μέθοδο **push( )** με παράμετρο ένα κενό string **('')** στον πίνακα **values** με αποτέλεσμα να προστίθεται ένα νέο πεδίο. Τέλος καλούμε το **setCodesteps(values)** για να ενημερώσουμε τον πίνακα **codesteps** με τα ήδη υπάρχοντα πεδία και το νέο πεδίο.

Στις γραμμές 172 έως 177 ορίζουμε τον event handler **const removeStepHandler = (i, event) => { }** και του περνάμε τα ορίσματα **i** και **event**. Τον συγκεκριμένο event handler τον συνδέουμε στο **onClick** σε κάθε κουμπί "Χ" των πεδίων "Κωδικοί ΣΤΕΠ" και επομένως εκτελείται όταν γίνεται κλικ σε κάποιο. Εντός του πραγματοποιούμε παρόμοια βήματα όπως στον προηγούμενο event handler, ωστόσο εδώ χρησιμοποιούμε τη μέθοδο **splice( )** στον τοπικό πίνακα **values** για να αφαιρέσουμε ένα πεδίο. Στη μέθοδο αυτή ενσωματώνουμε τις παραμέτρους **i** και **1**, όπου το **i** προσδιορίζει τη θέση του στοιχείου που θα ξεκινήσει η αφαίρεση, ενώ το **1** είναι ο αριθμός των πεδίων που θα αφαιρεθούν. Τέλος καλούμε το **setCodesteps(values)** για να ενημερώσουμε τον πίνακα **codesteps** με τις καινούριες αλλαγές.

Στις γραμμές 179 έως 188 δημιουργούμε τον event handler **let handleRegistryChange = (i, event) => { }**, τον οποίο τον συνδέουμε στα δυναμικά πεδία "Όνομα Μητρώου" και "ΑΜ/Κωδικός Εκπαιδευτή Στο Μητρώο". Σε αυτόν τον event handler ενσωματώνουμε τα ορίσματα **i** και **event**, όπου το **i** είναι ο δείκτης του αντίστοιχου **registry** object (Μητρώου) στον πίνακα που θα ενημερωθεί. Στο εσωτερικό του ορίζουμε τον τοπικό πίνακα **newRegistryValues** και του αντιγράφουμε τον πίνακα **registryValues** με χρήση της σύνταξης spread operator. Έπειτα πραγματοποιούμε έλεγχο **if**, όπου ελέγχουμε αν το input που εισήχθη η τιμή έχει χαρακτηριστικό **name="name"**. Έτσι ελέγχουμε αν η τιμή εισήχθη σε πεδίο "Όνομα Μητρώου". Αν ισχύει, τότε αποθηκεύουμε την εισαγόμενη τιμή στην ιδιότητα **name** του **registry** object που βρίσκεται στον δείκτη **i** στον πίνακα **newRegistryValues** (**newRegistryValues[i].registry['name'] = event.target.value**). Στη συνέχεια ελέγχουμε με ένα δεύτερο **if** αν το input που εισήχθη η τιμή έχει χαρακτηριστικό **name="code"**, δηλαδή ελέγχουμε αν η τιμή εισήχθη σε πεδίο "ΑΜ/Κωδικός Εκπαιδευτή Στο Μητρώο". Αν ισχύει, τότε αποθηκεύουμε την εισαγόμενη τιμή στην ιδιότητα **code** του αντίστοιχου **registry** στο πίνακα **newRegistryValues** (**newRegistryValues[i].code = event.target.value**). Τέλος αποθηκεύουμε τα **registry** objects με τις εισαγόμενες τιμές που περιέχει ο τοπικός πίνακας

newRegistryValues στον πίνακα **registryValues** καλώντας το **setRegistryValues(newRegistryValues)**.

Στις γραμμές 190 έως 193 κατασκευάζουμε τον event handler **let addRegistryFields = (event) => { }**, τον οποίο τον συνδέουμε στο **onClick** του κουμπιού "Προσθήκη Μητρώου" και επομένως εκτελείται όταν γίνεται κλικ στο κουμπί. Μέσα του καλούμε το **setRegistryValues([...registryValues, { registry: {name: ""}, code: "" }])** για να ενημερώσουμε τη μεταβλητή **registryValues** με έναν νέο πίνακα. Στον νέο πίνακα **registryValues** αντιγράφουμε τα υπάρχοντα δυναμικά πεδία "Όνομα Μητρώου" και "ΑΜ/Κωδικός Εκπαιδευτή Στο Μητρώο" με τις εισαγόμενες τιμές τους και στο τέλος του προσθέτουμε ένα νέο **registry** object (**{ registry: {name: ""}, code: "" }**). Αυτό έχει ως αποτέλεσμα να προστίθενται στο τέλος δύο νέα δυναμικά πεδία "Όνομα Μητρώου" και "ΑΜ/Κωδικός Εκπαιδευτή Στο Μητρώο".

Στις γραμμές 195 έως 200 ορίζουμε τον event handler **let removeRegistryFields = (i, event) => { }** με ορίσματα το **i** και το **event**. Αυτόν τον event handler τον συνδέουμε στο **onClick** σε κάθε κουμπί "X" των πεδίων "Όνομα Μητρώου" και "ΑΜ/Κωδικός Εκπαιδευτή Στο Μητρώο" και εκτελείται όταν πατιέται κάποιο κουμπί. Εντός του αφαιρούμε ένα μητρώο, δηλαδή ένα πεδίο "Όνομα Μητρώου" και ένα πεδίο "ΑΜ/Κωδικός Εκπαιδευτή Στο Μητρώο", πραγματοποιώντας παρόμοια βήματα όπως στον event handler **removeStepHandler**.

Στις γραμμές 202 έως 316 ορίζουμε τον event handler **const submitHandler = (event) => { }**, τον οποίο τον συνδέουμε στο event **onSubmit** του στοιχείου **<form>** για τη διαχείριση της υποβολής της φόρμας. Στον συγκεκριμένο event handler ενσωματώνουμε ως όρισμα το αντικείμενο **event** και μέσα του αξιοποιούμε το **event.preventDefault( )** για να αποτρέψουμε την προεπιλεγμένη συμπεριφορά του προγράμματος περιήγησης, η οποία θα προκαλούσε την επαναφόρτωση της σελίδας κατά την υποβολή της φόρμας. Έτσι έχουμε τη δυνατότητα να επικυρώσουμε κατά την υποβολή και να αποστείλουμε αίτημα στο mockAPI.

Στις γραμμές 204 έως 210 καλούμε τις συναρτήσεις ενημέρωσης **setEnteredCodeTouched(true)**, **setEnteredFirstTouched(true)**, **setEnteredLastTouched(true)**, **setEnteredStreetTouched(true)**, **setEnteredNumberTouched(true)**, **setEnteredPhoneTouched(true)** και **setEnteredEmailTouched(true)** για να ενημερώσουμε τις αντίστοιχες touched μεταβλητές κατάστασης με την τιμή **true**. Αυτές τις ενημερώσεις τις πραγματοποιούμε ακόμα κι αν ο χρήστης δεν έχει "αγγίξει" τα πεδία, διότι όταν υποβάλλει τη φόρμα, τότε επιβεβαιώνει την

ολοκλήρωση της. Επιπλέον, ο κύριος λόγος είναι για να ενεργοποιούμε την εμφάνιση των μηνυμάτων σφάλματος και των ενδείξεων στην περίπτωση που δεν έχουν συμπληρωθεί τα υποχρεωτικά πεδία.

Στις γραμμές 212 έως 272 πραγματοποιούμε ελέγχους **if** για την επικύρωση των πεδίων κατά την υποβολή. Ειδικότερα, στις γραμμές 212 έως 215 ορίζουμε μία δήλωση **if** με συνθήκη το **!enteredCodeIsValid**, όπου ελέγχουμε αν δεν είναι έγκυρη η τιμή που έχει εισαχθεί στο πεδίο "Κωδικός". Αν η συνθήκη είναι true, τότε εστιάζουμε στο πεδίο αξιοποιώντας την ιδιότητα **current** με τη μέθοδο **focus( )** στην αναφορά **codeRef** (**codeRef.current.focus()**). Επίσης χρησιμοποιούμε το **return** για να διακόψουμε την εκτέλεση του event handler και για να μην ολοκληρωθεί η υποβολή της φόρμας.

Στις γραμμές 217 έως 245 ορίζουμε παρόμοιες δηλώσεις **if** για να ελέγξουμε αν δεν είναι έγκυρες οι εισαγόμενες τιμές στα πεδία "Όνομα", "Επώνυμο", "Οδός", "Δήμος", "Τηλέφωνο" και "Email". Αν κάποια δεν είναι έγκυρη, τότε πραγματοποιούμε παρόμοια βήματα όπως στον προηγούμενο έλεγχο **if**.

Στις γραμμές 247 έως 250 ορίζουμε μία δήλωση **if**, στην οποία προσθέτουμε ως συνθήκη μία λογική πράξη AND (**&&**) μεταξύ του **!enteredCodecertIsValid** και της μεταβλητής κατάστασης **showCertification**. Με λίγα λόγια ελέγχουμε αν δεν έχει εισαχθεί έγκυρη τιμή στο πεδίο "Κωδικός Πιστοποίησης" και το switch "Πιστοποίηση ΕΟΠΠΕΠ" είναι ενεργοποιημένο. Αν η συνθήκη ικανοποιείται, τότε εστιάζουμε στο πεδίο χρησιμοποιώντας το **codecertRef.current.focus()** και χρησιμοποιούμε το **return** για να διακόψουμε την εκτέλεση του event handler και για να μην πραγματοποιηθεί η υποβολή.

Στις γραμμές 252 έως 262 επικυρώνουμε το δυναμικό πεδίο "Κωδικοί ΣΤΕΠ" κατά την υποβολή. Πιο συγκεκριμένα, δημιουργούμε μία δήλωση **if** με συνθήκη το **!codestepsIsValid && showCertification**, όπου ελέγχουμε αν δεν είναι έγκυρα τα πεδία "Κωδικοί ΣΤΕΠ" και το switch "Πιστοποίηση ΕΟΠΠΕΠ" είναι ενεργοποιημένο. Εντός του **if** ορίζουμε ένα εμφωλευμένο **if** με συνθήκη το **codesteps.length >= 1**, όπου ελέγχουμε αν ο πίνακας **codesteps** περιέχει ένα ή περισσότερα πεδία. Στη συνέχεια, μέσα στο εμφωλευμένο **if** ορίζουμε τον βρόχο **for (let elem of fieldRef.current.elements)** και εντός του πραγματοποιούμε έλεγχο **if**, ο οποίος εκτελείται για κάθε στοιχείο της φόρμας. Ειδικότερα, ελέγχουμε αν το χαρακτηριστικό **id** του στοιχείου είναι ίσο και ίδιου τύπου με το **'codestep'** και αν η εισαγόμενη τιμή του είναι κενή (**if (elem.id === 'codestep' && !elem.value.trim())**). Αν η

συνθήκη ικανοποιείται για κάποιο στοιχείο, τότε εστιάζουμε σε αυτό με το **elem.focus()** και διακόπτουμε την εκτέλεση του event handler με το **return**.

Στις γραμμές 264 έως 272 επικυρώνουμε κατά την υποβολή τα δυναμικά πεδία "Όνομα Μητρώου" και "ΑΜ/Κωδικός Εκπαιδευτή Στο Μητρώο". Για την ακρίβεια ορίζουμε ένα **if** με συνθήκη το **!registriesIsValid**, όπου ελέγχουμε αν δεν είναι έγκυρες οι τιμές που έχουν εισαχθεί στα δυναμικά πεδία. Μέσα στο **if** ορίζουμε πάλι τον βρόχο **for (let elem of fieldRef.current.elements)** και στο εσωτερικό του κατασκευάζουμε μία δήλωση **if**, η οποία εκτελείται για κάθε στοιχείο της φόρμας. Στην **if** ελέγχουμε αν το στοιχείο έχει την ιδιότητα **data-required** και αν η εισαγόμενη τιμή του είναι κενή (**if (elem.dataset.required && !elem.value.trim())**). Αν είναι αληθής, τότε χρησιμοποιούμε πάλι το **elem.focus()** και το **return**.

Στις γραμμές 274 έως 296 κατασκευάζουμε το αντικείμενο **data** και του αποθηκεύουμε με συγκεκριμένη δομή τα δεδομένα της φόρμας. Στις γραμμές 298 έως 307 δημιουργούμε τον νέο εκπαιδευτή ενηλίκων στο mockAPI. Πιο συγκεκριμένα, εκτελούμε ένα HTTP POST αίτημα προς το mockAPI αξιοποιώντας τη μέθοδο **post( )** της βιβλιοθήκης Axios. Στο **post( )** ενσωματώνουμε δύο ορίσματα, όπου το πρώτο όρισμα είναι το endpoint "https://61e53aed595afe00176e5423.mockapi.io/trainers" που θα αποθηκευτεί ο εκπαιδευτής, ενώ το δεύτερο είναι το αντικείμενο **data** που περιέχει τα δεδομένα του. Αν το αίτημα ολοκληρωθεί επιτυχώς, τότε εκτελείται ο κώδικας της callback συνάρτησης της **then( )** μεθόδου, όπου χρησιμοποιούμε το **navigate("/")** για να πλοηγηθούμε στην αρχική σελίδα. Διαφορετικά αν αποτύχει το αίτημα, τότε εκτελείται το τμήμα **catch( )**, όπου καλούμε το **setError(error)** για να αποθηκεύσουμε το HTTP σφάλμα στη μεταβλητή κατάστασης **error**.

Στις γραμμές 309 έως 315 καλούμε τις συναρτήσεις ενημέρωσης των touched με την τιμή **false** για να επαναφέρουμε τις αντίστοιχες touched μεταβλητές κατάστασης στην αρχική τους κατάσταση. Αυτό το πραγματοποιούμε για να μην εμφανίζονται τα μηνύματα σφάλματος και οι ενδείξεις μετά την υποβολή.

Στις γραμμές 319 έως 497 στο **return** τμήμα υλοποιούμε τη φόρμα με τα πεδία και το breadcrumb που προβάλλονται στην οθόνη. Εντός του **return** χρησιμοποιούμε το component **<Wrapper>** ως container και μέσα του περικλείουμε το περιεχόμενο της σελίδας. Ειδικότερα, στο εσωτερικό του στις γραμμές 322 έως 329 αξιοποιούμε το σημασιολογικό στοιχείο **<nav>** για να καθορίσουμε το breadcrumb της σελίδας. Μέσα στο **<nav>** χρησιμοποιούμε το **<ol>** στοιχείο ως γονέα και εντός του ορίζουμε δύο **<li>** στοιχεία για να δημιουργήσουμε το

breadcrumb. Στο πρώτο **<li>** κατασκευάζουμε τον σύνδεσμο "Αρχική" του breadcrumb με το **<button className="govgr-link">Αρχική</button>** και τον περικλείουμε στο **<Link>** component του React Router. Στο **<Link>** προσθέτουμε το χαρακτηριστικό **to="/"**, ώστε με το πάτημα του συνδέσμου να γίνεται πλοήγηση στην αρχική σελίδα. Στο δεύτερο **<li>** δημιουργούμε το τμήμα "Εισαγωγή" του breadcrumb που μας πληροφορεί σε ποια σελίδα βρισκόμαστε. Στα στοιχεία του breadcrumb προσθέτουμε το χαρακτηριστικό **className** και ενσωματώνουμε προκαθορισμένες κλάσεις CSS του Bootstrap και την custom κλάση **"gap-bottom"** για να μορφοποιήσουμε το στυλ του breadcrumb.

Στη γραμμή 331 προσθέτουμε την έκφραση **{error && <ErrorPage error={error}/>}**, όπου πραγματοποιούμε conditional rendering χρησιμοποιώντας τον λογικό τελεστή AND (**&&**). Στην έκφραση αυτή η μεταβλητή κατάστασης **error** είναι η συνθήκη και ελέγχουμε αν υπάρχει HTTP σφάλμα. Αν η συνθήκη είναι αληθής, τότε εκτελείται το component "ErrorPage" και προβάλλεται η σελίδα σφαλμάτων. Επίσης, στο "ErrorPage" προσθέτουμε το prop **error={error}** για να μεταβιβάσουμε το HTTP σφάλμα. Στη γραμμή 333 πραγματοποιούμε πάλι conditional rendering και ελέγχουμε με τη συνθήκη **!error** αν δεν υπάρχει HTTP σφάλμα. Αν δεν υπάρχει, τότε εμφανίζεται το περιεχόμενο της σελίδας.

Στις γραμμές 334 έως 495 χρησιμοποιούμε το στοιχείο **<form>** για να αναπτύξουμε τη φόρμα. Στο **<form>** προσθέτουμε το event **onSubmit** και του συνδέουμε τον event handler **submitHandler** για τη διαχείριση της υποβολής. Επίσης, προσθέτουμε το χαρακτηριστικό **ref** και του ενσωματώνουμε την αναφορά **fieldRef** για να αποκτήσουμε πρόσβαση στους DOM κόμβους των στοιχείων της φόρμας.

Μέσα στο **<form>** στις γραμμές 336 έως 340 ορίζουμε ένα στοιχείο ** ** και εντός του δημιουργούμε το πεδίο "Κωδικός". Στο ** ** προσθέτουμε το χαρακτηριστικό **className**, στο οποίο χρησιμοποιούμε τη σύνταξη template literals και του ενσωματώνουμε μία κλάση του Digigov CSS για στυλ του GOV.GR και μία custom κλάση. Επιπλέον, στο **className** ενσωματώνουμε την έκφραση **${codeInputIsInvalid ? 'govgr-form-group__error' : ''}**, στην οποία χρησιμοποιούμε τον τριαδικό τελεστή (**? :**) για να επιτύχουμε conditional rendering. Στην έκφραση ελέγχουμε αν η συνθήκη **codeInputIsInvalid** είναι αληθής, δηλαδή αν το πεδίο "Κωδικός" είναι μη έγκυρο. Αν είναι αληθής, τότε εφαρμόζεται δυναμικά η κλάση **'govgr-form-group__error'** και εμφανίζεται στα αριστερά του πεδίου μία κόκκινη κάθετη μπάρα. Εντός του ** ** στη γραμμή 337 δημιουργούμε την ετικέτα "Κωδικός*" χρησιμοποιώντας το στοιχείο **<label>**. Στο **<label>** προσθέτουμε το χαρακτηριστικό

className και του ενσωματώνουμε δύο κλάσεις του Digigov CSS για στυλ του GOV.GR. Ακόμα, προσθέτουμε το χαρακτηριστικό **htmlFor** με την τιμή **"code"** για να το συνδέσουμε με το **<input>** που έχει **id="code"**. Στη συνέχεια, στη γραμμή 338 τοποθετούμε την έκφραση **{codeInputIsInvalid && Λάθος:Πρέπει να εισάγετε τον Κωδικό. }**, όπου πραγματοποιούμε πάλι conditional rendering χρησιμοποιώντας τον λογικό τελεστή AND (**&&**) και ελέγχουμε αν το **codeInputIsInvalid** είναι αληθής. Αν είναι, τότε εμφανίζεται πάνω από το πεδίο το μήνυμα σφάλματος "Πρέπει να εισάγετε τον Κωδικό." Τέλος, στη γραμμή 339 κατασκευάζουμε με το στοιχείο **<input>** το πεδίο "Κωδικός" και του προσθέτουμε τα χαρακτηριστικά **className**, **id**, **name**, **type**, **value**, **ref** και **onChange**. Στο **className** ενσωματώνουμε δύο κλάσεις του Digigov CSS για να ακολουθήσουμε το στυλ του GOV.GR και την έκφραση **${codeInputIsInvalid ? 'govgr-error-input' : ''}**. Στην έκφραση πραγματοποιούμε πάλι conditional rendering και ελέγχουμε αν το **codeInputIsInvalid** είναι αληθής. Αν είναι, τότε εφαρμόζεται δυναμικά η κλάση **'govgr-error-input'** και το περίγραμμα του πεδίου γίνεται κόκκινο για να επισημανθεί ως μη έγκυρο. Το **id** καθορίζει το μοναδικό αναγνωριστικό του πεδίου και του εκχωρούμε τη μοναδική τιμή **"code"**. Το **name** καθορίζει το όνομα του πεδίου και του τοποθετούμε την τιμή **"code"**. Στο **type** προσθέτουμε το **"text"**, ώστε το **<input>** να είναι πεδίο εισαγωγής κειμένου, ενώ στο **value** ενσωματώνουμε τη μεταβλητή κατάστασης **enteredCode** για να μετατρέψουμε το στοιχείο σε ελεγχόμενο (controlled). Στο **ref** ενσωματώνουμε την αναφορά **codeRef** για να αποκτήσουμε πρόσβαση στον DOM κόμβο του πεδίου, ενώ στο **onChange** τον event handler **codeChangeHandler**.

Στις γραμμές 342 έως 361 δημιουργούμε τα πεδία "Όνομα", "Επώνυμο" και "Πατρώνυμο". Αρχικά ορίζουμε ένα στοιχείο **<fieldset>** για να ομαδοποιήσουμε τα συγκεκριμένα πεδία, επειδή σχετίζονται μεταξύ τους. Μέσα στο **<fieldset>** ορίζουμε ένα στοιχείο ** ** και του προσθέτουμε το **className={`gap-name-bottom ${firstInputIsInvalid || lastInputIsInvalid ? 'govgr-form-group__error' : '' }`}**, όπου πραγματοποιούμε conditional rendering και ελέγχουμε αν το πεδίο "Όνομα" ή το πεδίο "Επώνυμο" είναι μη έγκυρα. Αν κάποιο από τα πεδία είναι μη έγκυρο, τότε εφαρμόζεται δυναμικά η κλάση **'govgr-form-group__error'** και εμφανίζεται στα αριστερά των πεδίων μία κόκκινη κάθετη μπάρα. Εντός του ** ** ορίζουμε τρία ένθετα ** **, στα οποία δημιουργούμε τα πεδία "Όνομα", "Επώνυμο" και "Πατρώνυμο". Ειδικότερα, εντός του πρώτου και του δεύτερου ** ** χρησιμοποιούμε τα ίδια στοιχεία και χαρακτηριστικά όπως στο πεδίο "Κωδικός" για να δημιουργήσουμε τις ετικέτες και τα πεδία "Όνομα" και "Επώνυμο". Ωστόσο τους ενσωματώνουμε μοναδικές τιμές, μεταβλητές

κατάστασης, αναφορές και event handlers. Στο εσωτερικό του τρίτου ** ** δημιουργούμε την ετικέτα "Πατρώνυμο" με το **<label>** και του προσθέτουμε το **htmlFor="father"** για να το συνδέσουμε με το στοιχείο που έχει **id="father"**. Έπειτα κατασκευάζουμε το πεδίο "Πατρώνυμο" με το **<input>** και του προσθέτουμε τα χαρακτηριστικά **className**, **id**, **name**, **type**, **value** και **onChange**. Στο **id** και **name** προσθέτουμε την τιμή **"father"**, ενώ στο **type** την τιμή **"text"**. Στο **value** ενσωματώνουμε τη μεταβλητή **enteredFather** και στο **onChange** τον event handler **fatherChangeHandler**. Σε αντίθεση με τα προηγούμενα πεδία, σε αυτό δεν ενσωματώνουμε κάποια αναφορά και δεν χρησιμοποιούμε conditional rendering, διότι το πεδίο είναι προαιρετικό.

Στις γραμμές 363 έως 366 υλοποιούμε την ετικέτα και το πεδίο "Επάγγελμα" με παρόμοιο τρόπο όπως το πεδίο "Πατρώνυμο", ωστόσο τους ενσωματώνουμε μοναδικές τιμές.

Στις γραμμές 368 έως 431 δημιουργούμε τα πεδία "Οδός", "Αριθμός", "Τηλέφωνο", "Email" και την αναπτυσσόμενη λίστα "Δήμος" και τα περικλείουμε μέσα σε ένα στοιχείο **<fieldset>** για να τα ομαδοποιήσουμε. Πιο συγκεκριμένα, εντός του **<fieldset>** στη γραμμή 369 κατασκευάζουμε την επικεφαλίδα "Στοιχεία Επικοινωνίας" με το στοιχείο **<h3>** και την περικλείουμε μέσα σε ένα στοιχείο **<legend>** για να την ορίσουμε ως λεζάντα της ενότητας. Στις γραμμές 370 έως 430 χρησιμοποιούμε ένα ** ** ως γονέα και εντός του ορίζουμε πέντε ** **, στα οποία δημιουργούμε τα πεδία και την αναπτυσσόμενη λίστα. Στο ** ** τοποθετούμε το **className={`${streetInputIsInvalid || municipalityInputIsInvalid || phoneInputIsInvalid || emailInputIsInvalid ? 'govgr-form-group__error' : '' }`}**, στο οποίο χρησιμοποιούμε conditional rendering και ελέγχουμε αν το πεδίο "Οδός" ή η αναπτυσσόμενη λίστα "Δήμος" ή το πεδίο "Τηλέφωνο" ή το πεδίο "Email" είναι μη έγκυρα. Αν κάποιο είναι μη έγκυρο, τότε εφαρμόζεται δυναμικά η κλάση **'govgr-form-group__error'** και προβάλλεται στα αριστερά των πεδίων και της αναπτυσσόμενης λίστας μία κόκκινη κάθετη μπάρα. Ειδικότερα, εντός του ** ** στη γραμμή 371 εισάγουμε με το **<h4>** την επικεφαλίδα "Διεύθυνση".

Στις γραμμές 373 έως 377 εντός του ** ** κατασκευάζουμε την ετικέτα και το πεδίο "Οδός" χρησιμοποιώντας τα ίδια στοιχεία και χαρακτηριστικά όπως στα πεδία "Κωδικός", "Όνομα" και "Επώνυμο". Η διαφορά είναι ότι τους ενσωματώνουμε μοναδικές τιμές. Στις γραμμές 379 έως 382 μέσα στο ** ** υλοποιούμε την ετικέτα και το πεδίο "Αριθμός" όπως τα πεδία "Πατρώνυμο" και "Επάγγελμα", ωστόσο εκχωρούμε σε αυτά διαφορετικές τιμές.

Στις γραμμές 384 έως 417 εντός του ** ** δημιουργούμε με το **<label>** την ετικέτα "Δήμος" και του προσθέτουμε το **htmlFor="municipality"**. Στη συνέχεια χρησιμοποιούμε την έκφραση **{municipalityInputIsInvalid && Λάθος:Πρέπει να εισάγετε το Δήμο. }** για να επιτύχουμε conditional rendering. Έπειτα αξιοποιούμε τα components **<Autocomplete>**, **<Box>** και **<StyledTextField>** της βιβλιοθήκης Material UI για να δημιουργήσουμε την αναπτυσσόμενη λίστα "Δήμος". Στο **<Autocomplete>** προσθέτουμε τα χαρακτηριστικά **id**, **options**, **autoHighlight**, **getOptionLabel**, **renderOption**, **renderInput**, και **onChange**. Στο **id** εκχωρούμε την τιμή **"municipality"**, ενώ στο **options** ενσωματώνουμε τη μεταβλητή **dimoi** για να φορτώσουμε ως επιλογές τους δήμους. Να αναφερθεί πάλι ότι η μεταβλητή **dimoi** είναι ένας πίνακας (array) που περιλαμβάνει αντικείμενα (objects), όπου το κάθε αντικείμενο είναι ένας δήμος. Το **autoHighlight** το χρησιμοποιούμε για να επισημαίνεται αυτόματα η πρώτη επιλογή της αναπτυσσόμενης λίστας. Στο **getOptionLabel** ορίζουμε τον τρόπο που θα εμφανίζεται η επιλεγμένη τιμή στο πεδίο, ενώ στο **renderOption** χρησιμοποιούμε το **<Box>** component και προσαρμόζουμε την απεικόνιση της κάθε επιλογής. Στο **onChange** ενσωματώνουμε τον event handler **municipalityChangeHandler**, ενώ στο **renderInput** προσαρμόζουμε την εμφάνιση του πεδίου. Συγκεκριμένα στο **renderInput** ενσωματώνουμε μία callback συνάρτηση, στην οποία περνάμε ως παράμετρο το αντικείμενο **params** και στο εσωτερικό της αξιοποιούμε το styled component **<StyledTextField>**. Στο **<StyledTextField>** αντιγράφουμε τις ιδιότητες του **params** χρησιμοποιώντας το **{...params}**, το οποίο είναι σύνταξη spread operator. Ακόμα του προσθέτουμε το **inputRef={municipalityRef}** για να αποκτήσουμε πρόσβαση στον DOM κόμβο της αναπτυσσόμενης λίστας και τα χαρακτηριστικά **inputProps**, **sx** και **variant** για περαιτέρω εφαρμογή ιδιοτήτων και προσαρμογή του στυλ. Στο **sx** προσθέτουμε την CSS ιδιότητα **border** και της εκχωρούμε την έκφραση **`${municipalityInputIsInvalid ? '3px solid #cc2e2e' : '2px solid black'}`**. Στην έκφραση χρησιμοποιούμε τον τριαδικό τελεστή για να επιτύχουμε conditional rendering και ελέγχουμε αν η αναπτυσσόμενη λίστα είναι μη έγκυρη. Αν είναι, τότε εφαρμόζεται στην ιδιότητα **border** η τιμή **'3px solid #cc2e2e'** και το περίγραμμα του πεδίου γίνεται κόκκινο για να επισημανθεί ως μη έγκυρο. Διαφορετικά εφαρμόζεται η προεπιλεγμένη τιμή **'2px solid black'**.

Στις γραμμές 419 έως 423 εντός του ** ** χρησιμοποιούμε πάλι το **<label>** για να δημιουργήσουμε την ετικέτα "Τηλέφωνο*" και του προσθέτουμε **htmlFor="phone"**. Έπειτα πραγματοποιούμε conditional rendering με την έκφραση **{phoneInputIsInvalid && <p**

className="govgr-error-message">Λάθος:Πρέπει να εισάγετε το Τηλέφωνο και να είναι 10 ψηφία.</p>}, όπου ελέγχουμε αν το **phoneInputIsInvalid** είναι αληθής. Αν είναι αληθής, τότε εμφανίζεται το μήνυμα σφάλματος "Πρέπει να εισάγετε το Τηλέφωνο και να είναι 10 ψηφία.". Τέλος υλοποιούμε το πεδίο "Τηλέφωνο" με το **<input>** και του προσθέτουμε τα χαρακτηριστικά **className**, **type**, **id**, **name**, **value**, **ref** και **onChange**, όπως στα inputs "Κωδικός", "Όνομα", "Επώνυμο" και "Οδός". Ωστόσο στα χαρακτηριστικά αυτά ενσωματώνουμε μοναδικές τιμές. Στο **type** εκχωρούμε την τιμή **"tel"**, ώστε το **<input>** να είναι πεδίο εισαγωγής αριθμού τηλεφώνου και για να παρουσιάζεται στις κινητές συσκευές ένα αριθμητικό πληκτρολόγιο. Επίσης με το **"tel"** το πρόγραμμα περιήγησης παρέχει βασική επικύρωση στο πεδίο. Εκτός από τα χαρακτηριστικά που αναφέρθηκαν, στο **<input>** προσθέτουμε επιπλέον τα **inputMode**, **minLength** και **maxLength**. Στο **inputMode** εκχωρούμε το **"numeric"** για να παρουσιάζεται ακόμα πιο ειδικό αριθμητικό πληκτρολόγιο στις κινητές συσκευές. Στο **minLength** και **maxLength** προσθέτουμε την τιμή **"10"**, ώστε το ελάχιστο και μέγιστο μήκος των εισαγόμενων χαρακτήρων να είναι δέκα.

Στις γραμμές 425 έως 429 στο εσωτερικό του ** ** κατασκευάζουμε την ετικέτα και το πεδίο "Email" με παρόμοιο τρόπο όπως τα πεδία "Κωδικός", "Όνομα", "Επώνυμο" και "Οδός". Ωστόσο, οι διαφορές είναι ότι εκχωρούμε στα χαρακτηριστικά τους μοναδικές τιμές και ότι στο **type** προσθέτουμε την τιμή **"email"**, ώστε το **<input>** να είναι πεδίο εισαγωγής email. Επιπλέον με το **type="email"** το πρόγραμμα περιήγησης εκτελεί βασική επικύρωση στο πεδίο.

Στις γραμμές 433 έως 465 κατασκευάζουμε τον διακόπτη "Πιστοποίηση ΕΟΠΠΕΠ", το πεδίο "Κωδικός Πιστοποίησης", το δυναμικό πεδίο "Κωδικοί ΣΤΕΠ" και το κουμπί "Προσθήκη ΣΤΕΠ" και τα περιλαμβάνουμε μέσα σε ένα στοιχείο **<fieldset>** για να τα ομαδοποιήσουμε. Πιο αναλυτικά, μέσα στο **<fieldset>** στη γραμμή 434 εισάγουμε την επικεφαλίδα "Στοιχεία Πιστοποίησης ΕΟΠΠΕΠ" με το **<h3>** και το περικλείουμε στο **<legend>** για να ορίσουμε την επικεφαλίδα ως λεζάντα της ενότητας.

Στη γραμμή 436 δημιουργούμε τον διακόπτη "Πιστοποίηση ΕΟΠΠΕΠ" αξιοποιώντας το component **<Switch>** της βιβλιοθήκης Material UI και του προσθέτουμε τα χαρακτηριστικά **id**, **checked**, **onClick** και **inputProps**. Στο **id** τοποθετούμε την τιμή **"certification"**, ενώ στο **onClick** τον event handler **showCertificationHandler**. Στο **checked** ενσωματώνουμε τη μεταβλητή κατάστασης **showCertification**, ώστε ο διακόπτης να ενεργοποιείται ή να

απενεργοποιείται με βάση την τιμή του **showCertification** αλλά και για να μετατρέψουμε το component σε ελεγχόμενο. Στη γραμμή 437 υλοποιούμε με το **<label>** την ετικέτα "Πιστοποίηση ΕΟΠΠΕΠ" του διακόπτη και του προσθέτουμε το **htmlFor="certification"** για να το συνδέσουμε με τον διακόπτη.

Στη γραμμή 438 πραγματοποιούμε conditional rendering με τη χρήση του λογικού τελεστή AND (**&&**) και ελέγχουμε την τιμή του **showCertification**, δηλαδή αν είναι ενεργοποιημένος ο διακόπτης. Αν είναι αληθής, τότε εμφανίζεται το τμήμα κώδικα (…) των γραμμών 439 έως 464 μετά το **&&**, δηλαδή τα πεδία της πιστοποίησης και το κουμπί "Προσθήκη ΣΤΕΠ". Ειδικότερα, στις γραμμές 439 έως 463 χρησιμοποιούμε ένα ** ** ως γονέα και μέσα του ορίζουμε ένα ** ** και ένα **<button>**. Στο ** ** προσθέτουμε το **className={`${(showCertification && codecertInputIsInvalid) || (showCertification && codestepInputIsInvalid) ? 'govgr-form-group__error' : '' }`}**, στο οποίο ελέγχουμε αν ο διακόπτης είναι ενεργοποιημένος και το πεδίο "Κωδικός Πιστοποίησης" είναι μη έγκυρο ή αν ο διακόπτης είναι ενεργοποιημένος και κάποιο από τα πεδία "Κωδικοί ΣΤΕΠ" είναι μη έγκυρο. Αν ο έλεγχος ικανοποιείται, τότε εφαρμόζεται δυναμικά η κλάση **'govgr-form-group__error'** και προβάλλεται στα αριστερά των πεδίων της πιστοποίησης μία κόκκινη κάθετη μπάρα. Στη συνέχεια, εντός του ** ** στις γραμμές 441 έως 445 ορίζουμε ένα ** **, όπου στο εσωτερικό του κατασκευάζουμε την ετικέτα και το πεδίο "Κωδικός Πιστοποίησης" με παρόμοιο τρόπο όπως τα πεδία "Κωδικός", "Όνομα", "Επώνυμο" και "Οδός". Βέβαια τους ενσωματώνουμε διαφορετικές τιμές.

Στις γραμμές 447 έως 460 ορίζουμε ένα δεύτερο ** **, όπου εντός του αρχικά χρησιμοποιούμε το **<label>** για να κατασκευάσουμε την ετικέτα "Κωδικοί ΣΤΕΠ*" και του προσθέτουμε το **htmlFor="codestep"**. Έπειτα πραγματοποιούμε conditional rendering και ελέγχουμε την τιμή του **codestepInputIsInvalid**, δηλαδή αν κάποιο πεδίο από τους κωδικούς ΣΤΕΠ είναι μη έγκυρο. Αν το **codestepInputIsInvalid** είναι αληθής, τότε προβάλλεται το μήνυμα σφάλματος "Πρέπει να εισάγετε τους Κωδικούς ΣΤΕΠ.". Στη συνέχεια, στις γραμμές 450 έως 459 δημιουργούμε το δυναμικό πεδίο "Κωδικοί ΣΤΕΠ" αξιοποιώντας τη μέθοδο **map( )** στον πίνακα **codesteps**. Στη **map( )** ενσωματώνουμε ως παράμετρο μία callback συνάρτηση, η οποία εκτελείται για το κάθε στοιχείο που υπάρχει στον πίνακα **codesteps**. Στην callback συνάρτηση τοποθετούμε τα ορίσματα **field** και **idx**, όπου το **field** αντιπροσωπεύει την τιμή του αντίστοιχου πεδίου, ενώ το **idx** είναι ο δείκτης του αντίστοιχου πεδίου στον πίνακα. Εντός της callback επιστρέφουμε με το **return( )** ένα πεδίο και ένα κουμπί "X" για το κάθε στοιχείο του πίνακα. Συγκεκριμένα, μέσα στο **return( )** χρησιμοποιούμε ως γονέα ένα ** **, στο οποίο

προσθέτουμε το χαρακτηριστικό **key={`${"step"}-${idx}`}** για να γίνονται σωστές ενημερώσεις στο DOM και εντός του ορίζουμε ακόμα ένα ** **. Εντός του ένθετου ** ** δημιουργούμε το δυναμικό πεδίο με το **<input>**, στο οποίο προσθέτουμε τα χαρακτηριστικά **className**, **id**, **type**, **value** και **onChange**. Στο **className** τοποθετούμε τις κλάσεις **govgr-input**, **govgr-input--width-10**, **input-step** και την έκφραση **${field.trim() === '' ? 'govgr-error-input' : ''}**. Στην έκφραση ελέγχουμε αν η τιμή του αντίστοιχου πεδίου είναι ίση και ίδιου τύπου με κενό string. Αν είναι, τότε εφαρμόζεται δυναμικά η κλάση **'govgr-error-input'** με αποτέλεσμα να γίνεται κόκκινο το περίγραμμα του αντίστοιχου πεδίου και να επισημαίνεται ως μη έγκυρο. Στο **id** εισάγουμε την τιμή **"codestep"** και στο **type** το **"text"** ώστε το **<input>** να είναι πεδίο εισαγωγής κειμένου. Στο **value** ενσωματώνουμε την τιμή **{field || ""}** ώστε το **<input>** να περιέχει την αντίστοιχη εισαγόμενη τιμή ή την προεπιλεγμένη κενή τιμή. Στο **onChange** ενσωματώνουμε μία callback συνάρτηση, στην οποία περνάμε ως όρισμα το **e** και εντός της καλούμε τον event handler **stepChangeHandler( )** με ορίσματα το **idx** και το **e**. Έπειτα υλοποιούμε με το **<button>** το κουμπί "Χ" και του προσθέτουμε τα χαρακτηριστικά **className** και **onClick**. Στο **onClick** ενσωματώνουμε μία callback με όρισμα το **e** και εντός της καλούμε τον event handler **removeStepHandler(idx, e)**. Τέλος, στη γραμμή 462 χρησιμοποιούμε πάλι το **<button>** με τα χαρακτηριστικά **className** και **onClick** για να κατασκευάσουμε το κουμπί "Προσθήκη ΣΤΕΠ". Στο **onClick** ενσωματώνουμε τον event handler **addStepHandler**.

Στις γραμμές 467 έως 491 αναπτύσσουμε τα δυναμικά πεδία "Όνομα Μητρώου", "ΑΜ/Κωδικός Εκπαιδευτή Στο Μητρώο" και το κουμπί "Προσθήκη Μητρώου". Αυτά τα περικλείουμε μέσα σε ένα στοιχείο **<fieldset>** για να τα ομαδοποιήσουμε. Ειδικότερα, εντός του **<fieldset>** στη γραμμή 468 εισάγουμε την επικεφαλίδα "Μητρώα" με τη χρήση του **<h3>** και την περικλείουμε μέσα σε ένα **<legend>** για να την ορίσουμε ως τίτλο της συγκεκριμένης ενότητας.

Στις γραμμές 469 έως 489 ορίζουμε ως γονέα ένα ** ** και του προσθέτουμε το **className={`${registriesInputIsInvalid ? 'govgr-form-group__error' : '' }`}**, στο οποίο ελέγχουμε αν κάποιο πεδίο από τα μητρώα είναι μη έγκυρο. Αν κάποιο είναι μη έγκυρο, τότε εφαρμόζεται δυναμικά η κλάση **'govgr-form-group__error'** και εμφανίζεται στα αριστερά των μητρώων μία κόκκινη κάθετη μπάρα. Εντός του ** ** στις γραμμές 470 και 471 πραγματοποιούμε conditional rendering και ελέγχουμε πάλι αν κάποιο από τα μητρώα είναι μη έγκυρο. Αν ισχύει, τότε προβάλλονται τα μηνύματα σφάλματος "Πρέπει να συμπληρωθούν όλα τα πεδία." και "Πρέπει να εισάγετε το Όνομα Μητρώου και το ΑΜ Εκπαιδευτή.". Έπειτα,

στις γραμμές 472 έως 488 δημιουργούμε τα δυναμικά πεδία "Όνομα Μητρώου" και "ΑΜ/Κωδικός Εκπαιδευτή Στο Μητρώο" χρησιμοποιώντας τη μέθοδο **map( )** στον πίνακα **registryValues**. Στη μέθοδο **map( )** περνάμε ως παράμετρο μία callback συνάρτηση με ορίσματα το **element** και το **index**, όπου το **element** αντιπροσωπεύει το αντίστοιχο registry object (Μητρώο) στον πίνακα **registryValues**, ενώ το **index** είναι ο δείκτης του. Μέσα στην callback επιστρέφουμε δύο πεδία και ένα κουμπί "X" για το κάθε registry object στον πίνακα. Πιο συγκεκριμένα, μέσα στην callback χρησιμοποιούμε το ** ** ως γονέα και εντός του ορίζουμε ένα ** ** και ένα **<button>**. Στο εσωτερικό του ** ** στις γραμμές 476 έως 479 ορίζουμε ένα ** **, όπου εντός του αρχικά κατασκευάζουμε με το **<label>** την ετικέτα "Όνομα Μητρώου*" και της προσθέτουμε το χαρακτηριστικό **htmlFor={`regname${index}`}** για να το συνδέσουμε με το αντίστοιχο **<input>** που έχει **id={`regname${index}`}**. Στη συνέχεια, δημιουργούμε το δυναμικό πεδίο "Όνομα Μητρώου" με το **<input>** στοιχείο και του προσθέτουμε τα χαρακτηριστικά **className**, **id**, **type**, **name**, **data-required**, **value** και **onChange**. Στο **className** προσθέτουμε το **{`govgr-input govgr-!-width-three-quarter ${element.registry['name'].trim() === '' ? 'govgr-error-input' : ''}`}**, στο οποίο πραγματοποιούμε πάλι conditional rendering και ελέγχουμε αν η τιμή της ιδιότητας **registry['name']** του αντίστοιχου registry object είναι ίση και ίδιου τύπου με κενό string. Με άλλα λόγια αν το αντίστοιχο πεδίο "Όνομα Μητρώου" περιέχει κενή τιμή. Αν ισχύει , τότε εφαρμόζεται δυναμικά η κλάση **'govgr-error-input'** και το περίγραμμα του αντίστοιχου πεδίου γίνεται κόκκινο για να επισημανθεί ως μη έγκυρο. Στο **id** εκχωρούμε την τιμή **{`regname${index}`}** και στο **name** την τιμή **"name"**. Στο **type** προσθέτουμε την τιμή **"text"**, ώστε το **<input>** να είναι πεδίο εισαγωγής κειμένου. Το **data-required="true"** μας βοηθάει στην επικύρωση. Στο **value** ενσωματώνουμε την τιμή **{element.registry['name'] || ""}** ώστε το **<input>** να περιέχει την αντίστοιχη τιμή που εισάγεται ή την προεπιλεγμένη κενή τιμή. Στο **onChange** ενσωματώνουμε μία callback, στην οποία περνάμε ως όρισμα το **e** και στο εσωτερικό της καλούμε τον event handler **handleRegistryChange(index, e)**.

Στις γραμμές 481 έως 484 ορίζουμε ακόμα ένα ** **, όπου εντός του κατασκευάζουμε την ετικέτα και το δυναμικό πεδίο "ΑΜ/Κωδικός Εκπαιδευτή Στο Μητρώο" χρησιμοποιώντας τα ίδια στοιχεία και χαρακτηριστικά όπως στο δυναμικό πεδίο "Όνομα Μητρώου". Οι διαφορές είναι ότι στο χαρακτηριστικό **htmlFor** του **<label>** και στο **id** του **<input>** τοποθετούμε το **{`regcode${index}`}**. Στα χαρακτηριστικά **name** και **value** του **<input>** ενσωματώνουμε τις τιμές **"code"** και **{element.code || ""}** αντίστοιχα. Ακόμα, στο **className** του **<input>**

ελέγχουμε την τιμή της ιδιότητας **code** του αντίστοιχου registry object. Έπειτα, στη γραμμή 486 χρησιμοποιούμε το **<button>** με τα χαρακτηριστικά **className** και **onClick** για να δημιουργήσουμε το κουμπί "X". Στο **onClick** ενσωματώνουμε μία callback με όρισμα το **e**, όπου καλούμε τον event handler **removeRegistryFields(index, e)**. Στη συνέχεια, στη γραμμή 490 υλοποιούμε με το **<button>** το κουμπί "Προσθήκη Μητρώου", το οποίο είναι το τελευταίο στοιχείο σε αυτό το **<fieldset>**. Στο **<button>** προσθέτουμε τα χαρακτηριστικά **className** και **onClick**, όπου στο **onClick** ενσωματώνουμε τον event handler **addRegistryFields**.

Τέλος, στη γραμμή 493 δημιουργούμε το κουμπί "Αποθήκευση" αξιοποιώντας πάλι το **<button>** με το χαρακτηριστικό **className**. Επιπλέον σε αντίθεση με τα προηγούμενα **<button>**, σε αυτό προσθέτουμε το χαρακτηριστικό **type="submit"** για να καθορίσουμε ότι αυτό είναι το κουμπί της υποβολής της φόρμας.

```
1. import React, { useState, useRef, useEffect } from "react";
2. import { Link, useNavigate } from "react-router-dom";
3. import axios from "axios";
4. import Wrapper from "./UI/Wrapper";
5. import ErrorPage from "./ErrorPage";
6. import Box from '@mui/material/Box';
7. import Autocomplete from '@mui/material/Autocomplete';
8. import StyledTextField from "../styled-textfield/StyledTextField";
9. import Switch from '@mui/material/Switch';
10.import "./Form.css";
11.import dimoi from "../json/dimoi";
12.
13.const Form = () => {
14.
15.  const [enteredCode, setEnteredCode] = useState('');
16.  const [enteredCodeTouched, setEnteredCodeTouched] = useState(false);
17.  const codeRef = useRef();
18.
19.  const [enteredFirst, setEnteredFirst] = useState('');
20.  const [enteredFirstTouched, setEnteredFirstTouched] = useState(false);
21.  const firstRef = useRef();
22.
23.  const [enteredLast, setEnteredLast] = useState('');
24.  const [enteredLastTouched, setEnteredLastTouched] = useState(false);
25.  const lastRef = useRef();
26.
27.  const [enteredFather, setEnteredFather] = useState('');
28.  const [enteredOccupation, setEnteredOccupation] = useState('');
29.
30.  const [enteredStreet, setEnteredStreet] = useState('');
31.  const [enteredStreetTouched, setEnteredStreetTouched] = useState(false);
```

```
32.  const streetRef = useRef();
33.
34.  const [enteredNumber, setEnteredNumber] = useState('');
35.  const [enteredNumberTouched, setEnteredNumberTouched] = useState(false);
36.
37.  const [enteredMunicipality, setEnteredMunicipality] = useState(null);
38.  const municipalityRef = useRef();
39.
40.  const [enteredPhone, setEnteredPhone] = useState('');
41.  const [enteredPhoneTouched, setEnteredPhoneTouched] = useState(false);
42.  const phoneRef = useRef();
43.
44.  const [enteredEmail, setEnteredEmail] = useState('');
45.  const [enteredEmailTouched, setEnteredEmailTouched] = useState(false);
46.  const emailRef = useRef();
47.
48.  const [enteredCodecert, setEnteredCodecert] = useState('');
49.  const codecertRef = useRef();
50.
51.  const [codesteps, setCodesteps] = useState(['']);
52.
53.  const [registryValues, setRegistryValues] = useState([]);
54.
55.  const fieldRef = useRef();
56.
57.  const [showCertification, setShowCertification] = useState(false);
58.
59.  const [error, setError] = useState(null);
60.
61.  useEffect(() => {
62.    if (!showCertification) {
63.      setEnteredCodecert("");
64.      setCodesteps([""]);
65.    }
66.  }, [showCertification]);
67.
68.  let navigate = useNavigate();
69.
70.  const enteredCodeIsValid = enteredCode.trim() !== '';
71.  const codeInputIsInvalid = !enteredCodeIsValid && enteredCodeTouched;
72.
73.  const enteredFirstIsValid = enteredFirst.trim() !== '';
74.  const firstInputIsInvalid = !enteredFirstIsValid && enteredFirstTouched;
75.
76.  const enteredLastIsValid = enteredLast.trim() !== '';
77.  const lastInputIsInvalid = !enteredLastIsValid && enteredLastTouched;
78.
```

```
79.  const enteredStreetIsValid = ((enteredStreet.trim() === '' &&
   enteredNumber.trim() === '') || (enteredStreet.trim() !== '' &&
   enteredNumber.trim() !== '')) || enteredNumber.trim() === '';
80.  const streetInputIsInvalid = !enteredStreetIsValid;
81.
82.  const enteredMunicipalityIsValid = ((enteredMunicipality === null &&
   enteredStreet.trim() === '') || (enteredMunicipality !== null &&
   enteredStreet.trim() !== '')) || enteredStreet.trim() === '';
83.  const municipalityInputIsInvalid = !enteredMunicipalityIsValid;
84.
85.  const enteredPhoneIsValid = enteredPhone.match(/^[0-9]{10}$/);
86.  const phoneInputIsInvalid = !enteredPhoneIsValid && enteredPhoneTouched;
87.
88.  const enteredEmailIsValid = enteredEmail.trim() !== '' &&
89.  enteredEmail.match(/^(([^<>()[\]\\.,;:\s@"]+(\.[^<>()[\]\\.,;:\s@"]+)*)|(
   ".+"))@((\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}])|(([a-zA-Z\-0-
   9]+\.)+[a-zA-Z]{2,}))$/);
90.  const emailInputIsInvalid = !enteredEmailIsValid && enteredEmailTouched;
91.
92.  const enteredCodecertIsValid = enteredCodecert.trim() !== '';
93.  const codecertInputIsInvalid = !enteredCodecertIsValid;
94.
95.  const codestepsIsValid = codesteps.length >= 1 &&
   codesteps.every(codestep => codestep.trim() !== '');
96.  const codestepInputIsInvalid = !codestepsIsValid;
97.
98.  const registriesIsValid = registryValues.every(registry =>
   registry.registry.name.trim() !== '' && registry.code.trim() !== '');
99.  const registriesInputIsInvalid = !registriesIsValid;
100.
101.
102.    const codeChangeHandler = (event) => {
103.      setEnteredCodeTouched(true);
104.      setEnteredCode(event.target.value);
105.    }
106.
107.    const firstChangeHandler = (event) => {
108.      setEnteredFirstTouched(true);
109.      setEnteredFirst(event.target.value);
110.    }
111.
112.    const lastChangeHandler = (event) => {
113.      setEnteredLastTouched(true);
114.      setEnteredLast(event.target.value);
115.    }
116.
117.    const fatherChangeHandler = (event) => {
118.      setEnteredFather(event.target.value);
```

```
119.     }
120.
121.     const occupationChangeHandler = (event) => {
122.       setEnteredOccupation(event.target.value);
123.     }
124.
125.     const streetChangeHandler = (event) => {
126.       setEnteredStreetTouched(true);
127.       setEnteredStreet(event.target.value);
128.     }
129.
130.     const numberChangeHandler = (event) => {
131.       setEnteredNumberTouched(true);
132.       setEnteredNumber(event.target.value);
133.     }
134.
135.     const municipalityChangeHandler = (event, newInputValue) => {
136.       setEnteredMunicipality(newInputValue);
137.     }
138.
139.     const phoneChangeHandler = (event) => {
140.       setEnteredPhoneTouched(true);
141.       setEnteredPhone(event.target.value);
142.     }
143.
144.
145.     const emailChangeHandler = (event) => {
146.       setEnteredEmailTouched(true);
147.       setEnteredEmail(event.target.value);
148.     }
149.
150.     const showCertificationHandler = (event) => {
151.       event.preventDefault();
152.       setShowCertification(prevState => !prevState);
153.     }
154.
155.     const codecertChangeHandler = (event) => {
156.       setEnteredCodecert(event.target.value);
157.     }
158.
159.     const stepChangeHandler = (i, event) => {
160.       const values = [...codesteps];
161.       values[i] = event.target.value;
162.       setCodesteps(values);
163.     }
164.
165.     const addStepHandler = (event) => {
166.       event.preventDefault();
```

```
167.       const values = [...codesteps];
168.       values.push('');
169.       setCodesteps(values);
170.     }
171.
172.     const removeStepHandler = (i, event) => {
173.       event.preventDefault();
174.       const values = [...codesteps];
175.       values.splice(i, 1);
176.       setCodesteps(values);
177.     }
178.
179.     let handleRegistryChange = (i, event) => {
180.       let newRegistryValues = [...registryValues];
181.       if (event.target.name === 'name') {
182.         newRegistryValues[i].registry['name'] = event.target.value;
183.       };
184.       if (event.target.name === 'code') {
185.         newRegistryValues[i].code = event.target.value;
186.       };
187.       setRegistryValues(newRegistryValues);
188.     }
189.
190.     let addRegistryFields = (event) => {
191.       event.preventDefault();
192.       setRegistryValues([...registryValues,  { registry: {name: ""},
    code: "" }]);
193.     }
194.
195.     let removeRegistryFields = (i, event) => {
196.       event.preventDefault();
197.       let newRegistryValues = [...registryValues];
198.       newRegistryValues.splice(i, 1);
199.       setRegistryValues(newRegistryValues);
200.   }
201.
202.     const submitHandler = (event) => {
203.       event.preventDefault();
204.       setEnteredCodeTouched(true);
205.       setEnteredFirstTouched(true);
206.       setEnteredLastTouched(true);
207.       setEnteredStreetTouched(true);
208.       setEnteredNumberTouched(true);
209.       setEnteredPhoneTouched(true);
210.       setEnteredEmailTouched(true);
211.
212.       if (!enteredCodeIsValid) {
213.         codeRef.current.focus();
```

```
        return;
      }

      if (!enteredFirstIsValid) {
        firstRef.current.focus();
        return;
      }

      if (!enteredLastIsValid) {
        lastRef.current.focus();
        return;
      }

      if (!enteredStreetIsValid) {
        streetRef.current.focus();
        return;
      }

      if (!enteredMunicipalityIsValid) {
        municipalityRef.current.focus();
        return;
      }

      if (!enteredPhoneIsValid) {
        phoneRef.current.focus();
        return;
      }

      if (!enteredEmailIsValid) {
        emailRef.current.focus();
        return;
      }

      if (!enteredCodecertIsValid && showCertification) {
        codecertRef.current.focus();
        return;
      }

      if (!codestepsIsValid && showCertification) {
        if (codesteps.length >= 1) {
          for (let elem of fieldRef.current.elements) {
            if (elem.id === 'codestep' && !elem.value.trim()) {
              elem.focus();
              return;
            }
          }
        }
        return;
```

```
262.       }
263.
264.       if (!registriesIsValid) {
265.         for (let elem of fieldRef.current.elements) {
266.           if (elem.dataset.required && !elem.value.trim()) {
267.             elem.focus();
268.             return;
269.           }
270.         }
271.         return;
272.       }
273.
274.       const data = {
275.         code: enteredCode,
276.         name: {
277.           first: enteredFirst,
278.           last: enteredLast,
279.           father: enteredFather,
280.         },
281.         occupation: enteredOccupation,
282.         contact: {
283.           address: {
284.             street: enteredStreet,
285.             number: enteredNumber,
286.             municipality: enteredMunicipality
287.           },
288.           phone: enteredPhone,
289.           email: enteredEmail
290.         },
291.         certification: {
292.           code: enteredCodecert,
293.           step: codesteps
294.         },
295.         registries: registryValues
296.       }
297.
298.       axios
299.         .post("https://61e53aed595afe00176e5423.mockapi.io/trainers",
    data)
300.         .then((response) => {
301.           console.log(response);
302.           setError(null);
303.           navigate("/");
304.         })
305.         .catch((error) => {
306.           setError(error);
307.         });
308.
```

```
309.       setEnteredCodeTouched(false);
310.       setEnteredFirstTouched(false);
311.       setEnteredLastTouched(false);
312.       setEnteredStreetTouched(false);
313.       setEnteredNumberTouched(false);
314.       setEnteredPhoneTouched(false);
315.       setEnteredEmailTouched(false);
316.     }
317.
318.
319.     return (
320.       <Wrapper>
321.
322.         <nav aria-label="breadcrumb" className="gap-bottom">
323.           <ol className="breadcrumb">
324.             <li className="breadcrumb-item fs-5">
325.               <Link to="/"><button className="govgr-
    link">Αρχική</button></Link>
326.             </li>
327.             <li className="breadcrumb-item fs-5
    active"><button>Εισαγωγή</button></li>
328.           </ol>
329.         </nav>
330.
331.         {error && <ErrorPage error={error}/>}
332.
333.         {!error &&
334.         <form onSubmit={submitHandler} ref={fieldRef}>
335.
336.           <div className={`govgr-form-group gap-bottom
    ${codeInputIsInvalid ? 'govgr-form-group__error' : ''}`}>
337.             <label className="govgr-label govgr-!-font-weight-bold"
    htmlFor="code">Κωδικός*</label>
338.             {codeInputIsInvalid && <p className="govgr-error-
    message"><span className="govgr-visually-hidden">Λάθος:</span>Πρέπει να
    εισάγετε τον Κωδικό.</p>}
339.             <input className={`govgr-input govgr-!-width-three-quarter
    ${codeInputIsInvalid ? 'govgr-error-input' : ''}`} id="code" name="code"
    type="text" value={enteredCode} ref={codeRef} onChange={codeChangeHandler}
    />
340.           </div>
341.
342.           <fieldset>
343.             <div className={`gap-name-bottom ${firstInputIsInvalid ||
    lastInputIsInvalid ? 'govgr-form-group__error' : '' }`}>
344.               <div className="govgr-form-group">
345.                 <label className="govgr-label govgr-!-font-weight-bold"
    htmlFor="first">Όνομα*</label>
```

```
346.                 {firstInputIsInvalid && <p className="govgr-error-
   message"><span className="govgr-visually-hidden">Λάθος:</span>Πρέπει να
   εισάγετε το Όνομα.</p>}
347.                 <input className={`govgr-input govgr-!-width-three-
   quarter ${firstInputIsInvalid ? 'govgr-error-input' : ''}`} id="first"
   name="first" type="text" value={enteredFirst} ref={firstRef}
   onChange={firstChangeHandler} />
348.               </div>
349.
350.               <div className="govgr-form-group margin-minus-top">
351.                 <label className="govgr-label govgr-!-font-weight-bold"
   htmlFor="last">Επώνυμο*</label>
352.                 {lastInputIsInvalid && <p className="govgr-error-
   message"><span className="govgr-visually-hidden">Λάθος:</span>Πρέπει να
   εισάγετε το Επώνυμο.</p>}
353.                 <input className={`govgr-input govgr-!-width-three-
   quarter ${lastInputIsInvalid ? 'govgr-error-input' : ''}`} id="last"
   name="last" type="text" value={enteredLast} ref={lastRef}
   onChange={lastChangeHandler} />
354.               </div>
355.
356.               <div className="govgr-form-group margin-minus-top">
357.                 <label className="govgr-label govgr-!-font-weight-bold"
   htmlFor="father">Πατρώνυμο</label>
358.                 <input className="govgr-input govgr-!-width-three-
   quarter" id="father" name="father" type="text" value={enteredFather}
   onChange={fatherChangeHandler}/>
359.               </div>
360.             </div>
361.           </fieldset>
362.
363.           <div className="govgr-form-group gap-top">
364.             <label className="govgr-label govgr-!-font-weight-bold"
   htmlFor="occupation">Επάγγελμα</label>
365.             <input className="govgr-input govgr-!-width-three-quarter"
   id="occupation" name="occupation" type="text" value={enteredOccupation}
   onChange={occupationChangeHandler}/>
366.           </div>
367.
368.           <fieldset>
369.             <legend><h3 className="govgr-heading-m">Στοιχεία
   Επικοινωνίας</h3></legend>
370.             <div className={`${streetInputIsInvalid ||
   municipalityInputIsInvalid || phoneInputIsInvalid || emailInputIsInvalid ?
   'govgr-form-group__error' : '' }`}>
371.               <h4 className="govgr-heading-s margin-minus-
   top">Διεύθυνση</h4>
372.
```

```
373.              <div className="govgr-form-group">
374.                <label className="govgr-label govgr-!-font-weight-bold"
   htmlFor="street">Οδός</label>
375.                {streetInputIsInvalid && <p className="govgr-error-
   message"><span className="govgr-visually-hidden">Λάθος:</span>Πρέπει να
   εισάγετε την Οδό.</p>}
376.                <input className={`govgr-input govgr-!-width-three-
   quarter ${streetInputIsInvalid ? 'govgr-error-input' : ''}`} id="street"
   name="street" type="text" ref={streetRef} value={enteredStreet}
   onChange={streetChangeHandler} />
377.              </div>
378.
379.              <div className="govgr-form-group">
380.                <label className="govgr-label govgr-!-font-weight-bold"
   htmlFor="number">Αριθμός</label>
381.                <input className="govgr-input govgr-!-width-three-
   quarter" id="number" name="number" type="text" value={enteredNumber}
   onChange={numberChangeHandler} />
382.              </div>
383.
384.              <div className="govgr-form-group gap-bottom">
385.                <label className="govgr-label govgr-!-font-weight-bold"
   htmlFor="municipality">Δήμος</label>
386.                {municipalityInputIsInvalid && <p className="govgr-error-
   message"><span className="govgr-visually-hidden">Λάθος:</span>Πρέπει να
   εισάγετε το Δήμο.</p>}
387.                <Autocomplete
388.                id="municipality"
389.                options={dimoi}
390.                autoHighlight
391.                getOptionLabel={(option) => `ΔΗΜΟΣ ${option.name}`}
392.                renderOption={(props, option) => (
393.                  <Box
394.                    component="li"
395.                    {...props}
396.                  >
397.                    ΔΗΜΟΣ {option.name}
398.                  </Box>
399.                )}
400.                renderInput={(params) => (
401.                  <StyledTextField
402.                    {...params}
403.                    inputProps={{
404.                      ...params.inputProps,
405.                      autoComplete: "off", // disable autocomplete and
   autofill,
406.                      style: {
407.                        padding: 0,
```

```
                    },
                  }}
                  sx={{border: `${municipalityInputIsInvalid ? '3px solid #cc2e2e' : '2px solid black'}` }}
                  inputRef={municipalityRef}
                  variant="outlined"
                />
              )}
              onChange={municipalityChangeHandler}
              />
            </div>

            <div className="govgr-form-group">
              <label className="govgr-label govgr-!-font-weight-bold" htmlFor="phone">Τηλέφωνο*</label>
              {phoneInputIsInvalid && <p className="govgr-error-message"><span className="govgr-visually-hidden">Λάθος:</span>Πρέπει να εισάγετε το Τηλέφωνο και να είναι 10 ψηφία.</p>}
              <input className={`govgr-input govgr-!-width-three-quarter ${phoneInputIsInvalid ? 'govgr-error-input' : ''}`} type="tel" id="phone" name="phone" inputMode="numeric" minLength="10" maxLength="10" value={enteredPhone} ref={phoneRef} onChange={phoneChangeHandler} />
            </div>

            <div className="govgr-form-group">
              <label className="govgr-label govgr-!-font-weight-bold" htmlFor="email">Email*</label>
              {emailInputIsInvalid && <p className="govgr-error-message"><span className="govgr-visually-hidden">Λάθος:</span>Πρέπει να εισάγετε το Email και να είναι έγκυρο.</p>}
              <input className={`govgr-input govgr-!-width-three-quarter ${emailInputIsInvalid ? 'govgr-error-input' : ''}`} type="email" id="email" name="email" value={enteredEmail} ref={emailRef} onChange={emailChangeHandler} />
            </div>
          </div>
        </fieldset>

        <fieldset>
          <legend><h3 className="govgr-heading-m">Στοιχεία Πιστοποίησης ΕΟΠΠΕΠ</h3></legend>

          <Switch id="certification" checked={showCertification} onClick={showCertificationHandler} inputProps={{ 'aria-label': 'controlled' }} />
          <label className="govgr-label govgr-!-font-weight-bold cert-label" htmlFor="certification">Πιστοποίηση ΕΟΠΠΕΠ</label>
          {showCertification && (
```

```
439.             <div>
440.               <div className={`${(showCertification &&
   codecertInputIsInvalid) || (showCertification && codestepInputIsInvalid)  ?
   'govgr-form-group__error' : '' }`}>
441.                 <div className="govgr-form-group">
442.                   <label className="govgr-label govgr-!-font-weight-bold"
   htmlFor="codecert">Κωδικός Πιστοποίησης*</label>
443.                   {codecertInputIsInvalid && <p className="govgr-error-
   message"><span className="govgr-visually-hidden">Λάθος:</span>Πρέπει να
   εισάγετε τον Κωδικό Πιστοποίησης.</p>}
444.                   <input className={`govgr-input govgr-!-width-three-
   quarter ${codecertInputIsInvalid ? 'govgr-error-input' : ''}`}
   id="codecert" name="codecert" type="text" value={enteredCodecert}
   ref={codecertRef} onChange={codecertChangeHandler} />
445.                 </div>
446.
447.                 <div className="govgr-form-group">
448.                   <label className="govgr-label govgr-!-font-weight-bold"
   htmlFor="codestep">Κωδικοί ΣΤΕΠ*</label>
449.                   {codestepInputIsInvalid && <p className="govgr-error-
   message"><span className="govgr-visually-hidden">Λάθος:</span>Πρέπει να
   εισάγετε τους Κωδικούς ΣΤΕΠ.</p>}
450.                     {codesteps.map((field, idx) => {
451.                       return (
452.                       <div key={`${"step"}-${idx}`}>
453.                         <div className="flex-row">
454.                           <input className={`govgr-input govgr-input--
   width-10 input-step ${field.trim() === '' ? 'govgr-error-input' : ''}`}
   id="codestep" type="text" value={field || ""} onChange={e =>
   stepChangeHandler(idx, e)} />
455.                           <button className="govgr-btn govgr-btn-warning
   remove-step" onClick={(e) => removeStepHandler(idx, e)}>X</button>
456.                         </div>
457.                       </div>
458.                       );
459.                     })}
460.                 </div>
461.               </div>
462.               <button className="govgr-btn govgr-btn-secondary button-
   step" onClick={addStepHandler}>Προσθήκη ΣΤΕΠ</button>
463.             </div>
464.             )}
465.           </fieldset>
466.
467.           <fieldset>
468.             <legend><h3 className="govgr-heading-m margin-
   top">Μητρώα</h3></legend>
```

```
469.            <div className={`${registriesInputIsInvalid  ? 'govgr-form-
    group__error' : '' }`}>
470.              {registriesInputIsInvalid && <p className="govgr-error-
    message"><span className="govgr-visually-hidden">Λάθος:</span>Πρέπει να
    συμπληρωθούν όλα τα πεδία.</p>}
471.              {registriesInputIsInvalid && <p className="govgr-error-
    message"><span className="govgr-visually-hidden">Λάθος:</span>Πρέπει να
    εισάγετε το Όνομα Μητρώου και το ΑΜ Εκπαιδευτή.</p>}
472.              {registryValues.map((element, index) => (
473.                <div key={index} className="flex-row registry-margin-
    bottom">
474.                  <div className="registry-flex-basis">
475.
476.                    <div className="govgr-form-group">
477.                      <label className="govgr-label govgr-!-font-weight-
    bold" htmlFor={`regname${index}`}>Όνομα Μητρώου*</label>
478.                      <input className={`govgr-input govgr-!-width-three-
    quarter ${element.registry['name'].trim() === '' ? 'govgr-error-input' :
    ''}`} id={`regname${index}`} type="text" name="name" data-required="true"
    value={element.registry['name'] || ""} onChange={e =>
    handleRegistryChange(index, e)} />
479.                    </div>
480.
481.                    <div className="govgr-form-group">
482.                      <label className="govgr-label govgr-!-font-weight-
    bold" htmlFor={`regcode${index}`}>ΑΜ/Κωδικός Εκπαιδευτή Στο Μητρώο*</label>
483.                      <input className={`govgr-input govgr-!-width-three-
    quarter ${element.code.trim() === '' ? 'govgr-error-input' : ''}`}
    type="text" id={`regcode${index}`} name="code" data-
    required="true"  value={element.code || ""} onChange={e =>
    handleRegistryChange(index, e)} />
484.                    </div>
485.                  </div>
486.                  <button className="govgr-btn govgr-btn-warning remove-
    registry" onClick={(e) => removeRegistryFields(index, e)}>X</button>
487.                </div>
488.              ))}
489.            </div>
490.            <button className="govgr-btn govgr-btn-secondary button-
    registry" onClick={addRegistryFields}>Προσθήκη Μητρώου</button>
491.          </fieldset>
492.
493.          <button className="govgr-btn govgr-btn-primary btn-center"
    type="submit">Αποθήκευση</button>
494.
495.        </form>}
496.      </Wrapper>
497.    );
```

```
498.   };
499.
500.   export default Form;
```

Αρχείο-component Trainer.js

Στο συγκεκριμένο αρχείο ορίζουμε το function component **const Trainer = ( ) => { return( ); }** και δημιουργούμε το component "Trainer" που είναι η σελίδα της προβολής των στοιχείων του εκπαιδευτή.

Μέσα στο component στη γραμμή 9 καλούμε το **useParams( )** hook του React Router και αυτό μας επιστρέφει ένα αντικείμενο (object) με τις δυναμικές παραμέτρους του τρέχοντος URL. Στη συνέχεια, με χρήση της σύνταξης object destructuring **const { id }** λαμβάνουμε το αντίστοιχο id από τις παραμέτρους του αντικειμένου και το αποθηκεύουμε στη μεταβλητή **id**. Για παράδειγμα, αν το τρέχον URL είναι "http://localhost:3000/trainers/5", τότε στη μεταβλητή **id** θα αποθηκευτεί το 5.

Στις γραμμές 10 έως 24 ορίζουμε states για την αποθήκευση των στοιχείων του εκπαιδευτή αξιοποιώντας το **useState( )** hook. Ειδικότερα, στη γραμμή 10 καλούμε το **useState( )** με αρχική τιμή το κενό string **('')**, το οποίο όπως έχει ξανά αναφερθεί μας επιστρέφει έναν πίνακα με δύο στοιχεία. Αυτά τα στοιχεία τα αποθηκεύουμε σε δύο μοναδικές μεταβλητές αξιοποιώντας τη σύνταξη array destructuring **const [enteredCode, setEnteredCode]**. Το συγκεκριμένο state το χρησιμοποιούμε για την αποθήκευση του στοιχείου "Κωδικός" του εκπαιδευτή. Ακόμα, η μεταβλητή κατάστασης **enteredCode** περιέχει την τρέχουσα τιμή του στοιχείου "Κωδικός", ενώ το **setEnteredCode** είναι η συνάρτηση για την ενημέρωση της.

Στις γραμμές 11 έως 23 δηλώνουμε με παρόμοιο τρόπο τα υπόλοιπα states αλλά με διαφορετικές μεταβλητές και αρχικοποιήσεις. Αυτά τα states τα χρησιμοποιούμε για την αποθήκευση των στοιχείων "Όνομα", "Επώνυμο", "Πατρώνυμο", "Επάγγελμα", "Οδός", "Αριθμός", "Δήμος", "Τηλέφωνο", "Email", "Κωδικός Πιστοποίησης", "Κωδικοί ΣΤΕΠ", "Ονόματα Μητρώων" και "ΑΜ/Κωδικοί Εκπαιδευτή Στα Μητρώα" αντίστοιχα. Επιπλέον, στη γραμμή 24 το **const [error, setError] = useState(null);** είναι για την αποθήκευση των HTTP σφαλμάτων.

Στις γραμμές 27 έως 50 ορίζουμε το **useEffect( )** hook για να ανακτήσουμε τα στοιχεία του αντίστοιχου εκπαιδευτή από το mockAPI. Στο **useEffect( )** περνάμε δύο ορίσματα, όπου το πρώτο είναι μία callback συνάρτηση και το δεύτερο είναι ο πίνακας των εξαρτήσεων. Εντός της callback συνάρτησης στέλνουμε ένα HTTP GET αίτημα προς το mockAPI με χρήση της μεθόδου **get( )** του Axios, στην οποία μέθοδο ενσωματώνουμε το endpoint `https://61e53aed595afe00176e5423.mockapi.io/trainers/${id}` που είναι αποθηκευμένα τα στοιχεία του εκπαιδευτή. Στο endpoint το δυναμικό τμήμα **${id}** αντικαθίσταται με την τιμή της μεταβλητής **id**. Αν το αίτημα είναι επιτυχές, τότε εκτελείται η callback συνάρτηση της **then( )** μεθόδου, όπου μέσα της στις γραμμές 31 έως 44 καλούμε τις συναρτήσεις ενημέρωσης για να αποθηκεύσουμε στις αντίστοιχες μεταβλητές κατάστασης (state variables) τα ανακτηθέντα στοιχεία του αντίστοιχου εκπαιδευτή. Στη γραμμή 42 αποθηκεύουμε στη μεταβλητή **codesteps** τους κωδικούς ΣΤΕΠ καλώντας τη συνάρτηση **setCodesteps( )** με όρισμα το **response.data.certification.step.join(", ")**. Στο όρισμα χρησιμοποιούμε τη μέθοδο **join( )** με όρισμα το **(", ")** για να αποθηκεύσουμε ως ένα ενιαίο string τους κωδικούς ΣΤΕΠ και να τους διαχωρίσουμε με κόμμα και κενό.

Στη γραμμή 43 αποθηκεύουμε τα ονόματα μητρώων στη μεταβλητή **enteredRegistriesName** καλώντας το **setEnteredRegistriesName(response.data.registries.map((registry) => registry.registry.name).join(", "));**. Στο όρισμα της συνάρτησης αξιοποιούμε στον πίνακα **response.data.registries** τη μέθοδο **map( )** και της ενσωματώνουμε μία callback συνάρτηση. Στην callback της **map( )** περνάμε ως όρισμα το αντικείμενο **registry** που αντιπροσωπεύει το κάθε μητρώο στον πίνακα **response.data.registries** και εντός της δηλώνουμε το **registry.registry.name** για να εξάγουμε και να αποθηκεύσουμε το κάθε όνομα μητρώου στη μεταβλητή **enteredRegistriesName**. Επιπρόσθετα, κάνουμε χρήση της μεθόδου **join(", ")** για να τα αποθηκεύσουμε ως ένα ενιαίο string και να τα διαχωρίσουμε με κόμμα και κενό. Στη γραμμή 44 πραγματοποιούμε παρόμοια βήματα για να αποθηκεύσουμε τους κωδικούς των μητρώων στη μεταβλητή **enteredRegistriesCode**. Διαφορετικά αν το αίτημα αποτύχει, τότε εκτελείται η callback της **catch( )** μεθόδου, όπου μέσα της αποθηκεύουμε το HTTP σφάλμα στη μεταβλητή **error** με το **setError(error)**.

Τέλος, στον πίνακα των εξαρτήσεων προσθέτουμε ως εξάρτηση τη μεταβλητή **id** με αποτέλεσμα το **useEffect( )** να εκτελείται κατά την αρχική τοποθέτηση του component στο DOM αλλά και κάθε φορά που αλλάζει η τιμή του **id**.

Στις γραμμές 52 έως 135 το **return( )** τμήμα είναι αυτό που προβάλλεται στην οθόνη, δηλαδή το breadcrumb και η λίστα με τα συνολικά στοιχεία του εκπαιδευτή. Στο εσωτερικό του **return( )** χρησιμοποιούμε το <**Wrapper**> component ως container και εντός του στις γραμμές 54 έως 61 χρησιμοποιούμε το σημασιολογικό στοιχείο <**nav**> για να προσδιορίσουμε ότι αυτό είναι το breadcrumb της σελίδας. Εντός του <**nav**> ορίζουμε ως γονέα ένα <**ol**> στοιχείο και μέσα του ορίζουμε δύο <**li**> για να κατασκευάσουμε το breadcrumb. Στο πρώτο <**li**> χρησιμοποιούμε το **<button className="govgr-link">Αρχική</button>** για να κατασκευάσουμε το τμήμα "Αρχική" στο breadcrumb και το τοποθετούμε μέσα στο <**Link**> component για να το μετατρέψουμε σε σύνδεσμο. Στο <**Link**> προσθέτουμε την ιδιότητα **to="/"**, ώστε όταν γίνεται κλικ στον σύνδεσμο να πραγματοποιείται μετάβαση στην αρχική σελίδα. Στο δεύτερο <**li**> υλοποιούμε το τελευταίο τμήμα του breadcrumb, στο οποίο τοποθετούμε το <**button**>**{enteredCode}**</**button**> για να εμφανίζεται δυναμικά ο κωδικός του αντίστοιχου εκπαιδευτή.

Στη γραμμή 63 χρησιμοποιούμε την έκφραση **{error && <ErrorPage error={error}/>}** για να πραγματοποιήσουμε conditional rendering. Στην έκφραση ελέγχουμε αν η τιμή της μεταβλητής **error** είναι true. Αν είναι true, τότε εκτελείται το component "ErrorPage" με αποτέλεσμα να εμφανίζεται η σελίδα των σφαλμάτων.

Στη γραμμή 65 πραγματοποιούμε πάλι conditional rendering, ωστόσο τώρα ελέγχουμε αν είναι true το **!error**, δηλαδή την περίπτωση που η τιμή της μεταβλητής **error** είναι false. Αν το **!error** είναι true, τότε προβάλλεται το περιεχόμενο της συγκεκριμένης σελίδας.

Στη γραμμή 67 κατασκευάζουμε την επικεφαλίδα "Στοιχεία Εκπαιδευτή" με τη χρήση του στοιχείου <**h2**>.

Στις γραμμές 68 έως 125 ορίζουμε ένα <**dl**> στοιχείο ως γονέα για να δημιουργήσουμε τη λίστα με τα συνολικά στοιχεία του εκπαιδευτή. Εντός του <**dl**> ορίζουμε ένα <**div**> για κάθε στοιχείο του εκπαιδευτή, όπου στο καθένα προσθέτουμε το **className="govgr-summary-list__row"** για να χωριστούν τα στοιχεία σε σειρές. Στη συνέχεια, στο εσωτερικό του κάθε <**div**> χρησιμοποιούμε το στοιχείο <**dt className="govgr-summary-list__key**"> για να δημιουργήσουμε την ετικέτα του στοιχείου και το <**dd className="govgr-summary-list__value**"> για να εμφανίσουμε την πραγματική τιμή του στοιχείου. Για παράδειγμα, εντός του πρώτου <**div**> στο <**dt**> δημιουργούμε την ετικέτα "Κωδικός", ενώ στο <**dd**> τοποθετούμε το **{enteredCode}** για να εμφανίσουμε τον κωδικό του εκπαιδευτή. Να σημειωθεί ότι στα στοιχεία που μπορεί να μην διαθέτει ο εκπαιδευτής, προσθέτουμε στα <**dd**>

τους εκφράσεις, στις οποίες πραγματοποιούμε conditional rendering χρησιμοποιώντας τον τριαδικό τελεστή. Για παράδειγμα, στο **<dd>** του πατρώνυμου τοποθετούμε την έκφραση **{enteredFather !== "" ? enteredFather : "-"}**, όπου ελέγχουμε αν η τιμή του **enteredFather** είναι διαφορετική του κενού, δηλαδή αν ο εκπαιδευτής διαθέτει πατρώνυμο. Αν διαθέτει, τότε εκτελείται το **enteredFather** και εμφανίζεται το πατρώνυμο του εκπαιδευτή. Διαφορετικά εμφανίζεται μία παύλα (-).

Τέλος, στις γραμμές 127 έως 131 χρησιμοποιούμε το ** ** ως γονέα και εντός του δημιουργούμε το κουμπί "Επεξεργασία" με το **<button type="button" className="btn btn-primary btn-show">Επεξεργασία</button>**. Αυτό το κουμπί το περιλαμβάνουμε εντός του component **<Link to={`/trainers/update/${id}`}>**, ώστε με το πάτημα του κουμπιού να γίνεται πλοήγηση στη σελίδα της επεξεργασίας των στοιχείων του αντίστοιχου εκπαιδευτή.

```
1. import React, { useState, useEffect } from "react";
2. import { useParams, Link } from "react-router-dom";
3. import axios from "axios";
4. import Wrapper from "./UI/Wrapper";
5. import ErrorPage from "./ErrorPage";
6. import "./Read.css";
7.
8. const Trainer = () => {
9.   const { id } = useParams();
10.  const [enteredCode, setEnteredCode] = useState('');
11.  const [enteredFirst, setEnteredFirst] = useState('');
12.  const [enteredLast, setEnteredLast] = useState('');
13.  const [enteredFather, setEnteredFather] = useState('');
14.  const [enteredOccupation, setEnteredOccupation] = useState('');
15.  const [enteredStreet, setEnteredStreet] = useState('');
16.  const [enteredNumber, setEnteredNumber] = useState('');
17.  const [enteredMunicipality, setEnteredMunicipality] = useState(null);
18.  const [enteredPhone, setEnteredPhone] = useState('');
19.  const [enteredEmail, setEnteredEmail] = useState('');
20.  const [enteredCodecert, setEnteredCodecert] = useState('');
21.  const [codesteps, setCodesteps] = useState([]);
22.  const [enteredRegistriesName, setEnteredRegistriesName] = useState([]);
23.  const [enteredRegistriesCode, setEnteredRegistriesCode] = useState([]);
24.  const [error, setError] = useState(null);
25.
26.
27.  useEffect(() => {
28.    axios
29.      .get(`https://61e53aed595afe00176e5423.mockapi.io/trainers/${id}`)
30.      .then((response) => {
```

```
31.           setEnteredCode(response.data.code);
32.           setEnteredFirst(response.data.name.first);
33.           setEnteredLast(response.data.name.last);
34.           setEnteredFather(response.data.name.father);
35.           setEnteredOccupation(response.data.occupation);
36.           setEnteredStreet(response.data.contact.address.street);
37.           setEnteredNumber(response.data.contact.address.number);
38.           setEnteredMunicipality(response.data.contact.address.municipality
   );
39.           setEnteredPhone(response.data.contact.phone);
40.           setEnteredEmail(response.data.contact.email);
41.           setEnteredCodecert(response.data.certification.code);
42.           setCodesteps(response.data.certification.step.join(", "));
43.           setEnteredRegistriesName(response.data.registries.map((registry)
   => registry.registry.name).join(", "));
44.           setEnteredRegistriesCode(response.data.registries.map((registry)
   => registry.code).join(", "));
45.           setError(null);
46.       })
47.       .catch((error) => {
48.         setError(error);
49.       });
50.   }, [id]);
51.
52.   return (
53.     <Wrapper>
54.       <nav aria-label="breadcrumb" className="gap-bottom">
55.         <ol className="breadcrumb">
56.           <li className="breadcrumb-item fs-5">
57.             <Link to="/"><button className="govgr-
   link">Αρχική</button></Link>
58.           </li>
59.           <li className="breadcrumb-item fs-5
   active"><button>{enteredCode}</button></li>
60.         </ol>
61.       </nav>
62.
63.       {error && <ErrorPage error={error}/>}
64.
65.       {!error &&
66.       <>
67.         <h2 className="govgr-heading-l">Στοιχεία Εκπαιδευτή</h2>
68.         <dl className="govgr-summary-list">
69.           <div className="govgr-summary-list__row">
70.             <dt className="govgr-summary-list__key">Κωδικός</dt>
71.             <dd className="govgr-summary-list__value">{enteredCode}</dd>
72.           </div>
73.           <div className="govgr-summary-list__row">
```

```
74.             <dt className="govgr-summary-list__key">Όνομα</dt>
75.             <dd className="govgr-summary-list__value">{enteredFirst}</dd>
76.           </div>
77.           <div className="govgr-summary-list__row">
78.             <dt className="govgr-summary-list__key">Επώνυμο</dt>
79.             <dd className="govgr-summary-list__value">{enteredLast}</dd>
80.           </div>
81.           <div className="govgr-summary-list__row">
82.             <dt className="govgr-summary-list__key">Πατρώνυμο</dt>
83.             <dd className="govgr-summary-list__value">{enteredFather !== ""
    ? enteredFather : "-"}</dd>
84.           </div>
85.           <div className="govgr-summary-list__row">
86.             <dt className="govgr-summary-list__key">Επάγγελμα</dt>
87.             <dd className="govgr-summary-list__value">{enteredOccupation
    !== "" ? enteredOccupation : "-"}</dd>
88.           </div>
89.           <div className="govgr-summary-list__row">
90.             <dt className="govgr-summary-list__key">Οδός</dt>
91.             <dd className="govgr-summary-list__value">{enteredStreet !== ""
    ? enteredStreet : "-"}</dd>
92.           </div>
93.           <div className="govgr-summary-list__row">
94.             <dt className="govgr-summary-list__key">Αριθμός</dt>
95.             <dd className="govgr-summary-list__value">{enteredNumber !== ""
    ? enteredNumber : "-"}</dd>
96.           </div>
97.           <div className="govgr-summary-list__row">
98.             <dt className="govgr-summary-list__key">Δήμος</dt>
99.             <dd className="govgr-summary-list__value">{enteredMunicipality
    !== null ? enteredMunicipality.name : "-"}</dd>
100.            </div>
101.            <div className="govgr-summary-list__row">
102.              <dt className="govgr-summary-list__key">Τηλέφωνο</dt>
103.              <dd className="govgr-summary-
    list__value">{enteredPhone}</dd>
104.            </div>
105.            <div className="govgr-summary-list__row">
106.              <dt className="govgr-summary-list__key">Email</dt>
107.              <dd className="govgr-summary-
    list__value">{enteredEmail}</dd>
108.            </div>
109.            <div className="govgr-summary-list__row">
110.              <dt className="govgr-summary-list__key">Κωδικός
    Πιστοποίησης</dt>
111.              <dd className="govgr-summary-list__value">{enteredCodecert
    !== "" ? enteredCodecert : "-"}</dd>
112.            </div>
```

```
113.            <div className="govgr-summary-list__row">
114.              <dt className="govgr-summary-list__key">Κωδικοί ΣΤΕΠ</dt>
115.              <dd className="govgr-summary-list__value">{codesteps !== ""
    ? codesteps : "-"}</dd>
116.            </div>
117.            <div className="govgr-summary-list__row">
118.              <dt className="govgr-summary-list__key">Ονόματα
    Μητρώων</dt>
119.              <dd className="govgr-summary-
    list__value">{enteredRegistriesName !== "" ? enteredRegistriesName : "-
    "}</dd>
120.            </div>
121.            <div className="govgr-summary-list__row">
122.              <dt className="govgr-summary-list__key">ΑΜ/Κωδικοί
    Εκπαιδευτή Στα Μητρώα</dt>
123.              <dd className="govgr-summary-
    list__value">{enteredRegistriesCode !== "" ? enteredRegistriesCode : "-
    "}</dd>
124.            </div>
125.          </dl>
126.
127.          <div className="text-center">
128.            <Link to={`/trainers/update/${id}`}>
129.              <button type="button" className="btn btn-primary btn-
    show">Επεξεργασία</button>
130.            </Link>
131.          </div>
132.        </>}
133.
134.      </Wrapper>
135.    );
136.  };
137.
138.  export default Trainer;
```

Αρχείο-component Update.js

Στο συγκεκριμένο αρχείο κατασκευάζουμε τη σελίδα της επεξεργασίας των στοιχείων του εκπαιδευτή, δηλαδή τη φόρμα με τα πεδία που καλείται να επεξεργαστεί ο χρήστης ώστε να επεξεργαστεί τα στοιχεία ενός εκπαιδευτή. Αρχικά, εντός του δημιουργούμε το function component "Update" χρησιμοποιώντας τη JavaScript arrow function **const Update = ( ) => { return( ); }**.

Εντός του component χρησιμοποιούμε τον ίδιο κώδικα που χρησιμοποιούμε και στο component "Form" αλλά με μερικές διαφορές. Ειδικότερα, στη γραμμή 15 ορίζουμε το **const { id } = useParams()**, όπου καλούμε το **useParams( )** hook του React Router και αυτό μας επιστρέφει ένα αντικείμενο με τις δυναμικές παραμέτρους της τρέχουσας διεύθυνσης URL. Έπειτα, εξάγουμε το αντίστοιχο id από τις παραμέτρους του αντικειμένου και το αποθηκεύουμε στη μεταβλητή **id** αξιοποιώντας τη σύνταξη object destructuring **const { id }**. Για παράδειγμα, αν η τρέχουσα διεύθυνση URL είναι "http://localhost:3000/trainers/update/2", τότε στη μεταβλητή **id** θα αποθηκευτεί το 2.

Στις γραμμές 17 έως 43 χρησιμοποιούμε το **useEffect( )** hook για να ανακτήσουμε τα στοιχεία του αντίστοιχου εκπαιδευτή από το mockAPI. Στο **useEffect( )** περνάμε δύο ορίσματα, όπου το πρώτο είναι μία callback συνάρτηση και το δεύτερο είναι ο πίνακας με τις εξαρτήσεις. Στο εσωτερικό της callback εκτελούμε ένα HTTP GET αίτημα προς το mockAPI χρησιμοποιώντας τη μέθοδο **get( )** της βιβλιοθήκης Axios, στην οποία προσθέτουμε το endpoint `https://61e53aed595afe00176e5423.mockapi.io/trainers/${id}` που είναι αποθηκευμένα τα στοιχεία του αντίστοιχου εκπαιδευτή. Στο endpoint το **${id}** αντικαθίσταται με την τιμή της μεταβλητής id. Αν το αίτημα ολοκληρωθεί με επιτυχία, τότε εκτελείται ο κώδικας μέσα στην callback της **then( )** μεθόδου, όπου στις γραμμές 21 έως 33 καλούμε τις συναρτήσεις ενημέρωσης **setEnteredCode(response.data.code)**, **setEnteredFirst(response.data.name.first)**, **setEnteredLast(response.data.name.last)**, **setEnteredFather(response.data.name.father)**, **setEnteredOccupation(response.data.occupation)**, **setEnteredStreet(response.data.contact.address.street)**, **setEnteredNumber(response.data.contact.address.number)**, **setEnteredMunicipality(response.data.contact.address.municipality)**, **setEnteredPhone(response.data.contact.phone)**, **setEnteredEmail(response.data.contact.email)**, **setEnteredCodecert(response.data.certification.code)**, **setCodesteps(response.data.certification.step)** και **setRegistryValues(response.data.registries)** για να αποθηκεύσουμε τα ανακτηθέντα δεδομένα του αντίστοιχου εκπαιδευτή στις αντίστοιχες μεταβλητές κατάστασης. Ο λόγος που το πραγματοποιούμε αυτό είναι για να προσυμπληρώσουμε τα πεδία της φόρμας με τα υπάρχοντα στοιχεία. Στις γραμμές 35 έως 37 ορίζουμε μία δήλωση **if** με συνθήκη το **response.data.certification.code !== ""**, στην οποία ελέγχουμε αν ο Κωδικός Πιστοποίησης

του εκπαιδευτή είναι διαφορετικός του κενού string. Με άλλα λόγια ελέγχουμε αν ο εκπαιδευτής διαθέτει Κωδικό Πιστοποίησης. Αν ισχύει, τότε καλούμε το **setShowCertification(true)** ώστε να είναι ενεργοποιημένος ο διακόπτης "Πιστοποίηση ΕΟΠΠΕΠ" και να εμφανίζονται τα πεδία της πιστοποίησης. Διαφορετικά αν το αίτημα αποτύχει, τότε εκτελείται ο κώδικας της **catch( )** μεθόδου, όπου καλούμε το **setError(error)** για να αποθηκεύσουμε το HTTP σφάλμα στη μεταβλητή κατάστασης **error**. Τέλος, στον πίνακα με τις εξαρτήσεις ενσωματώνουμε τη μεταβλητή **id**, ώστε το **useEffect( )** να εκτελείται κατά την αρχική τοποθέτηση του component στο DOM και όποτε αλλάζει η τιμή του **id**.

Εντός του event handler της υποβολής **const submitHandler = (event) => { }**, στις γραμμές 328 έως 337 ενημερώνουμε τον υπάρχοντα εκπαιδευτή ενηλίκων στο mockAPI. Πιο αναλυτικά, εκτελούμε ένα HTTP PUT αίτημα προς το mockAPI χρησιμοποιώντας τη μέθοδο **put( )** της βιβλιοθήκης Axios. Στην **put( )** ενσωματώνουμε δύο ορίσματα, όπου το πρώτο όρισμα είναι το endpoint `https://61e53aed595afe00176e5423.mockapi.io/trainers/${id}` που θέλουμε να γίνει η ενημέρωση, ενώ το δεύτερο είναι το αντικείμενο **data** που περιλαμβάνει τα ενημερωμένα δεδομένα του εκπαιδευτή. Αν το αίτημα ολοκληρωθεί με επιτυχία, τότε εκτελείται η callback συνάρτηση της **then( )** μεθόδου, όπου στο εσωτερικό της καλούμε το **navigate( )** με όρισμα το **`/trainers/${id}`** ώστε μετά την ενημέρωση να πλοηγηθούμε στη σελίδα της προβολής των στοιχείων του αντίστοιχου εκπαιδευτή. Διαφορετικά αν το αίτημα δεν ολοκληρωθεί με επιτυχία, τότε εκτελείται η callback της **catch( )** μεθόδου, στην οποία καλούμε το **setError(error)** για να αποθηκεύσουμε το HTTP σφάλμα στη μεταβλητή **error**.

Μέσα στο **return( )** στις γραμμές 352 έως 362 χρησιμοποιούμε το σημασιολογικό στοιχείο **<nav>** και εντός του δημιουργούμε το breadcrumb της σελίδας. Στο εσωτερικό του **<nav>** χρησιμοποιούμε το **<ol>** στοιχείο ως γονέα και εντός του ορίζουμε τρία **<li>** στοιχεία. Στο πρώτο **<li>** υλοποιούμε με το **<button className="govgr-link">Αρχική</button>** τον σύνδεσμο "Αρχική" του breadcrumb και τον περιτυλίγουμε με το **<Link>** component του React Router. Στο **<Link>** προσθέτουμε το χαρακτηριστικό **to="/"** ώστε με το πάτημα του συνδέσμου να πλοηγούμαστε στην αρχική σελίδα. Στο δεύτερο **<li>** κατασκευάζουμε με το **<button className="govgr-link">{enteredCode}</button>** το δεύτερο τμήμα του breadcrumb, στο οποίο προβάλλεται δυναμικά ο κωδικός του αντίστοιχου εκπαιδευτή. Αυτό το τμήμα το περικλείουμε μέσα στο **<Link to={`/trainers/${id}`}>** component, ώστε με το πάτημα να γίνεται μετάβαση πίσω στη σελίδα της προβολής των στοιχείων του αντίστοιχου εκπαιδευτή. Στο τρίτο **<li>** υλοποιούμε με το **<button>Επεξεργασία</button>** το τμήμα

"Επεξεργασία" του breadcrumb που είναι το τελευταίο και μας ενημερώνει σε ποια σελίδα βρισκόμαστε.

Τέλος, στη γραμμή 372 προσθέτουμε το χαρακτηριστικό **disabled** στο **<input>** "Κωδικός", προκειμένου το πεδίο να είναι απενεργοποιημένο στη συγκεκριμένη σελίδα και να μην δίνεται η δυνατότητα της επεξεργασίας του.

```
1. import React, { useState, useRef, useEffect } from "react";
2. import { useNavigate, useParams, Link} from "react-router-dom";
3. import axios from "axios";
4. import Wrapper from "./UI/Wrapper";
5. import ErrorPage from "./ErrorPage";
6. import Box from '@mui/material/Box';
7. import Autocomplete from '@mui/material/Autocomplete';
8. import StyledTextField from "../styled-textfield/StyledTextField";
9. import Switch from '@mui/material/Switch';
10.import "./Form.css";
11.import dimoi from "../json/dimoi";
12.
13.const Update = () => {
14.
15.    const { id } = useParams();
16.
17.    useEffect(() => {
18.      axios
19.        .get(`https://61e53aed595afe00176e5423.mockapi.io/trainers/${id}`)
20.        .then((response) => {
21.          setEnteredCode(response.data.code);
22.          setEnteredFirst(response.data.name.first);
23.          setEnteredLast(response.data.name.last);
24.          setEnteredFather(response.data.name.father);
25.          setEnteredOccupation(response.data.occupation);
26.          setEnteredStreet(response.data.contact.address.street);
27.          setEnteredNumber(response.data.contact.address.number);
28.          setEnteredMunicipality(response.data.contact.address.municipality
   );
29.          setEnteredPhone(response.data.contact.phone);
30.          setEnteredEmail(response.data.contact.email);
31.          setEnteredCodecert(response.data.certification.code);
32.          setCodesteps(response.data.certification.step);
33.          setRegistryValues(response.data.registries);
34.
35.          if (response.data.certification.code !== "") {
36.            setShowCertification(true);
37.          }
38.          setError(null);
39.        })
```

```
40.         .catch((error) => {
41.           setError(error);
42.         });
43.     }, [id]);
...

232.   const submitHandler = (event) => {
233.       event.preventDefault();

...

304.       const data = {
305.         code: enteredCode,
306.         name: {
307.           first: enteredFirst,
308.           last: enteredLast,
309.           father: enteredFather,
310.         },
311.         occupation: enteredOccupation,
312.         contact: {
313.           address: {
314.             street: enteredStreet,
315.             number: enteredNumber,
316.             municipality: enteredMunicipality
317.           },
318.           phone: enteredPhone,
319.           email: enteredEmail
320.         },
321.         certification: {
322.           code: enteredCodecert,
323.           step: codesteps
324.         },
325.         registries: registryValues
326.       }
327.
328.       axios
329.         .put(`https://61e53aed595afe00176e5423.mockapi.io/trainers/${id}`
    , data)
330.         .then((response) => {
331.           console.log(response);
332.           setError(null);
333.           navigate(`/trainers/${id}`);
334.         })
335.         .catch((error) => {
336.           setError(error);
337.         });

...
```

```
346.     }
347.
348.
349.     return (
350.       <Wrapper>
351.
352.         <nav aria-label="breadcrumb" className="gap-bottom">
353.           <ol className="breadcrumb">
354.             <li className="breadcrumb-item fs-5">
355.               <Link to="/"><button className="govgr-
    link">Αρχική</button></Link>
356.             </li>
357.             <li className="breadcrumb-item fs-5">
358.               <Link to={`/trainers/${id}`}><button className="govgr-
    link">{enteredCode}</button></Link>
359.             </li>
360.             <li className="breadcrumb-item fs-5
    active"><button>Επεξεργασία</button></li>
361.           </ol>
362.         </nav>
363.
364.         {error && <ErrorPage error={error}/>}
365.
366.         {!error &&
367.         <form onSubmit={submitHandler} ref={fieldRef}>
368.
369.           <div className={`govgr-form-group gap-bottom
    ${codeInputIsInvalid ? 'govgr-form-group__error' : ''}`}>
370.             <label className="govgr-label govgr-!-font-weight-bold"
    htmlFor="code">Κωδικός*</label>
371.             {codeInputIsInvalid && <p className="govgr-error-
    message"><span className="govgr-visually-hidden">Λάθος:</span>Πρέπει να
    εισάγετε τον Κωδικό.</p>}
372.             <input className={`govgr-input govgr-!-width-three-quarter
    ${codeInputIsInvalid ? 'govgr-error-input' : ''}`} id="code" disabled
    name="code" type="text" value={enteredCode} ref={codeRef}
    onChange={codeChangeHandler} />
373.           </div>

...

529.   </form>}
530.       </Wrapper>
531.     );
532.   };
533.
534.   export default Update;
```

Αρχείο-component ErrorPage.js

Στο αρχείο αυτό ορίζουμε το **const ErrorPage = (props) => { return ( ); }** και κατασκευάζουμε το component "ErrorPage", το οποίο είναι η σελίδα σφαλμάτων και το αξιοποιούμε στα components "Read", "Form", "Trainer" και "Update". Στο component περνάμε ως παράμετρο το αντικείμενο **props** για να λάβουμε τα δεδομένα που έχουμε μεταβιβάσει από τα άλλα components.

Στο **return** κομμάτι χρησιμοποιούμε το component **<Wrapper>** ως container και εντός του περιλαμβάνουμε το περιεχόμενο της σελίδας σφαλμάτων. Πιο συγκεκριμένα, στο εσωτερικό του εισάγουμε δύο επικεφαλίδες με το στοιχείο **<h1>** και μία παράγραφο με το **<p>**. Στο πρώτο **<h1>** τοποθετούμε την έκφραση **{props.error.response && `${props.error.response.status} ${props.error.response.data}`}**, ενώ στο δεύτερο προσθέτουμε την έκφραση **{!props.error.response && `${props.error.message}`}**. Στις εκφράσεις χρησιμοποιούμε τον λογικό τελεστή AND (**&&**) για να πραγματοποιήσουμε conditional rendering και ελέγχουμε την τιμή του **props.error.response**. Αν είναι αληθής, τότε εκτελείται το **`${props.error.response.status} ${props.error.response.data}`** του πρώτου **<h1>** με αποτέλεσμα να προβάλλεται το αντίστοιχο HTTP μήνυμα σφάλματος. Διαφορετικά αν είναι ψευδής, τότε εκτελείται το **`${props.error.message}`** του δεύτερου **<h1>** και εμφανίζεται κάποιο μήνυμα σφάλματος, για παράδειγμα το "Network Error". Στο **<p>** τοποθετούμε το **{props.error.message}** για να προβάλλουμε μία σύντομη περιγραφή του HTTP μηνύματος σφάλματος. Ακόμα κάνουμε χρήση του component **<ScrollToTop>**, ώστε όταν φορτώνεται η σελίδα να μετακινείται η κάθετη μπάρα κύλισης στην κορυφή.

```
1. import Wrapper from "../components/UI/Wrapper";
2. import ScrollToTop from "./ScrollToTop";
3.
4. const ErrorPage = (props) => {
5.     return (
6.         <Wrapper>
7.             <ScrollToTop />
8.             <h1 className="govgr-heading-xl">{props.error.response &&
   `${props.error.response.status} ${props.error.response.data}`}</h1>
```

```
9.              <h1 className="govgr-heading-xl">{!props.error.response &&
   `${props.error.message}`}</h1>
10.             <p className="govgr-body">{props.error.message}</p>
11.         </Wrapper>
12.     );
13. }
14.
15. export default ErrorPage;
```

Αρχείο-component Wrapper.js

Στο συγκεκριμένο αρχείο ορίζουμε τη JavaScript arrow function **const Wrapper = (props) => { return ( ); }** για να κατασκευάσουμε το function component "Wrapper". Αυτό το component το χρησιμοποιούμε ως wrapper ή container στα components "Form", "Update", "Trainer" και "ErrorPage". Εντός του **return** χρησιμοποιούμε ως γονέα το HTML στοιχείο ** ** και εσωτερικά του ορίζουμε δύο ένθετα ** **. Σε κάθε στοιχείο προσθέτουμε το χαρακτηριστικό **className** και ενσωματώνουμε έτοιμες κλάσεις του Digigov CSS για τη διάταξη και για να ανταποκρίνεται δυναμικά το περιεχόμενο των components που "περιτυλίγει".

Στο function component ενσωματώνουμε το αντικείμενο **props** ως όρισμα προκειμένου να αξιοποιήσουμε το **props.children**, το οποίο είναι ένα ειδικό prop της React. Το **props.children** το τοποθετούμε μέσα σε **{ }** (curly braces), διότι είναι JavaScript έκφραση που θέλουμε να την χρησιμοποιήσουμε στον JSX κώδικα. Ο λόγος που το αξιοποιούμε είναι για να εμφανίζεται το περιεχόμενο που "περιτυλίγει" το component "Wrapper" στα άλλα components. Γενικά χωρίς αυτό δεν θα εμφανιζόταν το περιεχόμενο, επειδή το component δεν γνωρίζει τα παιδιά-στοιχεία του. Τέλος, τα πλεονεκτήματα της δημιουργίας του είναι η αποφυγή διπλότυπου κώδικα και η δυνατότητα επαναχρησιμοποίησης.

```
1. const Wrapper = (props) => {
2.   return (
3.     <div className="govgr-width-container">
4.       <div className="govgr-main-wrapper govgr-layout-wrapper__full-height
   govgr-layout-wrapper">
5.         <div className="govgr-grid-column-two-thirds">
6.         {props.children}
7.         </div>
8.       </div>
9.     </div>
```

```
10.   );
11.};
12.
13.export default Wrapper;
```

Αρχείο-component StyledTextField.js

Σε αυτό το αρχείο προσαρμόζουμε το στυλ εμφάνισης του **TextField** component της βιβλιοθήκης Material UI, δηλαδή το πεδίο της αναπτυσσόμενης λίστας "Δήμος", ώστε να ταιριάζει με τα υπόλοιπα πεδία. Μέσα σε αυτό δημιουργούμε το component "StyledTextField" και χρησιμοποιούμε τη συνάρτηση **styled( )** με δύο ορίσματα. Το πρώτο όρισμα είναι το component **TextField** που θέλουμε να μορφοποιήσουμε και το δεύτερο είναι ένα object που περιέχει τους κανόνες στυλ.

```
1. import TextField from '@mui/material/TextField';
2. import { outlinedInputClasses } from "@mui/material/OutlinedInput";
3. import { inputLabelClasses } from "@mui/material/InputLabel";
4. import { styled } from "@mui/material/styles";
5.
6. const StyledTextField = styled(TextField)({
7.     [`& .${outlinedInputClasses.root}
   .${outlinedInputClasses.notchedOutline}`]: {
8.       border: "none",
9.       borderRadius: 0,
10.    },
11.    [`& .${outlinedInputClasses.root}.${outlinedInputClasses.focused}
   .${outlinedInputClasses.notchedOutline}`]: {
12.      borderColor: "black",
13.    },
14.    [`& .${outlinedInputClasses.input}`]: {
15.      color: "black",
16.    },
17.    [`& .${outlinedInputClasses.root}.${outlinedInputClasses.focused}
   .${outlinedInputClasses.input}`]: {
18.      color: "black"
19.    },
20.    [`& .${inputLabelClasses.outlined}`]: {
21.      color: "black"
22.    },
23.    [`&:hover .${inputLabelClasses.outlined}`]: {
24.      color: "black"
25.    },
```

```
26.     [`& .${inputLabelClasses.outlined}.${inputLabelClasses.focused}`]: {
27.       color: "black"
28.     }
29.   });
30.
31.export default StyledTextField;
```

Αρχείο-dimoi.js

Στο συγκεκριμένο αρχείο δημιουργούμε τον πίνακα (array) **dimoi** που περιέχει αντικείμενα (objects), όπου το κάθε αντικείμενο αντιπροσωπεύει έναν δήμο και έχει τις ιδιότητες **code** και **name**. Αυτόν τον πίνακα τον χρησιμοποιούμε στα components "Read", "Form" και "Update" και ειδικότερα τον ενσωματώνουμε στο Autocomplete component "Δήμος" για να φορτώσουμε ως επιλογές τους δήμους.

Για τη δημιουργία του αρχικά κατεβάσαμε το αρχείο excel "Αντιστοίχιση Καποδίστριας – Καλλικράτης" από τη διεύθυνση "https://geodata.gov.gr/dataset/antistoikhese-ton-pro-kapodistriakon-ota-se-ota-tou-skhediou-kapodistrias-kai-se-demous-tou-pro" και αντιγράψαμε τις στήλες "ΚΩΔΙΚΟΣ ΝΕΟΥ ΟΤΑ", "ΟΝΟΜΑ ΝΕΟΥ ΟΤΑ" και "ΠΕΡΙΦΕΡΕΙΑΚΗ ΕΝΟΤΗΤΑ" σε ένα καινούριο excel αρχείο. Στη συνέχεια, αφαιρέσαμε τα διπλότυπα και μετατρέψαμε το καινούριο excel σε μορφή JSON στην ηλεκτρονική διεύθυνση "https://beautifytools.com/excel-to-json-converter.php". Τέλος, μετατρέψαμε τα JSON δεδομένα σε JavaScript αντικείμενα και τα προσθέσαμε στον πίνακα.

```
const dimoi = [
  {
    code: "9001",
    name: "ΔΟΞΑΤΟΥ (Π.Ε. ΔΡΑΜΑΣ)",
  },
  {
    code: "9002",
    name: "ΔΡΑΜΑΣ (Π.Ε. ΔΡΑΜΑΣ)",
  },
  {
    code: "9003",
    name: "ΚΑΤΩ ΝΕΥΡΟΚΟΠΙΟΥ (Π.Ε. ΔΡΑΜΑΣ)",
  },
  {
    code: "9004",
```

```
    name: "ΠΑΡΑΝΕΣΤΙΟΥ (Π.Ε. ΔΡΑΜΑΣ)",
  },
  {
    code: "9005",
    name: "ΠΡΟΣΟΤΣΑΝΗΣ (Π.Ε. ΔΡΑΜΑΣ)",
  },


...



  {
    code: "9321",
    name: "ΚΑΝΤΑΝΟΥ - ΣΕΛΙΝΟΥ(Π.Ε. ΧΑΝΙΩΝ)",
  },
  {
    code: "9322",
    name: "ΚΙΣΣΑΜΟΥ (Π.Ε. ΧΑΝΙΩΝ)",
  },
  {
    code: "9323",
    name: "ΠΛΑΤΑΝΙΑ (Π.Ε. ΧΑΝΙΩΝ)",
  },
  {
    code: "9324",
    name: "ΣΦΑΚΙΩΝ (Π.Ε. ΧΑΝΙΩΝ)",
  },
  {
    code: "9325",
    name: "ΧΑΝΙΩΝ (Π.Ε. ΧΑΝΙΩΝ)",
  },
];

export default dimoi;
```

Αρχείο index.css

```
1. body {
2.   margin: 0;
3.   font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', 'Roboto',
   'Oxygen',
4.     'Ubuntu', 'Cantarell', 'Fira Sans', 'Droid Sans', 'Helvetica Neue',
```

```
    sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  box-sizing: border-box;
}

code {
  font-family: source-code-pro, Menlo, Monaco, Consolas, 'Courier New',
    monospace;
}
```

Αρχείο Read.css

```
.margin-bottom {
  margin-bottom: 30px;
}

.margin-right {
  margin-right: 5rem;
}

.gap-top {
  margin-top: 100px;
}

.space-top {
  margin-top: 75px;
}

.div-clear {
  display: flex;
  align-items: flex-end;
}

.space-between {
  display: flex;
  justify-content: space-between;
}

.wrap {
  display: flex;
  flex-wrap: wrap;
}

```

```
.btn-page-margin {
  margin-right: 0.75rem;
  margin-left: 0.75rem;
}

.div-create {
  text-align: center;
}

.btn-create {
  display: inline-block;
  margin-right: 0;
  margin-bottom: 0;
}

.div-show {
  text-align: end;
}

.text-center {
  text-align: center;
}

.text-bold {
  font-weight: 500;
}

@media (min-width: 640px){
  .govgr-summary-list__key {
      width: 70%;
  }
}
```

Αρχείο Form.css

```
.gap-bottom {
    margin-bottom: 70px;
}

.gap-name-bottom {
    margin-bottom: 38px;
}

.margin-minus-top {
    margin-top: -10px;
```

```
}

.button-step {
    display: inline-block;
    margin: 3px 0 10px 30px;
    font-size: 14px;
    padding: 10px;
}

.remove-step {
    display: inline-block;
    margin-left: 15px;
    margin-bottom: auto;
    font-size: 14px;
    padding: 10px 15px;
}

.remove-registry {
    display: inline-block;
    margin-top: 30px;
    margin-left: 15px;
    font-size: 14px;
    padding: 10px 15px;
    align-self: center;
}

.flex-row {
    display: flex;
}

.registry-flex-basis {
    flex-basis: 85%;
}

.input-step {
    margin: 10px 0;
}

.input-step:first-child {
    margin-top: 0;
}

.button-certification {
    display: inline-block;
    font-size: 14px;
    padding: 10px;
}

```

```
.button-registry {
    display: inline-block;
    font-size: 14px;
    padding: 10px;
    margin-top: 40px;
    margin-bottom: 100px;
    margin-left: 30px;
}

.registry-margin-bottom {
    margin-bottom: 40px;
}

.registry-margin-bottom:last-child {
    margin-bottom: 0;
}

.btn-center {
    margin: auto;
    margin-bottom: 10rem;
}

.cert-label {
    display: inline-block;
}

.margin-top {
    margin-top: 32px;
}

@media screen and (max-width: 371px) {
    .input-step {
        width: 80%;
    }
}
```