

ΙΟΝΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ



Πτυχιακή Εργασία

Ανάπτυξη Web Εφαρμογής Σύμφωνα με την Τεχνική MVC
(Model-View-Controllers) στην Πλευρά του Browser

Σειράν Οσμάν (Π2009023)

Επιβλέπων: Μιχαήλ Στεφανιδάκης

23 Σεπτεμβρίου 2021

Επιβλέπων

Μιχαήλ Στεφανιδάκης, Αναπληρωτής Καθηγητής,
Τμήμα Πληροφορικής

Τριμελής Επιτροπή

Μιχαήλ Στεφανιδάκης, Αναπληρωτής Καθηγητής,
Τμήμα Πληροφορικής

Κωνσταντίνος Οικονόμου, Καθηγητής,
Τμήμα Πληροφορικής

Θεόδωρος Ανδρόνικος, Αναπληρωτής Καθηγητής,
Τμήμα Πληροφορικής

ΠΕΡΙΛΗΨΗ

Στόχος της πτυχιακής εργασίας είναι η δημιουργία διαδικτυακής εφαρμογής με πρότυπο σχεδίασης την τεχνική Model-View-Controller (MVC) και την υλοποίησή της στα μοντέρνα frameworks Angular και React, αναδεικνύοντας τις προσεγγίσεις που ακολουθούν κατά την ανάπτυξη και τη λειτουργία της εφαρμογής. Ο σκοπός είναι να κάνουμε εμφανή τον λόγο για τη χρήση της τεχνικής MVC, τις δυνατότητες που μας παρέχει στην ανάπτυξη διαδικτυακών εφαρμογών, τον τρόπο δημιουργίας και σύνδεσης των επιμέρους λειτουργιών, καθώς και τα πλεονεκτήματα που προσφέρει στην ανάπτυξη διαδικτυακών εφαρμογών.

Για την ανάδειξη και την επίτευξη του στόχου σε προγραμματιστικό περιβάλλον, κατασκευάσαμε την ίδια δοκιμαστική εφαρμογή Single Page Application σε κάθε ένα από τα υπό δοκιμή frameworks. Η εφαρμογή διαχειρίζεται δεδομένα φοιτητών προς αποφοίτηση, καταχωρώντας τα στοιχεία και τον τίτλο της πτυχιακής τους εργασίας. Υλοποιήσαμε τις λειτουργίες της διαχείρισης των δεδομένων, της διεπαφής χρήστη, καθώς και τη διασύνδεση μεταξύ τους. Παρόλο που κάθε framework έχει διαφορετική λογική σχεδιασμού και εκτέλεσης κώδικα καθώς και διαφορετικό τρόπο συγγραφής εφαρμογών, η αρχιτεκτονική MVC που ακολουθείται και στα δύο καθιστά ευκολότερη τη διαχείριση και την επέκταση των εφαρμογών όπως και την επίλυση προβλημάτων κατά την ανάπτυξη.

ABSTRACT

The aim of this thesis is to create an online application following the Model-View-Controller (MVC) design pattern in two modern frameworks, Angular and React, highlighting the development approaches and differences in operation for the aforementioned application. The purpose is to present the advantages of the MVC pattern usage and explain how to create and connect the individual functions, as well as to present the benefits it offers in development of web applications.

To achieve the above goal in a practical programming environment, we have built a sample Single Page Application in each of the frameworks tested, an application that manages data of students like student details and the title of their thesis. We have implemented the data management functions, user interfaces and the interconnection between them as well. Each framework has a different logic for the design and code execution and requires a different style of programming, but following in both cases the MVC pattern the codebase of an application can be easily maintained and expanded.

Περιεχόμενα

Κεφάλαιο Α΄	1
Α΄.1 Εισαγωγή.....	1
Α΄.2 MVC Design Patterns.....	1
Α΄.3 Single Page Apps (SPAs).....	2
Α΄.4 Σκοπός της ΠΕ	4
Κεφάλαιο Β΄	5
Β΄.1 MVC frameworks.....	5
Β΄.2 Angular.....	5
Β΄.3 React.....	6
Κεφάλαιο Γ΄	7
Γ΄.1 Περιγραφή της Εφαρμογής.....	7
Γ΄.1.1 Προαπαιτούμενα Υλοποίησης.....	8
Γ΄.1.2 RESTful backend.....	8
Γ΄.1.3 Υλοποίηση backend.....	9
Γ΄.2 Υλοποίηση SPA με Angular.....	13
Γ΄.2.1 Εγκατάσταση	13
Γ΄.2.2 Συστατικά εφαρμογής	14
Γ΄.2.2.1 Components.....	15
Γ΄.2.2.2 Templates	25
Γ΄.2.2.3 Directives.....	28
Γ΄.2.2.4 Dependency Injection (Services).....	29
Γ΄.2.3 Forms.....	35
Γ΄.2.4 Routing	38
Γ΄.3 Υλοποίηση SPA με React.....	40
Γ΄.3.1 Εγκατάσταση	40
Γ΄.3.2 JSX	40
Γ΄.3.3 Συστατικά εφαρμογής	41
Γ΄.3.3.1 Components.....	42
Γ΄.3.3.2 HTTP Requests.....	52
Γ΄.3.3.3 Routing	56
Γ΄.3.4 Form Validation.....	59
Συμπεράσματα.....	62

Βιβλιογραφία.....	63
-------------------	----

Κεφάλαιο Α΄

Α΄.1 Εισαγωγή

Όλο και περισσότερες διαδικτυακές εφαρμογές κάνουν την εμφάνισή τους στο διαδίκτυο χάρη στη γρήγορη ανάπτυξη της τεχνολογίας, στην πληθώρα πλεονεκτημάτων που προσφέρουν αλλά και στην ανταγωνιστικότητα των εταιριών στην οικονομία, οι οποίες ώθησαν στην ανάπτυξη και στη δημιουργία νέων τεχνικών για την κάλυψη των αναγκών τους.

Μερικές από τις πιο δημοφιλείς τάσεις οι οποίες επικρατούν στην ανάπτυξη διαδικτυακών εφαρμογών έχουν σκοπό τη βελτίωση της εμπειρίας του χρήστη, τη διευκόλυνση της χρήσης τους, τη διατήρηση του χρήστη στην εφαρμογή προκαλώντας το ενδιαφέρον του αλλά κυρίως, στην κάλυψη και των πιο απαιτητικών αναγκών στο διαδίκτυο. Μία από αυτές τις τάσεις είναι η ανάπτυξη εφαρμογών Single-page Application (SPA) σύμφωνα με την τεχνική Model-View-Controller (MVC).

Α΄.2 MVC Design Patterns

Model-View-Controller (MVC) ονομάζεται το πρότυπο ανάπτυξης λογισμικού με έμφαση στη δημιουργία διεπαφών χρήστη όπου η λογική της εφαρμογής απαρτίζεται από τρία συνδεδεμένα μεταξύ τους στοιχεία.

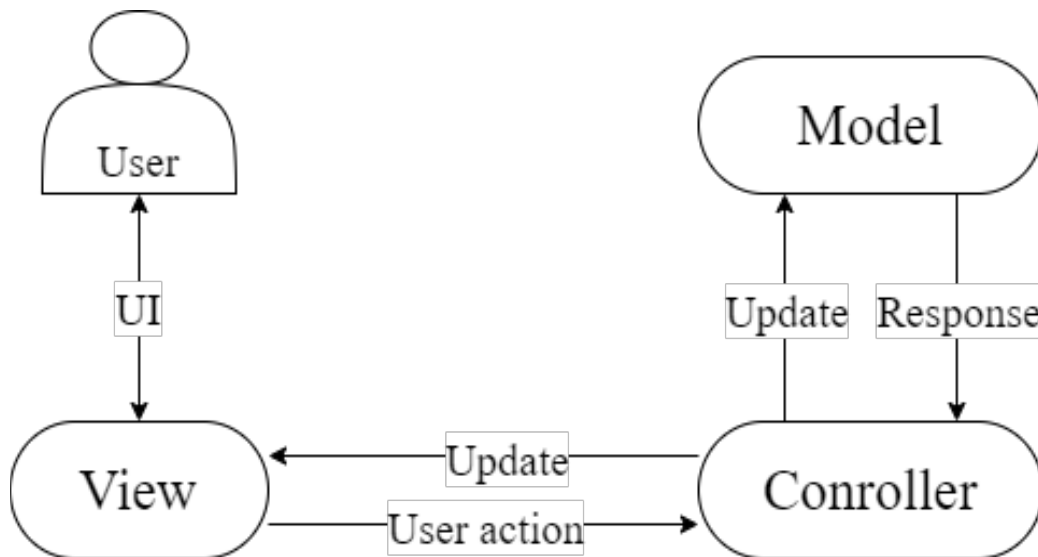
- **Model** ή Μοντέλο είναι το κύριο συστατικό της τεχνικής αυτής, καθώς απαρτίζει τα δεδομένα που προβάλλονται στον χρήστη.
- **View** ή Διεπαφή της εφαρμογής με τον χρήστη είναι η αναπαράσταση και προβολή των δεδομένων (Models) της εφαρμογής για τον σκοπό που έχει αναπτυχθεί.
- **Controller** ή Ελεγκτής είναι το κομμάτι της εφαρμογής που ευθύνεται για την ανταπόκριση στις ενέργειες του χρήστη, εκτελεί ενέργειες σχετικά με την αλλαγή των δεδομένων και ενημερώνει τα δεδομένα της διεπαφής ή ενημερώνει απευθείας την διεπαφή για περαιτέρω λειτουργικότητες της εφαρμογής.

Η γενική ιδέα της τεχνικής MVC απαρτίζεται από τη διεπαφή η οποία είναι το κυριότερο κομμάτι όπου αλληλεπιδρά ο χρήστης με την εφαρμογή, τα δεδομένα τα οποία δρομολογούνται

από τον server για να προβληθούν στον χρήστη μέσω της διεπαφής και ο Controller είναι ο ελεγκτής για την διαχείριση και ενημέρωση της διεπαφής ή των δεδομένων προς προβολή.

Κατά την εκκίνηση της εφαρμογής ο ελεγκτής ενημερώνει τη διεπαφή με τα δεδομένα (Model) προς προβολή είτε αυτά είναι στατικά είτε τα ζητά από κάποιον εξυπηρετητή (server), στη συνέχεια δημιουργείται η διεπαφή την οποία βλέπει ο χρήστης. Οποιαδήποτε ενέργεια γίνεται από τον χρήστη σχετικά με τις λειτουργικότητες της εφαρμογής πυροδοτεί την εκτέλεση του ελεγκτή για ενημέρωση δεδομένων, αλλαγή περιεχομένου ή αλλαγή σελίδας. Στην περίπτωση ενημέρωσης ή καταχώρησης δεδομένων ο ελεγκτής ενημερώνει τα δεδομένα (Model) που πιθανώς βρίσκονται σε βάση δεδομένων και στη συνέχεια ενημερώνει την διεπαφή με τα καινούργια δεδομένα για να προβληθούν στον χρήστη.

Παρακάτω δίνεται μια σχηματική προσέγγιση λειτουργιών της τεχνικής MVC.

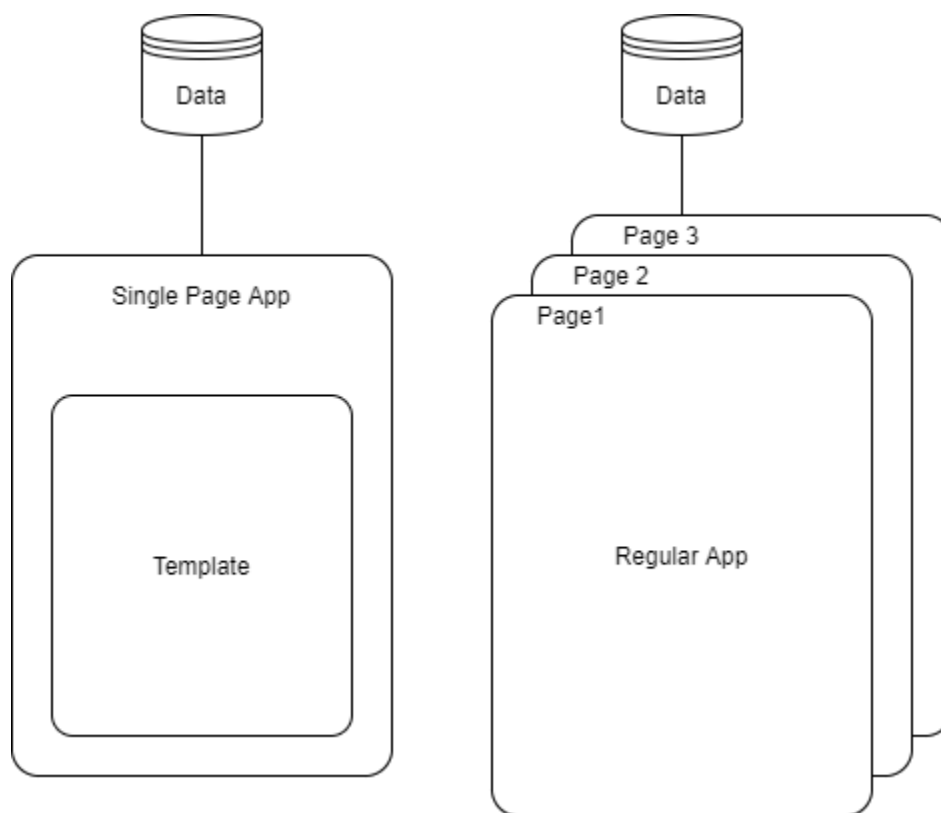


Σχήμα Α'2. 1: Σχεδιάγραμμα MVC

A'3 Single Page Apps (SPAs)

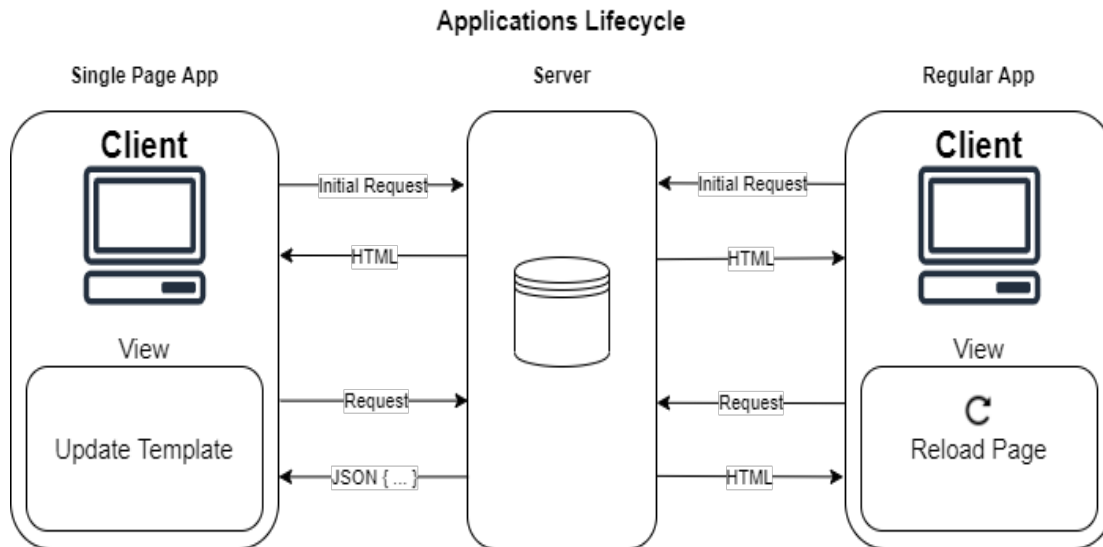
Single page applications είναι οι διαδικτυακές εφαρμογές των οποίων ο απαραίτητος για την εκτέλεσή τους κώδικας φορτώνεται μια φορά στην πλευρά του περιηγητή. Οι εφαρμογές SPA ανανεώνουν το περιεχόμενο ή τις λειτουργίες τους δυναμικά, δίχως να δώσουν την αίσθηση των μεταβάσεων που έχουν οι απλές εφαρμογές όπως την επαναφόρτωση της σελίδας κατά την περιήγηση στην εφαρμογή.

Εικονική αναπαράσταση λειτουργίας σε SPA και σε απλή διαδικτυακή εφαρμογή.



Σχήμα Α'3.1: Λειτουργία SPA

Στο επόμενο σχήμα δίνεται η ανάλυση με τον τρόπο εκτέλεσης μιας SPA και μιας απλής εφαρμογής όπου κατά την περιήγηση, αριστερά στο σχήμα (SPA) το περιεχόμενο αλλάζει δυναμικά σε κάθε αίτημα στον Server ενώ στα δεξιά η αλλαγή περιεχομένου γίνεται φορτώνοντας εξ'ολοκλήρου τη σελίδα με το νέο περιεχόμενο.



Σχήμα Α'3. 2: Κύκλος ζωής εφαρμογών

Α'.4 Σκοπός της ΠΕ

Στόχος της Πτυχιακής Εργασίας είναι να κατασκευάσουμε μια διαδικτυακή εφαρμογή SPA σε πλατφόρμες ανάπτυξης που ακολουθούν τα πρότυπα σχεδίασης λογισμικού MVC (MVC frameworks), να αναδείξουμε τις δυνατότητες που μας προσφέρουν αυτές οι αναπτυξιακές πλατφόρμες και να συγκρίνουμε τις όποιες διαφορές υπάρχουν μεταξύ τους μέσω της ίδιας εφαρμογής.

Στο κεφάλαιο Β' θα αναφερθούμε στα MVC frameworks τα οποία θα χρησιμοποιήσουμε για να αναδείξουμε όλους τους παραπάνω στόχους, εν συνεχεία, στο κεφάλαιο Γ' ακολουθεί η υλοποίηση της εφαρμογής με περαιτέρω ανάλυση στην εγκατάσταση και στον τρόπο λειτουργίας του κάθε framework. Το κείμενο κλείνει με την παρουσίαση των συμπερασμάτων της εργασίας και με μια αναφορά σε μελλοντικές επεκτάσεις της.

Κεφάλαιο Β΄

Β΄.1 MVC frameworks

Κατά καιρούς η ανάπτυξη διαδικτυακών εφαρμογών έρχεται αντιμέτωπη με πάρα πολλές προκλήσεις όπου αυτές μεταβάλλονται σύμφωνα με τις ανάγκες των πελατών και των καταναλωτών. Κάπως έτσι ξεκίνησε η δημιουργία των MVC frameworks, είτε βιβλιοθήκες κώδικα για να καλύψουν συγκεκριμένες ανάγκες και λειτουργίες με μια εύκολη ενσωμάτωση στην εφαρμογή, είτε έχοντας εξελιχθεί σε ολοκληρωμένες πλατφόρμες ανάπτυξης διευκολύνοντας έτσι τη δημιουργία διαδικτυακών εφαρμογών.

Καθώς προσφέρεται μια πληθώρα MVC frameworks στον Παγκόσμιο Ιστό, επιλέξαμε με βάση τα σύγχρονα δεδομένα δύο από τα πιο δημοφιλή και επικρατέστερα στον κλάδο αυτόν MVC frameworks, το Angular και το React. Τα δύο αυτά frameworks περιγράφουμε στη συνέχεια και υλοποιούμε την διαδικτυακή εφαρμογή σε κάθε ένα από αυτά με την προσέγγιση της τεχνικής MVC.

Β΄.2 Angular

Το Angular είναι πλατφόρμα ανάπτυξης με δυνατότητα δημιουργίας διαδικτυακών εφαρμογών μιας σελίδας (SPA). Αναπτύχθηκε με επικεφαλή την ομάδα Angular από την Google, είναι δωρεάν ως προς τη χρήση του και υποστηρίζεται από μια μεγάλη κοινότητα προγραμματιστών και εταιριών.

Αρχικά δημοσιεύτηκε ως βιβλιοθήκη AngularJS, εύκολο στην ενσωμάτωση σε ήδη υπάρχουσες εφαρμογές, επεκτάσιμο, λειτουργικό με άλλες βιβλιοθήκες, με ευκολία στη χρήση του, ένα ευανάγνωστο περιβάλλον για ανάπτυξη διαδικτυακής εφαρμογής με περιεχόμενο που αλλάζει δυναμικά, παρέχοντας λειτουργικότητες με λιγότερο κώδικα προγραμματισμού. Στη συνέχεια εξελίχθηκε σε μια πλατφόρμα ανάπτυξης περιλαμβάνοντας την MVC δομή, συμπεριλαμβάνοντας και βιβλιοθήκες με χαρακτηριστικά όπως ευκολία περιήγησης σε άλλα περιεχόμενα, διαχείριση φόρμας, διαχείριση αιτήσεων με εξυπηρετητές και άλλα, τα οποία διευκολύνουν τη δημιουργία μιας ολοκληρωμένης διαδικτυακής εφαρμογής SPA.

Η δοκιμαστική εφαρμογή υλοποιήθηκε στην εκδοχή Angular 10 (10.1.3) και σε λειτουργικό σύστημα Windows.

B'.3 React

Το React είναι μια βιβλιοθήκη ανάπτυξης διεπαφών χρήστη όπου, η δημιουργία της διεπαφής γίνεται δυναμικά σε πραγματικό χρόνο στη γλώσσα προγραμματισμού JavaScript δίχως στατικό κώδικα HTML και ιδανικό για την δημιουργία διαδικτυακών εφαρμογών μιας σελίδα(SPA).

Αναπτύχθηκε από την ομάδα του Facebook, της διάσημης εφαρμογής κοινωνικής δικτύωσης, είναι δωρεάν ως προς τη χρήση του και υποστηρίζεται επίσης από μια μεγάλη κοινότητα προγραμματιστών και εταιριών.

Μια ιδιαιτερότητα η οποία καθιστά το React ενδιαφέρον είναι ο τρόπος διαχείρισης των δεδομένων για την αναπαράσταση στο πηγαίο έγγραφο (DOM). Κατά την εκτέλεση της εφαρμογής και τη δημιουργία της δομής DOM στον περιηγητή, δημιουργείται και μια ψηφιακή δομή (virtual DOM) στην πλευρά της εφαρμογής όπου, σε κάθε ενέργεια μέσω της οποίας γίνεται αλλαγή των δεδομένων, αντί να αντικαθίσταται εξολοκλήρου η σελίδα στον περιηγητή, συμψηφίζεται η ψηφιακή δομή με την πραγματική, υπολογίζοντας τις διαφορές τους και η ενημέρωση της σελίδας επιτυγχάνεται αντικαθιστώντας μόνο τα απαραίτητα στοιχεία.

Η δοκιμαστική εφαρμογή υλοποιήθηκε στην εκδοχή React 16 (16.13.1) και σε λειτουργικό σύστημα Windows.

Κεφάλαιο Γ'

Γ'.1 Περιγραφή της Εφαρμογής

Η δοκιμαστική εφαρμογή υλοποιήθηκε για να αναδείξει την τεχνική MVC. Το πρακτικό κομμάτι των υλοποιήσεων θα περιγραφεί στις επόμενες ενότητες κάθε πλατφόρμας αναλυτικά, ενώ ο πηγαίος κώδικας είναι διαθέσιμος και στο GitHub για ανάλυση.

<https://github.com/Seiran91/GraduationProject.git>

Η γενική ιδέα της υλοποίησης είναι να φτιάξουμε μια εφαρμογή που θα διαχειρίζεται τα στοιχεία φοιτητών για την πτυχιακή εργασία που έχει αναλάβει να κάνει ο καθένας, τα οποία απαρτίζονται από το ονοματεπώνυμο, το ηλεκτρονικό ταχυδρομείο και τον τίτλο της πτυχιακής εργασίας που έχει καταχωρηθεί. Στην αρχική οθόνη εμφανίζουμε τη λίστα με τα ονόματα των φοιτητών που είναι καταχωρημένοι στη βάση δεδομένων, ενώ με το πάτημα στο όνομα της καταχώρησης εμφανίζονται τα στοιχεία του φοιτητή. Υλοποιήθηκε η δυνατότητα εισαγωγής, επεξεργασίας και διαγραφής καταχωρήσεων τα οποία είναι διαθέσιμα εφόσον έχει γίνει αυθεντικοποίηση του χρήστη στην εφαρμογή. Για την επίδειξη των δικαιωμάτων ανά χρήστη έχουν καταχωρηθεί ο Διαχειριστής (User: admin, Password: admin) και ένας απλός λογαριασμός (User: demo, Password: demo). Εισαγωγή νέας εγγραφής ή επεξεργασία των υπαρχόντων καταχωρήσεων μπορούν να γίνουν και από τους δύο λογαριασμούς, ενώ διαγραφή καταχώρησης γίνεται μόνο από τον Διαχειριστή. Αφού έχει γίνει είσοδος στην εφαρμογή, στην αρχική οθόνη εμφανίζεται το κουμπί νέας εγγραφής, όπου μπορούμε να καταχωρήσουμε καινούργια στοιχεία στη βάση δεδομένων. Κατά την επιλογή μιας καταχώρησης εμφανίζονται τα στοιχεία του φοιτητή με τα κουμπιά επεξεργασίας και διαγραφής, αναλόγως με τον χρήστη, και εφόσον υπάρχει αυθεντικοποίηση μπορούμε να επιλέξουμε τη δυνατότητα επεξεργασίας για ενημέρωση των στοιχείων αυτών.

Γ'.1.1 Προαπαιτούμενα Υλοποίησης

Προαπαιτούμενα για την ανάπτυξη της εφαρμογής είναι η γνώση των γλωσσών προγραμματισμού HTML, CSS, JavaScript, εγκατάσταση του Node.js το οποίο περιλαμβάνει τον διαχειριστή πακέτων npm, με ένα από τα μεγαλύτερα μητρώα λογισμικού ελεύθερου κώδικα, καθώς και η χρήση ενός συντάκτη κειμένου της αρεσκείας μας για τη συγγραφή κώδικα όπως για παράδειγμα Sublime Text, Visual Studio Code ή άλλα. Ο χειρισμός των πακέτων επιτυγχάνεται μέσω της διεπαφής γραμμής εντολών η οποία είναι εγκατεστημένη από προεπιλογή στο Node.js και οι πλατφόρμες ανάπτυξης Angular και React εξαρτώνται από αυτά τα πακέτα για την υλοποίηση της εφαρμογής.

Η εγκατάσταση του Node.js επιτυγχάνεται με την λήψη του αρχείου εγκατάστασης από την επίσημη ιστοσελίδα του

<https://nodejs.org>

για το λειτουργικό σύστημα Windows, καθώς σε αυτό αναπτύχθηκε η δοκιμαστική εφαρμογή.

Γ'.1.2 RESTful backend

Το κομμάτι της εφαρμογής που ασχολείται με την αποστολή των δεδομένων και τη διαχείριση των αιτημάτων για δεδομένα εκτελείται στην πλευρά του εξυπηρετητή γι' αυτό και ονομάζεται backend ή server-side κομμάτι. Για να θεωρηθεί μια εφαρμογή RESTful πρέπει να χρησιμοποιεί συγκεκριμένα πρότυπα για τη λειτουργία της όπως HTTP methods (GET, POST, PUT, DELETE), Uniform Resource Identifiers (URIs), JavaScript Object Notation (JSON) και Extensible Markup Language (XML).

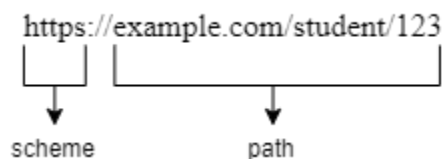
Με το πρότυπο της HTTP εκτελούμε τη μέθοδο GET για να μας αποσταλούν τα δεδομένα από τον εξυπηρετητή, η μέθοδος POST στέλνει δεδομένα στον εξυπηρετητή προς επεξεργασία, η μέθοδος PUT ενημερώνει τα δεδομένα που ήδη υπάρχουν στον εξυπηρετητή και η μέθοδος DELETE διαγράφει ένα υπάρχον στοιχείο.

HTTP	
GET	Ανάγνωση δεδομένων
POST	Εισαγωγή δεδομένων
PUT	Ενημέρωση δεδομένων
DELETE	Διαγραφή δεδομένων

Σχήμα Γ'1. 1: HTTP μέθοδοι και η σημασία τους

Καθώς τα δεδομένα είναι στον εξυπηρετητή, στις μεθόδους HTTP πρέπει να προσδιορίσουμε και την πληροφορία στην οποία αναφερόμαστε και αυτό επιτυγχάνεται με το URI. Το URI στην ουσία είναι ο σύνδεσμος (URL) που αναφέρεται στην πηγή, με την διαδρομή όπου υπάρχουν τα δεδομένα και τον αναγνωριστικό κωδικό ο οποίος είναι μοναδικός για κάθε στοιχείο.

Παρακάτω δίνεται παράδειγμα για το URI το οποίο απαρτίζεται από το πρωτόκολλο (https) που εφαρμόζεται και τη διαδρομή όπου είναι το αναγνωριστικό του στοιχείου που αιτούμαστε.



Σχήμα Γ'1. 2: Παράδειγμα URI

Γ'.1.3 Υλοποίηση backend

Το κομμάτι του εξυπηρετητή αναπτύχθηκε στη γλώσσα PHP και διαχειρίζεται τα αιτήματα GET, POST, PUT, DELETE τα οποία εφαρμόζονται στη συνδεδεμένη βάση δεδομένων MySQL.

Η υλοποίηση ξεκινά με τη δημιουργία του αρχείου “connection.php” και μέσα στο αρχείο ο πηγαίος κώδικας περιτυλίσσεται από τις ετικέτες αρχής(<?php) και τέλους(>) για παράδειγμα

```
<?php echo "Hello" ?>
```

Αρχικά συνδεόμαστε στη βάση δεδομένων με την εντολή

```
$conn = new mysqli($servername, $username, $password, $db) or  
die("Connection failed");
```

όπου οι μεταβλητές \$servername, \$username, \$password, \$db είναι τα στοιχεία της βάσης δεδομένων στην οποία θα συνδεθούμε.

Έπειτα υλοποιούμε τα μέρη που χειρίζονται τα αιτήματα HTTP.

```
$data = json_decode(file_get_contents('php://input'));

if($_SERVER['REQUEST_METHOD'] == "GET")
{
    $uri_path = parse_url($_SERVER['REQUEST_URI'], PHP_URL_PATH);
    $uri_segments = explode('/', $uri_path);
    if(count($uri_segments) >= 4){
        $studentId = $uri_segments[3];

        $sql = "SELECT * FROM mytable WHERE id LIKE '$studentId'";
        $result = $conn->query($sql);
        if($result->num_rows > 0) {
            $ress1 = $result->fetch_assoc();
            echo json_encode($ress1);
        } else {
            http_response_code(404);
            echo "Error to found content!";
        }
    } else {
        $sql = "SELECT Name, id FROM mytable";
        $result = $conn->query($sql);
        if ($result->num_rows > 0) {
            // output data of each row
            while($row = $result->fetch_assoc()) {
                $ress2[] = $row;
            }
        } else {
            echo "0 results";
        }
        echo json_encode($ress2);
    }
}
```

Σχήμα Γ'1. 3: Μέθοδος GET

```

if($_SERVER['REQUEST_METHOD'] == "POST"){

    $name = $data->{'Name'};
    $title = $data->{'Title'};
    $email = $data->{'Email'};
    $sql = "INSERT INTO mytable (Name, Title, Email) VALUES ('$name', '$title', '$email')";
    if ($conn->query($sql) === TRUE) {
        $last_id = $conn->insert_id;
        echo $last_id;
    } else {
        echo "Error: " . $sql . "<br>" . $conn->error;
    }
}

```

Σχήμα Γ'1. 4: Μέθοδος POST

```

if ($_SERVER['REQUEST_METHOD'] == "PUT"){

    $uri_path = parse_url($_SERVER['REQUEST_URI'], PHP_URL_PATH);
    $uri_segments = explode('/', $uri_path);
    $id = $uri_segments[3];

    $name = $data->{'Name'};
    $title = $data->{'Title'};
    $email = $data->{'Email'};
    $sql = "UPDATE mytable SET Name = '$name',
                                Title = '$title',
                                Email = '$email'
                                WHERE id = '$id'";

    if ($conn->query($sql)=== TRUE){
        echo $id;
    }else {
        echo "Error: " . $sql . "<br>" . $conn->error;
    }
}

```

Σχήμα Γ'1. 5: Μέθοδος PUT

```

if ($_SERVER['REQUEST_METHOD'] == "DELETE"){

    $uri_path = parse_url($_SERVER['REQUEST_URI'], PHP_URL_PATH);
    $uri_segments = explode('/', $uri_path);

    $id = $uri_segments[3];
    $sql = "DELETE FROM mytable WHERE id = '$id'";
    if ($conn->query($sql)=== TRUE){
        echo $id;
    }else {
        echo "Error: " . $sql . "<br>" . $conn->error;
    }
}

```

Σχήμα Γ'1. 6: Μέθοδος DELETE

Τα δεδομένα που αποστέλλονται από τη δοκιμαστική εφαρμογή εκχωρούνται στην μεταβλητή “\$data”, το URI που αιτήθηκε εκχωρείται στην μεταβλητή “\$uri_path” ως συμβολοσειρά με τη μέθοδο της PHP “parse_url” από τον πίνακα πληροφοριών “\$_SERVER” που παρέχεται από τον εξυπηρετητή.

Με τη μέθοδο “explode” διαιρούμε το “uri_path” σε μικρότερες συμβολοσειρές με σημείο τομής την παράμετρο “/” και τις εκχωρούμε στη μεταβλητή “uri_segments” ως πίνακα έτσι ώστε, ανάλογα με τα δεδομένα που βρίσκονται στο URI, να αναφερόμαστε σε συγκεκριμένα δεδομένα.

Η εφαρμογή κατά την εκκίνηση στέλνει αίτημα ζητώντας δεδομένα από τον εξυπηρετητή με τη μέθοδο GET με το URI:

http://localhost:4200/students

Ο εξυπηρετητής μόλις λάβει το αίτημα θα εκτελέσει τον κώδικα στο Σχήμα Γ'1. 3 όπου διαιρείται το URI σε τρεις συμβολοσειρές(http:, localhost:4200, students), καταχωρείται σε κελιά του πίνακα και στη συνέχεια εκτελείται ο κώδικας όταν δεν ισχύει η συνθήκη “If”, ο οποίος επιστρέφει όλα τα ονόματα από τη βάση δεδομένων.

Όταν η εφαρμογή θέλει να απεικονίσει τα στοιχεία ενός φοιτητή στέλνει το URI:

http://localhost:4200/student/5

σε αυτήν την περίπτωση το URI πληροί τη συνθήκη και επιστρέφονται τα στοιχεία του φοιτητή με το αναγνωριστικό “id = 5”.

Γ'.2 Υλοποίηση SPA με Angular

Γ'.2.1 Εγκατάσταση

Το Angular είναι πλατφόρμα ανάπτυξης διαδικτυακών εφαρμογών και αναπτύχθηκε στη γλώσσα προγραμματισμού TypeScript, η οποία ουσιαστικά αναλύει τη σύνταξη κώδικα με υπόβαθρο την γλώσσα προγραμματισμού JavaScript με περαιτέρω λειτουργικότητες, διευκολύνοντας τη σύνταξη, τον εντοπισμό ή την επίλυση των σφαλμάτων κατά τη συγγραφή κώδικα αλλά το σημαντικότερο που πρέπει να επισημάνουμε είναι, ότι παρέχει τη δυνατότητα ορισμού τύπου μεταβλητών ή κλάσεων για να αποφευχθούν τυχόν συγχύσεις κατά τη χρήση ή επεξεργασία τους στον κώδικα καθώς προωθεί μεταδεδομένα που καθορίζουν τον τρόπο επεξεργασίας και χρήσης του στοιχείου κατά την εκτέλεση της εφαρμογής.

Εφόσον έχει γίνει η εγκατάσταση του Node.js όπως αναφέρεται στα προαπαιτούμενα, εγκαθιστούμε μέσω της γραμμής εντολών τη γραμμή εντολών του Angular, η οποία μας επιτρέπει να εκτελούμε εντολές που θα μας διευκολύνουν στην ανάπτυξη της εφαρμογής. Όλες οι εντολές εκτελούνται μέσω της γραμμής εντολών ενώ ως περιβάλλον σύνταξης επιλέξαμε το Visual Studio Code και στις παρακάτω ενότητες οι εικόνες έχουν τη διεπαφή του συντάκτη αυτού.

Η εγκατάσταση της γραμμής εντολών Angular επιτυγχάνεται πληκτρολογώντας την εντολή

```
npm install -g @angular/cli
```

στη γραμμή εντολών.

Επόμενο βήμα είναι η δημιουργία του περιβάλλοντος εργασίας και των αρχείων εκκίνησης πληκτρολογώντας την εντολή

```
ng new my-app-name
```

όπου στο

```
my-app-name
```

ορίζουμε το όνομα της εφαρμογής.

Αφού έχει δημιουργηθεί η εφαρμογή πλοηγούμαστε στη διαδρομή για την τοποθεσία της εφαρμογής στην γραμμή εντολών πληκτρολογώντας την εντολή

```
cd my-app-name
```

Τέλος με την εντολή

```
ng serve
```

γίνεται εκκίνηση του εξυπηρετητή ανάπτυξης που θα εμφανίσει την αρχική διεπαφή της εφαρμογής που έχουμε δημιουργήσει στη σελίδα του περιηγητή μας πληκτρολογώντας τον σύνδεσμο “localhost:4200”.

Εναλλακτικός τρόπος εκτέλεσης εντολών χωρίς τη γραμμή εντολών Angular είναι:

```
npm run ng command
```

Γ'.2.2 Συστατικά εφαρμογής

Η υλοποίηση της εφαρμογής προϋποθέτει την κατανόηση του τρόπου λειτουργίας πίσω από την βασική ιδέα του Angular η οποία, χωρίζει τη λογική του σχεδιασμού σε τμήματα:

- **Components**
- **Templates**
- **Directives**
- **Dependency Injection**

Η λογική της σχεδίασης και η διαχείριση των περιεχομένων μας δείχνει ήδη ότι εφαρμόζεται η τεχνική MVC όπου κάθε τμήμα έχει δικό του ρόλο τόσο κατά τη διάρκεια της υλοποίησης όσο και στη λειτουργία του.

Γ'.2.2.1 Components

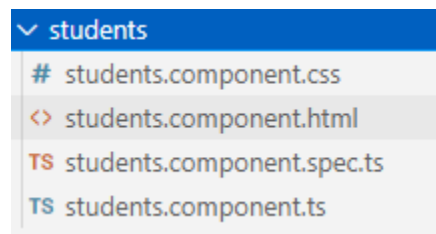
Τα Components είναι ο βασικός πυλώνας των εφαρμογών Angular καθώς περιέχουν τα συστατικά που αναδεικνύουν την τεχνική MVC.

Για τη δημιουργία ενός Component εκτελούμε την εντολή

ng generate component students

όπου στη συνέχεια δημιουργείται ο φάκελος με την ονομασία “students” με τα ακόλουθα από προεπιλογή περιεχόμενα:

- Αρχείο CSS
- Αρχείο HTML για την δομή
- Αρχείο TypeScript δομή προδιαγραφών πηγαίου κώδικα
- Αρχείο TypeScript πηγαίου κώδικα



Σχήμα Γ'2. 1: Προβολή αρχείων στο VSCode

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-students',
  templateUrl: './students.component.html',
  styleUrls: ['./students.component.css']
})

export class StudentsComponent implements OnInit {

  constructor(){

  }

  ngOnInit(){

  }
}
```

Σχήμα Γ'2. 2: Προβολή του Component students από το αρχείο 'students.component.ts'

Όπως στο Σχήμα Γ'2. 2 κάθε Component αποτελείται από:

- **@Component:** Ετικέτα που δηλώνει την κλάση StudentComponent ως τύπο Component παρέχοντας μεταδεδομένα για τη διαχείριση που πρέπει να έχει από το framework.
- **selector:** Είναι η ονομασία και συγχρόνως η ταυτότητα της ετικέτας που ορίζει τη θέση στην οποία θα δημιουργηθεί το Component στη δομή του DOM και θα προβληθεί εντός της ετικέτας αυτής το “template” του Component.
-

```
▼<app-students _ngghost-udn-c45>
  ►<div _ngcontent-udn-c45 class="studentList">...</div>
</app-students>
```

Σχήμα Γ'2. 3: selector in DOM

- **templateUrl:** Λαμβάνει ως μεταβλητή τη διαδρομή του αρχείου που περιέχει τη δομή “template” των δεδομένων που θέλουμε να εμφανίσουμε.
- **styleUrls:** Λαμβάνει ως μεταβλητή τη διαδρομή του αρχείου που περιέχει τον κώδικα για τη μορφοποίηση των δεδομένων που εμφανίζουμε στο “template”.

Η δημιουργία των Components μπορεί να γίνει και χωρίς την εκτέλεση κώδικα καθώς μπορούμε να τα δημιουργήσουμε και οι ίδιοι. Φυσικά, η χρήση της γραμμής εντολών είναι προτιμότερη καθώς εξοικονομούμε χρόνο σύνταξης και η δήλωση των Components στο συγκεντρωτικό αρχείο `app.module.ts`, όπου πρέπει να αναφερθούν τα συστατικά της εφαρμογής για την εκτέλεσή τους, γίνεται αυτόματα.

Τον ρόλο του Controller αναλαμβάνει η κλάση που δημιουργείται κατά την ονομασία του Component, στην προκειμένη περίπτωση είναι η κλάση “StudentsComponent” που απεικονίζεται στο Σχήμα Γ’2. 2. Ο πηγαίος κώδικας εντός της κλάσης καθορίζει την λειτουργικότητα και τη διαχείριση των δεδομένων του στοιχείου αυτού, το οποίο στη συνέχεια θα αναπαριστά τη λίστα με τις καταχωρήσεις των φοιτητών, με τα δεδομένα που λαμβάνουμε από το Model κομμάτι της εφαρμογής.

Η μέθοδος `“ngOnInit()”` ονομάζεται “Lifecycle hook method” και μας δίνει τη δυνατότητα να εκτελέσουμε πηγαίο κώδικα κατά την αρχικοποίηση του Component.

“Lifecycle hook methods” είναι διεπαφές που παρέχονται από το Angular, χρησιμοποιούνται για την εκτέλεση λειτουργιών εντός της κλάσης καθώς, έχουν την ιδιότητα να πυροδοτούνται κατά την δημιουργία, εκτέλεση, επεξεργασία και διαγραφή του Component.

Παραδείγματα των “hook methods” και η ακολουθία εκτέλεσης – πυροδότησης:

- **“ngOnChanges”**: Κατά την αλλαγή τιμής σε διασυνδεδεμένη μεταβλητή
- **“ngOnInit”**: Μετά την εκτέλεση του `ngOnChanges`
- **“ngDoCheck”**: Κατά την ανίχνευση αλλαγών από τον προγραμματιστή
- **“ngAfterContentInit”**: Αφού αρχικοποιηθεί το περιεχόμενο στο Component
- **“ngAfterContentChecked”**: Μετά από κάθε έλεγχο περιεχομένου του Component
- **“ngAfterViewInit”**: Αφού αρχικοποιηθεί η διεπαφή
- **“ngAfterViewChecked”**: Μετά από κάθε έλεγχο της διεπαφής
- **“ngOnDestroy”**: Πριν την αφαίρεση(καταστροφή) στοιχείου από το έγγραφο(DOM)

Ο πηγαίος κώδικας στο περιεχόμενο του κάθε Component εκτελείται μόνο όταν είναι ενεργό το συγκεκριμένο Component στο κυρίως σώμα της διεπαφής (DOM) και σε καμία άλλη περίπτωση.

Η εφαρμογή απαρτίζεται από τα παρακάτω Components και ρόλους:

- students – Αρχική σελίδα, λίστα με τις καταχωρήσεις φοιτητών
- student-details – Περιεχόμενα με τα στοιχεία φοιτητή
- login-form – Φόρμα αυθεντικοποίησης χρήστη
- register – Φόρμα νέας καταχώρησης
- pagenotfound – Σελίδα με κωδικό 404 μη εύρεσης περιεχομένου

StudentsComponent

```
export class StudentsComponent implements OnInit {
  searchText: string = "";
  students: Student[];

  constructor(public connectionService: ConnectionService) {
  }
  ngOnInit(): void {
    this.getList();
  }
  getList(): void {
    this.students = this.connectionService.students;
  }
}
```

Σχήμα Γ'2. 4: Student Component

Η κλάση StudentsComponent είναι υπεύθυνη για τη λήψη και καταχώρηση των δεδομένων για τη λίστα των φοιτητών που θα προβληθούν. Η λήψη και καταχώρηση των δεδομένων γίνεται με την “hook” μέθοδο

“ngOnInit ()”

η οποία εκτελείται κατά τη δημιουργία στιγμιότυπου του Component, εν συνεχεία εκτελεί την μέθοδο

“getList ()”

καταχωρώντας τα ληφθέντα δεδομένα από το Model κομμάτι στη μεταβλητή “students” τα οποία παρεμβάλλονται στη διεπαφή. Να σημειώσουμε πως η καταχώρηση των δεδομένων στην προκείμενη περίπτωση γίνεται από μια μεταβλητή του Model τμήματος, η οποία αρχικοποιείται με τα δεδομένα των φοιτητών κατά την εκκίνηση της εφαρμογής στο κύριο αρχείο “app.module.ts”, κατόπιν αιτήματος στον εξυπηρετητή που παρέχει τα δεδομένα.

Η μεταβλητή “**searchText**” είναι το πεδίο μέσω του οποίου ο χρήστης μπορεί να κάνει αναζητήσεις ονομάτων πληκτρολογώντας και ανανεώνει το περιεχόμενο της λίστας που είναι συνδεδεμένη σε πεδίο εισόδου (<input>) με διπλή διασύνδεση στο “template”.

Η λειτουργία αρχικοποίησης στο “app.module.ts” έχει ως εξής:

Εισάγουμε το “APP_INITIALIZER” στην αρχή του κώδικα

```
“import { NgModule, APP_INITIALIZER }” from '@angular/core' ”
```

και η κλάση “ConnectionService” η οποία είναι η υπηρεσία που δημιουργήσαμε εμείς και είναι υπεύθυνη για τα αιτήματα προς τον εξυπηρετητή

```
“import {ConnectionService} from './Services/connection.service' ”
```

Στη συνέχεια εισάγουμε τον κώδικα για την εκτέλεση κατά την εκκίνηση της αίτησης των δεδομένων από τον εξυπηρετητή στο πεδίο “providers” το οποίο εάν δεν υπάρχει το δημιουργούμε εμείς

```
“providers: [{ provide : APP_INITIALIZER, useFactory :  
initFunction, deps: [ConnectionService], multi : true }]”
```

- “APP_INITIALIZER”: χρησιμοποιείται για την εκτέλεση μεθόδου κατά την εκκίνηση της εφαρμογής
- “useFactory”: Η μέθοδος που θα εκτελεστεί
- “deps”: Η πηγή από την οποία εξαρτάται η μέθοδος για την εκτέλεση λειτουργιών
- “multi”: Για την υλοποίηση περαιτέρω αρχικοποιητών στο πεδίο “providers”

Στη συνέχεια δημιουργούμε μια συνάρτηση με το όνομα “**initFunction**”, όπου σε αυτήν εκτελούμε την μέθοδο που θα αιτηθεί από τον εξυπηρετητή τα δεδομένα των φοιτητών. Η συνάρτηση καταχωρείται στο πεδίο “**useFactory**” για την εκτέλεσή της.

```

imports: [
    BrowserModule,
    FormsModule,
    ReactiveFormsModule,
    AppRoutingModule,
    HttpClientModule
],
providers: [StudentsComponent, {provide :
APP_INITIALIZER, useFactory : initFunction, deps: [
ConnectionService] , multi : true}],
bootstrap: [AppComponent]
})
export class AppModule { }

export function initFunction(config :
ConnectionService)
{
    console.log("Initializer runs...");
    return () =>config.initDataFun();
}

```

Σχήμα Γ'2. 5: Αρχικοποίηση δεδομένων στο αρχείο app.module.ts

Στον τελικό πηγαίο κώδικα στο πεδίο “providers” από προεπιλογή δημιουργήθηκε το StudentsComponent, το οποίο είναι προαιρετικό στην προκειμένη περίπτωση.

Η συνάρτηση “**initFunction()**” εκτελεί τη μέθοδο “**initDataFun()**” η οποία είναι μέλος της υπηρεσίας και κλάσης ConnectionService. Η μέθοδος αυτή ζητάει από τον εξυπηρετητή τα δεδομένα και τα αποθηκεύει σε μια τοπική μεταβλητή “students”.

StudentDetailsComponent

Στο StudentDetailsComponent οι λειτουργίες που εκτελούνται είναι οι εξής:

- Εμφάνιση στοιχείων του φοιτητή (Ον/μο, Ηλ. ταχυδρομείο, Τίτλος πτυχιακής)
- Επεξεργασία στοιχείων
- Διαγραφή φοιτητή

Επιλέγοντας έναν φοιτητή, στη λίστα απεικονίζονται τα στοιχεία του σε μια νέα μορφή όπου έχουμε την δυνατότητα επεξεργασίας εφόσον έχει προηγηθεί είσοδος(admin ή demo).

Κατά τη δημιουργία του StudentDetailsComponent εκτελείται η εντολή

ngOnInit()

η οποία ακολούθως καλεί τη μέθοδο

getStudentDetails()

για να αιτηθεί από την υπηρεσία ConnectionService τα στοιχεία του φοιτητή με το Uniform Resource Identifier(URI) που περιέχει το αναγνωριστικό του φοιτητή.

```
ngOnInit(): void {
  this.userLogged = this.studentService.User.logged;
  this.getStudentDetails();
}

getStudentDetails(): void {
  const reqStudent = this.location.path();
  this.studentService.getStudentDetails(reqStudent)
    .subscribe(
      response => {this.detailedStudent = response,
this.createForm()},
      err => {console.log(err.message);}
    );
}
```

Σχήμα Γ'2. 6: Λήψη στοιχείων από την υπηρεσία ConnectionService

Στο Σχήμα Γ'2. 6 η υπηρεσία εισάγεται με την ονομασία “studentService” και εκτελεί τη μέθοδο “getStudentDetails()” με παράμετρο το αναγνωριστικό του φοιτητή(id), καταχωρώντας στην τοπική μεταβλητή “detailedStudent” τα στοιχεία που επιστρέφονται και παρεμβάλλοντάς τα στο “template”.

Για την επεξεργασία των στοιχείων απεικονίζεται μια φόρμα με τα στοιχεία προς επεξεργασία όπου, κατά την υποβολή των ενημερωμένων στοιχείων, πυροδοτείται η εκτέλεση της μεθόδου

“updateStudent()” του Component για την ενημέρωση στοιχείων στη βάση δεδομένων αλλά και τοπικά.

```
updateStudent(){
  var studentURL = this.location.path();
  var id = this.route.snapshot.paramMap.get("id");
  var index = this.studentService.students.findIndex(function(item, i){
    return item.id == +id;
  });

  this.studentService.updateStudent(studentURL, this.editForm.value)
  .subscribe(
    response => {
      console.log("Student with id: "+response+" updated succesfully!");
      console.log(this.editForm.value);
      this.studentService.students[index].Name = this.editForm.get('Name').value;
      this.detailedStudent.Name = this.editForm.get('Name').value;
      this.detailedStudent.Email = this.editForm.get('Email').value;
      this.detailedStudent.Title = this.editForm.get('Title').value;
      this.editShowHideBoolean();
    },
    err => {console.log(err)}
  );
}
```

Σχήμα Γ'2. 7: Ενημέρωση στοιχείων φοιτητή

Στη μέθοδο “updateStudent()” κατά την υποβολή των αλλαγών ενημερώνουμε τη βάση δεδομένων, την τοπική μεταβλητή “detailedStudent” και τη λίστα των φοιτητών που έχουμε αποθηκεύσει σε τοπική μεταβλητή στην υπηρεσία ConnectionService που περιέχει το όνομα του φοιτητή. Ο λόγος που το κάνουμε αυτό είναι για να αποφύγουμε τα άσκοπα αιτήματα στον εξυπηρετητή, γι’ αυτό και στο Σχήμα Γ'2. 4 η λήψη των δεδομένων για την απεικόνιση της λίστας γίνεται μεταφέροντας τα δεδομένα των φοιτητών με μια απλή καταχώρηση από μεταβλητή σε μεταβλητή όταν έχει προηγηθεί η λήψη τους με αίτημα στον εξυπηρετητή κατά την εκκίνηση της εφαρμογής.

Η ενημέρωση του “detailedStudent” γίνεται για να είναι εμφανείς οι αλλαγές που έχουμε κάνει άμεσα στα στοιχεία καθώς δεν αιτούμαστε ξανά στον εξυπηρετητή για τα νέα δεδομένα και για να αναδείξουμε τη δυναμική ενημέρωση των στοιχείων στη διεπαφή.

Στη λίστα των φοιτητών ενημερώνουμε μόνο το όνομα στο οποίο έχουν γίνει οι αλλαγές. Ο λόγος ενημέρωσης του ονόματος είναι ότι όταν επιστρέψουμε στην αρχική λίστα, εάν δεν έχει προηγηθεί η ενημέρωση και γίνουν αλλαγές στο πεδίο με το όνομα, η λίστα θα απεικονίζει τον φοιτητή που έχει ενημερωθεί με την αρχική καταχώρηση που είχε. Για να επιτευχθεί αυτό,

βρίσκουμε τον δείκτη θέσης στον οποίο βρίσκεται το πεδίο με το όνομα που πρέπει να ενημερωθεί, καθώς η λίστα με τα ονόματα είναι ένας μονοδιάστατος πίνακας από αντικείμενα δεδομένων με πεδία τιμών: id και Name.

Οι παράμετροι που χρειάζονται για την ενημέρωση της βάσης δεδομένων είναι το URI και τα ενημερωμένα στοιχεία της φόρμας, τα οποία αποστέλλονται στη μέθοδο της υπηρεσίας για να κάνει τις απαραίτητες ενέργειες στον εξυπηρετητή. Αφού η υπηρεσία αιτηθεί ενημέρωση στοιχείων, ο εξυπηρετητής επιστρέφει μια απάντηση αποδοχής ή απόρριψης. Στην περίπτωση απόρριψης εμφανίζεται το μήνυμα του σφάλματος ενώ στην περίπτωση της αποδοχής η βάση δεδομένων έχει ενημερωθεί οπότε ενημερώνουμε και τα τοπικά στοιχεία όπως το “detailedStudent” με τα καινούργια πεδία και τη λίστα των φοιτητών, καθώς έχει προηγηθεί η εύρεση του δείκτη θέσης του φοιτητή προς ενημέρωση με το ενημερωμένο όνομα, εάν αυτό έχει τροποποιηθεί.

Στη μέθοδο διαγραφής ακολουθούνται σχεδόν τα ίδια βήματα όπως και στην ενημέρωση με την διαφορά ότι η παράμετρος που αποστέλλεται στην υπηρεσία είναι μόνο το URI με το αναγνωριστικό του φοιτητή. Κατά την αποδοχή του αιτήματος διαγραφής στοιχείων από την υπηρεσία στην βάση δεδομένων, βρίσκουμε τον δείκτη θέσης του φοιτητή στον πίνακα της λίστας, τον αφαιρούμε και στη συνέχεια ανακατευθυνόμαστε στην αρχική οθόνη, όπου απεικονίζεται η λίστα με τα ονόματα των φοιτητών.

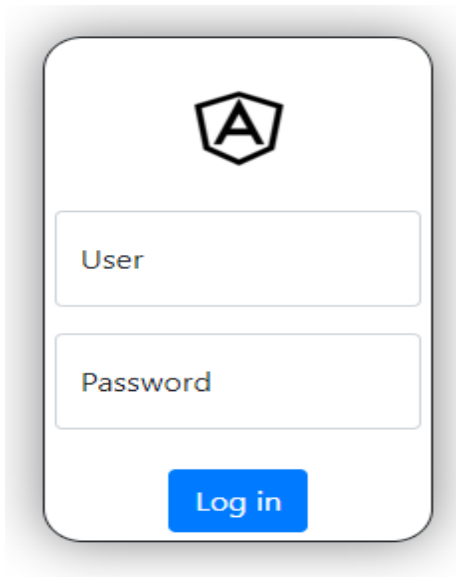
```
deleteStudent(): void {
  const reqStudent = this.location.path();
  const id = +this.route.snapshot.paramMap.get("id");
  const index = this.studentService.students.findIndex(function(item, i){
    return item.id == +id;
  });

  this.studentService.deleteStudent(reqStudent)
    .subscribe(
      response => {
        console.log("Student with id: "+response+" deleted succesfully!");
        this.studentService.students.splice(index,1);
        this.location.back();
      },
      err => {console.log(err)}
    );
}
```

Σχήμα Γ'2. 8: Διαγραφή στοιχείων φοιτητή

LoginFormComponent

Σε αυτό το Component ταυτοποιείται ο χρήστης για να μπορεί να εισάγει, επεξεργαστεί και να διαγράψει δεδομένα από τη βάση δεδομένων.



```
login() {  
  this.userService.Login(this.Loguser)  
    .subscribe( res =>{  
    if(res){  
      this.Loguser.user=null;  
      this.Loguser.password=null;  
    } else {  
      alert("Authentication failed!");  
    }  
  },  
  err =>{  
    console.log(err)  
  })  
}
```

Σχήμα Γ'2. 9: Αυθεντικοποίηση χρήστη

Η αυθεντικοποίηση πραγματοποιείται με την εισαγωγή των στοιχείων στα πεδία, έπειτα καλείται η μέθοδος “login()” που αποστέλλει τα στοιχεία στη βάση δεδομένων. Ο εξυπηρετητής ανάλογα με τα στοιχεία που έχουμε δώσει αποστέλλει την απάντηση Αληθής ή Ψευδής ή εάν υπάρξει κάποιο σφάλμα κατά τη διαδικασία αποστέλλεται ένδειξη σφάλματος. Σε περίπτωση επιτυχούς αυθεντικοποίησης επαναφέρουμε τα πεδία στην αρχική τους μορφή που είναι διπλά διασυνδεδεμένα με τα πεδία της μεταβλητής “Loguser”, σε περίπτωση μη αυθεντικοποίησης εμφανίζεται το μήνυμα “Authentication failed!” στον περιηγητή και σε περίπτωση σφάλματος τυπώνεται το σφάλμα στην κονσόλα του περιηγητή.

RegisterComponent

Αφού εισέλθουμε στην εφαρμογή ως αυθεντικοί χρήστες, στην αρχική λίστα εμφανίζεται το κουμπί “Register” απ’ όπου μπορούμε να εισάγουμε νέες εγγραφές στη βάση δεδομένων. Μόλις πατήσουμε το κουμπί εμφανίζεται μια φόρμα και αφού την συμπληρώσουμε πλήρως μπορούμε να εισάγουμε την καταχώρηση στη βάση δεδομένων. Κατά την εισαγωγή πυροδοτείται η εκτέλεση της μεθόδου “newStudent()” η οποία αποστέλλει τα δεδομένα στην υπηρεσία για να

αιτηθεί την εισαγωγή τους στη βάση δεδομένων. Εάν απορριφθεί το αίτημα εμφανίζουμε το σφάλμα στην κονσόλα του περιηγητή. Στην περίπτωση που ολοκληρώνεται η καταχώρηση στη βάση δεδομένων δημιουργούμε τοπικά την μεταβλητή “std” με τα πεδία “Name” και “id” , στη συνέχεια καταχωρούμε στο πεδίο “Name” το όνομα που συμπληρώσαμε στη φόρμα και στο πεδίο “id” το αναγνωριστικό της καταχώρησης, το οποίο επιστρέφεται από τον εξυπηρετητή. Στη συνέχεια εισάγουμε τη μεταβλητή “std” στον μονοδιάστατο πίνακα ενημερώνοντας τη λίστα των φοιτητών και στο τέλος επαναφέρουμε τη φόρμα στην αρχική της μορφή.

```
newStudent() {
  this.connectionService.addStudent(this.registerForm.value)
  .subscribe(
    data =>{
      /*The unary + operator its operand
      convert it to Number type otherwise
      we get type incompatibility error */
      const std = {Name: this.registerForm.get('Name').value, id: +data};
      //We push our registered student
      //in Students List array to update the list and view
      this.connectionService.students.push(std);
      alert("Student added succesfully with \n id: " + data + "\n Name: " + std.Name);
      this.resetForm();
    },
    err =>{
      console.log(err)
    }
  );
}
```

Σχήμα Γ'2. 10: Εισαγωγή νέας καταχώρησης

PageNotFoundComponent

Το συγκεκριμένο Component δεν περιέχει καμία λειτουργία εντός της κλάσης του και ο μόνος σκοπός που υπάρχει είναι να απεικονίζει το μήνυμα του κωδικού 404 στο “template” το οποίο σημαίνει ότι το περιεχόμενο που προσπαθούμε να προσπελάσουμε δεν υπάρχει.

Γ'.2.2.2 Templates

Το “template” αντιπροσωπεύει το View της τεχνικής MVC και προωθεί την απεικόνιση και τη διεπαφή με τον χρήστη που υλοποιούνται στο αρχείο “students.component.html” και στο Component αντίστοιχα. Στη σύνταξη του “template” ελέγχουμε τη ροή των δεδομένων από το

Component σε αυτό και το αντίθετο, για παράδειγμα η απεικόνιση της δομής και του κειμένου της μεταβλητής που δημιουργείται δυναμικά γίνεται με την παρεμβολή των στοιχείων που δημιουργούνται κατά την εκτέλεση της κλάσης όπως στο Σχήμα Γ'2. 11.

```
//file StudentsComponent.component.ts
export class StudentsComponent implements OnInit {
  Name = "Michael";
```

```
file students.component.html
<div>
  <h1>Hello {{ Name }}</h1>
</div>
```

Σχήμα Γ'2. 11: Παρεμβολή μεταβλητής στο View

Binding

Βασική λειτουργικότητα του Angular που παρέχεται στο “template” είναι η διασύνδεση (binding) των στοιχείων του πηγαίου εγγράφου (DOM) με τα περιεχόμενα της κλάσης, κάτι που δίνει τη δυνατότητα της διαχείρισης των τιμών και των λειτουργιών τους δυναμικά.

Διασύνδεση χαρακτηριστικών:

```
<button [disabled]="isUnchanged">Disabled button</button>
```

Η διασύνδεση των χαρακτηριστικών γίνεται με την εισαγωγή του σε αγκύλες και η τιμή του αλλάζει δυναμικά ανάλογα με την τιμή που έχει η μεταβλητή “isUnchanged” η οποία βρίσκεται στην κλάση που διαχειρίζεται αυτό το “template”. Διασύνδεση μπορεί να γίνει σε όλα τα χαρακτηριστικά της ετικέτας όπου μπορεί να εκχωρηθεί τιμή.

Διασύνδεση συμβάντων:

```
<button (click)="function()">Click me!</button>
```

Διασύνδεση διεπαφής και μεθόδου εισάγοντας το σε παρενθέσεις που ανταποκρίνεται στις ενέργειες του χρήστη (αριστερά) και εκτελούνται οι αλλαγές στην μέθοδο (δεξιά).

Διπλή διασύνδεση:

```
<input type="search" [(ngModel)]="searchText">
```

Στη διασύνδεση χαρακτηριστικών ή συμβάντων η ροή της πληροφορίας είναι μονόδρομη για το κάθε στοιχείο, στη διπλή διασύνδεση η πληροφορία ρέει αμφίδρομα καθώς αλλάζει η τιμή του στοιχείου ενεργοποιεί και την εκτέλεση συμβάντων σε αυτό. Για παράδειγμα στο πεδίο αναζήτησης στη δοκιμαστική εφαρμογή κάθε φορά που εισάγουμε ένα γράμμα ενεργοποιείται η μέθοδος αναζήτησης όπου η εισαγωγή είναι η αλλαγή τιμής του στοιχείου αυτού.

Pipes

“Pipes” είναι μεμονωμένες μέθοδοι που δέχονται μία παράμετρο και επιστρέφουν το αποτέλεσμα της παραμέτρου μετά τη μετατροπή. Μπορούν να εφαρμοστούν σε στοιχεία του “template” και μας επιτρέπουν τη μετατροπή των τιμών τους κατά την απεικόνιση, χωρίς την ανάγκη της δήλωσης τους καθώς παρέχονται τα απαραίτητα μεταδεδομένα στην πηγαίο κώδικα με την TypeScript επικεφαλίδα @Pipes και ενσωματώνονται απευθείας. Τέτοιου είδους μέθοδοι εφαρμόστηκαν στη δοκιμαστική εφαρμογή για την ταξινόμηση της λίστας και για το φιλτράρισμα των φοιτητών κατά την εύρεση στο πεδίο αναζήτησης.

```
// students.component.html
<li *ngFor=
  "let student of students | orderBy: 'Name' | filter: searchText">
  <span>{{student.Name}}</span>
</li>
```

Σχήμα Γ'2. 12: Χρήση των “Pipes”

Η σύνταξη και εφαρμογή των “Pipes” στο “template” γίνεται όπως στο Σχήμα Γ'2. 12 όπου η μεταβλητή “students” είναι η πηγή των δεδομένων που θα εφαρμοστεί η μετατροπή, “orderBy” είναι η μέθοδος “Pipe” και “Name” η παράμετρος με την οποία θα γίνει η μετατροπή που στην προκείμενη περίπτωση είναι αλφαβητικά.

Για την δημιουργία νέου “Pipe” εκτελούμε στη γραμμή εντολών την εντολή

```
"ng generate pipe name"
```

και έπειτα αναπτύσσουμε τη λειτουργικότητά του.

Γ'.2.2.3 Directives

Όπως έχουμε αναφέρει παραπάνω στην ενότητα Components, τα δεδομένα που θα απεικονίζονται είναι μια λίστα φοιτητών που θα αποστέλλονται από τον εξυπηρετητή, θα καταχωρούνται στον “Controller” κι έπειτα θα παρεμβάλλονται στο “View”. Επίσης στην ίδια διεπαφή αποκρύπτουμε το κουμπί της νέας καταχώρησης το οποίο είναι διαθέσιμο και το εμφανίζουμε εφόσον έχει γίνει ταυτοποίηση χρήστη. Καθώς τα δεδομένα των φοιτητών επιστρέφονται ως μονοδιάστατος πίνακας από τον εξυπηρετητή, για την απεικόνιση τους θα χρησιμοποιήσουμε τα directives “*ngIf” και “*ngFor” καθώς έχουν τη δυνατότητα να διαμορφώνουν τη δομή του κυρίως εγγράφου DOM δυναμικά προσθέτοντας, αφαιρώντας και ενημερώνοντας στοιχεία της δομής. Η απεικόνιση της λίστας επιτυγχάνεται ενσωματώνοντας το directive “*ngFor” στην ετικέτα που θέλουμε να εμφανίσουμε τα στοιχεία σε μορφή λίστας.

```
<li *ngFor="let student of students" >
  <span>{{student.Name}}</span>
</li>
```

Σχήμα Γ'2. 13: Χρήση *ngFor

Έπειτα από επιτυχή εκτέλεση του κώδικα στο Σχήμα Γ'2. 13 μπορούμε να δούμε τη δομή των στοιχείων όπως στο Σχήμα Γ'2. 14 παρακάτω, επιθεωρώντας τη λίστα που απεικονίστηκε στη οθόνη του χρήστη.

```
<ul _ngcontent-cuj-c45 class="list-
  ><li _ngcontent-cuj-c45>...</li>
  ><li _ngcontent-cuj-c45>...</li>
  ><li _ngcontent-cuj-c45>...</li>
  ><li _ngcontent-cuj-c45>...</li>
  ><li _ngcontent-cuj-c45>...</li>
  ><li _ngcontent-cuj-c45>...</li>
  ><li _ngcontent-cuj-c45>...</li>
  ><li _ngcontent-cuj-c45>...</li>
  ><li _ngcontent-cuj-c45>...</li>
  ><li _ngcontent-cuj-c45>...</li>
  ><li _ngcontent-cuj-c45>...</li>
  ><li _ngcontent-cuj-c45>...</li>
```

Σχήμα Γ'2. 14: Δομή DOM

Το directive “*ngIf” δέχεται καταστάσεις αληθούς ή ψευδούς τα οποία μπορεί να προέρχονται από μεταβλητές ή συναρτήσεις που παρέχονται από τον Controller που είναι υπεύθυνος για αυτό το στοιχείο στο οποίο έχει ενσωματωθεί. Ανάλογα με την κατάσταση που βρίσκεται απεικονίζει ή αποκρύπτει το στοιχείο διαγράφοντάς το από τη δομή του εγγράφου.

```
<div *ngIf="user.logged">
  <h1>Welcome {{ user.name }}</h1>
</div>
```

Σχήμα Γ'2. 15: Χρήση *ngIf

Στο Σχήμα Γ'2. 15 η μεταβλητή “logged” μόνο όταν είναι αληθής θα εμφανιστεί η επικεφαλίδα στην οθόνη του χρήστη.

Γ'.2.2.4 Dependency Injection (Services)

Dependency Injection είναι αρχεία πηγαίου κώδικα που θεωρούνται ως υπηρεσίες που παρέχουν δεδομένα και λειτουργικότητες συναρτήσεων στις κλάσεις των Components χωρίς να τις δημιουργούν οι ίδιες οι κλάσεις. Εισάγοντας την υπηρεσία αυτή στο πεδίο “constructor” μπορούν να εκτελεστούν μέθοδοι και ενέργειες της υπηρεσίας.

Η δημιουργία της υπηρεσίας γίνεται με την εντολή

```
“ng generate service connection”
```

όπου

```
“connection”
```

είναι η ονομασία που έχουμε δώσει και έχει την εξής δομή:

```
import { Injectable } from '@angular/core';

@Injectable({
  providedIn: 'root',
})
export class ConnectionService {
  constructor() { }
}
```

Σχήμα Γ'2. 16: Υπηρεσία ConnectionService

Σκοπός της υπηρεσίας είναι η εισαγωγή δεδομένων ή λειτουργιών στις κλάσεις των Components που εξαρτώνται από την υπηρεσία αυτή για την ομαλή λειτουργία τους. Η υπηρεσία στην εφαρμογή μας έχει τον ρόλο του Model. Η συγκεκριμένη υπηρεσία υλοποιήθηκε για να διαχειρίζεται τη σύνδεση μεταξύ εφαρμογής και εξυπηρετητή όπου επιστρέφονται τα δεδομένα προς επεξεργασία στον Controller που τα έχει αιτηθεί, εν συνεχεία αποστέλλονται στο View για απεικόνιση. Η διαχείριση των δεδομένων γίνεται με τη χρήση της κλάσης “HttpClient” η οποία συμπεριλαμβάνεται στην εγκατάσταση του Angular και χρησιμοποιείτε για εκτέλεση αιτημάτων GET, POST, PUT, DELETE όπου στην προκειμένη εφαρμογή, τα δεδομένα είναι οι καταχωρήσεις των φοιτητών που βρίσκονται στη βάση δεδομένων του εξυπηρετητή.

Η χρήση της κλάσης “HttpClient” γίνεται εφόσον την εισάγουμε στην αρχή του αρχείου της υπηρεσίας

```
import { HttpClient } from '@angular/common/http';
```

και προστεθεί στο πεδίο “constructor” της υπηρεσίας

```
constructor( private http: HttpClient ) {}
```

Η δήλωση της κλάσης σε μεταβλητή άλλης ονομασίας προκύπτει για τον προσδιορισμό και τη χρήση της κλάσης αυτής στην υπηρεσία ConnectionService για τη σύνταξη κώδικα στη συνέχεια, καθώς υλοποιούμε τα τμήματα αιτημάτων που θα πραγματοποιούνται για την λήψη, αποστολή, ενημέρωση και διαγραφή των δεδομένων στον εξυπηρετητή.

Η πρώτη κλήση στην υπηρεσία για τη λήψη δεδομένων από τον εξυπηρετητή γίνεται κατά την εκτέλεση της εφαρμογής όπου αναφερθήκαμε στο Σχήμα Γ'2. 5.

Για τη χρήση της υπηρεσίας `ComponentService` σε οποιοδήποτε τμήμα της εφαρμογής το οποίο εξαρτάται από τα δεδομένα που επιστρέφει, θα πρέπει να εισάγουμε στην αρχή του αρχείου την κλάση της υπηρεσίας και την διαδρομή του αρχείου στο οποίο βρίσκεται η κλάση

```
import {ConnectionService} from './Services/connection.service'
```

έπειτα να γίνει ο προσδιορισμός της στο πεδίο “constructor” με οποιοδήποτε όνομα που διευκολύνει τη χρήση της

```
constructor(private config: ConnectionService){}
```

Στο Σχήμα Γ'2. 5 καλούμε την μέθοδο “`initDataFun()`” η οποία βρίσκεται στην υπηρεσία `ConnectionService` για να αρχικοποιήσει τα δεδομένα που χρειάζεται το `StudentComponent` για την απεικόνιση της λίστας.

```
initDataFun(): Promise<Boolean> {  
  return new Promise<Boolean>((resolve)=>{  
    this.getStudentsList()  
    .subscribe(  
      data => { this.students = data },  
      err => { console.log("Something went wrong!\n" + err.message), resolve(false) },  
      () => {resolve(true)}  
    );  
  });  
}
```

Σχήμα Γ'2. 17: Καταχώρηση δεδομένων

Στο Σχήμα Γ'2. 17 υλοποιείται η μέθοδος “`initDataFun()`” με τις διεπαφές “`Promise`” όπου γίνεται η παραμετροποίηση της μεθόδου που θα εκτελέσει ασύγχρονες λειτουργίες. Το “`Promise`” καλείται να επιστρέψει αποτέλεσμα ολοκλήρωσης ή αποτυχίας για τη διαδικασία απόκτησης των δεδομένων.

Η μέθοδος αρχικοποίησης καλεί τη μέθοδο “`getStudentsList()`” η οποία επιστρέφει τα δεδομένα που έχει λάβει από τον εξυπηρετητή.

Ο λόγος που χρησιμοποιούμε τη μέθοδο “`subscribe()`” είναι για την πρόσβαση στα δεδομένα, καθώς αυτό που επιστρέφεται από τη μέθοδο “`getStudentList()`” είναι ένας τύπος σώματος το οποίο ονομάζεται “`Observables`” και η προσπέλαση των δεδομένων του γίνεται με αυτόν τον τρόπο.

Η “`subscribe()`” μέθοδος περιέχει τρία στάδια για τη διαχείριση των δεδομένων τα οποία αποκαλούνται “`callbacks`”

- next()
- error()
- complete()

Στην περίπτωση που το αίτημα γίνει αποδεκτό καταχωρούμε τα δεδομένα στην τοπική μεταβλητή “students”. Στην περίπτωση αποτυχίας εμφανίζουμε μήνυμα σφάλματος και ενημερώνουμε το “Promise” με αποτυχία και σταματά η συνέχεια εκτέλεσης του “subscribe()”. Τέλος στην περίπτωση ολοκλήρωσης του αιτήματος ενημερώνουμε το “Promise” με την επιτυχή ολοκλήρωση του αιτήματος.

Μέθοδος GET (Λίστα φοιτητών)

```
getStudentsList(): Observable <Student[]> {
    return this.http.get<Student[]>(this.baseUrl)
        .pipe(
            catchError((err) =>{
                return throwError(err);
            }));
}
```

Σχήμα Γ'2. 18: HTTP GET

Η διαδικασία αίτησης στον εξυπηρετητή φαίνεται στο Σχήμα Γ'2. 18 με τη μέθοδο “getStudentsList()” να εκτελεί τη μέθοδο GET της κλάσης HttpClient που προσδιορίσαμε στο “constructor” ως “http”. Η μέθοδος GET εκτελείται με παράμετρο τον σύνδεσμο από τον οποίο θα αιτηθούμε τα δεδομένα όπου στην προκειμένη περίπτωση είναι ο εξυπηρετητής που έχουμε υλοποιήσει εμείς ως RESTful backend (βλ. ενότητα Γ'.1.3 Υλοποίηση backend).

Όλες οι μέθοδοι της κλάσης HttpClient επιστρέφουν τα αποτελέσματα από τον εξυπηρετητή ως αντικείμενα “Observable” λόγω της αρχιτεκτονικής τους και ένας τρόπος για τη διαχείριση τους είναι να εισάγουμε την κλάση αυτή από τη βιβλιοθήκη όπου βρίσκεται

```
import {Observable, throwError} from 'rxjs';
```

Στη μέθοδο χρησιμοποιούμε την κλάση “Observable” για να δηλώσουμε τον τύπο των δεδομένων που επιστρέφονται. Η μέθοδος “throwError” προλαμβάνει την αποτυχία του “Observable” και εμφανίζει το μήνυμα σφάλματος.

Το “Student” και το “StudentDetails” που θα δούμε στη συνέχεια ονομάζονται “interfaces” και περιέχουν τον τύπο της δομής που πρέπει να έχει η μεταβλητή τέτοιου είδους με σκοπό την αποφυγή συντακτικών λαθών ή λανθασμένης εκχώρησης τιμών κατά την υλοποίηση και λειτουργία της εφαρμογής. Αυτό είναι προαιρετικό και υλοποιήθηκε με σκοπό δοκιμής.

```
// Interface provides type-checking
// of Student ONLY not creating of this!
// file: Student.ts
export interface Student {
  id: number;
  Name: string;
}

export interface StudentDetails {
  id: number;
  Name: string;
  Email: string;
  Title: string;
}
```

Σχήμα Γ'2. 19: Student and StudentDetails interface

Μέθοδος GET (Στοιχεία φοιτητή)

Κατά την επιλογή ενός φοιτητή από τη λίστα εκτελείται η μέθοδος “getStudentDetails()” που θα αιτηθεί δεδομένα με παραμέτρους τον σύνδεσμο του εξυπηρετητή και το αναγνωριστικό του συγκεκριμένου φοιτητή (id).

```
getStudentDetails(reqStudent: string): Observable <StudentDetails[]> {
  return this.http.get<StudentDetails[]>(`${this.baseUrl}${reqStudent}`)
    .pipe(
      catchError((err) =>{
        return throwError(err);
      })
    )
}
```

Σχήμα Γ'2. 20: HTTP GET Details

Μέθοδος POST

Εφόσον έχει προηγηθεί αυθεντικοποίηση του χρήστη στην εφαρμογή μπορούμε να εισάγουμε νέες καταχωρήσεις φοιτητών στη βάση δεδομένων. Κατά την εισαγωγή εκτελείται η μέθοδος “addStudent()” με παραμέτρους τον σύνδεσμο του εξυπηρετητή και τα στοιχεία της φόρμας που έχουμε συμπληρώσει.

```
addStudent(student: StudentDetails) {  
  return this.http.post<StudentDetails>(this.baseUrl, student)  
    .pipe(  
      catchError((err) =>{  
        return throwError(err.message);  
      })  
    )  
}
```

Σχήμα Γ'2. 21: HTTP POST

Μέθοδος PUT

Κατά την ενημέρωση στοιχείων εκτελείται η μέθοδος “updateStudent()” η οποία θα αιτηθεί για αλλαγές στη βάση δεδομένων με παραμέτρους τον σύνδεσμο του εξυπηρετητή, το αναγνωριστικό του φοιτητή και τα δεδομένα προς ενημέρωση.

```
updateStudent(studentURL: string, student: StudentDetails) {  
  return this.http.put<StudentDetails>(`${this.baseUrl}${studentURL}`, student)  
    .pipe(  
      catchError((err) =>{  
        return throwError(err.message);  
      })  
    )  
}
```

Σχήμα Γ'2. 22: HTTP PUT

Μέθοδος DELETE

Εάν ο χρήστης είναι διαχειριστής της εφαρμογής, κατά τη διαγραφή στοιχείων εκτελείται η μέθοδος “deleteStudents()”, η οποία θα αιτηθεί τη διαγραφή καταχώρησης από τη βάση δεδομένων με παράμετρο τον σύνδεσμο για τον εξυπηρετητή και το αναγνωριστικό στοιχείο του φοιτητή προς διαγραφή.

```

deleteStudent(reqStudent: string) {
  return this.http.delete(`${this.baseUrl}${reqStudent}`)
    .pipe(
      catchError((err) =>{
        return throwError(err.message);
      })
    )
}

```

Σχήμα Γ'2. 23: HTTP DELETE

Γ'2.3 Forms

Στη δοκιμαστική εφαρμογή υλοποιήθηκε η δυνατότητα εισαγωγή νέας καταχώρησης και ταυτοποίηση χρήστη, κάτι που επιτυγχάνεται με τη χρήση φόρμας για την εισαγωγή δεδομένων.

Για τη διαχείριση και επικύρωση των δεδομένων της φόρμας στο Angular υπάρχουν δύο προσεγγίσεις, reactive forms και template-driven forms.

Reactive Forms

- Η διαχείριση των μοντέλων γίνεται αποκλειστικά εντός της κλάσης.
- Η ροή των δεδομένων μεταξύ του “template” και της κλάσης είναι σύγχρονη.
- Η επικύρωση των δεδομένων επιτυγχάνεται με μεθόδους εντός της κλάσης.

Template-driven Forms

- Η διαχείριση των μοντέλων γίνεται στο “template”.
- Η ροή των δεδομένων είναι ασύγχρονη.
- Η επικύρωση των δεδομένων επιτυγχάνεται με Directives στο “template”.

Παράδειγμα υλοποίησης “Reactive” είναι οι φόρμες καταχώρησης στο RegisterComponent και η φόρμα επεξεργασίας στο StudentDetailsComponent, ενώ “Template-driven” είναι η φόρμα ταυτοποίησης του χρήστη στο LoginFormComponent.

Η χρήση των ελεγκτών φόρμας μπορεί να γίνει αφού έχουν καταχωρηθεί στο συγκεντρωτικό αρχείο λειτουργιών “app.module.ts” οι κλάσεις που εμπεριέχουν τις λειτουργίες αυτές

```

"import {FormsModule, ReactiveFormsModule} from '@angular/forms'"

```

και έχουν δηλωθεί στον πίνακα “imports” του “@NgModule”.

Reactive Forms

Για τη δημιουργία “Reactive Form” και την επικύρωση των δεδομένων εισάγουμε τις κλάσεις στο Component που θα δημιουργηθούν οι ελεγκτές

```
"import {FormBuilder, Validators} from '@angular/forms';"
```

Η κλάση “FormBuilder” είναι υπηρεσία που μας παρέχει τη δυνατότητα να δημιουργήσουμε ελεγκτές για τη διαχείριση της φόρμας και η κλάση “Validators” χρησιμοποιείται για τη διασφάλιση και επικύρωση των δεδομένων που εισάγονται από το χρήστη καθώς παρέχει βασικές λειτουργίες ελέγχου όπως την υποχρεωτική καταχώρηση τιμής στο πεδίο ελέγχου. Η υπηρεσία “FormBuilder” έχει τις μεθόδους “group()”, “control()” και “array()” οι οποίες δημιουργούν στιγμιότυπα των “FormGroup”, “FormControl” και “FormArray()” αντίστοιχα. Στην προκειμένη περίπτωση χρησιμοποιούμε μόνο τις μεθόδους “group()” και “control()”.

Η δημιουργία ελεγκτών επιτυγχάνεται με τον προσδιορισμό της κλάσης “FormBuilder” στο πεδίο “constructor”

```
"constructor(private fb: FormBuilder){}"
```

Στη συνέχεια δημιουργούμε στιγμιότυπο ενός συνόλου ελεγκτών με τις ονομασίες των μεταβλητών συνοδευόμενα με τις επικυρώσεις που θα εφαρμόζονται σε κάθε πεδίο.

```
pattern="^([a-z]+([0-9.\-_-]*[a-z0-9]+)*@[1]{a-z}+\.[1]{a-z}+)+$";
registerForm = this.fb.group({
  Name: ['', [Validators.required]],
  Email: ['', [Validators.required, Validators.pattern(this.pattern)]],
  Title: ['', [Validators.required]]
},
```

Σχήμα Γ'2. 24: Reactive FormBuilder and Validators

Στο Σχήμα Γ'2. 24 καταχωρούμε το σύνολο των ελεγκτών στη μεταβλητή “registerForm” ως μια ομάδα. Κάθε μεταβλητή δημιουργείται με τιμή που μπορούμε να αρχικοποιήσουμε εμείς (στην προκειμένη περίπτωση είναι κενή) και μπορεί να ακολουθείται με επικυρώσεις που πρέπει να πληρούνται όπως το “required” που δηλώνει την υποχρεωτικότητα τιμής και το “pattern” το οποίο έχει εισαχθεί για την ορθότητα της σύνταξης που πρέπει να έχει ένα ηλεκτρονικό

ταχυδρομείο, εφαρμόζοντας στην τιμή του πεδίου την κανονική έκφραση (Regular Expression) της μεταβλητής “pattern”.

Η σύνδεση της ομάδας ελεγκτών με το “template” επιτυγχάνεται δηλώνοντας τη συλλογή των ελεγκτών στο χαρακτηριστικό “formGroup”, το οποίο παρέχεται από το “ReactiveFormsModule” και παρακολουθεί τις τιμές της φόρμας και την κατάσταση επικύρωσης.

```
<form [formGroup]="registerForm">
```

Η σύνδεση της μεταβλητής του κάθε ελεγκτή με τα πεδία της φόρμας επιτυγχάνεται καταχωρώντας το όνομα της μεταβλητής στο χαρακτηριστικό “formControlName” εντός της ετικέτας εισόδου

```
<input type = "email" formControlName = "Email" >
```

Template-driven Forms

Κατά τη υλοποίηση “template-driven” φόρμας οι απαραίτητες λειτουργίες εισάγονται από το “FormsModule” που έχουμε δηλώσει στο συγκεντρωτικό αρχείο “app.module.ts”.

Η δημιουργία ελεγκτή φόρμας επιτυγχάνεται με τη δήλωση μεταβλητής με το χαρακτηριστικό “ngForm” το οποίο, είναι στιγμιοτύπο του “FormGroup” που παρέχεται από το “FormsModule” και ελέγχει τις τιμές και την επικύρωση της φόρμας

```
<form #submitForm="ngForm">
```

η σύνδεση της μεταβλητής με τον ελεγκτή εισόδου στη φόρμα επιτυγχάνεται με τη διπλή διασύνδεση στο χαρακτηριστικό. Καθώς ο χρήστης πληκτρολογεί, ενημερώνεται η μεταβλητή και η διεπαφή.

```
[ (ngModel) ]="Loguser.user"
```

Το “ngModel” δημιουργεί στιγμιοτύπο του “FormControl” και το συνδέει με το στιγμιοτύπο “FormGroup” που προηγήθηκε.

Στο LoginFormComponent υπάρχουν δύο πεδία προς συμπλήρωση για την είσοδο, το όνομα χρήστη και ο κωδικός χρήστη. Στα πεδία εφαρμόζεται το χαρακτηριστικό “required” της HTML5 που δηλώνει την υποχρεωτικότητα των πεδίων και στο κουμπί καταχώρησης έχουμε το

χαρακτηριστικό “disabled” της HTML διασυνδεδεμένο με την κατάσταση της φόρμας. Ο ελεγκτής της φόρμας, ανάλογα με την επικύρωση που εφαρμόζεται στα πεδία, επιστρέφει μια κατάσταση αληθούς ή ψευδούς, την οποία χρησιμοποιούμε για τη διαθεσιμότητα στο κουμπί καταχώρησης της φόρμας. Στην προκείμενη περίπτωση όταν τα πεδία ονόματος και κωδικού έχουν συμπληρωθεί και τα δύο, μόνο τότε είναι διαθέσιμο το κουμπί εισαγωγής.

Γ'2.4 Routing

Καθώς η ανάπτυξη της εφαρμογής υλοποιήθηκε με έμφαση στο SPA, η πλοήγηση στα Components και η απεικόνιση των περιεχομένων τους γίνεται με τη μέθοδο του routing. Δημιουργούμε ένα νέο “module” με την εντολή που θα δημιουργήσει το αρχείο “app-routing.module.ts”.

ng generate module app-routing

Εισάγουμε μέσα στο αρχείο την κλάση που έχει τις λειτουργίες της πλοήγησης:

```
import {Routes, RouterModule} from '@angular/router'
```

Θα πρέπει στη συνέχεια να εισαχθεί η κλάση με τις λειτουργίες στο αρχείο “app.module.ts”:

```
import { AppRoutingModule } from './Router/app-routing.module';
```

και να δηλωθεί το όνομα της κλάσης αυτής στον πίνακα “imports[]” του πεδίου @NgModule για την ορθή λειτουργία του.

Η υλοποίηση του “routing” μας επιτρέπει την πλοήγηση στο “template” του συγκεκριμένου Component με βάση το URL. Ακολούθως εισάγουμε και τα Components που θα απεικονίσουμε.

```

const routes: Routes = [
  { path: '', redirectTo: '/students', pathMatch: 'full' },
  { path: 'students', component: StudentsComponent, pathMatch: 'full'},
  { path: 'student/:id', component: StudentDetailsComponent, pathMatch: 'full'},
  { path: 'register', component: RegisterComponent, pathMatch: 'full' },
  { path: 'login', component: LoginFormComponent, pathMatch: 'full' },
  { path: '**', component: PagenotfoundComponent }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }

```

Σχήμα Γ'2. 25: Routing

Στο Σχήμα Γ'2. 25 απεικονίζεται ο πλήρης κώδικας με τις λειτουργίες της πλοήγησης και τα Components τα οποία εκτελούνται ανάλογα με τον σύνδεσμο που έχουμε προσδιορίσει.

Η πλοήγηση και η αλλαγή του συνδέσμου επιτυγχάνεται με την αλληλεπίδραση του χρήστη με συνδέσμους που έχουμε ορίσει εμείς, προσθέτοντας στα στοιχεία του “template” το χαρακτηριστικό “routerLink” και εκχωρώντας τον σύνδεσμο που θα αναπαριστά το Component που προσδιορίσαμε.

<button routerLink="/register" > Register </button>

Όλα τα περιεχόμενα των Components ορίζονται να απεικονιστούν στο κυρίως σώμα εγγράφου DOM με την ταύτιση συνδέσμου. Ο ορισμός της τοποθεσίας απεικόνισης στο “template” επιτυγχάνεται με την τοποθέτηση του “selector” που συμπληρώνει το περιεχόμενό του δυναμικά ανάλογα με τα δεδομένα του Component που ταυτίζεται.

<router-outlet></router-outlet>

Η ετικέτα “router-outlet” παρέχεται από το “RouterModule” και σκοπός της είναι να ορίζει σε ποιο σημείο της εφαρμογής μας θα εμφανίζονται τα δεδομένα μας. Στη δοκιμαστική εφαρμογή ο ορισμός του “selector” γίνεται στην αρχική οθόνη της εφαρμογής που είναι το αρχείο “app.component.html”

Γ.3 Υλοποίηση SPA με React

Γ.3.1 Εγκατάσταση

Εφόσον έχουμε εγκαταστήσει το Node.js όπως αναφέρεται στα προαπαιτούμενα εκτελούμε στη γραμμή εντολών την εντολή

```
npm create-react-app my-app-name
```

δημιουργώντας μια δομή σκελετού για το περιβάλλον ανάπτυξης στο φάκελο που έχουμε πλοηγηθεί στη γραμμή εντολών με την ονομασία που ορίζουμε στο πεδίο “my-app-name” της παραπάνω εντολής. Στη συνέχεια πλοηγούμαστε στον φάκελο με την εντολή

```
cd my-app-name
```

και εκτελώντας την εντολή

```
npm start
```

γίνεται εκκίνηση της εφαρμογής.

Γ.3.2 JSX

Η JSX είναι μια προσθήκη σύνταξης στη JavaScript που χρησιμοποιείται για τη δημιουργία διεπαφών με γραφή πηγαίου κώδικα που προσεγγίζει το “template” της HTML αλλά στην ουσία αναγνωρίζεται και εκτελείται ως πηγαίος κώδικας Javascript. Η χρήση της JSX είναι προαιρετική: χρησιμοποιείται με σκοπό τη διευκόλυνση ανάπτυξης διεπαφών, συνδέοντας την παρεμβολή δεδομένων και την εκτέλεση λειτουργιών σε στοιχεία του React τα οποία θα δημιουργούνται στο σώμα εγγράφου DOM και θα απεικονίζονται στον χρήστη.

Η χρήση της JSX μειώνει σε μεγάλο βαθμό τον χρόνο ανάπτυξης της εφαρμογής και τη δημιουργία στοιχείων προς απεικόνιση κατά τη σύνταξη κώδικα.

Δημιουργία στοιχείου React μέσω JSX :

```
const element = ( <h1 className = "header" > Welcome </h1> ) ;
```

Δημιουργία στοιχείου React χωρίς JSX :

```
const element = React.createElement('h1', {className: 'header'},  
'Welcome');
```

Γ'3.3 Συστατικά εφαρμογής

Το React είναι μια βιβλιοθήκη που έχει ως βασική ιδέα τη δημιουργία αυτόνομων διεπαφών για εφαρμογές μιας σελίδας (SPA) με τη χρήση των Components. Αυτό καθιστά το React εξαρτώμενο από βιβλιοθήκες τρίτων, καθώς δεν περιέχει λειτουργίες υποβολής αιτημάτων HTTP σε εξυπηρετητές ούτε και λειτουργίες πλοήγησης για τη διαχείριση πολλαπλών διεπαφών που ανανεώνουν το περιεχόμενο της εφαρμογής δυναμικά.

Τα τμήματα που απαρτίζουν τις λειτουργίες στη δοκιμαστική εφαρμογή είναι:

- **Components – React**
- **HTTP Requests – Axios library**
- **React Router – React-Router-DOM library**

Τα Components είναι οι διεπαφές που περιέχουν τις λειτουργίες απεικόνισης δεδομένων και τις λειτουργίες αλληλεπίδρασης με τον χρήστη.

Η βιβλιοθήκη Axios εγκαταστάθηκε για τη διαχείριση HTTP μεθόδων. Για τις ανάγκες της εφαρμογής υλοποιήσαμε τις μεθόδους GET, POST, PUT, DELETE για τη χρήση τους από τα Component της εφαρμογής και τα αποθηκεύσαμε στο αρχείο “requestService.js”, το οποίο χρησιμοποιείται ως υπηρεσία υποβολής αιτημάτων στον εξυπηρετητή.

Η βιβλιοθήκη React Router DOM εγκαταστάθηκε για τη διαχείριση συνδέσμων και λειτουργιών πλοήγησης.

Τα βασικά Components που απαρτίζουν την εφαρμογή:

- StudentList – Αρχική σελίδα με τις καταχωρήσεις φοιτητών
- DetailsView – Απεικόνιση στοιχείων φοιτητή
- Login – Φόρμα αυθεντικοποίησης χρήστη
- Register – Φόρμα καταχώρησης νέας εγγραφής
- Page404 – Εμφάνιση κωδικού 404 για ενημέρωση μη εύρεσης σελίδας

Γ'3.3.1 Components

Τα Components είναι το βασικότερο στοιχείο στο React κατά την υλοποίηση της δοκιμαστικής εφαρμογής καθώς αναλαμβάνουν τον ρόλο του ελεγκτή (Controller) και την προώθηση της διεπαφής (View) με βάση την τεχνική MVC.

Αναλυτικότερα τα Components είναι συναρτήσεις ή κλάσεις (ES6 class) που επιστρέφουν ως αποτέλεσμα τα στοιχεία React που θα απεικονιστούν στο χρήστη.

```
"function test(){ return <h1>This is a Test</h1> }"
```

Όπως αναφέραμε στο υποκεφάλαιο Β'3 React η απεικονιζόμενη διεπαφή δεν περιέχει στατικό κώδικα HTML αλλά δημιουργείται στον πηγαίο κώδικα ως στοιχείο React μέσω JSX που λειτουργεί ως προσέγγιση σε δομή HTML.

Η απεικόνιση των στοιχείων γίνεται με τη μέθοδο "ReactDOM.render()" το οποίο είναι "module" της βιβλιοθήκης ReactDOM που εγκαταστάθηκε μαζί με το React κατά τη δημιουργία του περιβάλλοντος ανάπτυξης και παρέχει λειτουργίες για την τροποποίηση της δομής στο πηγαίο έγγραφο (DOM).

```
ReactDOM.render(  
  <React.StrictMode>  
    <App />  
  </React.StrictMode>,  
  document.getElementById('root')  
);
```

Σχήμα Γ'3. 1: Χρήση ReactDOM

Στο Σχήμα Γ'3. 1 εφαρμόζεται η απεικόνιση του Component App στο αρχείο "index.js", το οποίο έχει παραμετροποιηθεί κατά την αυτόματη δημιουργία της δομής σκελετού. Η μέθοδος "render()" εκτελείται με δύο παραμέτρους, τα React στοιχεία προς απεικόνιση και την τοποθεσία στο πηγαίο έγγραφο (DOM) που θα απεικονιστούν τα React στοιχεία. Στην προκειμένη περίπτωση της δοκιμαστικής εφαρμογής, η πρώτη παράμετρος είναι το στοιχείο "App", μια συνάρτηση που επιστρέφει React στοιχεία και η υλοποίησή της επιτυγχάνεται μέσω JSX. Η δεύτερη παράμετρος είναι η τιμή "root" του χαρακτηριστικού "id" στην ετικέτα "div", Η ετικέτα αυτή εδρεύει στο "template" της αρχικής σελίδας "index.html" στο φάκελο "public" που έχει δημιουργηθεί κατά την αρχικοποίηση της εφαρμογής.

Η χρήση της μεθόδου απεικόνισης προϋποθέτει την εισαγωγή του “React ” και “ReactDOM” στο αρχείο όπου θα γίνει η χρήση της

```
import React from 'react';  
import ReactDOM from 'react-dom';
```

η χρήση των Components που δημιουργούνται σε ξεχωριστά αρχεία εισάγονται για τη χρήση με τον ίδιο τρόπο

```
import App from './App';
```

με την προϋπόθεση ότι εξάγονται για κοινή χρήση

```
function App() { return (<div> My Code </div>); }
```

```
export default App;
```

Στο React, για την υλοποίηση των Components δημιουργούμε αρχεία με κατάληξη αρχείων “.js” για τον προσδιορισμό τους ως JavaScript αρχεία. Στη δοκιμαστική εφαρμογή, για την ανάπτυξη των Components χρησιμοποιούμε κλάσεις που απαρτίζονται από τα τμήματα :

- **State**
- **Lifecycle methods**
- **React element**

StudentList Component

Στο τμήμα “State” προσδιορίζουμε και αρχικοποιούμε τις μεταβλητές οι οποίες θα χρησιμοποιηθούν για την αποθήκευση των δεδομένων προς επεξεργασία και απεικόνιση.

```

class StudentList extends React.Component{
  constructor(props){
    super(props);
    this.state = {
      students: [],
      filteredStudents: []
    }

    this.handleSearch = this.handleSearch.bind(this);
  }
}

```

Σχήμα Γ'3. 2: Τμήμα State του StudentList Component

Στη δοκιμαστική εφαρμογή το StudentList Component παρέχει τις λειτουργίες για την απεικόνιση της λίστας των φοιτητών. Στο Σχήμα Γ'3. 2 οι φοιτητές που επιστρέφονται από τον εξυπηρετητή καταχωρούνται στις μεταβλητές “students” και “filteredStudents”. Η μεταβλητή “filteredStudents” θα απεικονίζει τους φοιτητές που αναζητά ο χρήστης στο πεδίο αναζήτησης της εφαρμογής. Η δημιουργία του “constructor” δεν είναι υποχρεωτική εάν δεν χρησιμοποιήσουμε μεταβλητές αρχικοποίησης και μεθόδους λειτουργίας εντός της κλάσης οι οποίες χρειάζονται να συνδεθούν εντός του “constructor” για την ορθή λειτουργία τους. Το “props” είναι μεταβλητή εισόδου που περιέχει δεδομένα που μεταβιβάζονται από το γονικό στοιχείο σε αυτό το Component.

“Lifecycle methods” ονομάζονται οι μέθοδοι που εκτελούνται ανάλογα με το στάδιο της κατάστασης που βρίσκεται το Component. Για τον προσδιορισμό της κατάστασης του Component υπάρχουν τρία στάδια:

- Φόρτωση
- Ενημέρωση
- Διαγραφή

Σε κάθε στάδιο υπάρχουν συγκεκριμένες μέθοδοι που παρέχονται από το React οι οποίες πυροδοτούνται και μπορούμε να τις αξιοποιήσουμε για τις δικές μας λειτουργίες.

Για τις ανάγκες της δοκιμαστικής εφαρμογής σε όλα τα Components που θα διαχειριστούμε δεδομένα έχουμε χρησιμοποιήσει “Lifecycle” μεθόδους:

- constructor() – Αρχικοποίηση δεδομένων στη φόρτωση του Component
- componentDidMount() – Ενέργειες αφού φορτώσει το Component
- render() – Αποστολή React στοιχείου προς απεικόνιση

- `setState()` – Επεξεργασία των δεδομένων και πυροδότηση `render()`

Παραδείγματα “Lifecycle” μεθόδων ανά στάδιο και ακολουθία εκτέλεσης οι οποίες υλοποιήθηκαν στη δοκιμαστική εφαρμογή:

Κατά τη φόρτωση Component:

- `constructor()`
- `render()`
- `componentDidMount()`

Κατά την επεξεργασία του Component

- `setState()`
- `render()`

```
componentDidMount(){
  if(globalData === ''){
    const data = getData();
    data.then(res => {
      updateGlobalData(res);
      this.setState({students: res, filteredStudents: res});
    })
    .catch(console.error())
  } else {
    this.setState({students: globalData, filteredStudents: globalData});
  }
}
```

Σχήμα Γ'3. 3: Lifecycle `componentDidMount()` του `StudentList Component`

Στη “Lifecycle” μέθοδο “`componentDidMount`” εκτελείται η λειτουργία για την απόκτηση των δεδομένων από τον εξυπηρετητή και ενημερώνουμε τις τοπικές μεταβλητές “`students`” και “`filteredStudents`” που θα επεξεργαστούμε για την απεικόνιση της λίστας με τους φοιτητές και την υλοποίηση της λειτουργία αναζήτησης. Για τη λήψη των δεδομένων εκτελούμε τη μέθοδο “`getData()`” της υπηρεσίας που μας επιστρέφει τα δεδομένα από τον εξυπηρετητή. Για τη χρήση των μεθόδων της υπηρεσίας κάθε μέθοδος πρέπει να εισάγεται στο αρχείο που θα χρησιμοποιηθεί.

```
import { getData } from './requestService';
```

Για την αποφυγή άσκοπων αιτήσεων προς τον εξυπηρετητή έχουμε δημιουργήσει μια “global” μεταβλητή στο αρχείο “backendUrl.js”, στην οποία αποθηκεύουμε τα δεδομένα των φοιτητών και τα χρησιμοποιούμε για την απεικόνιση της λίστας δίχως να χρειάζεται η εφαρμογή να τα αιτηθεί ξανά.

```
render(){
  return (
    <div className="half-width">
      <NavLinks />
      <ul className="list-group">
        <input type="text" onChange={(event) =>this.handleSearch(event)} placeholder="Search..." />
        { this.state.filteredStudents.map( list =>
          <li key={list.id} className="list-group-item list-group-item-action">
            <Link to={`StudentDetails/${list.id}`}>
              <span >
                {list.Name}
              </span>
            </Link>
          </li>
        ) }
      </ul>
    </div>
  );
}
```

Σχήμα Γ'3. 4: Lifecycle render() του StudentList Component

Το τμήμα “React Element” αποτελεί το βασικότερο στοιχείο των Components καθώς περιέχει το React στοιχείο, το οποίο υλοποιείται εντός της “Lifecycle” μεθόδου “render()” και αναπτύσσεται η διεπαφή της Component που θα απεικονίζει τα δεδομένα των φοιτητών και θα αλληλεπιδρά με τον χρήστη.

Στο Σχήμα Γ'3. 4 υλοποιούμε τη διεπαφή που περιέχει το πεδίο αναζήτησης και τη λίστα των φοιτητών. Η διαχείριση και η χρήση συμβάντων ή χαρακτηριστικών στο React στοιχείο γίνεται σε σύνταξη “camelCase” και αγκύλες όπως:

HTML

```
<div class="list-group" onchange="function()"></div>
```

React

```
<div className="list-group" onChange={function()}></div>
```

Επίσης η παρεμβολή δεδομένων από μεταβλητές επιτυγχάνεται με αγκύλες:

```
<div> { Name } </div>
```

Κατά την αίτηση λήψης των καταχωρήσεων ο εξυπηρετητής επιστρέφει έναν μονοδιάστατο πίνακα με περιεχόμενο τα δεδομένα των φοιτητών σε μορφή JSON. Για την απεικόνιση των δεδομένων του πίνακα κάθε στοιχείο πρέπει να μετατραπεί σε στοιχείο React. Εφαρμόζουμε τη μέθοδο “map()” στη μεταβλητή με τα δεδομένα για την προσπέλαση των στοιχείων στον πίνακα, η οποία στη συνέχεια επιστρέφει έναν μονοδιάστατο πίνακα με τις μετατροπές που έχουν εφαρμοστεί σε κάθε στοιχείο. Κάθε στοιχείο μετατρέπεται μέσω JSX σε React στοιχείο που αντιπροσωπεύει ένα πεδίο της λίστας με αναγνωριστικό “key” το “id” του φοιτητή, με τιμή πεδίου το όνομα του φοιτητή και το σύνδεσμο πλοήγησης για την εμφάνιση των στοιχείων του. Το τελευταίο επιτυγχάνεται με τη μέθοδο “Link” της βιβλιοθήκης “React Router DOM” που θα αναλύσουμε σε επόμενη ενότητα.

Ως ένα σημαντικό κριτήριο για την ορθή λειτουργία της λίστας, το React απαιτεί την καταχώρηση αναγνωριστικού σε κάθε στοιχείο της. Αυτό επιτυγχάνεται καταχωρώντας το ειδικό χαρακτηριστικό “key” σε κάθε στοιχείο της λίστας. Τα χαρακτηριστικά “key” βοηθούν το React στον προσδιορισμό των στοιχείων που προστίθενται, επεξεργάζονται ή διαγράφονται από τη λίστα. Οι τιμές που καταχωρούνται στο χαρακτηριστικό πρέπει να είναι μοναδικές εντός της λίστας. Στην προκειμένη περίπτωση καταχωρούμε το αναγνωριστικό “id” του φοιτητή του οποίου διασφαλίζεται από τον εξυπηρετητή η μοναδικότητα στη βάση δεδομένων.

DetailsView Component

Στο “DetailsView” Component απεικονίζουμε τα στοιχεία του φοιτητή που επιλέξαμε από τη λίστα των καταχωρήσεων. Εκεί έχουμε τη δυνατότητα επεξεργασίας ή διαγραφής εφόσον έχει προηγηθεί είσοδος πιστοποιημένου χρήστη (admin ή demo). Οι λειτουργίες που μας παρέχει το “DetailsView” είναι οι εξής:

- Εμφάνιση στοιχείων φοιτητή (Ον/μο, ηλ.ταχυδρομείο, Τίτλο πτυχιακής)
- Επεξεργασία στοιχείων
- Διαγραφή φοιτητή

Αφού δημιουργηθεί το Component εκτελείται η μέθοδος “componentDidMount()” που θα αιτηθεί από την υπηρεσία τα στοιχεία του φοιτητή.

```

componentDidMount(){

    const url = this.props.match.url;
    getDetails(url).then(
        res => {
            this.setState({details: res});
        })
        .catch(error => {
            console.log(error);
        });
}

```

Σχήμα Γ'3. 5: Λήψη στοιχείων απο την υπηρεσία

Το αίτημα λήψης επιτυγχάνεται μέσω της μεθόδου “getDetails()” στην υπηρεσία αιτημάτων. Η μέθοδος “getDetails()” λειτουργεί με παράμετρο το URI του φοιτητή, το οποίο περιέχει το αναγνωριστικό “id” του φοιτητή στη βάση δεδομένων. Το URI του φοιτητή εκχωρείται από τη μεταβλητή εισόδου “props” που περιέχει την πληροφορία αυτή από το γονικό στοιχείο Router, στο οποίο θα αναφερθούμε στη συνέχεια. Αφού ολοκληρωθεί το αίτημα στον εξυπηρετητή καταχωρούμε τα στοιχεία στη μεταβλητή “details” με τη μέθοδο “setState()”, η οποία θα πυροδοτήσει την εκτέλεση της μεθόδου για την απεικόνιση των στοιχείων στον χρήστη.

Στην επεξεργασία των στοιχείων απεικονίζουμε τη φόρμα με τα στοιχεία του φοιτητή προς επεξεργασία όπου κατά την υποβολή καλείται η μέθοδος “updateDetails()” της υπηρεσίας με παραμέτρους το URI του φοιτητή και τα ενημερωμένα στοιχεία.

```

updateDetails(url, student)
    .then(() => {
        this.setState({details: {Name: name, Email: email, Title: title} });
        const index = globalData.findIndex(function(data, i){
            return data.id === id
        });
        var a = globalData;
        a[index].Name = this.state.details.Name;
        const temp = Object.assign(globalData, a);
        updateGlobalData(temp);
        this.showForm();
    })
    .catch(error => {
        console.log(error);
    });

```

Σχήμα Γ'3. 6: Ενημέρωση στοιχείων φοιτητή

Με την ολοκλήρωση του αιτήματος ενημερώνουμε τη μεταβλητή με τα στοιχεία του φοιτητή τοπικά στο Component με τη μέθοδο “setState()”, η οποία πυροδοτεί την εκτέλεση της μεθόδου “render()” και απεικονίζονται τα ενημερωμένα στοιχεία. Έπειτα ενημερώνουμε τη “global” μεταβλητή που απεικονίζει τη λίστα η οποία περιέχει το όνομα του συγκεκριμένου φοιτητή. Η εύρεση στοιχείων του φοιτητή στον πίνακα της “global” μεταβλητής επιτυγχάνεται με την εύρεση του δείκτη θέσης στον πίνακα και ενημέρωση του πεδίου ονόματος στο οποίο μπορεί να έχει γίνει επεξεργασία.

```
Delete(){
  const url = this.props.match.url;
  const id = this.props.match.params.id;
  deleteData(url)
    .then(() => {
      const index = globalData.findIndex(function(data, i){
        return data.id === id
      });
      var a = globalData;
      a.splice(index, 1);
      const temp = Object.assign(globalData, a);
      updateGlobalData(temp);

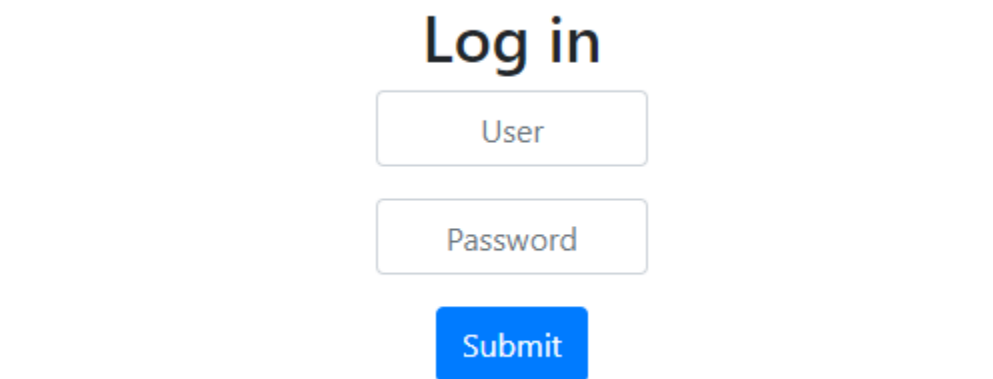
      this.props.history.push('/StudentList');
      console.log('The student deleted!');
    })
    .catch(error=>{
      console.log(error);
    });
}
```

Σχήμα Γ'3. 7: Διαγραφή φοιτητή

Κατά τη διαγραφή του φοιτητή εκτελείται η μέθοδος “Delete()” που καλεί από την υπηρεσία αιτημάτων τη μέθοδο διαγραφής “deleteData()” με παράμετρο το URI που περιέχει το αναγνωριστικό του φοιτητή. Αφού ολοκληρωθεί το αίτημα αποτελεσματικά, εκχωρούμε στη μεταβλητή “id” το αναγνωριστικό “id” του φοιτητή που θα χρησιμοποιήσουμε για την εύρεση του δείκτη θέσης στον πίνακα της “global” μεταβλητής, στη συνέχεια αφαιρούμε τα στοιχεία του φοιτητή από τον πίνακα ώστε όταν θα γίνει απεικόνιση της λίστας, αυτή να είναι ενημερωμένη.

Login Component

Το Login Component έχει ως λειτουργία την ταυτοποίηση χρήστη στην εφαρμογή για να μπορούμε να εισάγουμε, να επεξεργαστούμε και να διαγράψουμε δεδομένα από τη βάση δεδομένων.



The image shows a simple login interface. At the top, the text 'Log in' is displayed in a large, bold, black font. Below this, there are two rectangular input fields with rounded corners. The first field is labeled 'User' in a light gray font. The second field is labeled 'Password' in a light gray font. Below these two fields is a solid blue rectangular button with the word 'Submit' written in white text.

```
authReq(e){
    e.preventDefault();
    const user = document.getElementById('Username').value;
    const password = document.getElementById('Password').value;
    const data = {user,password};
    const authuser = {user: data.user, state: true};
    getUser(data)
        .then(response => {
            if(response){
                authentication(authuser);
                this.setState({User: user, Logged: true});
            } else {alert("Authentication failed!");}
        })
        .catch(console.error);
}
```

Σχήμα Γ'3. 8: Αυθεντικοποίηση χρήστη

Η διαδικασία αυθεντικοποίησης ξεκινά με την εισαγωγή των στοιχείων στα αντίστοιχα πεδία. Κατά την καταχώρηση εκτελείται η συνάρτηση “authReq()” με παράμετρο το συμβάν της καταχώρησης φόρμας “event” για τη διαχείριση των δεδομένων προτού ολοκληρωθεί η ενέργεια του συμβάντος εκτελώντας την μέθοδο “preventDefault()”. Στη συνέχεια καταχωρούμε τις τιμές των πεδίων “User” και “Password” στη μεταβλητή “data” σε μορφή “object” για την εύκολη διαχείρισή τους, έπειτα εκτελείται η μέθοδος “getUser()” αποστέλλοντας τα στοιχεία της φόρμας στην υπηρεσία για αυθεντικοποίηση από τον εξυπηρετητή. Αφού ολοκληρωθεί το αίτημα εκχωρείται η κατάσταση του χρήστη στο Component με τη μέθοδο “setState()” και εν συνεχεία

σε μια δημόσια μεταβλητή στο αρχείο “./backendUrl.js” με τη μέθοδο “authentication()” για κοινή χρήση της κατάστασης χρήστη από τα υπόλοιπα Components στην εφαρμογή. Σε περίπτωση απόρριψης εισόδου εμφανίζεται μήνυμα στον χρήστη για αποτυχία αυθεντικοποίησης ή σφάλματος.

Register Component

Στο Register Component γίνεται η διαχείριση της φόρμας εισαγωγής για νέα καταχώρηση στη βάση δεδομένων εφόσον έχει προηγηθεί είσοδος πιστοποιημένου χρήστη.

```
registerPerson(e){
  e.preventDefault();
  const name = document.getElementById('RegName').value;
  const email = document.getElementById('RegEmail').value;
  const title = document.getElementById('RegTitle').value;
  if((name.length && email.length && title.length) > 0){
    const student = {Name: name, Email: email, Title: title};
    sendData(student)
      .then(response => {
        if(Number.isInteger(response)){
          var a = globalData;
          a.push({Name: student.Name, id: JSON.stringify(response)});
          updateGlobalData(a);
          document.getElementById('RegName').value = '';
          document.getElementById('RegEmail').value = '';
          document.getElementById('RegTitle').value = '';
          alert("New data inserted");
        } else {alert(response)}
      })
      .catch(console.error());
  } else {
    alert("You have to fill all the fields!");
  }
}
```

Σχήμα Γ'3. 9: Εισαγωγή νέας καταχώρησης

Εφόσον εισέλθουμε στην εφαρμογή ως πιστοποιημένοι χρήστες, εμφανίζεται το κουμπί “Add Person” που οδηγεί στο πεδίο με τη φόρμα συμπλήρωσης νέας καταχώρησης. Συμπληρώνοντας όλα τα πεδία της φόρμας μπορούμε να καταχωρήσουμε τα δεδομένα στη βάση δεδομένων. Κατά την διαδικασία καταχώρησης εκτελείται η μέθοδος “registerPerson()” με παράμετρο το συμβάν καταχώρησης φόρμας “event”, στη συνέχεια εν συντομία γίνεται ένας τυπικός έλεγχος για την

ύπαρξη τιμών στα πεδία που καταχωρούνται σε τοπικές μεταβλητές και τέλος εκτελείται η μέθοδος “sendData()” με παράμετρο τα στοιχεία της φόρμας προς καταχώρηση. Αφού ολοκληρωθεί το αίτημα, ενημερώνουμε τη “global” μεταβλητή συμπεριλαμβάνοντας τη νέα καταχώρηση στο πίνακα και εμφανίζουμε το μήνυμα για την επιτυχή εισαγωγή στο χρήστη.

Page404 Component

Το συγκεκριμένο Component είναι μια μέθοδος που επιστρέφει μήνυμα μη εύρεσης σελίδας ως στοιχείο React για την απεικόνιση του στο χρήστη.

```
import React from "react";

export function NoMatch(){
  return (
    <h1>The requested URL doesn't exist in this App!</h1>
  );
}
```

Σχήμα Γ΄3. 10: Μήνυμα μη εύρεσης σελίδας

Γ΄3.3.2 HTTP Requests

Η διαχείριση υποβολής αιτημάτων GET, POST, PUT, DELETE στον εξυπηρετητή υλοποιήθηκε με τη χρήση της βιβλιοθήκης Axios, καθώς το React είναι μια βιβλιοθήκη δημιουργίας διεπαφών και δεν εμπεριέχει λειτουργίες διασύνδεσης με εξυπηρετητές και διαχείριση βάσεων δεδομένων.

<https://axios-http.com/>

Για την εγκατάσταση της βιβλιοθήκης Axios στον φάκελο της εφαρμογής πληκτρολογούμε στην γραμμή εντολών την εντολή:

```
"npm install axios"
```

με την προϋπόθεση η διαδρομή στη γραμμή εντολών να δείχνει τον φάκελο της εφαρμογής.

Επίσης, για τις ανάγκες της εφαρμογής εγκαθιστούμε τη βιβλιοθήκη “underscore” που θα χρησιμοποιηθεί για ταξινόμηση δεδομένων στη συνέχεια:

```
" npm install underscore "
```

Οι μέθοδοι της υποβολής αιτημάτων υλοποιήθηκαν στο αρχείο “requestService.js”, το οποίο χρησιμοποιείται ως υπηρεσία υποβολής αιτημάτων στα Components.

Για τη χρήση των μεθόδων HTTP εισάγουμε στο αρχείο τη βιβλιοθήκη Axios

```
" import axios from 'axios'; "
```

και για την λειτουργία ταξινόμησης των δεδομένων εισάγεται η βιβλιοθήκη “underscore”

```
" import _ from 'underscore'; "
```

Μέθοδος GET (Λίστα φοιτητών)

Η πρώτη υποβολή αιτήματος δεδομένων πραγματοποιείται κατά την εκκίνηση της εφαρμογής στο Component App στο αρχείο “App.js” για την αρχικοποίηση της “global” μεταβλητής

```
import { getData } from './Components/requestService';  
import { updateGlobalData } from './Components/backendUrl';  
  
function App() {  
  getData()  
  .then(res => updateGlobalData(res))  
  .catch(console.error());  
}
```

Σχήμα Γ'3. 11: Αρχικοποίηση global μεταβλητής

στη συνέχεια καλείται ξανά στο StudentList Component (Σχήμα Γ'3. 3) για την απόκτηση της λίστας φοιτητών με συνθήκη αποφυγής επανάληψης αίτησης εάν έχουν ήδη ληφθεί τα δεδομένα και απεικονίζονται στην αρχική σελίδα του χρήστη καλώντας τη μέθοδο “getData()” στην οποία υλοποιήθηκε η HTTP μέθοδος GET για τη λήψη των δεδομένων από τον εξυπηρετητή.

```

export async function getData(){
  return await axios.get(baseUrl)
    .then(
      res => {
        var sortedStudents = _.sortBy(res.data, "Name");
        return sortedStudents;
      })
    .catch( error => {
      console.log(error.error);
    });
}

```

Σχήμα Γ'3. 12: Μέθοδος GET λίστας φοιτητών

Η μέθοδος GET εκτελείται με παράμετρο τον σύνδεσμο του εξυπηρετητή που είναι καταχωρημένος στη μεταβλητή “baseUrl” στο αρχείο “./backendUrl.js”. Εφόσον γίνει αποδοχή του αιτήματος επιστρέφεται από τη βάση δεδομένων ένας μονοδιάστατος πίνακας με τα στοιχεία των φοιτητών. Προτού αποσταλούν τα δεδομένα πίσω στο Component υπόκεινται στη μέθοδο ταξινόμησης “sortBy” που παρέχεται από τη βιβλιοθήκη “underscore” και ταξινομεί τα δεδομένα που επιστράφηκαν από τον εξυπηρετητή με παράμετρο ταξινόμησης την αλφαβητική σειρά ονόματος.

Μέθοδος GET (Στοιχεία φοιτητή)

Κατά την επιλογή ενός φοιτητή από τη λίστα η εφαρμογή πλοηγείται στο DetailsView Component που απεικονίζει τα στοιχεία του φοιτητή δυναμικά. Η υποβολή αιτήματος για τα στοιχεία υλοποιήθηκε στη μέθοδο “getDetails()” όπου εκτελείται η HTTP μέθοδος GET με παράμετρο το URI του φοιτητή που περιέχει και το αναγνωριστικό κωδικό “id” στη βάση δεδομένων.

```

export async function getDetails(url){
  return await axios.get(baseUrl+url)
    .then(
      res => {
        return res.data;
      })
    .catch(error => {
      console.log(error);
    });
}

```

Σχήμα Γ'3. 13: Μέθοδος GET για τα στοιχεία φοιτητών

Κατά την ολοκλήρωση του αιτήματος επιστρέφονται τα στοιχεία στο Component ενώ σε περίπτωση σφάλματος τυπώνεται μήνυμα σφάλματος στην κονσόλα του περιηγητή.

Μέθοδος POST

Η μέθοδος POST υλοποιήθηκε για την εισαγωγή νέων καταχωρήσεων στη βάση δεδομένων και για την ταυτοποίηση χρηστών στην εφαρμογή. Η υποβολή νέας καταχώρησης υλοποιήθηκε στη μέθοδο “sendData()” όπου εκτελείτε η HTTP μέθοδος POST με παραμέτρους τον σύνδεσμο του εξυπηρετητή και τα δεδομένα της νέας καταχώρησης.

```
export async function sendData(reg){
  return await axios.post(baseUrl, reg)
    .then(response => {
      return response.data;
    })
    .catch(error => {
      console.log(error.error);
      return error.error;
    });
}
```

Σχήμα Γ'3. 14: Μέθοδος POST εισαγωγής νέας καταχώρησης

Η υποβολή αυθεντικοποίησης χρήστη υλοποιήθηκε στη μέθοδο “getUser()” που εκτελεί την HTTP μέθοδο POST με παραμέτρους το σύνδεσμο του εξυπηρετητή χρηστών και τα στοιχεία ταυτοποίησης, όπου ο σύνδεσμος για τον εξυπηρετητή ταυτοποίησης χρηστών είναι καταχωρημένος στη μεταβλητή “userUrl” στο αρχείο “./backendUrl”.

```
export async function getUser(user){
  return await axios.post(userUrl, user)
    .then(response => {
      return response.data;
    })
    .catch(error => {
      console.log(error.error);
      return error.error;
    });
}
```

Σχήμα Γ'3. 15: Μέθοδος POST αυθεντικοποίησης χρήστη

Μέθοδος PUT

Η μέθοδος PUT υλοποιήθηκε για την ενημέρωση της βάσης δεδομένων κατά την επεξεργασία των στοιχείων στην εφαρμογή. Κατά την υποβολή της ενημέρωσης στοιχείων καλείται η μέθοδος “updateDetails()” που εκτελεί την HTTP μέθοδο PUT με παραμέτρους το URI του φοιτητή και τα στοιχεία προς ενημέρωση στη βάση δεδομένων.

```
export async function updateDetails(url, student){
  return await axios.put(baseUrl+url, student)
    .then(res => {
      return res.data;
    })
    .catch(error => {
      console.log(error.error);
      return error.error;
    });
}
```

Σχήμα Γ'3. 16: Μέθοδος PUT

Μέθοδος DELETE

Η μέθοδος DELETE υλοποιήθηκε για τη διαγραφή στοιχείων από τη βάση δεδομένων. Κατά τη διαγραφή του φοιτητή καλείται η μέθοδος “deleteData()” όπου εκτελείται η HTTP μέθοδος DELETE με παράμετρο αποστολής το URI του φοιτητή προς διαγραφή.

```
export async function deleteData(url){
  return await axios.delete(baseUrl+url).then(res => {
    return res.data;
  })
  .catch(error=>{
    console.log(error.error);
  });
}
```

Σχήμα Γ'3. 17: Μέθοδος DELETE Διαγραφή δεδομένων

Γ'3.3.3Routing

Η δοκιμαστική εφαρμογή υλοποιήθηκε με έμφαση στο μοντέλο SPA. Αυτό καθιστά την εφαρμογή ικανή να ανανεώνει το περιεχόμενο της δυναμικά. Κατά την πλοήγηση του χρήστη στα Components δεν υπόκειται σε επαναφόρτωση η εφαρμογή στον περιηγητή του χρήστη. Η πλοήγηση στα Components επιτυγχάνεται με τη μέθοδο του “routing” όπου η απεικόνιση του περιεχομένου γίνεται με βάση το σύνδεσμο URL που έχει προσδιοριστεί σε κάθε Component. Για την υλοποίηση της λειτουργίας πλοήγησης εγκαθιστούμε τη βιβλιοθήκη “React Router DOM” όπου μας παρέχει τη δυνατότητα προσδιορισμού συνδέσμων για κάθε Component και δυνατότητα διαχείρισης συνδέσμων URL για απεικόνιση των περιεχομένων τους. Για την εγκατάσταση της βιβλιοθήκης πληκτρολογούμε στη γραμμή εντολών την εντολή:

```
" npm install react-router-dom "
```

Οι λειτουργίες του “routing” αναπτύσσονται εντός του App Component στο αρχείο “App.js”, καθώς είναι το κύριο React στοιχείο που απεικονίζεται με τη μέθοδο “ReactDOM.render()” στο αρχείο “index.js”. Αφού εγκαταστήσουμε τη βιβλιοθήκη εισάγουμε τις κλάσεις με τις λειτουργίες του “routing”

```
"import { BrowserRouter as Router, Redirect, Switch, Route } from  
'react-router-dom' "
```

έπειτα εισάγουμε όλα τα Components με τα περιεχόμενα που θέλουμε να απεικονίσουμε κατά την πλοήγηση του χρήστη

```
import StudentList from './Components/StudentsList';  
import DetailsView from './Components/DetailsView';  
import RegLink from './Components/RegisterView';  
import Login from './Components/Login';  
import { NoMatch } from './Components/404error';
```

Σχήμα Γ'3. 18: Εισαγωγή Components στο App.js

Η λειτουργία του “routing” υλοποιείται στη “lifecycle” μέθοδο “render()” που επιστρέφει τον υλοποιημένο κώδικα ως React στοιχείο.

```

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <h2>React</h2>
        <img src={logo} className="App-logo" alt="logo" />
        <h2>MVC</h2>
      </header>
      <Router>
        <Switch>
          <Route exact path = "/">
            <Redirect to = "/StudentList" />
          </Route>
          <Route exact path = "/StudentList" component = {StudentList} />
          <Route exact path = "/StudentDetails/:id" component = {DetailsView} />
          <Route exact path = "/RegLink" component = {RegLink} />
          <Route exact path = "/Login" component = {Login} />
          <Route path = "*" component={NoMatch} />
        </Switch>
      </Router>
    </div>
  );
}

```

Σχήμα Γ'3. 19: Routing

Στο Σχήμα Γ'3. 19 απεικονίζεται ο πλήρης πηγαίος κώδικας του Router στο αρχείο “App.js”. Η πλοήγηση στα Components επιτυγχάνεται με συνδέσμους που έχουμε ορίσει εμείς σε στοιχεία των Components με το React στοιχείο “Link” που παρέχεται από το React Router DOM. Παράδειγμα χρήσης “Link” συναντήθηκε στο StudentList Component (Σχήμα Γ'3. 4) όπου για να γίνει η χρήση του προηγήθηκε η εισαγωγή του στο αρχείο

```

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

```

Παράδειγματα πλοήγησης αποτελούν επίσης οι σύνδεσμοι για την απεικόνιση της φόρμας εισαγωγής και της φόρμας εισόδου χρήστη που υλοποιήθηκαν στο “NavLinks.js” και απεικονίστηκαν εμφωλευμένα στο StudentList Component όπως απεικονίζεται στο Σχήμα Γ'3. 4 ως React στοιχείο “<NavLinks />”.

```

function NavLinks(){
  let user = authUser();
  if(user.logged){
    return(
      <nav>
        <Link to={'/Login'}>
          <button className="btn logged-btn">
            Welcome {user.user}<i className="fa fa-user"></i>
          </button>
        </Link>
        <Link to={'/RegLink'}>
          <button className="btn logged-btn">
            <i className="fa fa-plus-square">Add Person</i>
          </button>
        </Link>
      </nav>
    );
  }
}

```

Σχήμα Γ'3. 20: Πλοήγηση Link

Γ'3.4 Form Validation

Για την επικύρωση των στοιχείων φόρμας υλοποιήθηκαν μέθοδοι ελέγχου για την ορθή είσοδο στοιχείων όπου ειδοποιείται ο χρήστης με κατάλληλο μήνυμα. Στο Register Component όπου αναπτύχθηκε η φόρμα εισαγωγής νέας καταχώρησης, κατά την επιλογή καταχώρησης φοιτητή κάθε πεδίο ελέγχεται για την πληρότητα δεδομένων με συνθήκη “if()”

```

registerPerson(e){
  e.preventDefault();
  const name = document.getElementById('RegName').value;
  const email = document.getElementById('RegEmail').value;
  const title = document.getElementById('RegTitle').value;
  if((name.length && email.length && title.length) > 0){
    const student = {Name: name, Email: email, Title: title};
    sendData(student)
      .then(response => {

```

Σχήμα Γ'3. 21: Έλεγχος πεδίων φόρμας για δεδομένα

Για κάθε πεδίο ξεχωριστά υλοποιήθηκαν μέθοδοι για την εμφάνιση μηνυμάτων που ειδοποιούν τον χρήστη για την υποχρεωτικότητα των πεδίων εφόσον επιλέξουν κάποιο από αυτά και

περάσουν στο επόμενο δίχως συμπλήρωση στοιχείων. Επίσης στο πεδίο “Email” ελέγχεται η πληρότητα αλλά και η ορθότητα σύνταξης του ηλεκτρονικού ταχυδρομείου με σύγκριση κανονικής έκφρασης (“Regular expression”).

```
checkName(event){
  this.setState({touched: true});
  if(event.target.value !== ''){
    this.setState({nameErr: null});
  } else {
    this.setState({nameErr: "Name required!"});
  }
}
checkEmail(event){
  const val = event.target.value;
  const check = /^[a-zA-Z]+([0-9.\-_\]*[a-z0-9]+)*@[1][a-z]+\.{1}[a-z]+$/;
  this.setState({touched: true});

  if(val === ''){
    this.setState({emailErr: "Email is required"});
  }
  else if (!check.test(val)) {
    this.setState({emailErr: "Email is not valid!"});
  }
  else {
    this.setState({emailErr: null});
  }
}
checkTitle(event){
  this.setState({touched: true});
  if(event.target.value !== ''){
    this.setState({titleErr: null});
  } else {
    this.setState({titleErr: "Title required!"});
  }
}
```

Σχήμα Γ'3. 22: Έλεγχος πεδίων φόρμας

Οι μεταβλητές “nameErr”, “emailErr” και “titleErr” χρησιμοποιήθηκαν για την απεικόνιση ειδοποιήσεων στον χρήστη, οι οποίες διασυνδέθηκαν στη διεπαφή του χρήστη με την παρεμβολή δεδομένων.

```

<form onSubmit={this.registerPerson} className="register-form">
  <div className="form-group">
    <label htmlFor="Name">Name</label>
    <input type="text" className="form-control"
      onBlur={this.checkName} id="RegName"
      placeholder="Enter Full Name" />
    <p className="error">{this.state.nameErr}</p>
  </div>

```

Σχήμα Γ'3. 23: Πεδίο φόρμας Register Component

Στο Σχήμα Γ'3. 23 απεικονίζεται εν συντομία η μορφή του πεδίου "Name" στη φόρμα καταχώρησης, όπου επιτυγχάνεται η σύνδεση μεθόδων με συμβάντα των πεδίων. Κατά την καταχώρηση της φόρμας εκτελείται η μέθοδος "registerPerson" για την εισαγωγή των στοιχείων στη βάση δεδομένων ενώ κατά την επιλογή του πεδίου "Name" χωρίς την συμπλήρωση τιμής και όταν εστιάσουμε σε κάποιο άλλο πεδίο, πυροδοτείται η εκτέλεση της μεθόδου "checkName".

Συμπεράσματα

Ακολουθώντας την τεχνική Model-View-Controller προσδιορίσαμε τις λειτουργίες και τα σημεία που πρέπει να αναπτυχθεί ο πηγαίος κώδικας με την αντίστοιχη λογική της εφαρμογής. Αυτός ο διαχωρισμός μας βοήθησε στη διαχείριση της πολυπλοκότητας κατά την υλοποίηση, καθώς διευκόλυνε την τροποποίηση, την επέκταση και την επίλυση σφαλμάτων στα περιεχόμενα της εφαρμογής.

Υλοποιώντας την εφαρμογή Single Page Application στα frameworks Angular και React, είδαμε τους τρόπους προσέγγισης με τους οποίους διαχειρίζονται τα αντίστοιχα frameworks την προβολή των στοιχείων, την επεξεργασία δεδομένων, την δημιουργία διεπαφών, την ανάπτυξη λειτουργιών καθώς και τον τρόπο επικοινωνίας μεταξύ τους. Η ενημέρωση των περιεχομένων δυναμικά και ο τρόπος διαχείρισής τους από τα frameworks καθιστά δυνατή τη δημιουργία διαδικτυακών εφαρμογών που μπορούν να διαχειριστούν πολλές προβολές (Views) συγχρόνως δίχως να επιβαρύνεται η επίδοση της εφαρμογής δραματικά, καθώς και τη δημιουργία πολύπλοκων διεπαφών για την αλληλεπίδραση με τον χρήστη.

Με τον κλάδο της τεχνολογίας να αναπτύσσεται και τις τεχνολογικές υποδομές να βελτιώνονται συνεχώς, η υλοποίηση εφαρμογών Single page application με την τεχνική MVC και τη δυνατότητα διαχείρισης δυναμικού περιεχομένου καθιστά ολοένα και πιο εφικτή τη δημιουργία πολύπλοκων διαδικτυακών εφαρμογών, οι οποίες λειτουργούν στον περιηγητή εξίσου καλά με τα τοπικά εγκατεστημένα (native) προγράμματα.

Ως μελλοντική εργασία θα μπορούσαμε στην δοκιμαστική εφαρμογή να υλοποιήσουμε την καλύτερη διαχείριση των καταχωρήσεων, να επιχειρήσουμε την ανάπτυξη και βελτίωση του συστήματος ως ψηφιακή βιβλιοθήκη, όπως επίσης και να προσθέσουμε ένα πλήρες σύστημα καταγραφής αλλαγών και διατήρησης αντιγράφων ασφαλείας.

Βιβλιογραφία

- Angular Observable*. (n.d.). Retrieved from TEK TUTORIALSHUB:
<https://www.tektutorialshub.com/angular/angular-observable-tutorial-using-rxjs/>
- Angular platform*. (n.d.). Retrieved from <https://angular.io/>
- Axios library*. (n.d.). Retrieved from Axios-HTTP: <https://axios-http.com/>
- Bootstrap. (n.d.). Retrieved from Build with Bootstrap: <https://getbootstrap.com/>
- CommonJS*. (n.d.). Retrieved from NodeJs: <https://nodejs.org/docs/latest/api/modules.html>
- Curry, E., & Grace, P. (2008, April 25). *Flexible Self-Management Using the Model-View-Controller Pattern*. Retrieved from IEEE Xplore:
<https://ieeexplore.ieee.org/abstract/document/4497770>
- Mozilla. (n.d.). *CSS Tutorial*. Retrieved from MDN Web Docs: <https://developer.mozilla.org/en-US/docs/Web/CSS>
- Mozilla. (n.d.). *HTML Tutorial*. Retrieved from MDN Web Docs:
<https://developer.mozilla.org/en-US/docs/Web/HTML>
- Mozilla. (n.d.). *Javascript Tutorial*. Retrieved from MDN Web Docs:
<https://developer.mozilla.org/en-US/docs/Web/JavaScript>
- Mozilla. (n.d.). *Regular Expressions*. Retrieved from MDN: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions
- Node.js*. (n.d.). Retrieved from <https://nodejs.org/en/>
- React framework*. (n.d.). Retrieved from <https://reactjs.org/>
- React Router Web*. (n.d.). Retrieved from React Router:
<https://reactrouter.com/web/guides/quick-start>
- Reactive Extension Library for JavaScript*. (n.d.). Retrieved from RxJS: <https://rxjs.dev/>
- Regex Editor*. (n.d.). Retrieved from Regexr: <https://regexr.com/>
- TypeScript, Javascript syntax for types*. (n.d.). Retrieved from TypeScript:
<https://www.typescriptlang.org/>
- W3school. (n.d.). *CSS Tutorial*. Retrieved from W3schools: <https://www.w3schools.com/css>
- W3school. (n.d.). *HTML Tutorial*. Retrieved from W3schools: <https://www.w3schools.com/html/>
- W3school. (n.d.). *JavaScript Tutorial*. Retrieved from W3schools: <https://www.w3schools.com/js>
- W3school. (n.d.). *PHP Tutorials*. Retrieved from W3schools: <https://www.w3schools.com/php/>

Wikipedia. (n.d.). *CommonJS*. Retrieved from Wikipedia:
<https://en.wikipedia.org/wiki/CommonJS>

Wikipedia. (n.d.). *ECMA Script*. Retrieved from <https://en.wikipedia.org/wiki/ECMAScript>

Wikipedia. (n.d.). *Model–view–controller*. Retrieved from Wikipedia :
<https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller>

Wikipedia. (n.d.). *Representational state transfer*. Retrieved from Wikipedia:
https://en.wikipedia.org/wiki/Representational_state_transfer

Wikipedia. (n.d.). *Single-page Applications*. Retrieved from Wikipedia:
https://en.wikipedia.org/wiki/Single-page_application