ΔΙΕΘΝΕΣ ΠΑΝΕΠΙΣΤΗΜΙΟ ΤΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

«Ολοκληρωμένο Πληροφοριακό Σύστημα Διδακτικού προσωπικού πανεπιστημιακού τμήματος»

Των φοιτητών
Καρανικόλα Στυλιανή
Αρ. Μητρώου:154459
και
Κεμεντζετζίδου Γεωργία
Αρ. Μητρώου:154467

Επιβλέπων
Σιδηρόπουλος Αντώνης
Επίκουρος Καθηγητής

Ημερομηνία 04/09/2022

Τίτλος Δ.Ε. Ολοκληρωμένο Πληροφοριακό Σύστημα Διδακτικού προσωπικού πανεπιστημιακού τμήματος

Κωδικός Δ.Ε. 21169

Ονοματεπώνυμο φοιτητή/τών Καρανικόλα Στυλιανή
Κεμεντζετζίδου Γεωργία
Ονοματεπώνυμο εισηγητή Σιδηρόπουλος Αντώνης

Ημερομηνία ανάληψης Δ.Ε. 05/03/2021
Ημερομηνία περάτωσης Δ.Ε. 04/09/2022

Βεβαιώνω ότι είμαι ο συγγραφέας αυτής της εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, έχω καταγράψει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών, εικόνων και κειμένου, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επιπλέον, βεβαιώνω ότι αυτή η εργασία προετοιμάστηκε από εμένα προσωπικά, ειδικά ως διπλωματική εργασία, στο Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του ΔΙ.ΠΑ.Ε.

Η έγκριση της διπλωματικής εργασίας από το Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του Διεθνούς Πανεπιστημίου της Ελλάδος, δεν υποδηλώνει απαραιτήτως και αποδοχή των απόψεων του συγγραφέα, εκ μέρους του Τμήματος.

Πρόλογος

Κάθε χρόνο η ανάπτυξη εφαρμογών διαχείρισης δεδομένων γίνεται όλο και πιο αναγκαία. Πλέον πολλές υπηρεσίες παρέχονται ηλεκτρονικά, καθώς όλο και περισσότερα αρχεία γράφονται και αποθηκεύονται σε δίσκους και όχι στο χαρτί. Αυτός είναι και ο λόγος που όλο ένα και περισσότερες ηλεκτρονικές πλατφόρμες και εφαρμογές δημιουργούνται καθημερινά για την εξυπηρέτηση των χρηστών και την διαχείριση των πληροφοριών και των αρχείων τους. Οι web εφαρμογές υλοποιούνται για να μπορεί πλέον ο χρήστης να κάνει μια ρουτίνα πιο εύκολα και πιο γρήγορα Αυτή η πτυχιακή εργασία διαθέτει ότι χρειάζεται ένας φοιτητής για να δει την βάση μιας ολοκληρωμένης web εφαρμογής που δεν έχει ιδιαίτερες διαφορές από αντίστοιχες εφαρμογές που τρέχουν παραγωγικά. Είναι μια πολύ καλή προετοιμασία για να μπορέσει ένας φοιτητής να ανταπεξέλθει γρήγορα στις ανάγκες του τομέα του. Η συγκεκριμένη εργασία μπορεί να διευκολύνει τον πρόεδρο ενός τμήματος πανεπιστημίου να ελέγξει και να διαχειριστεί τους καθηγητές. Η αρχιτεκτονική των μελλοντικών web εφαρμογών που θα κληθεί ένας φοιτητής να υλοποιήσει θα είναι παρόμοια με αυτή την αρχιτεκτονική που υπάρχει στην πτυχιακή εργασία. Συνεπώς, η συγκεκριμένη πτυχιακή αποτελεί τον συνδετικό κρίκο της εκπαίδευσης ενός φοιτητή με την μελλοντική του εργασία.

Περίληψη

Κατά την στιγμή της συγγραφής η ανάγκη για ένα διαδικτυακό σύστημα διαχείρισης προσωπικού είναι είναι πιο εμφανής από ποτέ. Το παραδοσιακό σύστημα διαχείρισης βασίζεται στην καταγραφή των προγραμμάτων σε έγραφα τα οποία πρέπει να παραμετροποιουνται ανά τακτά χρονικά διαστήματα πράγμα που μπορεί να οδηγήσει σε ανακρίβειες και σπάταλη πόρων. Επιπλέον σε περιπτώσεις ειδικών υγειονομικών συνθηκών όπως η πρόσφατη πανδημία η διαδικασία αυτή γίνεται σχεδόν αδύνατη καθώς η πρόσβαση στα απαραίτητα έγραφα είναι πολύ πιο δύσκολη.
Για την αντιμετώπιση των προβλημάτων αυτών σχεδιάσαμε μια διαδικτυακή εφαρμογή που επιτρέπει την διαχείριση του διδακτικού προσωπικού. Η εφαρμογή αποτελείται από δυο μέρη, ένα μέρος που είναι υπεύθυνο για την διαχείριση και αποθήκευση των δεδομένων (back-end) και ένα μέρος που είναι υπεύθυνο για την προβολή τους (front-end). Ο διαχωρισμός αυτός έχει ως σκοπό την διευκόλυνση της ανάπτυξης της εφαρμογής, την μεγαλύτερη επεκτασιμότητα της και την βελτίωση της διαχείρισης και της απόδοσης της. Ο σκοπός αυτός επιτυγχάνεται καθώς τα δυο μέρη είναι ανεξάρτητα το ένα από το άλλο και κατά συνέπεια μπορούν να παραμετροποιηθουν χωρίς αυτό να έχει επιπτώσεις στο άλλο. Ένα επιπλέον πλεονέκτημα του διαχωρισμού αυτού είναι η ευκολία με την οποία μπορούν να αναπτυχθούν εφαρμογές για άλλες πλατφόρμες, όπως τα κινητά τηλεφωνά, καθώς το μόνο που χρειάζεται είναι μια αντίστοιχη σε αυτήν front-end εφαρμογή . Τα δυο μέρη συμφωνούν μόνο στο πρωτόκολλο επικοινωνίας που χρησιμοποιούν για την ανταλλαγή δεδομένων. Ο συνδυασμός αυτών των δυο μερών την εφαρμογής προσφέρει μια πλατφόρμα μέσω της οποίας ο υπεύθυνος για τις αναθέσεις του τμήματος θα μπορεί με ασφάλεια και ευκολία να διαχειριστεί το ενεργό διδακτικό προσωπικό άλλα και να προσθέσει καινούργιο.

« Integrated Information System for teaching staff of a university department »

Karanikola Styliani

Kementzetzidou Georgia

Abstract

At the time of writing the need for a web-based personnel management system is more apparent than ever. The traditional management system relies on recording data in documents which must be configured at regular intervals which can lead to inaccuracies and wasted resources. Moreover, in cases of special health conditions such as the recent pandemic, this process becomes almost impossible as access to the necessary documents is much more difficult.

To address these problems, we have designed a web application that allows the management of teaching staff. The application consists of two parts, a part responsible for managing and storing the data (back-end) and a part responsible for viewing it (front-end). This separation is intended to facilitate the development of the application, to make it more scalable and to improve its management and performance. This purpose is achieved as the two parts are independent of each other and can therefore be configured without affecting the other. A further advantage of this separation is the ease with which applications can be developed for other platforms, such as mobile phones, as all that is needed is a corresponding front-end application. The two parties only agree on the communication protocol they use to exchange data. The combination of these two parts of the application offers a platform through which the person in charge of the department's assignments can safely and easily manage the active teaching staff and add new ones.

Ευχαριστίες

Σε αυτό το σημείο και πριν προχωρήσουμε στην παρουσίαση της πτυχιακής μας εργασίας, θα θέλαμε να ευχαριστήσουμε ορισμένους ανθρώπους που συνεργαστήκαμε, που μας συμβούλεψαν και που μας έδειξαν τον δρόμο για την ολοκλήρωση της παρούσας πτυχιακής.

Η παρούσα εργασία, αποτελεί τη Διπλωματική Εργασία στο πλαίσιο των σπουδών μας στο τμήμα "Μηχανικών πληροφορικής και ηλεκτρονικών συστημάτων" του ΔΙΠΑΕ, υπό την επίβλεψη του επίκουρου καθηγητή Σιδηρόπουλου Αντώνη, τον οποίο θα θέλαμε να ευχαριστήσουμε θερμά πρώτα απ' όλα για την εμπιστοσύνη που μας έδειξε με την ανάθεση της εργασίας, την καθοδήγηση και την συμβολή του καθ' όλη την διάρκεια εκπόνησης αλλά και για την γενικότερη βοήθειά του στην μέχρι τώρα σταδιοδρομία μας.

Τέλος, θα θέλαμε να ευχαριστήσουμε τις οικογένειές μας, τους φίλους μας, τους συμφοιτητές μας και τους συνεργάτες μας που μας βοήθησαν όχι μόνο με οδηγίες και γνώσεις αλλά και με την ψυχολογική υποστήριξη που λαμβάναμε καθημερινά μέχρι την ολοκλήρωση της παρούσας πτυχιακής.

Περιεχόμενα

Κατάλογος Εικόνων

Κεφάλαιο 1ο: Εισαγωγή

1.1 Περιγραφή του υπό μελέτη προβλήματος

Είναι γνωστό ότι ένα πανεπιστήμιο διαθέτει εξαιρετικά μεγάλο προσωπικό από καθηγητές. Η διαχείριση των μαθημάτων που θα διδάσκουν ανά εξάμηνο καθώς και οι άδειες τους είναι αρκετά περίπλοκη. Πρέπει να γίνει ισάξια διαμοίραση αναθέσεων και να εξυπηρετηθούν όσο το δυνατό περισσότεροι. Ένας πρόεδρος ενός τμήματος πανεπιστημίου θα πρέπει να μπορεί να διαχειρίζεται και να παρακολουθεί την κατάσταση που επικρατεί για τον κάθε καθηγητή ξεχωριστά αλλά και για όλο το τμήμα συνολικά. Όταν κάποιος καθηγητής είναι καταλληλότερος να διδάξει ένα μάθημα θα πρέπει αυτός να το αναλάβει. Αυτό σημαίνει ότι κάποιος άλλος καθηγητής θα πρέπει να διδάξει μαθήματα που δίδασκε μέχρι στιγμής ο άλλος καθηγητής. Ποια μαθήματα είναι αυτά; Ποιος έχει χρόνο; Πόσες διδακτικές ώρες έχει ο καθηγητής την εβδομάδα; Ερωτήματα που πρέπει να απαντηθούνε εύκολα στον διαχειριστή του προγράμματος των μαθημάτων. Πόσες άδειες δικαιούται ένας καθηγητής; Πόσο αναγκαία είναι η πραγματοποίηση της; Χρειάζεται να καλυφθεί το κενό και αν ναι ποιος καθηγητής μπορεί να το καλύψει; Τα απαραίτητα δεδομένα για την λύση του προβλήματος θα πρέπει να υπάρχουν και να είναι εύκολα διαχειρίσιμα. Χρειάζεται μια ηλεκτρονική υπηρεσία προβολής, διαχείρισης και επεξεργασίας αυτών των δεδομένων για την πραγματοποίηση ενός επιτυχούς προγράμματος μαθημάτων και αδειών.

1.2 Σκοποί και στόχοι της εργασίας

Ξεκινώντας από την ασφάλεια των δεδομένων , η εφαρμογή θα πρέπει να διασφαλίζει ότι μόνο εξουσιοδοτούμενο προσωπικό θα μπορεί να δει και να διαχειριστεί τα στοιχειά των καθηγητών. Δεύτερον πρέπει να διασφαλίζεται ότι οι διαδικασίες ανανέωσης και τερματισμού συμβάσεων για το κάθε είδος προσωπικού και οι καταχωρίσεις άδειων θα γίνονται αυτόματα χωρίς να πρέπει να παρέμβει ο χρήστης του συστήματος. Επιπλέον καθώς η εφαρμογή θα χρησιμοποιείται μέσω ιστότοπου πρέπει να σιγουρέψουμε ότι η εμπειρία του χρήστη θα είναι βέλτιστη υλοποιώντας σύγχρονες τεχνικές παρουσίασης δεδομένων και ενημέρωσης του χρήστη για την κατάσταση των ενεργειών του. Τέλος η εφαρμογή πρέπει να είναι εύκολα διαχειρίσιμη και αναβαθμίσιμη για να μπορεί να τρέχει χωρίς κάποιο ιδιαίτερο κόστος και με περιθώρια για μελλοντικές προσθήκες.

Για να φέρουμε εις πέρας τον στόχο μας αποφασίσαμε να ακολουθήσουμε μια από τις πιο σύγχρονες τεχνικές σχεδιασμού εφαρμογών, δηλαδή των διαχωρισμό της εφαρμογής σε δυο μέρη ένα που θα είναι υπεύθυνο για την διαχείριση των δεδομένων και ένα που θα είναι υπεύθυνο για την προβολή τους. Με αυτό τον τρόπο αξιοποιήσουμε τον χρόνο ανάπτυξης πολύ πιο αποδοτικά και θα αποκτήσουμε εμπειρία παρόμοια με πραγματικά περιβάλλοντα εργασίας οπού ένας προγραμματιστής πρέπει να συχνά να χρησιμοποιήσει πολλαπλές εφαρμογές τρίτων για την επίτευξη των στόχων του. Ο διαχωρισμός αυτός αποφασίσαμε να γίνει με τον εξής τρόπο. Η φοιτήτρια Κεμεντζετζίδου Γεωργία θα αναλάβει την δημιουργία του μέρους που θα διαχειρίζεται και θα αποθηκεύει τα δεδομένα ενώ η φοιτήτρια Καρανικόλα Στυλιανή θα αναλάβει το μέρος της που θα τα προβάλει. Οι δυο αυτές εφαρμογές θα είναι ανεξάρτητες η μια από την άλλη και στο μόνο που θα συμφωνούν είναι το πρωτόκολλο επικοινωνίας και την μορφή την οποία θα έχουν τα δεδομένα που θα ανταλλάσσουν μεταξύ τους. Για τον σκοπό αυτό πρέπει πριν ξεκινήσει η ανάπτυξη των εφαρμογών να θέσουμε μαζί την μορφή των διάφορων αντικειμένων που θα μεταφέρουν τα δεδομένα μας. Το πρωτόκολλο επικοινωνίας θα είναι το HTTP αφού οι εφαρμογές θα επικοινωνούν μέσω του διαδικτύου. Τέλος ένα σημαντικό κομμάτι, στο οποίο θα πρέπει να συνεργαστούμε για την ανάπτυξη του, είναι η διαδικασία της ταυτοποίησης των χρηστών. Η διαδικασία αυτή είναι κρίσιμη για την ορθή λειτουργία της εφαρμογής μας και κατ' επέκταση θα πρέπει

να βρούμε μαζί ένα τρόπο με το οποίο τα δεδομένα που την αφορούν θα μεταφέρονται με ασφάλεια και ακεραιότητα μεταξύ των δυο εφαρμογών.

1.3 Βασικοί ορισμοί

Παρακάτω θα αναλυθούν εν συντομία οι τεχνολογίες και οι γλώσσες προγραμματισμού που χρησιμοποιήθηκαν για την υλοποίηση της λύσης.

1.3.1. Framework

Ορισμός

Framework είναι ένα «πλαίσιο», που αναφέρεται κυρίως σε μια βασική δομή στήριξης στην οποία χτίζονται άλλα πράγματα, όπως οι εφαρμογές. Τα frameworks χρησιμοποιούνται συνήθως για την ανάπτυξη διαδικτυακών εφαρμογών. Συγκεκριμένα ονομάζονται web frameworks ή web application frameworks, είναι πλαίσια λογισμικού που έχουν σχεδιαστεί για να υποστηρίζουν την ανάπτυξη εφαρμογών Ιστού, συμπεριλαμβανομένων των υπηρεσιών ιστού, των πόρων ιστού και των API Ιστού. Αποτελούνται από βιβλιοθήκες οι οποίες είναι κατάλληλες για να επιτελέσουν έναν συγκεκριμένο σκοπό στο ενιαίο προς δημιουργία πρόγραμμα.

Συμβάλουν στην σωστή διαχείριση και "συνεργασία" διάφορων επί μέρους στοιχείων που αποτελούν μια εφαρμογή όπως sessions, cookies, βάσεις δεδομένων, αυθεντικοποίηση καθώς και σύνδεση με εξωτερικές για την εφαρμογή υπηρεσίες όπως είναι το twitter ή ένα google account.

Πλεονεκτήματα

Κάθε προγραμματιστής επιλέγει να χρησιμοποιήσει ένα συγκεκριμένο framework, για να εξυπηρετήσει τον σκοπό και τις απαιτήσεις της εφαρμογής που χρειάζεται να διεκπεραιώσει. Τα περισσότερα όμως frameworks έχουν κοινά πλεονεκτήματα, ασχέτως αν χρησιμοποιούνται περισσότερο ή λιγότερο από τους προγραμματιστές. Έτσι, τα κυριότερα πλεονεκτήματα της χρήσης των frameworks είναι:

- Εξοικονόμηση χρόνου υλοποίησης
- Καλύτερη υποστήριξη από την κοινότητα του framework επιλογής
- Πρόληψη προγραμματιστικού αδιέξοδου
- Απλοποιημένες διαδικασίες testing
- Χρήση μοντέλων MVC
- Εύκολη εκμάθηση
- Ασφάλεια εφαρμογής
- Επεκτασιμότητα
- Ευκολία στην υλοποίηση

1.3.2. Back-end

Ως back-end ονομάζεται το μέρος μιας εφαρμογής με το οποίο ο τελικός χρήστης δεν αλληλεπιδρά άμεσα. Ο σκοπός του μέρους αυτού είναι να επικοινωνεί με μια η πολλές εφαρμογές front-end, με βάσεις δεδομένων η και με άλλες εφαρμογές τύπου back-end ώστε να προσφέρει στον χρήστη τα δεδομένα που αναζητεί και οποιαδήποτε λειτουργία θέλει να κάνει σε αυτά. Οι λειτουργίες και οι δυνατότητες ενός back-end χρησιμοποιούνται έμμεσα από χρήστες μέσω μιας front-end εφαρμογής.

1.3.3 JSON

Σημείωση αντικειμένου JavaScript (JSON) είναι μια ανοιχτή τυπική μορφή αρχείου και μορφή ανταλλαγής δεδομένων, που χρησιμοποιεί ανθρώπινο αναγνώσιμο κείμενο για την αποθήκευση και τη μετάδοση αντικειμένων δεδομένων που αποτελούνται από ζεύγη χαρακτηριστικών-τιμών και τύπους

δεδομένων συστοιχίας (ή οποιαδήποτε άλλη σειριοποιήσιμη τιμή) [17]. Είναι μια πολύ κοινή μορφή δεδομένων, με ένα ευρύ φάσμα εφαρμογών, όπως η αντικατάσταση του XML σε συστήματα AJAX. Χρησιμοποιείται κυρίως για την σειριοποίηση και αποστολή δεδομένων μέσω του διαδικτύου , καθώς δεν χρησιμοποιεί πολλούς πόρους λόγω του μικρού όγκου του αρχείου που παράγει. Προήλθε από JavaScript, αλλά πολλές σύγχρονες γλώσσες προγραμματισμού περιλαμβάνουν κώδικα για τη δημιουργία και ανάλυση δεδομένων μορφής JSON. Ο επίσημος τύπος μέσων Διαδικτύου για το JSON είναι το application / json. Τα ονόματα αρχείων JSON χρησιμοποιούν την επέκταση. Json.

1.3.4 Laravel

Εισαγωγή στην Laravel

Η γλώσσα προγραμματισμού που χρησιμοποιήθηκε για την υλοποίηση της εφαρμογής στην παρούσα πτυχιακή, στο backend ήταν PHP, χρησιμοποιώντας το Laravel framework [7]. Η Laravel χρησιμοποιεί διαφορετική σύνταξη σε σχέση με την απλή και κλασσική χρήση της PHP, χωρίς όμως να δυσκολεύει το έργο των προγραμματιστών.

Η Laravel έχει ως στόχο να γίνει η διαδικασία ανάπτυξης των εφαρμογών πιο ευχάριστη για τον προγραμματιστή, λιγότερο πολύπλοκη και χωρίς να θέτει σε κίνδυνο τη λειτουργικότητα της εφαρμογής. Για το σκοπό αυτό, η Laravel, συνδυάζει τα καλύτερα στοιχεία των framework ιστού, συμπεριλαμβανομένων framework που υλοποιούνται σε άλλες γλώσσες, όπως το Ruby on Rails, το ASP.NET MVC και το Sinatra.

Η Laravel είναι εύκολα προσβάσιμη, αλλά αξιόπιστη παρέχοντας ισχυρά εργαλεία που χρειάζονται για μεγάλες και απαιτητικές εφαρμογές. Προσφέρει έναν συνδυασμό από τα καλύτερα εφόδια – εργαλεία που χρειάζονται οι προγραμματιστές για να δημιουργήσουν και να φέρουν εις πέρας οποιαδήποτε εφαρμογή τους ανατεθεί, όπως είναι η αναστροφή του container ελέγχου, που προσφέρει την μέγιστη επεκτασιμότητα στην εφαρμογή, το αξιόπιστο σύστημα μετεγκατάστασης και η υποστήριξη δοκιμών μονάδας.

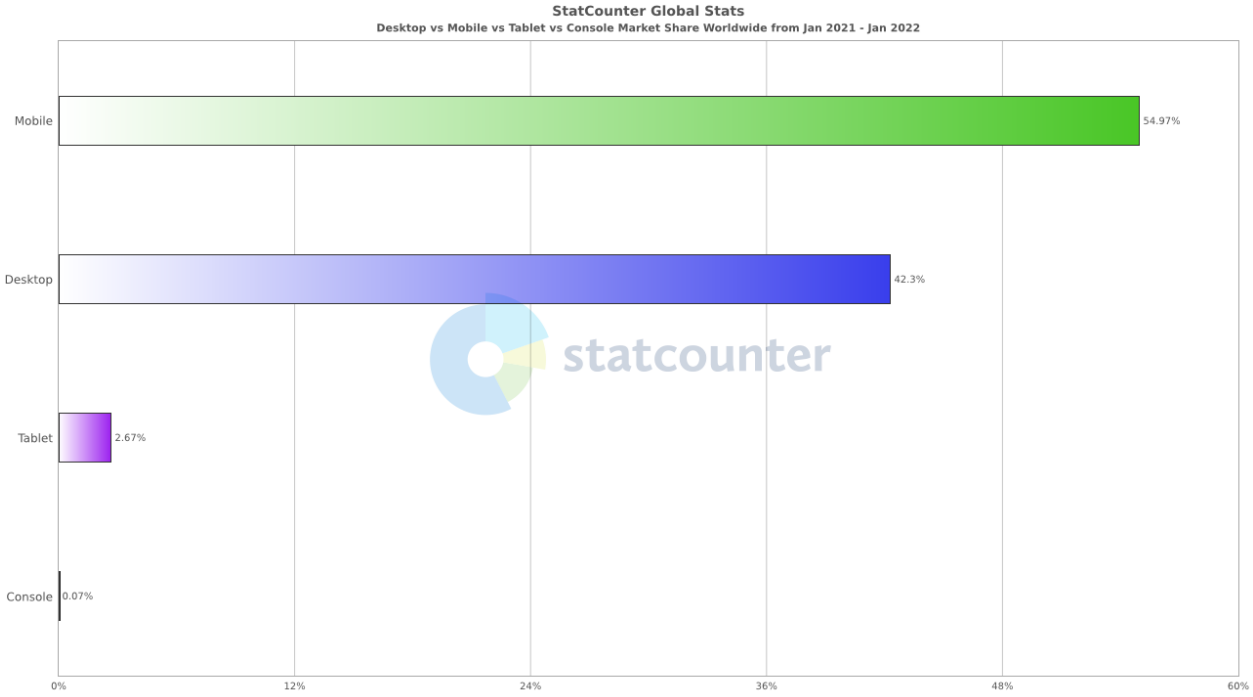
1.3.5 PHP

Όπως αναφέρθηκε παραπάνω, η γλώσσα προγραμματισμού της Laravel είναι η PHP (Hypertext Preprocessor). Η PHP είναι μια γλώσσα προγραμματισμού και χρησιμοποιείται κυρίως για τη δημιουργία δυναμικών σελίδων και web εφαρμογών. Η PHP που δημιουργήθηκε εξ ολοκλήρου από τον Rasmus Lerdorf, το 1994, χαρακτηρίζεται ως “scripting γλώσσα”. Λειτουργεί άψογα σε οποιοδήποτε λειτουργικό σύστημα (Windows, Linux) και εξελίσσεται συνεχώς.

Για να μπορεί η PHP να εκτελείται από πλευρά server χρησιμοποιείται ένα ειδικό πρόγραμμα που ονομάζεται apache. Η Laravel PHP είναι φυσικά μία επίσης server side γλώσσα προγραμματισμού. Είναι ακριβώς ίδια με την κλασσική php, με την διαφορά ότι χρησιμοποιώντας την Laravel ο προγραμματιστής δεν χρειάζεται να έχει άριστες γνώσεις php για να τελειοποιήσει μία εφαρμογή. Συνοψίζοντας, σαν γλώσσα προγραμματισμού, η PHP Laravel είναι κατάλληλη για ανάπτυξη από την πλευρά του διακομιστή επειδή είναι σταθερή, γρήγορη και εύκολη στην εκμάθηση.

Η PHP δεν χρησιμοποιείται για την ανάπτυξη desktop εφαρμογών. Είναι πολύ σημαντικό να μπορεί μία ιστοσελίδα ή μία web εφαρμογή να έχει σωστή εμφάνιση και να λειτουργεί άψογα σε όλες τις συσκευές τις οποίες μπορεί να χρησιμοποιήσει ένας χρήστης. Οι περισσότεροι χρήστες, στις μέρες μας, χρησιμοποιούν ολοένα και περισσότερο τα κινητά τους τηλέφωνα. Παρ’ όλα αυτά η χρήση σταθερού υπολογιστή συνεχίζει να έχει μεγάλη απήχηση σε πολλούς χρήστες. Σύμφωνα με στατιστική έρευνα, η

οποία πραγματοποιήθηκε με δείγμα σε παγκόσμιο επίπεδο κατά τη διάρκεια του έτους 2021, υπήρξε μεγαλύτερη κίνηση στο διαδίκτυο μέσω τον κινητών τηλεφώνων (54.97%) σε σχέση με τους ηλεκτρονικούς υπολογιστές (42.3%) , ενώ αν και σε μικρό ποσοστό(2.67%), υπάρχουν χρήστες που επιλέγουν να χρησιμοποιούν tablet. Συμπερασματικά, είναι σημαντικό να υπάρχει η δυνατότητα ανάπτυξης διαδικτυακών εφαρμογών που μπορούν να λειτουργήσουν εύρυθμα σε όλες τις συσκευές είτε είναι φορητές είτε σταθεροί υπολογιστές.



Εικόνα 1-Διάγραμμα ποσοστών χρήσης κινητών συσκευών και υπολογιστή 2021-2022

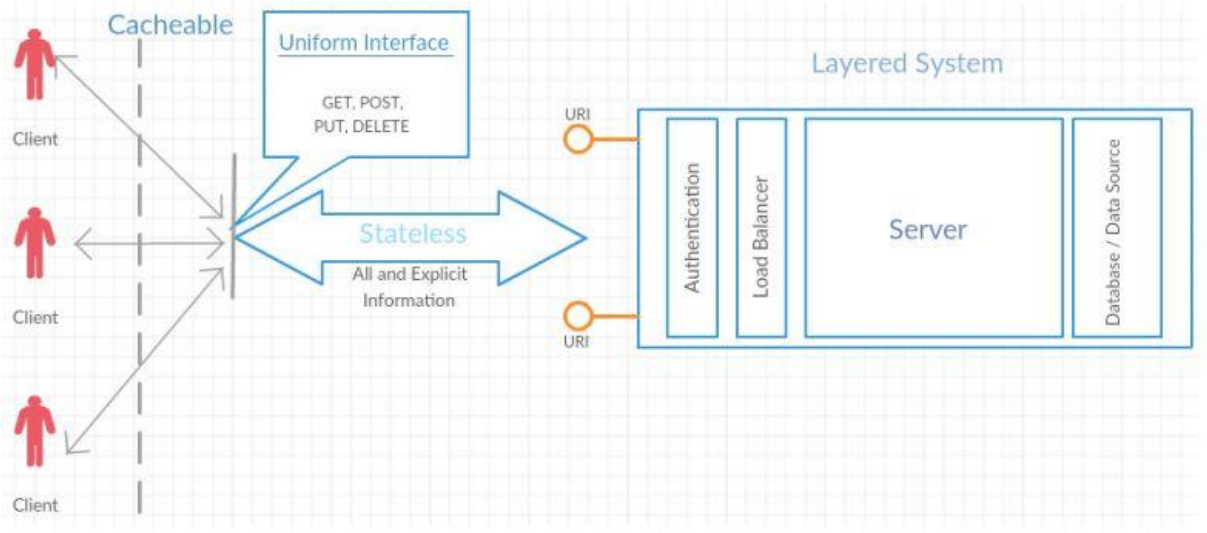
1.3.6 Composer

Ο Composer είναι ένα εργαλείο για να διαχειρίζονται οι προγραμματιστές, τα διάφορα πακέτα της PHP [13]. Πιο συγκεκριμένα, για την Laravel, ο composer είναι ένα εργαλείο που περιλαμβάνει όλες τις εξαρτήσεις και τις βιβλιοθήκες. Χρησιμοποιείται, λοιπόν, για τη διαχείριση των εξαρτήσεων του, οι οποίες καταγράφονται στο αρχείο composer.json που τοποθετείται στον φάκελο προέλευσης. Η βασική λειτουργία του είναι να κατεβάσει και να ενημερώσει τα PHP πακέτα που χρησιμοποιούνται σε μία εφαρμογή και συμβάλει στην ανάπτυξή της ακολουθώντας τους κανόνες και τις διαδικασίες του Laravel framework. Ακόμα οι third-party βιβλιοθήκες μπορούν να εγκατασταθούν εύκολα χρησιμοποιώντας τον composer.

1.3.7 Rest

Rest ονομάζετε ένα πρότυπο σχεδιασμού για APIs που έχει σκοπό να διευκολύνει την επικοινωνία με εφαρμογές front-end και την απλοποίηση σχεδιασμού του back-end. Μια εφαρμογή που έχει σχεδιαστεί βάση αυτού του προτύπου δίνει την δυνατότητα στον server και τον client να αναπτυχθούν ξεχωριστά, χωρίς να γνωρίζουν για την ύπαρξη του αλλού [6]. Αυτό σημαίνει ότι κάθε μεριά μπορεί να παραμετροποιηθεί χωρίς αυτό να έχει επιπτώσεις σε οποιαδήποτε άλλη εφαρμογή επικοινωνεί με αυτή. Το μόνο στο οποίο πρέπει να συμφωνούν το back-end και οι εφαρμογές του front-end είναι ο τρόπος με

τον οποίο θα επικοινωνούν και την μορφή που θα έχουν τα μεταξύ τους μηνύματα. Η αγνωστική σχέση μεταξύ των δυο εφαρμογών ονομάζεται statelessness και οδηγεί σε APIs που είναι πιο αξιόπιστα, καταναλώνουν λιγότερους πόρους και είναι πιο επεκτάσιμα.



Εικόνα 2-Rest

1.3.8. Front-end

Ως front-end αναφέρεται στο μέρος της εφαρμογής το οποίο περιέχει τα γραφικά (User Interface) με τα οποία αλληλοεπιδρά ο χρήστης. Σε μια διαδικτυακή εφαρμογή το κομμάτι αυτό αναφέρεται στην ιστοσελίδα και περιέχει ότι υπάρχει σε αυτή όπως εικόνες, κουμπιά, κείμενο, φόρμες κτλ. Οι γλώσσες προγραμματισμού που χρησιμοποιούνται για την ανάπτυξη τέτοιων εφαρμογών είναι οι HTML,CSS και JavaScript. Έκτος από την προβολή γραφικών το front-end είναι υπεύθυνο και για την ανάκτηση των δεδομένων που ζητά ο χρήστης από το back-end και την επεξεργασία τους αναλόγως της ανάγκες της εφαρμογής.

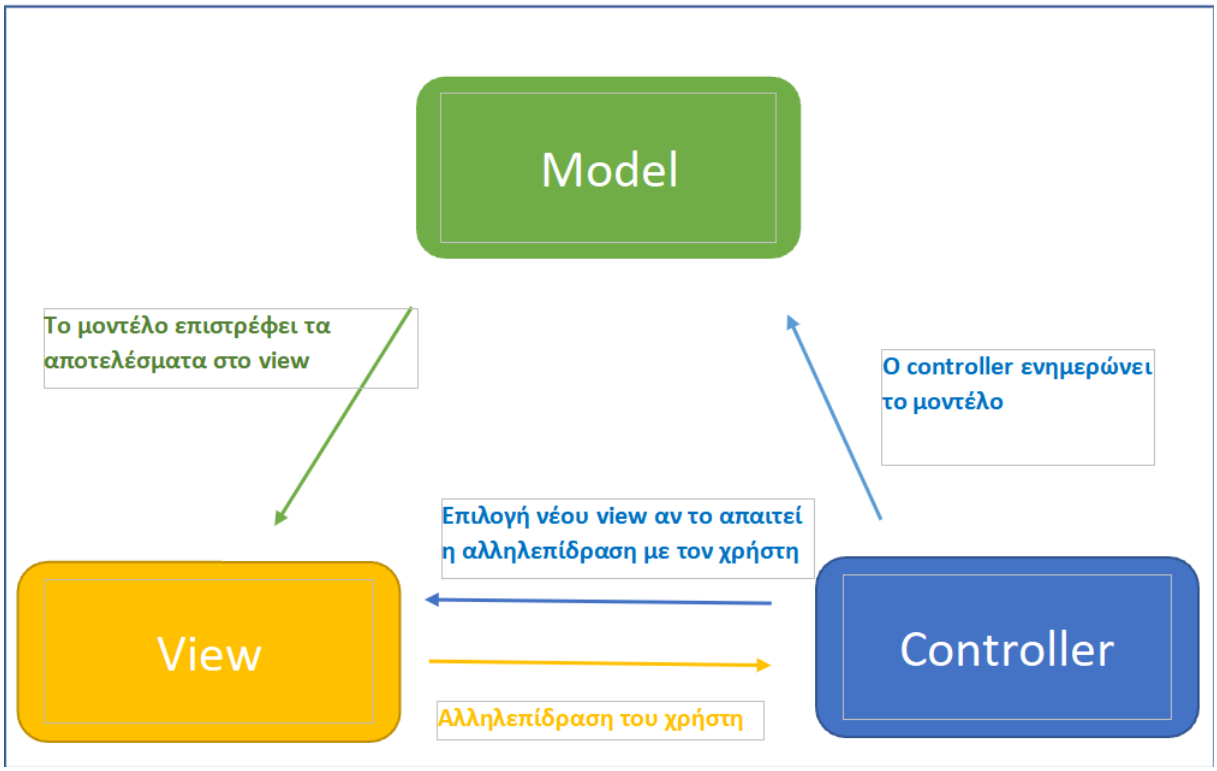
1.3.9 Angular

Η Angular είναι ένα framework JavaScript ανοιχτού κώδικα φτιαγμένο με Typescript που δημιούργησε και διαχειρίζεται η Google [9]. Το framework αυτό χρησιμοποιείται για την κατασκευή εφαρμογών client με HTML και Typescript. Μερικά από τα πλεονεκτήματα της Angular είναι το dependency injection, το two-way data binding και τα directives. Η Angular δημιουργήθηκε με βάση την αρχιτεκτονική Model-View-Controller με αποτέλεσμα όταν τα δεδομένα αλλάζουν στο Model να αλλάζουν και στο View. Αυτό έγινε δυνατό μέσω των directives τα οποία πρόσθεσαν επιπλέον λειτουργίες στο Document Object Model (DOM) επιτρέποντας την δημιουργία δυναμικού περιεχομένου μέσα στον HTML κώδικά.

1.3.10 Model-View-Controller (MVC)

Το Model View Controller (MVC) είναι ένα μοντέλο αρχιτεκτονικής λογισμικού το οποίο συμβάλει στη δημιουργία περιβαλλόντων αλληλεπίδρασης χρήστη. Το μοντέλο αυτό διακρίνεται από τρία ξεχωριστά μέρη:

- Model
- View
- Controller



Εικόνα 3-Model View Controller

MODEL

Το μοντέλο, είναι ένα κομμάτι του συστήματος της εφαρμογής, που ελέγχει όλες τις διεργασίες που σχετίζονται με την συλλογή δεδομένων. Το μοντέλο είναι υπεύθυνο για την επικοινωνία με την βάση δεδομένων και για άλλες εργασίες στην εφαρμογή όπως η επικύρωση δεδομένων, η αυθεντικοποίηση και η ταυτοποίηση δεδομένων. Περιέχει τεχνικές πρόσβασης σε δεδομένα από τις βάσεις δεδομένων και διαχειρίζεται τις βιβλιοθήκες κλάσεων προκειμένου να ελαχιστοποιηθεί η δυσκολία στην χρήση του κώδικα.

Συνεπώς, τα αρχεία που περιέχονται στον φάκελο model είναι κλάσεις οι οποίες σχετίζονται με την database. Στην εφαρμογή ιστού, όσοι πίνακες είναι αποθηκευμένοι στη βάση δεδομένων, τόσα είναι αντίστοιχα και τα αρχεία των κλάσεων. Τα αρχεία αυτά, περιέχουν πληροφορίες, συχνά μεθόδους, σχετικά με αντικείμενα του πίνακα καθώς επίσης και με κάποιες ενέργειες που πραγματοποιούνται κατά την υλοποίηση της εφαρμογής. (Για παράδειγμα εισαγωγή δεδομένων, διαγραφή δεδομένων, ενημέρωση δεδομένων κ.α.).

VIEW

Το View, είναι υπεύθυνο για την παρουσίαση των δεδομένων στην τελική και κατάλληλη μορφή για τον χρήστη. Συνήθως, είναι μια ιστοσελίδα HTML. Είναι η τελική οπτική παρουσίαση των δεδομένων μιας διαδικτυακής εφαρμογής που έχει λάβει ο Controller με τις απαραίτητες αλλαγές και πληροφορίες που έλαβε από το Μοντέλο.

CONTROLLER

Ο Controller, λειτουργεί ως συντονιστής – διαμεσολαβητής μεταξύ View και Model. Με άλλα λόγια, είναι το λογικό επίπεδο της εφαρμογής. Είναι υπεύθυνος για την επεξεργασία των αιτημάτων από τον χρήστη στην εφαρμογή. Τα αιτήματα αυτά απαιτούν συχνά πολύπλοκες διεργασίες. Έτσι, ο Controller επικοινωνεί αρχικά με το Model και στη συνέχεια στέλνει το τελικό αποτέλεσμα προς εμφάνιση στο View.

Σε μία διαδικτυακή εφαρμογή η διαδικασία αλληλεπίδρασης ξεκινάει από ένα url. Δίνοντας ένα url, ουσιαστικά καλείται μια μέθοδος της εφαρμογής του κατάλληλου controller. Στην συνέχεια το αρχείο "Routing" είναι υπεύθυνο για τη σωστή λειτουργία καθώς έχει διπλό ρόλο:

- Είναι υπεύθυνο για την κλήση της κατάλληλης μεθόδου.
- Είναι το αρχείο που περιέχει όλα τα actions.

Για παράδειγμα εάν ένα action έχει σκοπό την εμφάνιση μίας συγκεκριμένης φόρμας επικοινωνίας, τότε με την βοήθεια του controller γίνεται η ανάκτηση των δεδομένων από την βάση δεδομένων και στη συνέχεια τα αποτελέσματα στέλνονται και εμφανίζονται στο View.

1.3.11 HTML

Η HTML είναι γλωσσά καθορισμού διάταξης ενός εγγράφου και ορισμού προδιαγραφών υπερσυνδέσμων [1]. Ορίζει την σύνταξη και την τοποθέτηση ειδικών ενσωματωμένων οδηγιών που δεν προβάλλονται από το πρόγραμμα περιήγησης(browser) άλλα του υποδεικνύουν το πως να προβάλει τα περιεχόμενα του εγγράφου, συμπεριλαμβανόμενων κειμένου, εικόνων, και άλλων βοηθητικών μέσων. Οι οδηγίες αυτές αποτελούνται από μια ομάδα προκαθορισμένων ετικετών τις οποίες τοποθετούμε μέσα στα σύμβολα μεγαλύτερο και μικρότερο από (<ετικέτα>).Οι ετικέτες αυτές λειτουργούν ανά ζεύγη και μεταξύ τους συνήθως τοποθετείτε το περιεχόμενο που θα προβληθεί στην ιστοσελίδα. Επίσης προσφέρει έναν τρόπο να κάνει το έγγραφο διαδραστικό μεσώ της χρήσης ειδικών υπερσυνδέσμων, που το συνδέουν με άλλα έγγραφα η ιντερνετικούς πόρους στο ίδιο σύστημα η σε κάποιο απομακρυσμένο. Η HTML είναι βασισμένη στην SGML, η οποία δημιουργήθηκε με τον σκοπό να είναι μια γλωσσά σήμανσης για όλες τις ανάγκες. Το πρόβλημα με την SGML όμως ήταν πως ήταν τόσο ευρεία και περιεκτική που την έκανε πολύ δύσκολο να χρησιμοποιηθεί. Γι' αυτό τον λόγο η HTML ακολουθεί κάποιους από τους κανόνες που θέτει η SGML αγνοώντας κάποιους άλλους αποφεύγοντας έτσι τα ποιο εξειδικευμένα χαρακτηριστικά της και κάνοντας την πιο εύχρηστη. Παρέχει επίσης την δυνατότητα ενσωμάτωσης κώδικα άλλων γλωσσών όπως είναι η JavaScript μεσώ της οποίας το στατικό περιεχόμενο της HTML μπορεί να γίνει δυναμικό προσθέτοντας έτσι ένα επιπλέον φάσμα δυνατοτήτων. Οι επιλογές μορφοποίησης των γραφικών που δημιουργούνται από την HTML θα μπορούσαν να αναφερθούν ως περιορισμένες, για τον λόγο αυτό ο οργανισμός W3C που διατηρεί την HTML προτείνει την χρήση CSS για σκοπούς αλλαγής του προεπιλεγμένου τρόπου παρουσίασης του περιεχόμενου.

1.3.12 CSS

Η CSS (Cascading Style Sheets – διαδοχικά φύλλα ύφους ή επάλληλα φύλλα ύφους) προσφέρουν την δυνατότητα στον προγραμματιστή να αλλάξει την παρουσίαση ενός εγγράφου. Χωρίς ένα έγγραφο κάποιου είδους η γλωσσά αυτή δεν έχει σκοπό αφού δεν θα είχε κάτι για να παρουσιάσει [2]. Ως έγγραφο βεβαία μπορούν να θεωρηθούν πολλά πράγματα από PDF έως και ιστοσελίδες. Μάλιστα ακόμα και τα προγράμματα περιήγησης(broswer) χρησιμοποιούν CSS για την μορφοποίηση των γραφικών στοιχείων τους. Παρ' ολ' αυτά η πλέον διαδομένη χρήση της γλωσσάς είναι για την μορφοποίηση ιστοσελίδων και κατ' επέκταση κώδικα HTML. Αυτό συμβαίνει γιατί επιτρέπει πολύ πιο σύνθετα γραφικά απ' ότι η HTML, είναι εύκολη στην χρήση και απαιτεί λιγότερο κώδικα για να κάνει

παρόμοιες λειτουργίες και παρέχει διατάξεις για κανόνες που αντικρούονται(cascade). Το τελευταίο προτέρημα είναι και ο λόγος για τον οποίοι ονομάζεται η γλωσσά ως Cascading.

1.3.13 JavaScript

Η JavaScript είναι στενά συνδεδεμένη με τα προγράμματα περιήγησης (browser) πράγμα που την κάνει μια από τις ποιο διαδεδομένες γλώσσες προγραμματισμού στον κόσμο. Η γλωσσά αυτή χρησιμοποιεί κάποιες συμβάσεις που χρησιμοποιούνται και στην Java όμως στην πραγματικότητα έχει περισσότερα κοινά με γλώσσες όπως η Lisp και η Scheme. Μάλιστα την αποκαλούν και Lisp με μορφή C [4]. Η προέλευση της αυτή την κάνει μια αξιοσημείωτα ισχυρή γλωσσά προγραμματισμού. Μερικά από τα χαρακτηρίστηκα της είναι ότι οι μέθοδοι της αποτελούν αντικείμενα πρώτης τάξης (δηλαδή οντότητες που μπορούν να δημιουργηθούν κατά την εκτέλεση του προγράμματος, να περαστούν ως παράμετροι, να επιστραφούν από μια μέθοδο η να ανατεθούν σε μια μεταβλητή), το κατά πλειονότητα στατικό πεδίο δράσης (lexical or static scope) και η χρήση λάμδα εκφράσεων (ανώνυμες μέθοδοι που υλοποιούν την λειτουργικότητα που περιγράφει μια διεπαφή). Ένα από τα πιο αμφιλεγόμενα χαρακτηρίστηκα της JavaScript είναι το loose typing, δηλαδή οι μεταβλητές δεν πρέπει να ορίζουν τον τύπο των δεδομένων που θα έχουν. Το χαρακτηριστικό αυτό μπορεί να είναι ανησυχητικό για προγραμματιστές που έρχονται από strongly typed γλώσσες , καθώς ένα μεγάλο μέρος σφαλμάτων που προέρχονται από τύπους δεν θα εντοπισθεί κατά την μεταγλώττιση, όμως ταυτόχρονα δίνει την δυνατότητα στον προγραμματιστή να σχεδίαση πολύπλοκες δομές αντικειμένων χωρίς να ανησυχεί για έννοιες όπως η κληρονομικότητα. Οι αδυναμίες του loose typing μπορούν να αντιμετωπισθούν μεσώ σωστού testing του κώδικα πράγμα που θεωρείτε σχεδόν αναγκαίο στις μέρες μας. Η JavaScript χρησιμοποιείται και σε εφαρμογές εκτός ιστοσελίδων — τέτοια παραδείγματα είναι τα έγγραφα PDF, οι εξειδικευμένοι φυλλομέτρησες (site-specific browsers) και οι μικρές εφαρμογές της επιφάνειας εργασίας (desktop widgets).

1.3.14 Bootstrap

Η Bootstrap είναι μια συλλογή από χρήσιμα κομμάτια κώδικα σε HTML,CSS και JavaScript. Είναι επίσης ένα frontend framework που δίνει την δυνατότητα στους προγραμματιστές να δημιουργήσουν γρηγορά πλήρης σύγχρονες ιστοσελίδες. Αυτό ουσιαστικά επιτυγχάνεται μειώνοντας το CSS κώδικα που χρειάζεται στο ελάχιστο, δίνοντας περισσότερο χρόνο στον προγραμματιστή για να σχεδίαση την ιστοσελίδα. Μερικά από τα πλεονεκτήματα που προσφέρει είναι η έτοιμη δυναμική διάταξη και τα έτοιμα κομμάτια ιστοσελίδας που παρέχει όπως οι μπάρες πλοήγησης.

1.3.15 Typescript

Η Typescript είναι μια αντικειμενοστραφείς γλώσσα προγραμματισμού, βασισμένη στην JavaScript, που περιλαμβάνει ισχυρή πληκτρολόγηση [8]. Τα πλεονεκτήματα της σε σχέση με άλλες παροιμίες γλώσσες προγραμματισμού είναι ότι ουσιαστικά είναι μια επέκταση της JavaScript. Ο μεταφρασμένος Typescript κώδικας μπορεί να χρησιμοποιηθεί από απλή JavaScript και αντίστοιχα ένα αρχείο με την επέκταση .js μπορεί να μετονομαστεί σε .ts και να μεταφραστεί. Αυτή η ιδιότητα την Typescript την δίνει την δυνατότητα να τρέξει σε οποιοδήποτε μέσο (ιστότοπο, λειτουργικό σύστημα) δέχεται JavaScript χωρίς την ανάγκη για ξεχωριστό runtime environment.

1.3.16. SQL

Η γλωσσά προγραμματισμού SQL(Structured Query Language) και οι σχεσιακές βάσεις δεδομένων που βασίζονται σε αυτήν αποτελούν έναν από τους θεμελιώδεις λίθους των τεχνολογιών που έχουν κάνει την βιομηχανία των ηλεκτρόνιων μέσων, όπως υπολογιστές και κινητά, τόσο διαδεδομένα [4]. Σκοπός

της είναι να οργανώσει, να διαχειριστεί και να ανατρέξει τα δεδομένα που αποθηκεύονται σε μια βάση δεδομένων. Συγκεκριμένα η SQL είναι σχεδιασμένη για να δουλεύει με έναν τύπο βάσεων δεδομένων που ονομάζονται σχεσιακές. Όταν μια οντότητα επιθυμεί να λάβει δεδομένα από μια βάση δεδομένων χρησιμοποιεί SQL για να κάνει το αίτημα και το πρόγραμμα που είναι υπεύθυνο για την διαχείριση της βάσης (DBMS) το επεξεργάζεται, άνακτα τα δεδομένα και τα επιστρέφει σε αυτήν. Η διαδικασία αυτή ονομάζεται database query(ερώτημα στην βάση), εξ ου και το όνομα Structured *Query* Language

1.3.17. Intergraded development environment (IDE)

Τα IDEs είναι εφαρμογές που παρέχουν περιεκτικά εργαλεία για την ανάπτυξη εφαρμογών [20]. Ο σκοπός τους είναι να μειώσουν τους πόρους που χρειάζονται για την σύνθεση προγραμματιστικών εργαλείων προσφέροντας τις ιδίες δυνατότητες σε ένα πακέτο. Με αυτόν τον τρόπο μειώνεται ο χρόνος προδιαταξης και αυξάνεται η παραγωγικότητα.

1.3.17.1. Visual Studio Code

Το VisualStudioCode είναι ένας ελαφρύς και γρήγορος επεξεργαστής κώδικα που υποστηρίζει μεγάλο εύρος γλωσσών προγραμματισμού. Επιλέχθηκε καθώς περιλαμβάνει έναν επεξεργαστή κώδικα που υποστηρίζει το IntelliSense, καθώς και την αναδιαμόρφωση του κώδικα. Το ενσωματωμένο πρόγραμμα εντοπισμού σφαλμάτων λειτουργεί τόσο ως πρόγραμμα εντοπισμού σφαλμάτων την ώρα που εκτελείτε η εφαρμογή όσο και ως εργαλείο εντοπισμού σφαλμάτων κατά την διαδικασία δημιουργίας της. Άλλα ενσωματωμένα εργαλεία περιλαμβάνουν έναν σχεδιαστή κώδικα για την κατασκευή εφαρμογών GUI, σχεδιαστές ιστοσελίδων, σχεδιαστές τάξεων και σχεδιαστές σχήματος βάσης δεδομένων. Δέχεται προσθήκες που επεκτείνουν τη λειτουργικότητα σχεδόν σε κάθε επίπεδο - συμπεριλαμβανομένης της προσθήκης υποστήριξης για συστήματα ελέγχου διαφορετικής εκδοχής κώδικα (όπως το Subversion και το Git) και της προσθήκης νέων συνόλων εργαλείων. Το Visual Studio υποστηρίζει 36 διαφορετικές γλώσσες προγραμματισμού και επιτρέπει στον επεξεργαστή κώδικα και το πρόγραμμα εντοπισμού σφαλμάτων να υποστηρίζει (σε διαφορετικούς βαθμούς) σχεδόν οποιαδήποτε γλώσσα προγραμματισμού, υπό την προϋπόθεση ότι υπάρχει μια υπηρεσία για συγκεκριμένη γλώσσα.

1.3.18 Dependency Injection

Το Dependency Injection είναι ένα πρότυπο σχεδιασμού κώδικα στο οποίο ένα αντικείμενο δέχεται τα αντικείμενα στα οποία εξαρτάται. Ο σκοπός του είναι να ξεχωρίσει την διαδικασία την δημιουργίας ενός αντικειμένου και της χρήσης του οδηγώντας σε χαλαρά συνδεδεμένο κώδικα (loosely coupled code) . Το πρότυπο διασφαλίζει ότι ένα αντικείμενο που θέλει να χρησιμοποιήσει μια διεργασία δεν θα πρέπει να ξέρει πως να την δημιουργήσει. Αντιθέτως η ζητούμενη διεργασία θα παρέχεται από εξωτερικό κώδικα (injector) , ο οποίος δεν είναι εμφανής στον αποδέχοντα κώδικα. Επειδή ο αποδέκτης δεν δημιουργεί η βρίσκει την ζητούμενη διεργασία μονός του χρειάζεται μόνο να αναφερθεί στην διεπαφή (interface) της διεργασίας, πράγμα που κάνει την ευκολότερη την αλλαγή των διεργασιών κατά την εκτέλεση αφού δεν χρειάζεται να επαναμεταφραστεί όλος ο πηγαίος κώδικας. Η χρήση του προτύπου συμβάλει στην δημιουργία επαναχρησιμοποιήσιμου και πιο ευκολά διατηρήσιμου κώδικα

1.4 Παραδοτέα της εργασίας

Τα παραδοτέα της εργασίας αποτελούνται από των κώδικα του back-end σε PHP και τον κώδικα του front end με angular. Επίσης μαζί με αυτά υπάρχει το αρχείο της βάσης δεδομένων τύπου MySQL που δημιουργήσαμε και το εγχειρίδιο χρήσης του API σε ξεχωριστό έγγραφο.

Κεφάλαιο 2ο: Ανάλυση απαιτήσεων εφαρμογής

2.1 Εισαγωγή

Οι απαιτήσεις μιας εφαρμογής διαχωρίζονται σε δυο νοητά μέρη, τις λειτουργικές και της μη λειτουργικές. Οι λειτουργικές απαιτήσεις μιας εφαρμογής περιγράφουν τις μεθόδους που πρέπει να εκτελεί. Ως μέθοδο ορίζουμε απλά τις εισόδους που θα παίρνει, τις λειτουργίες που θα κάνει με αυτά και το τελικό αποτέλεσμα που θα επιστρέφει. Σκοπός τους είναι να βοηθήσουν στην κατανόηση της επιθυμητής λειτουργείας που θα πρέπει να κάνει η εφαρμογή. Οι μη λειτουργικές απαιτήσεις αντιπροσωπεύουν μια σειρά κανόνων μέσω τον οποίων θα κρίνεται αν η εφαρμογή μας δουλεύει όπως αναμένεται. Οι απαιτήσεις αυτές είναι απαραίτητες για την διασφάλιση της σταθερότητας και της αποτελεσματικότητας της εφαρμογής καθώς η μη τήρηση τους μπορεί να οδηγήσει σε αποτυχία κάλυψης των αναγκών ενός χρήστη. Η βασική διαφορά αυτών τον δύο απαιτήσεων είναι ότι μια λειτουργική απαίτηση ορίζει ένα σύστημα η ένα μέρος του ενώ μια μη λειτουργική ορίζει την απόδοση που θα πρέπει να έχει ένα σύστημα.

2.2 Λειτουργικές απαιτήσεις

Ο σκοπός της εφαρμογής που θα αναπτυχθεί είναι να παρέχει σε έναν εξουσιοδοτημένο άτομο την δυνατότητα να διαχειρίζεται τους καθηγητές του πανεπιστήμιου και τα μαθήματα που αυτοί διδάσκουν. Η εφαρμογή αυτή θα πρέπει να μπορεί να χρησιμοποιηθεί μέσω του διαδικτύου και θα αποτελείται από δυο μέρη. Ένα γραφικό περιβάλλον μέσω του οποίου ο χρήστης θα μπορεί να πραγματοποιεί τις διεργασίες που επιθυμεί και μια πηγή πληροφοριών, από την οποία το γραφικό περιβάλλον θα δέχεται τα δεδομένα, που θα μπορεί να χρησιμοποιηθεί ξεχωριστά για την προσθήκη μελλοντικών εφαρμογών. Με γνώμονα τα παραπάνω καταλήξαμε ότι οι λειτουργικές απαιτήσεις τις εφαρμογής μας θα πρέπει να είναι οι εξής:

- Ο χρήστης θα αποτελεί μέρος του πανεπιστήμιου και θα πρέπει να ταυτοποιείται μέσω του ιδρυματικού του λογαριασμού.
- Ο χρήστης θα πρέπει να έχει πρόσβαση στα στοιχεία όλων των καθηγητών του πανεπιστήμιου.
- Ο χρήστης θα μπορεί να αλλάξει τα μαθήματα τα οποία διδάσκει ένας καθηγητής για ένα εξάμηνο.
- Ο χρήστης θα μπορεί να μεταβάλει την κατάσταση ενός καθηγητή από μόνιμο σε έκτακτο και αντίστροφος.
- Ο χρήστης θα μπορεί να δει τα συμβόλαια που έχει ένας καθηγητής και να τα επεξεργαστεί.
- Ο χρήστης θα μπορεί να δει τις αδειές ενός καθηγητή και να τις επεξεργαστεί.
- Θα πρέπει να υπάρχει ένα υποσύστημα το οποίο θα μπορεί να επιστέφει και να δέχεται τις πληροφορίες που χρειάζονται για να πραγματοποιηθούν οι παραπάνω διεργασίες.

2.3 Μη λειτουργικές απαιτήσεις

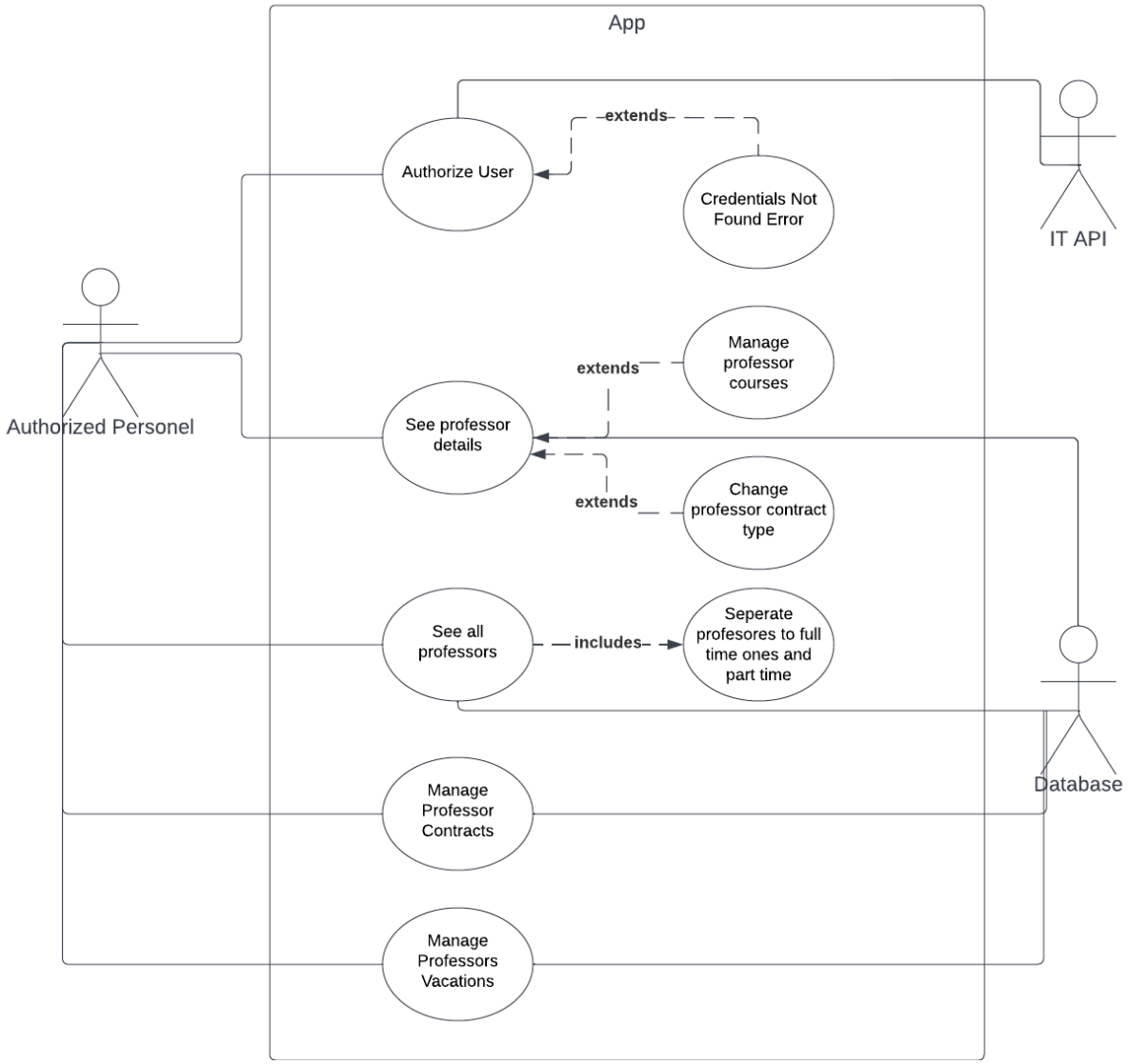
Οι μη λειτουργικές απαιτήσεις μια εφαρμογής που τρέχει σε πρόγραμμα περιήγησης θα πρέπει να είναι οι εξής :

- Οι σελίδες θα πρέπει να φορτώνουν σε πολύ λίγο χρόνο < 1s.
- Οι προστατευμένοι πόροι θα πρέπει να είναι προσβάσιμη μόνο σε ταυτοποιημένο προσωπικό.
- Ο τελικός χρήστης θα πρέπει να ενημερώνεται για την ποριά και τα αποτελέσματα των διεργασιών του.
- Οι χρήστες θα πρέπει να μπορούν να προηγηθούν ευκολά μέσω προφανών γραφικών.
- Η εφαρμογή θα πρέπει να είναι ευκολά επεκτάσιμη.

2.4 Διαγράμματα UML

2.4.1 Διάγραμμα περιπτώσεων χρήσης (Use case)

Στην γλώσσα μοντελοποίησης UML τα διαγράμματα περίπτωσης χρήσης χρησιμοποιούνται για να περιγράψουν την δυναμική συμπεριφορά ενός συστήματος σε υψηλό επίπεδο. Ενθυλακώνουν τις δυνατότητες του συστήματος εμπεριέχοντας περιπτώσεις χρήσης , εξωτερικούς παράγοντες που ονομάζονται actors και της μεταξύ τους σχέσεις. Ένα διάγραμμα περιπτώσεων χρήσης μπορεί να προσδιορίσει τους διαφορετικούς τύπους χρηστών που αλληλοεπιδρούν με το σύστημα και τις διαφορετικές δυνατότητες που μπορεί να έχουν σε αυτό. Παρ' όλα αυτά τα διαγράμματα περιπτώσεις χρήσης δεν εντρίβουν στις λεπτομερείς του συστήματος και για αυτό συνηθώς συνοδεύονται και από άλλους τύπους διαγραμμάτων.



Εικόνα 4 - Διάγραμμα περιπτώσεων χρήσης (Use Case)

Κεφάλαιο 3ο: Σχεδιασμός της Εφαρμογής

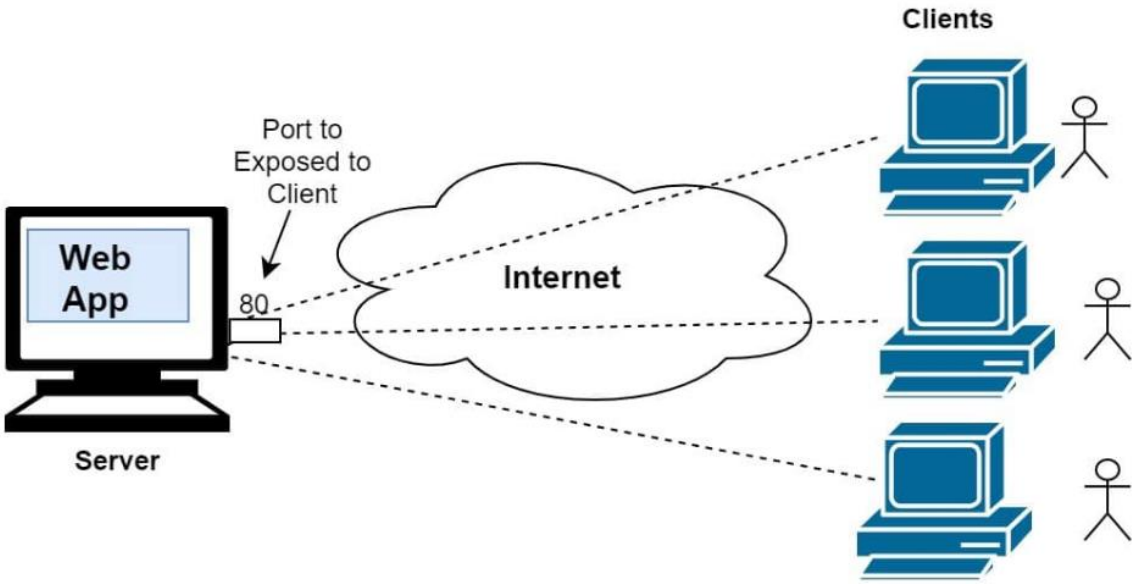
3.1 Εισαγωγή

Στο κεφαλαίο αυτό θα αναλύσουμε τις μεθοδολογίες και τα πρότυπα σχεδιασμού που χρησιμοποιήσαμε για την δημιουργία της τελικής εφαρμογής. Για την ανάπτυξη μιας εφαρμογής που τρέχει μέσω διαδικτύου, χρησιμοποιείται κυρίως το πρότυπο σχεδιασμού Model View Controller (MVC) με δυο διαφορετικές προσεγγίσεις. Την ύπαρξη και των τριών μερών του προτύπου αυτού στην ίδια εφαρμογή η τον διαχωρισμό τους σε δυο, η και περισσότερες εφαρμογές, οπού μια από αυτές θα επεξεργάζεται τα δεδομένα που παίρνει από τον χρήστη και θα τα αποθηκεύει σε κάποια βάση ενώ η άλλη θα προβάλει τα δεδομένα που επεξεργάζεται η πρώτη.

Η ύπαρξη των τριών πλευρών του προτύπου MVC στην ιδιά εφαρμογή αποτελούσε τον προκαθορισμένο τρόπο ανάπτυξης εφαρμογών διαδικτύου για πολύ καιρό. Μάλιστα ακόμα και σήμερα πολλοί ιστότοποι χρησιμοποιούν αυτήν την μεθοδολογία καθώς παλαιότερες εφαρμογές αναβαθμίζονται αλλά προστίθενται και καινούργιες. Το πλεονέκτημα αυτής της μεθοδολογίας είναι η ευκολία με την οποία μπορεί να δημιουργηθεί μια εφαρμογή καθώς τα controller μας επιστρέφουν views στον χρήση τα οποία περνούν τα δεδομένα από τα models δημιουργούν την HTML σελίδα [10]. Καθώς όμως αναπτύσσονταν οι σχετικές τεχνολογίες προέκυψαν κάποια προβλήματα που δεν μπορούσαν να αντιμετωπισθούν με εύκολο τρόπο. Αρχικά δεν ήταν εύκολο για τους προγραμματιστές του UI να σχεδιάσουν ιστοσελίδες καθώς ο κώδικας HTML έπρεπε να διαμορφωθεί ώστε να συμπίπτει με τις προδιαγραφές του View κάθε διαφορετικού framework που είχε χρησιμοποιηθεί. Επιπλέον, η δημιουργία δυναμικών ιστοσελίδων ήταν πολύ πιο δύσκολη καθώς απαιτούσε την χρήση JavaScript για την υλοποίηση της λογικής των δυναμικών λειτουργιών η οποία στην συνέχεια θα έπρεπε να επικοινωνήσει με κάποια άλλη μέθοδο κάποιου controller για να πάρει τα δεδομένα που απαιτούνταν η κάποιο νέο View. Τέλος , ένα πολύ βασικό μειονέκτημα είναι ότι δεν υποστηρίζονταν οι εφαρμογές για κινητά τηλεφωνά. Οι διαχειριστές της εφαρμογής μπορούσαν είτε να παραμετροποιήσουν τα VIEW τους για να τρέχουν στα μικρά παράθυρα των προγραμμάτων περιήγησης των κινητών, πράγμα που δεν ήταν θεμιτό λόγο την κακής απόδοσης που είχαν τα κινητά και κατ' επέκταση οδηγούσε σε κακή εμπειρία χρήστη, είτε να δημιουργήσουν μια εφαρμογή σχεδόν από την αρχή που θα επέστρεφε μόνο δεδομένα και όχι VIEW. Τα μειονεκτήματα αυτά αντιμετωπίσθηκαν σε μεγάλο βαθμό αναβαθμίζοντας την λειτουργικότητα των frameworks με προ εγκατεστημένες μεθόδους JavaScript και προσθήκη περιστέρων τροπών διαμόρφωσης του VIEW ήταν όμως προφανές πως έπρεπε δημιουργηθεί ένα νέο πρότυπο ανάπτυξης εφαρμογών διαδικτύου.

Η λύση στα προβλήματα που αναλύθηκαν παραπάνω ήταν ο διαχωρισμός του view από το controller και το model. Την θέση του view θα έπαιρνε ένα επίπεδο που θα επέστρεφε μόνο δεδομένα σε κανονισμένη μορφή και το view θα διαχειριζόταν από μια εντελώς διαφορετική εφαρμογή η οποία θα δούλευε πάλι με το MVC πρότυπο αλλά δεν θα ανησυχούσε για την επεξεργασία των δεδομένων, μόνο για την προβολή τους. Μέσω του διαμοιρασμού αυτού προκύπτει ο server (back-end) και η client εφαρμογή (front-end). Τα πλεονεκτήματα από τον διαχωρισμό αυτό είναι πολλά. Αρχικά μειώνει τους υπολογιστικούς πόρους που χρειάζεται ο server μας καθώς η διαμόρφωση του view γίνεται πλέον από τον κάθε client ξεχωριστά. Επίσης μειώνεται κατά πολύ το μέγεθος των αρχείων που επιστρέφει ο server καθώς δεν χρειάζεται να στείλει ένα ολόκληρο HTML αρχείο που μπορεί να περιέχει εκατοντάδες γραμμές κώδικα αλλά μόνο τα δεδομένα που θα πρέπει να τοποθετηθούν σε αυτό. Επιπλέον, αφαίρει την ευθηνή για την διαχείριση των σφαλμάτων του client από τον server, αποφεύγοντας το πέσιμο του σε περίπτωση που κάτι πάει στραβά, όπως με την προβολή περίπλοκων γραφικών στο view. Από την

πλευρά του client τα πλεονεκτήματα είναι επίσης πολλά. Αρχικά μπορούν να αναπτυχθούν διαφορετικού τύπου εφαρμογές για το ίδιο back-end χρησιμοποιώντας διαφορετικά frameworks πράγμα που πολλαπλασιάζει τον πιθανό αριθμό χρηστών που θα έχουν πρόσβαση στην εφαρμογή μας. Με αυτόν τον τρόπο ο ιδιόκτητης της εφαρμογής έχει μεγαλύτερο κέρδος σε σχέση με το ενωμένο πρότυπο αλλά και ο τελικός χρήστης έχει πολύ καλύτερη εμπειρία αφού είναι πιο πιθανό να βρει μια client εφαρμογή η οποία θα καλύπτει τις ανάγκες του ιδίου αλλά και της συσκευής του καλυτέρα.



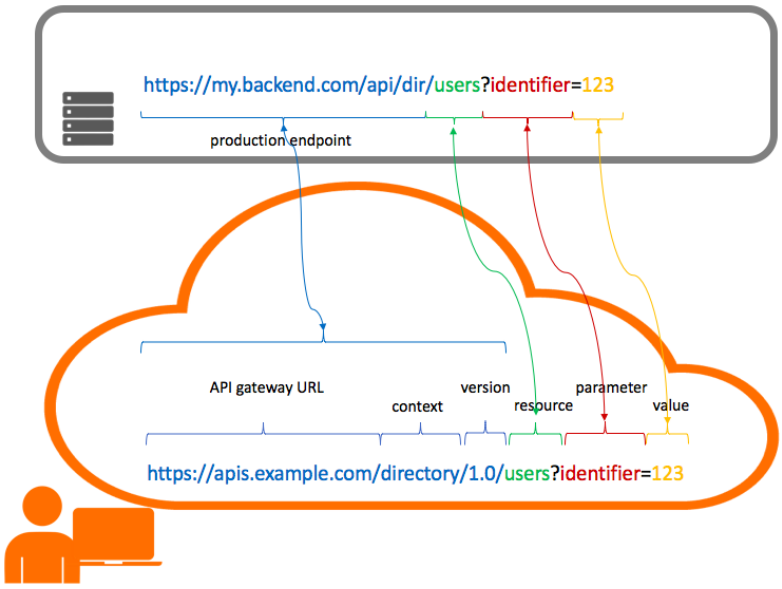
Εικόνα 5 - Το back-end τρέχει σε ένα μηχάνημα και επιστρέφει δεδομένα από οποίων client του τα ζητήσει ενώ η client εφαρμογή τρέχει στην συσκευή του τελικού χρήστη (υπολογιστής, κινητό τηλέφωνο)

Παρόλα τα προτερήματα του διαχωρισμού client και server υπάρχουν και μερικά αρνητικά. Καθώς οι εφαρμογές αναπτύσσονται ξεχωριστά πρέπει να υπάρξει κάποιο σημείο οπού όλες οι πλευρές θα συμφωνούν. Η διαδικασία ανάλυσης για να βρεθούν τα σημεία αυτά, όπως το πρωτόκολλο επικοινωνίας, η μορφή στην οποία θα παρουσιάζονται τα δεδομένα κτλ. , μεταξύ όλων το εμπλεκομένων μπορεί να είναι πολύ δύσκολη. Για τον λόγο αυτό έχουν οριστεί κάποια πρότυπα που πρέπει να ακολουθεί κυρίως το back-end ώστε να μην χρειάζεται να γίνουν τέτοιες διαδικασίες. Αρχικά η επικοινωνία μέσω του διαδικτύου πρέπει να γίνεται μέσω τον μεθόδων του Http ακολουθώντας το πρότυπο CRUD (Create Read Update Delete). Αναλυτικότερα η βασικές μέθοδοι του που χρησιμοποιούνται είναι οι εξής:

- GET: Αναπαριστά το Read και χρησιμοποιείται για την ανάκτηση ενός υπάρχον πόρου
- POST: Αναπαριστά το Create και χρησιμοποιείται για την προσθήκη ενός νέου πόρου
- PUT/PATCH: Αναπαριστούν το Update και χρησιμοποιούνται για την αντικατάσταση η την επεξεργασία ενός υπάρχον πόρου αντίστοιχα.
- DELETE: Αναπαριστά το Delete και χρησιμοποιείται για την διαγραφή ενός πόρου.

Ο server θα πρέπει να έχει έναν controller για κάθε πόρο που διαχειρίζεται στον οποίο θα υπάρχει υλοποίηση για κάθε μια από της παραπάνω μεθόδους που υποστηρίζει. Επιπλέον το URL που αντιστοιχεί στον πόρο αυτό θα πρέπει να έχει μια συγκεκριμένη μορφή. Στην αρχή θα πρέπει να υπάρχει το domain στου server ακολουθούμενο από το όνομα του πόρου ακολουθούμενο από το κλειδί του πόρου, αν η λειτουργία αφορά κάποιον μοναδικό. Επιπλέον χαρακτηρίστηκα όπως το version του server

μας και τυχόν παράμετροι μπορούν να βρίσκονται σε αυτό ανάλογα τον server και την μέθοδο που καλείτε.



Εικόνα 6 - Ανάλυση URL back-end για κάποιον πόρο

Τέλος, οι απαντήσεις του server θα πρέπει να είναι επί το πλείστων σε μορφή JSON. Αποτελεί καλή πρακτική ο δημιουργός του backend να παρέχει ένα έγγραφο στο οποίο αναλύει όλα τα endpoints της εφαρμογής του και της μεθόδους Http που δέχονται και την μορφή που θα έχουν οι πόροι που επιστρέφει ως JSON αντικείμενα. Ακολουθώντας τα πρότυπα αυτά δεν χρειάζεται να υπάρξει κάποια επικοινωνία μεταξύ των δημιουργών του back-end και των front-end εφαρμογών.

Έχοντας υπόψιν μας ότι αναλύσαμε παραπάνω και με γνώμονα τα ζητούμενα της εφαρμογής αποφασίσαμε πως ο πιο σωστός τρόπος ανάπτυξης της είναι ο διαχωρισμός της σε τρία επιμέρους κομμάτια. Τον σχεδιασμό του back-end με PHP και Laravel για την επεξεργασία κ αποθήκευση των δεδομένων, τον σχεδιασμό του front-end με Angular για την προβολή των δεδομένων και τελικά την ασφάλεια της εφαρμογής που απαιτεί την συνεργασία και των δυο άλλων κομματιών ακολουθώντας συγκεκριμένα πρότυπα.

3.2 Υλοποίηση front-end με Angular

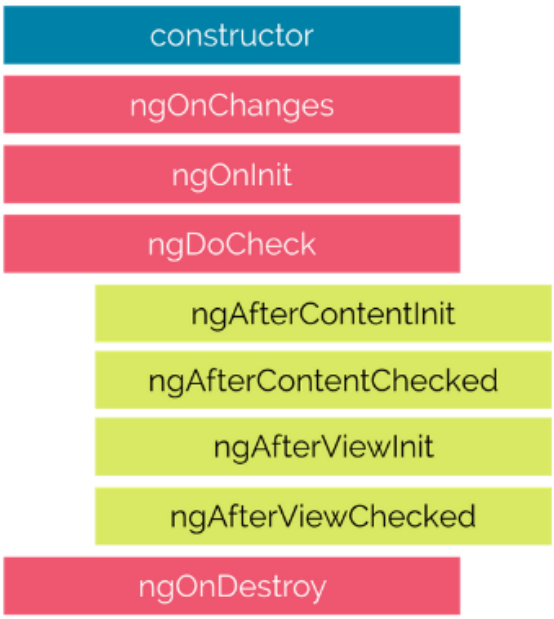
Το βασικό στοιχείο μιας angular εφαρμογής αποτελούν τα components. Κάθε angular εφαρμογή έχει τουλάχιστον ένα component, το οποίο ονομάζεται root και συνδέει όλα τα υπόλοιπα component με το document object model (DOM) [14]. Κάθε component ορίζει μια κλάση η οποία περιέχει λογική και δεδομένα σχετικά με αυτό, και συσχετίζεται με ένα HTML υπόδειγμα που ορίζει την όψη που θα έχει όταν προβληθεί. Ένα component μπορεί να αντιστοιχεί σε μια ολόκληρη σελίδα η και σε μόνο ένα μικρό μέρος μιας αποτελώντας δηλαδή παιδί ενός αλλού component. Επίσης κάθε component χρησιμοποιεί services του παρέχουν λειτουργικότητα που δεν συνδέεται απευθείας με την προβολή στοιχείων. Τα service αυτά μπορούν να γίνουν injected στα components κάνοντας έτσι τον κώδικα μας πιο αποδοτικό, ευανάγνωστο και διατηρήσιμο.

Στην εφαρμογή που αναπτύξαμε επιλέξαμε κάθε component να αποτελεί μια σελίδα της εφαρμογής. Συνολικά υλοποιήσαμε τέσσερα component

- Professors

- Contract-details
- Professor-details
- Permit-details

Κάθε component, έχει ένα συγκεκριμένο κύκλο ζωής κατά τον οποίο μας παρέχει κάποιες μεθόδους τις οποίες μπορούμε να χρησιμοποιήσουμε αναλόγως τις ανάγκες μας



Εικόνα 7 - Angular Lifecycle Hooks

Δυο από τις ποιο σημαντικές μεθόδους κατά τον κύκλο αυτό είναι η ngOnInit και η ngOnDestroy. Η ngOnInit καλείτε οπότε ξεκινάει η αρχικοποίηση ενός component ενώ η ngOnDestroy καλείτε ακριβώς πριν η Angular καταστρέψει το component. Η ngOnDestroy χρησιμοποιείται κυρίως για να "καθαρίσουμε" ότι θέλουμε πριν προχωρήσουμε σε κάποιο άλλο component, όπως ανοιχτά subscriptions που άμα δεν κλείσουμε κατά την κλήση της μεθόδου αυτής θα συνεχίσουν να τρέχουν και μπορούν να προκαλέσουν memory leak. Η μέθοδος ngOnInit χρησιμοποιείται κυρίως για την αρχικοποίηση μεταβλητών και λειτουργιών που είναι σχετικές με την Angular.

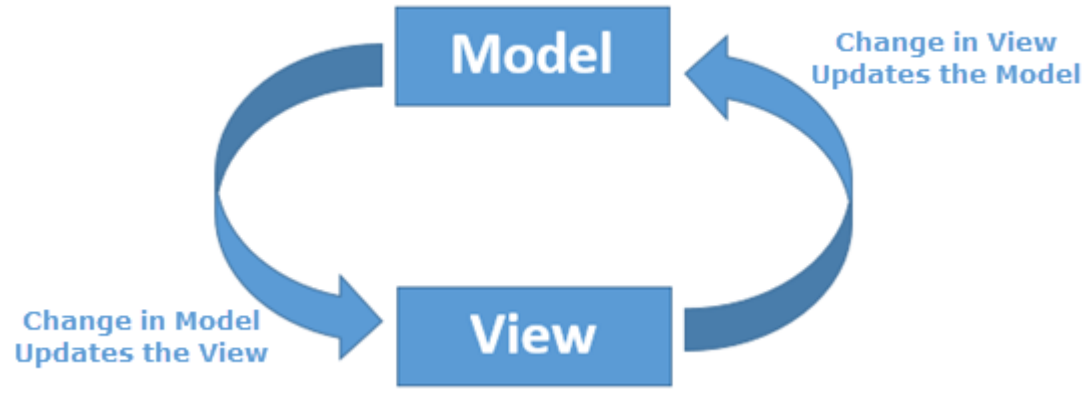
Στην εφαρμογή μας χρησιμοποιούμε το ngOnInit για την αρχικοποίηση των μεταβλητών που θα χρειαστούμε στην συνέχεια και για να κάνουμε τα request στο back-end τα οποία θα φέρνουν τα απαραίτητα δεδομένα για την αρχική διαμόρφωση της ιστοσελίδας. Τα request στο back-end γίνονται μέσω services τα οποία αντιστοιχούνε σε κάθε μοντέλο που επεξεργαζόμαστε και προβάλουμε. Τα μοντέλα αυτά είναι διεπαφές που περιγράφουν τον σκελετό ενός αντικειμένου που υπάρχει στο back-end. Οπότε για παράδειγμα στην αρχική σελίδα την εφαρμογής μας που προβάλουμε μια λίστα από καθηγητές μέσα στην ngOnInit καλούμε την μέθοδο getProfessors του professorsService, το οποίο έχουμε κάνει inject στον constructor του component professors, για να μας επιστρέψει την λίστα από το back-end.

```typescript
export interface IPermit {
    prof_id: number,
    title: string,
    description: string,
    from: string,
    until: string
}
```

```php
protected $fillable = [
  'prof_id',
  'title',
  'description',
  'from',
  'until'
];
```

Εικόνα 8 - Το μοντέλο Permit στο front και το back-end

Αφού έχουμε πλέον τα δεδομένα που θέλουμε να προβάλουμε στον χρήστη πρέπει με κάποιον τρόπο να τα ενσωματώσουμε στον HTML κώδικα μας. Για τον σκοπό αυτό η angular μας παρέχει προδιάγραφες HTML με angular κώδικα που μπορούν να μετατρέψουν στοιχεία HTML πριν αυτά προβληθούν. Οι δυο βασικές προδιάγραφες που μας παρέχονται είναι τα directives που μας επιτρέπουν να γράψουμε λογική μέσα σε HTML κώδικα και τα σύμβολα για binding μέσω των οποίων μπορούμε να συνδέσουμε τα δεδομένα μας με το DOM. Πριν την προβολή μια σελίδας η angular επεξεργάζεται την λογική που έχουμε γράψει στα directive μας και την μετατρέπει σε κανονικό HTML κώδικα χρησιμοποιώντας τα δεδομένα που έχουμε κάνει bind μαζί με τα directives. Συνεχίζοντας το παράδειγμα από την σελίδα με όλους τους καθηγητές αφού πήραμε την λίστα από το back-end την κάνουμε bind στον HTML κώδικα χρησιμοποιώντας ένα mat-table στο οποίο θέτουμε το property datasource την λίστα μας. Το property αυτό του mat-table υποδεικνύει πως έχει γίνει bind με δεδομένα από των κώδικα καθώς βρίσκεται μέσα σε '[ ]'. Σε περίπτωση που θέλουμε να κάνουμε bind δεδομένα στον Html κώδικα σε μέρος οπού θα βρισκόταν στατικά δεδομένα μπορούμε να χρησιμοποιήσουμε το όνομα μια μεταβλητής κατευθείαν στην HTML αρκεί να την βάλουμε μέσα σε '{}'. Αν αντί για το mat-table θέλαμε να δείξουμε δεδομένα που βρίσκονται σε μια λίστα θα μπορούσαμε να χρησιμοποιήσουμε το directive *ngFor, που όπως μας προϊδεάζει το όνομα του εκτελεί ένα for loop για κάθε στοιχείο της λίστας που του κάνουμε bind. Οι τρόποι binding που αναλυθήκαν παραπάνω είναι μονόπλευρη. Δηλαδή αλλάζοντας την μεταβλητή που βρίσκεται μέσα σε {} στον κώδικα θα αλλάξει την τιμή που προβάλλεται στην HTML αλλά αν αλλάξουμε την τιμή αυτή στην HTML δεν θα αλλάξει στον κώδικα. Το πρόβλημα αυτό λύνεται μέσω του two way data binding που προσφέρει η Angular και είναι πολύ χρήσιμο για την άμεση μεταβίβαση των δεδομένων από το DOM στον κώδικα. Τα binding για το two way data binding είναι '[(όνομα μεταβλητής)]'



Εικόνα 9 - Two way data binding

Στην εφαρμογή μας χρησιμοποιούμε two way data binding κυρίως για την ανταλλαγή δεδομένων μεταξύ φορμών. Στο component contract-details για παράδειγμα που μας προβάλει όλες τις συμβάσεις ενός καθηγητή δίνουμε την δυνατότητα στον χρήστη να προσθέσει και να επεξεργαστεί μια σύμβαση μέσω μια HTML φόρμας. Στον κώδικα του component έχουμε δημιουργήσει ένα αντικείμενο τύπου Contract το οποίο έχει πεδία για κάθε στοιχείο μιας συμβάσης που συνδέουμε με την φόρμα μέσω του προτύπου αυτού.

Τέλος η angular μας παρέχει ένα service που μας επιτρέπει να θέσουμε τον τρόπο πλοήγησης μέσα στην εφαρμογή μας μέσω γνωστών προτύπων πλοήγησης που χρησιμοποιούνται σε προγράμματα περιήγησης. Το service αυτό ονομάζεται Router και παίζει πολύ σημαντικό ρολό για την σωστή λειτουργία της εφαρμογής μας. Ο router κάθε φορά που ο χρήστης πραγματοποιεί μια λειτουργία που θα τον κάνει να μεταβεί σε άλλη σελίδα της εφαρμογής μας, όπως το να πατήσει ένα link, θα καταλάβει πως πάει να γίνει αλλαγή και θα καλέσει ολόκληρο το view της συγκεκριμένης σελίδας και όχι απλά την σελίδα. Επίσης είναι υπεύθυνος για το lazy loading των εξαρτήσεων ενός component , δηλαδή θα κάνει load ένα module μόνο αν κρίνει πως αυτό χρειάζεται στην παρούσα κατάσταση της εφαρμογής επιτυγχάνοντας έτσι καλύτερη απόδοση της. Για τον ορισμό των κανόνων πλοήγησης συσχετίζουμε μονοπάτια πλοήγησης με τα component μας. Ένα μονοπάτι χρησιμοποιεί σύνταξη τύπου URL

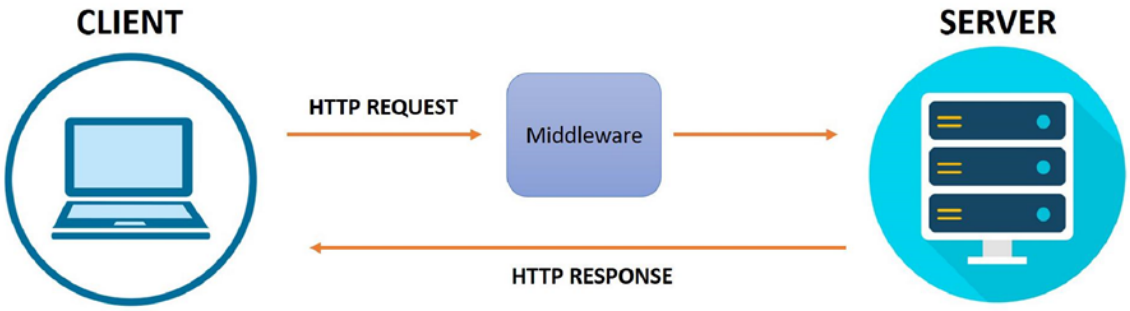
3.3 Υλοποίηση back-end με PHP και Laravel

Το backend(server) της εφαρμογής έχει υλοποιηθεί βάση των προδιαγραφών που αναλύσαμε κατά την εισαγωγή. Αυτό σημαίνει πως το application τρέχει μόνιμα σε έναν server και περιμένει κλήσεις από το front-end έτσι ώστε να απαντήσει και να κάνει και τις προβλεπόμενες κινήσεις ανάλογα με την κλήση που έγινε. Υπεύθυνοι για την υποδοχή των κλήσεων είναι οι controllers. Για κάθε οντότητα της εφαρμογής, υπάρχει και ένας controller και ανάλογα την οντότητα μπορεί να υποστηρίζει τις εξής λειτουργίες:

- Fetch όλων των εγγραφών που υπάρχουν σε έναν πίνακα μιας οντότητας
- Αποθήκευση μίας καινούριας εγγραφής
- Ενημέρωση μιας ήδη υπάρχουσας εγγραφής(update)
- Διαγραφή μιας εγγραφής

Οι controllers στην Laravel [19] οργανώνουν την λογική που χρειάζεται για την ολοκλήρωση ενός request σε μια βολική κλάση . Κάθε κλάση σύμφωνα με τα πρότυπα σχεδιασμού που ακολουθούμε πρέπει να περιέχει μεθόδους που αφορούν έναν συγκεκριμένο πόρο. Οι μέθοδοι αυτοί των κλάσεων αντιστοιχίζονται σε URLs, μέσω τον οποίων μπορούν να χρησιμοποιηθούν από μια front-end εφαρμογή, από το αρχείο routes/api.php. Στο αρχείο αυτό ορίζουμε το URL στο οποίο θέλουμε να ακούει μια μέθοδος, το όνομα της κλάσης που αντιπροσωπεύει τον controller που βρίσκεται η μέθοδος και το όνομα της μεθόδου. Επιπλέον μπορούμε να θέσουμε δυναμικές παραμέτρους στο URL όπως ένα κλειδί βάζοντας το όνομα της μεταβλητής μέσα σε “{}” και στην συνέχεια περνώντας μια μεταβλητή με το ίδιο όνομα ως παράμετρο στην μέθοδο του controller.

Ο σκοπός των controller λοιπών είναι να διαχειρίζονται τα request, πολλές φορές όμως θέλουμε να επεξεργαστούμε τα εισερχόμενα request πριν αυτά φτάσουν στους controllers. Για παράδειγμα δεν θα θέλαμε ένας χρήστης που δεν έχει τα καταλληλά headers στο request του να φτάσει στην μέθοδο ενός προστατευμένου controller. Επίσης δεν μπορούμε να γράψουμε ειδικό κώδικα για την διαχείριση αυτών τον καταστάσεων μέσα στους controllers καθώς αυτό θα πήγαινε ενάντια στην πρότυπο σχεδιασμού που ακολουθούμε και θα οδηγούσε σε δυσανάγνωστο κώδικα. Για τον λόγο αυτό η Laravel μας προσφέρει ένα επίπεδο στην αρχιτεκτονική του back-end μας που ονομάζεται middleware. Το middleware αποτελείται από μια λίστα κλάσεων που τρέχουν με συγκεκριμένη σειρά και σε συγκεκριμένες περιπτώσεις ανάλογος το είδος τους και τις ανάγκες μας.



Εικόνα 10 – Middleware

Δημιουργώντας μια καινούργια κλάση middleware μέσω της αντίστοιχης εντολής της Laravel μας παρέχεται η μέθοδος handle η οποία έχει δυο ορίσματα. Το request που έγινε από το front-end και το next που αφορά την επομένη κλάση middleware στην σειρά. Εμείς δημιουργήσαμε μια κλάση τέτοιου τύπου που έχει ως σκοπό την ανακατεύθυνση του χρήστη στην αρχική σελίδα του front-end μας σε περίπτωση που έχει ειδή τακτοποιηθεί και κατ' επέκταση έχει ειδή ένα access token. Η σειρά και η κατάσταση στη οποία θα τρέξει μια κλάση τύπου middleware ορίζεται στο αρχείο kernel.php.

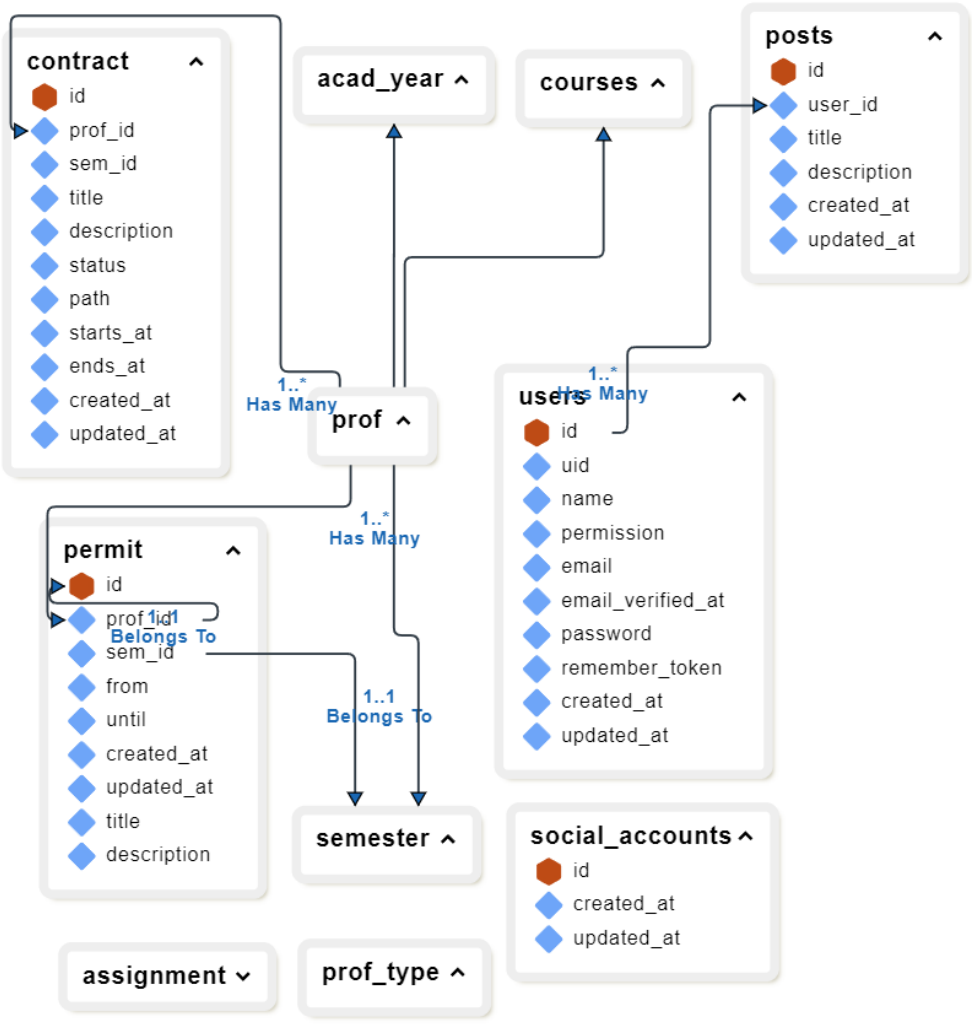
Μια πολύ σημαντική ευθηνή του back-end είναι η σύνδεση με την βάση δεδομένων και η ανάκτηση των δεδομένων. Ο παραδοσιακός τρόπος ανάκτησης δεδομένων είναι με την χρήση SQL ερωτήσεων (queries) στην βάση κάθε φορά που αυτό χρειάζεται. Σε ένα σύστημα σαν το δικό μας όμως οπού υπάρχουν πολλά μοντέλα, και κατ' επέκταση πολλοί πίνακες στην βάση, κάτι τέτοιο θα απαιτούσε αμέτρητες γραμμές κώδικα οι οποίες θα είχαν μικρές διάφορες μεταξύ τους. Για να αποφύγουμε κάτι τέτοιο χρησιμοποιούμε ένα Object-Relational-Mapper(ORM). Τα ORMs ενθυλακώνουν τον SQL κώδικα που απαιτείτε και μας προσφέρουν την δυνατότητα να δουλέψουμε κατευθείαν με τα αντίστοιχα αντικείμενα που χρησιμοποιούμε. Το ORM που χρησιμοποιήσαμε ονομάζεται Eloquent και απαιτεί την αντιστοίχιση κάθε μοντέλου τις εφαρμογής μας σε ένα table. Σαν model ονομάζουμε ένα αντικείμενο που κάνει extend την κλάση Model από το πακέτο Eloquent και περιέχει πεδία που το χαρακτηρίζουν

Τα models που δημιουργηθήκαν για την εφαρμογή μας είναι τα εξής:

- AcademyYear
- Assignment
- Contract
- Courses
- Permit
- Professor
- Semester
- SocialAccount

Τα models αυτά μπορούν να χρησιμοποιηθούν για την κατασκευή query σε PHP χωρίς την χρήση SQL. Με αυτόν τον τρόπο το κάθε σχετικό controller μπορεί να ανατρέχει άμεσα την βάση με μια μικρή καθαρή εντολή για να λάβει τα δεδομένα που χρειάζεται. Σχεδόν πάντα όμως τα models που δημιουργούμε έχουν σχέση μεταξύ τους. Για παράδειγμα ένας καθηγητής μπορεί να έχει από ένα έως πολλά συμβόλαια. Όταν όμως πάμε να λάβουμε τον καθηγητή το μόνο που θα μας επιστραφεί είναι τα πεδία που περιγράφονται στο model του και τίποτα άλλο. Αν γράφαμε κώδικα SQL θα έπρεπε να

εκτελέσουμε δυο select για να έχουμε το επιθυμητό αποτέλεσμα. Μέσω όμως τον περισσότερων ORM και κατ' επέκταση του Eloquent μας παρέχεται η δυνατότητα να ορίσουμε τις σχέσεις αυτές με απλές μεθόδους. Συνεχίζοντας το παράδειγμα η σχέση του καθηγητή με τα συμβόλαια διασφαλίζεται μέσω της μεθόδου hasMany() η οποία παίρνει σαν παραμέτρους το model με το οποίο έχει την σχέση ,το ξένο κλειδί και το τοπικό κλειδί. Παρακάτω φαίνεται το σχήμα ERD όλων των μοντέλων της εφαρμογής μας που δημιουργήσαμε [16]. Όπως εξηγήσαμε παραπάνω παρατηρούμε τις σχέσεις one to many του model professor αλλά και την σχέση Belongs To. Ορίζοντας αυτήν την σχέση μεταξύ δυο model επιτυγχάνουμε την δημιουργία αντιστρόφων query σε σχέση με το hasMany. Δηλαδή μπορούμε να βρούμε τον πατερά του παιδιού μέσω του κλειδιού του πατερά(ξένου κλειδιού). Πιο συγκεκριμένα μπορούμε από ένα model τύπου permit να βρούμε το model τύπου semester στο οποίο αναφέρεται.



Εικόνα 11 - Διάγραμμα ERD των μοντέλων του back-end

Τα models μας για να δουλέψουν σωστά πρέπει να αντιστοιχίζονται σε ένα table στην βάση δεδομένων. Η βάση που χρησιμοποιούμε αποτελείται από τους πίνακες μιας είδη υπάρχον βάσης οι οποίοι περιείχαν πραγματικά δεδομένα από καθηγητές του πανεπιστήμιου μας. Κάποια επιπλέον όμως μοντέλα που χρειαστήκαμε για την δημιουργία της εφαρμογής δεν υπήρχαν ειδή στην βάση και γιατρό τον λόγο

έπρεπε να τα δημιουργήσουμε. Για να το κάνουμε αυτό χρησιμοποιήσαμε migrations. Τα migration είναι μια σειρά από κανόνες που περιγράφονται σε κλάσης, με PHP κώδικα και όχι SQL, που μας επιτρέπουν να διαχειριστούμε τους πίνακες στην βάση δεδομένων μας. Μέσω αυτών δημιουργήσαμε τα table που έλειπαν και αλλάξαμε τα είδη υπάρχον για να είναι συμβατά με την επιπλέον πληροφορία που θα προσθέσει η εφαρμογή μας. Τα migration τρέχουν με μια συγκεκριμένη σειρά ώστε να μην αντιμετωπίσουμε κάποιο SQL σφάλμα.

Τέλος αναπόσπαστο μέρος κάθε API αποτελεί το error handling. Σε περίπτωση που κάτι πάει στραβά κατά την εκτέλεση ενός request το back-end πρέπει να έχει έναν τρόπο για να ενημέρωση το front-end για αυτό. Αν δεν ενημέρωση το front-end ο χρήστης που έκανε το request θα παραμείνει στην ιδιά οθόνη χωρίς να γνωρίζει γιατί δεν μπορεί να προχωρήσει με την εργασία του πράγμα που θα οδηγήσει σε κακή εμπειρία χρήστη. Για να αποφύγουμε κάτι τέτοιο πρέπει να πιάνουμε τα σφάλματα που συμβαίνουν κατά την εκτέλεση των μεθόδων των controller και ανάλογα την φύση τους να επιστρέφουμε ένα αντικείμενο τύπου σφάλματος στο front-end που ακολουθεί το πρότυπο του Http. Το πρότυπο αυτό έχει μια σειρά από κωδικούς που περιγράφουν τον τύπο της απάντησης που λαμβάνουν από το back-end. Περιεκτικά η σειρά 200 χρησιμοποιείται για την ενημέρωση επιτυχούς ολοκλήρωσης του request ενώ η σειρά 400 και 500 χρησιμοποιείται σε περιπτώσεις διάφορων σφαλμάτων. Έτσι μέσω ενός try-catch block στην μέθοδο login του PassportAuthController πιάνουμε το σφάλμα αν συμβεί και αναλόγως τον κωδικό του στέλνουμε ένα αντίστοιχο μήνυμα για να ενημερώσουμε το front-end και τον χρήστη.

3.4 Αναλυτική εξήγηση endpoint που προσφέρει το back-end

3.4.1 ProfessorController

3.4.1.1. GET api/professors

Αντιστοιχεί στην μέθοδο index() η οποία είναι υπεύθυνη για να φέρει μία λίστα με όλες τους καθηγητές που υπάρχουν στην βάση μαζί με τα εξάμηνα και τις ακαδημαϊκές χρονιές κατά τις οποίες εργαζόντουσαν στο πανεπιστήμιο.

3.4.1.2. GET api/professor/{id}

Αντιστοιχεί στην μέθοδο fetchById($id) η οποία επιστρέφει τον καθηγητή με το συγκεκριμένο id που παίρνει σαν παράμετρο. Μαζί με τα στοιχεία του καθηγητή επιστρέφει και τα εξάμηνα τα οποία εργάστηκε, τα μαθήματα που διδάσκει , τα συμβόλαια που έχει (ενεργά και μη) και τυχόν αδειές που μπορεί αν έχει.

3.4.1.3. POST api/professor/edit/{id}

Αντιστοιχεί στην μέθοδο editType($id) η οποία είναι υπεύθυνη για την αλλαγή του τύπου του καθηγητή με το συγκεκριμένο id από μόνιμο σε έκτακτο και αντίστροφα. Αφού πραγματοποιήσει την αλλαγή επιστρέφει πίσω τον καθηγητή.

3.4.2 AcademyYearController

3.4.1.4. GET api/acad_list

Αντιστοιχεί στην μέθοδο index() η οποία επιστρέφει μια λίστα με όλα τα ακαδημαϊκά έτη που υπάρχουν στην βάση μας.

3.4.3 ContractController

3.4.1.5. GET api/contract/show/{id}

Αντιστοιχεί στην μέθοδο show($id) η οποία βρίσκει το συμβόλαιο με το συγκεκριμένο id στην βάση και αφού μετατρέψει της ημερομηνίες έναρξης και λήξης στο επιθυμητό format το επιστρέφει.

3.4.1.6. POST api/contract/create

Αντιστοιχεί στην μέθοδο store() η οποία παίρνει σαν παράμετρο ολόκληρο το request στο οποίο θα περιέχονται τα στοιχεία την συμβάσης. Αρχικά ελέγχει αν το request περιέχει κάποιο πεδίο με το όνομα prof_id και αν το βρει συνεχίζει με τις λειτουργίες της, διαφορετικά τερματίζει. Αν υπάρχει λοιπόν το πεδίο δημιουργεί ένα νέο αντικείμενο τύπου Contract και αρχικοποιεί όλα του τα πεδία με τα αντίστοιχα που περιέχει το request. Σε περίπτωση που κάποιο από αυτά δεν υπάρχει στο request θέτει την τιμή του στο αντικείμενο ως null. Μόλις αναθέσουμε τις τιμές, χρησιμοποιώντας την τωρινή ημερομηνία κάνουμε έναν έλεγχο με την ημερομηνία που ξεκίνησε η σύμβαση και την ημερομηνία που τελειώνει η σύμβαση. Εάν η τωρινή ημερομηνία βρίσκεται ενδιάμεσα τότε σημαίνει ότι είναι ενεργή οπότε στο πεδίο “status” δίνουμε την τιμή 1 αλλιώς δίνουμε 0. Τέλος αποθηκεύουμε την σύμβαση και επιστρέφουμε το νέο αντικείμενο.

3.4.1.7. POST api/contract/update/{id}

Αντιστοιχεί στην μέθοδο update() η οποία δέχεται σαν όρισμα το request με τα στοιχεία ενός ανανεωμένου συμβολαίου και το id που υποδεικνύει πια εγγραφή πρέπει να τροποποιηθεί στην βάση. Αρχικά βρίσκει την είδη υπάρχουσα εγγραφή και ανανεώνει τα πεδία της με τις καινούργιες τιμές ελέγχοντας αν αυτή η τιμή υπάρχει στο request για να αποφευχθούν σφάλματα. Στην συνέχεια ελέγχει τις ημερομηνίες για να δει αν είναι ενεργή η σύμβαση όπως και στην παραπάνω μέθοδο. Έπειτα ελέγχει αν έχει σταλεί εικόνα με το request, στην περίπτωση που υπάρχει διαγράφουμε την παλιά, αν αυτή υπήρχε. Στην συνέχεια βρίσκουμε το extension της καινούργιας εικόνας και μέσω ενός array με επιτρεπόμενα extension ελέγχουμε αν το δικό της επιτρέπετε. Σε περίπτωση που επιτρέπεται, αποθηκεύουμε την εικόνα με ένα μοναδικό όνομα και θέτουμε στο συμβόλαιο το path για να μπορέσουμε να την λάβουμε στο μέλλον, σε περίπτωση που δεν επιτρέπεται επιστρέφουμε σφάλμα. Τέλος επιστέφει το ανανεωμένο συμβόλαιο.

3.4.1.8. POST api/contract/download

Αντιστοιχεί στην μέθοδο download() η κατεβάζει στον client το αρχείο που ζήτησε μέσω του request.

3.4.4 PermitController

3.4.1.9. POST api/permit/create

Αντιστοιχεί στην μέθοδο store() η οποία παίρνει σαν παράμετρο το request με τα στοιχεία μιας καινούργιας αδείας. Αρχικά ελέγχει αν υπάρχει το πεδίο prof_id στο request, αν το βρει δημιουργεί ένα νέο αντικείμενο τύπου Permit και θέτει στα πεδία του τις τιμές που περιέχει αν αυτές υπάρχουν. Τέλος αφού φέρει τις ημερομηνίες στην επιθυμητή μορφή αποθηκεύει το αντικείμενο και το επιστρέφει.

3.4.1.10. POST api/permit/update/{id}

Αντιστοιχεί στην μέθοδο update() που περιέχει σαν παράμετρο το request με την ανανεωμένη άδεια και ένα id που αντιστοιχεί στην άδεια που πρέπει να αλλαχθεί. Αφού βρει την υπάρχον άδεια από την βάση, της θέτει τις καινούργιες τιμές που παίρνει από το request, την αποθηκεύει και την επιστρέφει.

3.4.5 SemesterController

3.4.1.11. GET api/semester_list

Αντιστοιχεί στην μέθοδο index() η οποία επιστρέφει μια λίστα με όλα τα εξάμηνα που υπάρχουν στην βάση.

3.4.6 PassportAuthController

3.4.1.12. GET api/sign-in

Αντιστοιχεί στην μέθοδο signIn και κάνει redirect τον χρήστη στην σελίδα του πανεπιστήμιου για να βάλει τα διαπιστευτήρια του και να τακτοποιηθεί.

3.4.1.13. GET api/sign-in/redirect

Η συνάρτηση redirect είναι αυτή που είναι υπεύθυνη να δημιουργήσει το url που θα ανακατευθυνθεί ο χρήστης όταν θα προσπαθήσει να κάνει login και θα πρέπει να το κάνει μέσω της σελίδας του πανεπιστημίου. Για αυτή την διαδικασία πρέπει ο χρήστης να έχει κάνει πρώτα ένα τοπικό login για να πάρει ένα token από την εφαρμογή που θα χρησιμοποιηθεί μετέπειτα.

3.4.1.14. POST api/login

Αντιστοιχεί στην μέθοδο login() η οποία δέχεται σαν παράμετρο έναν χρήστη και ελέγχει αν αυτός ο χρήστης υπάρχει στην βάση. Αν δεν υπάρχει δημιουργεί και αποθηκεύει έναν βάση των στοιχείων , ενώ αν υπάρχει ανανεώνει τα πεδία του. Έπειτα μέσα σε ένα try-catch block προσπαθεί να κάνει login τον χρήστη. Η δυνατότητα αυτή μας δίνεται καθώς η κλάση User κάνει extend την Authenticatable. Σε περίπτωση που μέσα στην try κάτι δεν πάει καλά τότε το πρόγραμμα αυτόματα πάει στην catch όπου γίνεται logout και καταστρέφεται το session. Τέλος ανάλογα με τον κωδικό του σφάλματος επιστρέφει το ανάλογο μήνυμα.

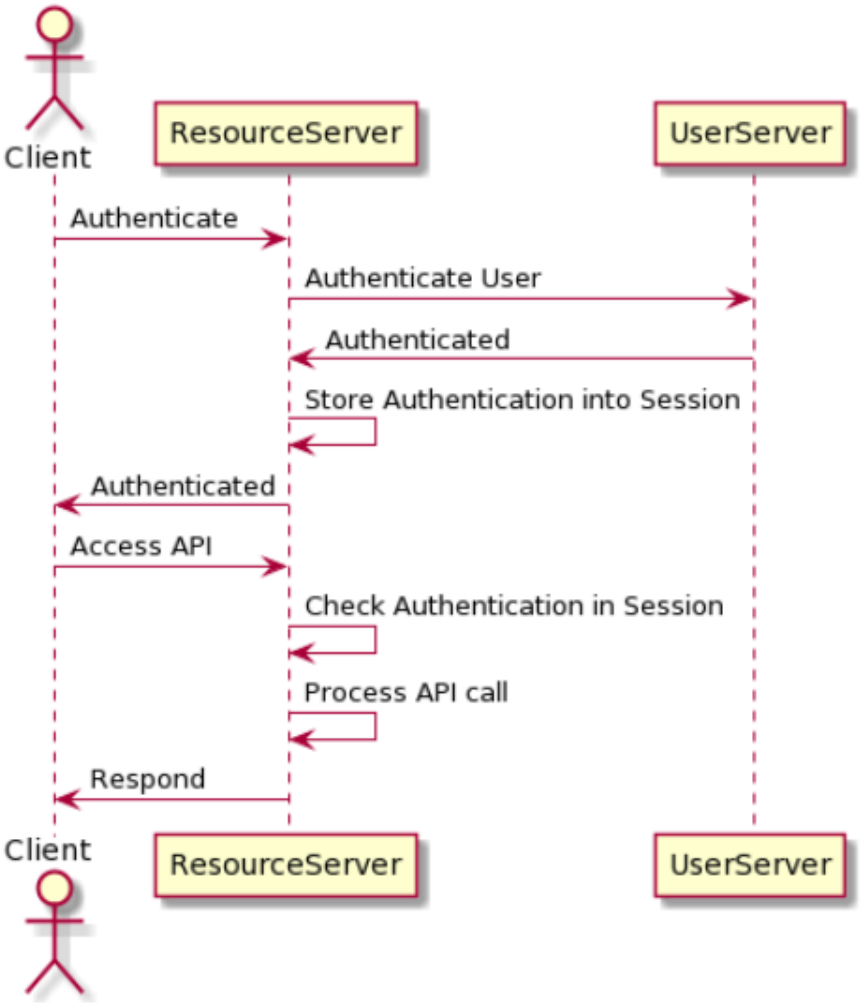
3.5 Ασφάλεια και επιλογή μεθόδου ταυτοποίησης

Σημαντικό μέρος της εφαρμογής που αναπτύχθηκε αποτελεί το σύστημα ταυτοποίησης χρηστών. Για την πραγματοποίηση σημαντικών λειτουργειών που αφορούν προσωπικά δεδομένα εργαζομένων μιας επιχείρησης η οργανισμού, μέσω του διαδικτύου, είναι απαραίτητο να υπάρχει ένα σύστημα που πιστοποιεί ότι το άτομο που προσπαθεί να κάνει τις λειτουργίες αυτές έχει τις αντίστοιχες αρμοδιότητες. Ο σκοπός αυτός επιτυγχάνεται μέσω τις παροχής στον χρήστη διαπιστευτηρίων τα οποία εκδίδει η εφαρμογή η τα δημιουργεί μονός του, μέσω κάποιας φόρμας εγγραφής. Τα διαπιστευτήρια αυτά χρησιμοποιούνται για την ταυτοποίηση του χρήστη στο σύστημα κατά την διαδικασία της σύνδεσης. Παρ' ολ' αυτά υπάρχουν κάποια σημαντικά προβλήματα που πρέπει να αντιμετωπισθούν. Αρχικά ο χρήστης αφού πιστοποίηση ότι έχει δικαίωμα πρόσβασης στο σύστημα στην συνέχεια θα θέλει να δει και να επεξεργαστεί δεδομένα που βρίσκονται σε αυτό. Όμως, καθώς περιγράφουμε μια διαδικτυακή εφαρμογή, κάθε ερώτηση που κάνουμε στο σύστημα θα απαιτεί ξανά τα διαπιστευτήρια του χρήστη. Κάτι τέτοιο θα οδηγούσε σε πολύ κακή εμπειρία χρήστη, σε σπάταλη υπολογιστικών πόρων, αφού το σύστημα μας θα έπρεπε να μετατρέψει τα διαπιστευτήρια σε κωδικοποιημένη μορφή και να τα ανατρέξει στην βάση, και σε προβλήματα ασφάλειας. Επίσης πρέπει να λάβουμε υπόψη και τα δικαιώματα που έχει ο χρήστης καθώς μπορεί να έχει δικαίωμα να δει τα δεδομένα αλλά να μην έχει το αντίστοιχο για να τα επεξεργαστεί. Για την επίλυση των προβλημάτων αυτών πρέπει να

δημιουργήσουμε ένα σύστημα ταυτοποίησης χρηστών το οποίω δέχεται τα διαπιστευτήρια κατά την ταυτοποίηση του χρήστη και στην συνέχεια για κάθε μετέπειτα ζητούμενο του, από την εφαρμογή, τον ταυτοποιεί αυτόματα χωρίς κοστοβόρες διαδικασίες, χωρίς να ανατρέξει την βάση δεδομένων και χωρίς να διακινδυνεύει λειτουργίες στις οποίες ο συγκεκριμένος χρήστης μπορεί να μην έχει πρόσβαση.

3.5.1 Session based Authentication

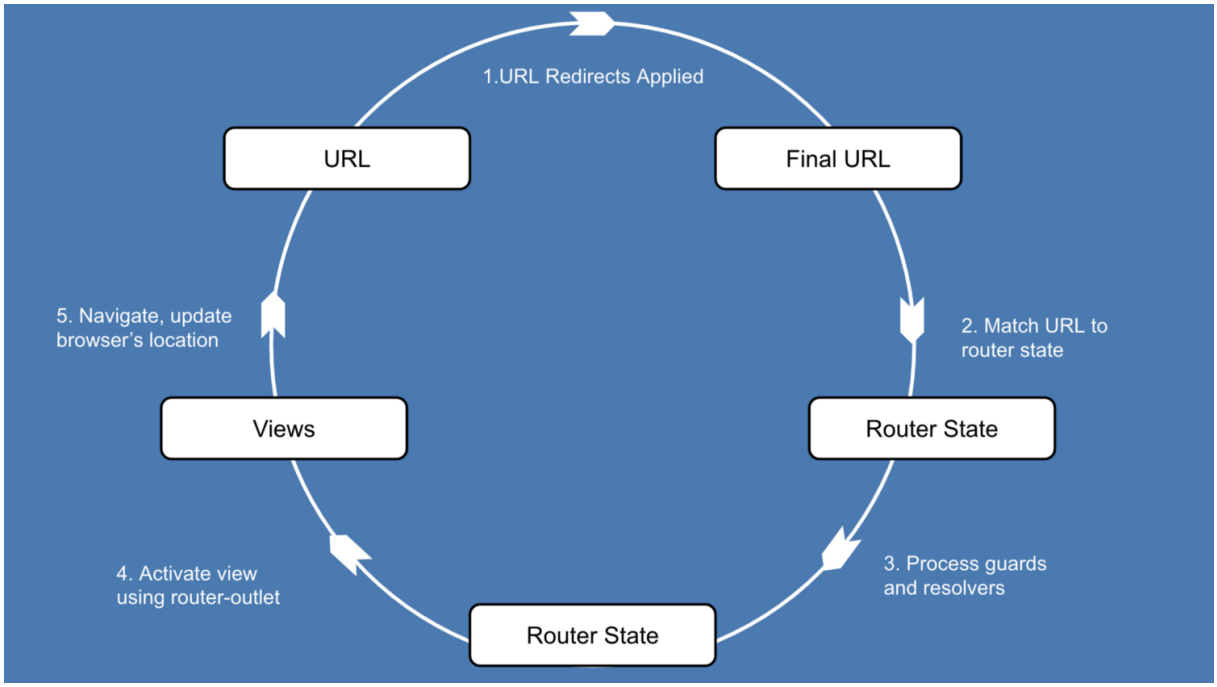
Μια από τις πιο διαδεδομένες και ευκολά υλοποιήσιμες μέθοδοι ταυτοποίησης χρηστών είναι τα sessions [11]. Η μέθοδος αυτή λύνει τα προβλήματα που θέσαμε παραπάνω αποθηκεύοντάς τα στοιχεία του χρήστη σε ένα session στον webserver και προσφέροντας ένα session id στον χρήστη με το οποίο κάθε επόμενο request του μπορεί να ταυτοποιηθεί.



Εικόνα 12 - Ταυτοποίηση βάση session

Στο session μπορούμε να αποθηκεύσουμε δεδομένα όπως το όνομα του χρήστη ,τα δικαιώματα του, την ημερομηνία τελευταίας σύνδεσης κ.ο.κ και να τα χρησιμοποιήσουμε για να κάνουμε διάφορες λειτουργίες που μπορεί να χρειάζονται κατά την υλοποίηση ενός request. Το session του κάθε χρήστη έχει ένα session id το οποίο παρέχεται και στον χρήστη ως cookie και χρησιμοποιείται για την ταυτοποίηση του. Η διαδικασία αυτή είναι απλή καθώς το μόνο που χρειάζεται είναι να γίνει έλεγχος για το αν ένα session με το συγκεκριμένο id υπάρχει στον websever οπού και ο χρήστης ταυτοποιείται ενώ αν δεν υπάρχει δεν ταυτοποιείται και το request απορρίπτεται.

Παρ ολ αυτά η μέθοδος αυτή έχει και τα μειονεκτήματα της. Αρχικά όλα τα δεδομένα για κάθε συνδεδεμένο χρήστη αποθηκεύονται στην μνήμη του server πράγμα που περιορίζει το πλήθος των χρηστών που μπορεί να είναι ενεργοί. Επιπλέον κάνει τον διαμοιρασμό του φόρτου εργασίας (load balancing) σε πολλαπλούς server πολύ πιο δύσκολο καθώς θα πρέπει να υπάρχει ένα σύστημα που θα ενημερώνει τα ενεργά session σε όλους του server. Ένα άλλο σημαντικό μειονέκτημα είναι ότι σε περίπτωση που ο server πέσει θα χαθούν και όλα τα δεδομένα των ενεργών sessions αν δεν έχουν αποθηκευτεί σε κάποια βάση, πράγμα που απαιτεί περεταίρω πόρους. Λαμβάνοντας υπόψιν ότι αναφέραμε παραπάνω γίνεται εμφανές ότι ενώ τα sessions λύνουν τα περισσότερα από τα προβλήματα μας, έχοντας όμως σχετικά υψηλό κόστος και δυσκολεύοντας την μελλοντική επέκταση των δυνατοτήτων της εφαρμογής.

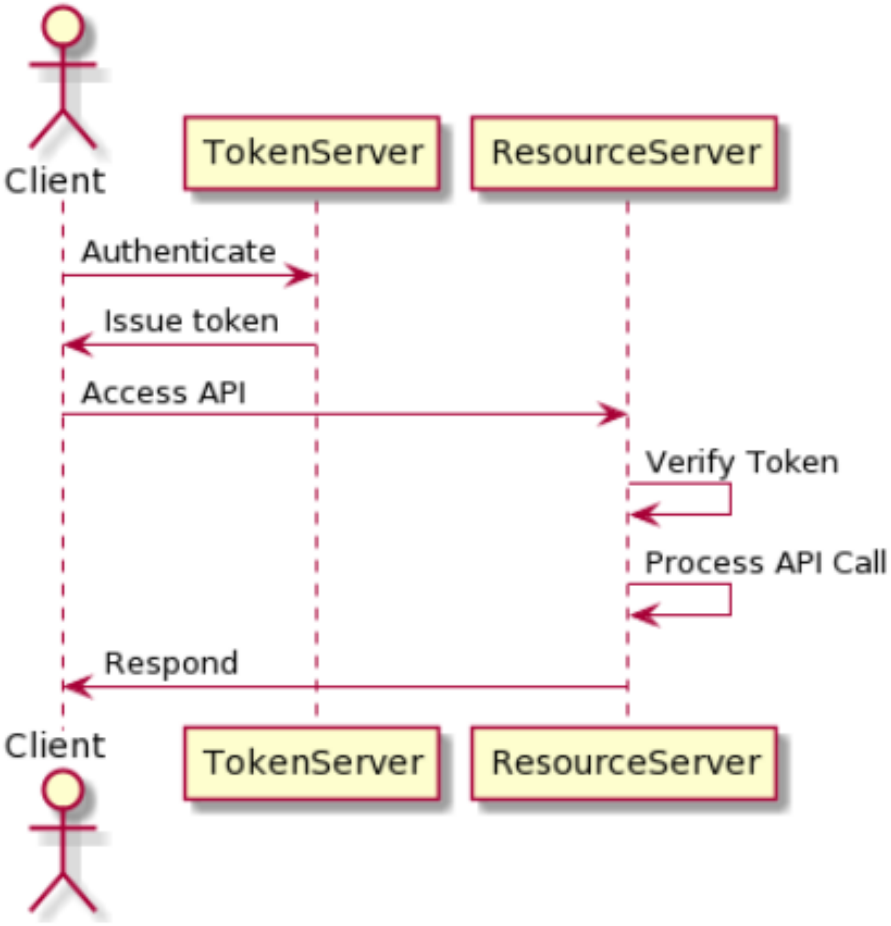


Εικόνα 13 - Πλοήγηση σε μια angular εφαρμογή

3.5.2 Token based Authentication

Η μέθοδος που σημειώνει ραγδαία άνοδο στην αφομοίωση σε συστήματα ταυτοποίησης είναι η χρήση token [11]. Όπως και με τα sessions τα tokens προσφέρουν λύση στο πρόβλημα ταυτοποίησης του χρήστη ανταλλάζοντας ουσιαστικά τα διαπιστευτήρια του με ένα token το οποίο μπορεί να χρησιμοποιήσει ο server για να τον ταυτοποίηση χωρίς να χρειάζεται να κάνει όλη την διαδικασία της αυθεντικοποίησης των στοιχείων του. Τα προαπετούμενα για να λειτουργήσει ένα τέτοιο σύστημα είναι να υπάρχει ειδική υποδομή στον server για την δημιουργία του token και ένας τρόπος για την επιβεβαίωση της ακεραιότητας του token που επιστρέφει ο χρήστης μαζί με κάθε request του. Αν ο server καλύψει αυτά τα ζητούμενα αυτόματος έχει πλεονέκτημα σε σχέση με την χρήση sessions καθώς δεν χρειάζεται να αποθηκεύσει τίποτα στην μνήμη του και δεν πρέπει να ανησυχεί για το τι θα γίνει σε περίπτωση που πέσει, καθώς δεν είναι απαραίτητο να γνωρίζει ποια είναι τα ενεργά token. Επιπλέον πλεονέκτημα που αποκτά είναι, αφού δεν χρειάζεται να αποθηκεύει τα tokens, είναι ότι μπορεί να διαχειριστή πολύ πιο ευκολά μεγάλο όγκο χρηστών, αφού ουσιαστικά το μόνο που χρειάζεται είναι να διαπιστωθεί αν το token είναι αυθεντικό η όχι. Επίσης, η διαδικασία έκδοσης των token μπορεί να ανατεθεί σε μια ξεχωριστή εφαρμογή, η οποία θα επικοινωνεί με όλους τους ενδιαφερομένους server και θα τους ενημερώνει μόνο για τον τρόπο ελέγχου της ακεραιότητας του token επιτυγχάνοντας έτσι

την υποστήριξει πολλαπλών server που μπορεί να αφορούν της λειτουργίες μια σελίδας με μόνο ένα σύστημα ταυτοποίησης.



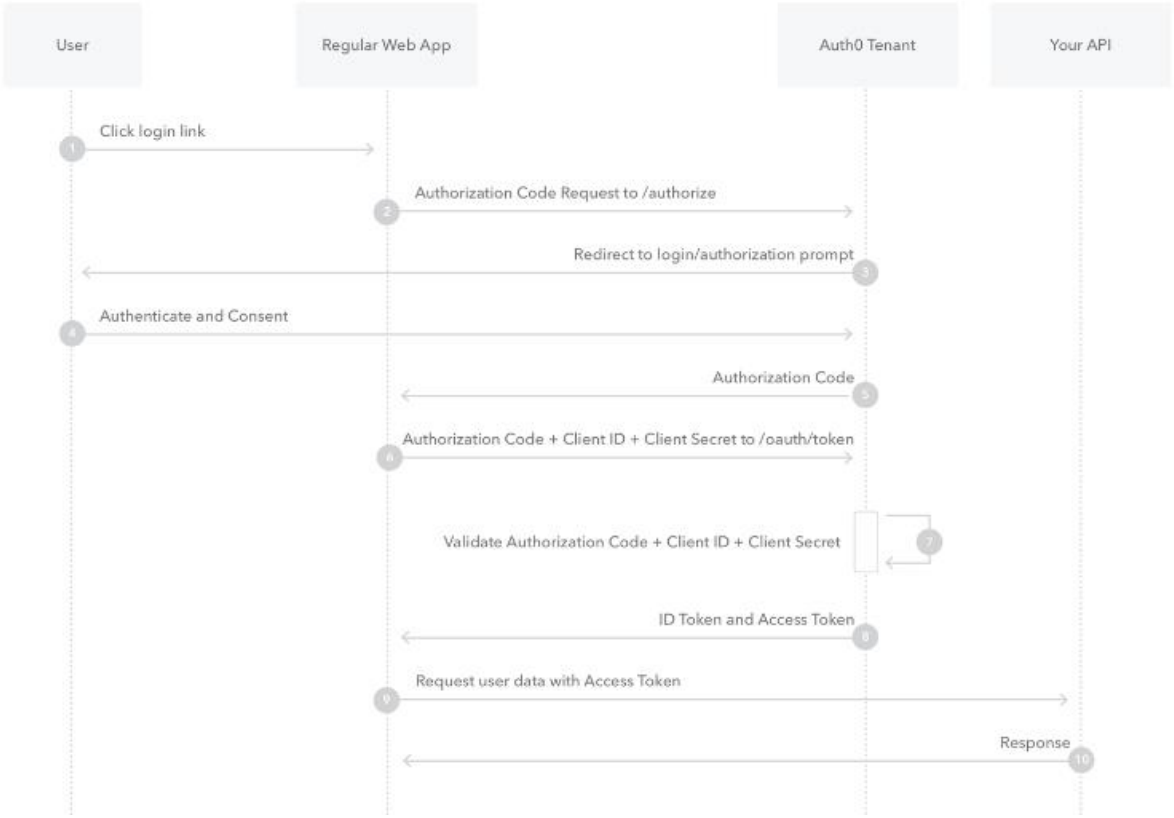
Εικόνα 14 - Ταυτοποίηση βάση token

Παρόλα τα προτερήματα της χρήσης token υπάρχουν και μερικοί περιορισμοί που πρέπει να διευθετηθούν για να μπορέσουμε να τα χρησιμοποιήσουμε σωστά. Αρχικά κάτι στο οποίο είναι υποδεέστερα σε σχέση με τα sessions είναι το μέγεθος τους το οποίο είναι συνήθως πολύ μεγαλύτερο σε σχέση με το cookie του session και κατ επέκταση απαιτεί περισσότερους πόρους από τον χρήστη για την μεταφορά του. Επιπλέον επειδή υπεύθυνος για την αποθήκευση του token είναι συνήθως ο χρήστης υπάρχει κίνδυνος υποκλοπής του από τρίτους πράγμα που σημαίνει πως ουσιαστικά αποκτούν πρόσβαση στους πόρους που έχει δικαίωμα να δει ο χρήστης καθώς για τον server το token παραμένει ενεργό.

3.5.3 OAuth2.0

Για την επίλυση των περιορισμών και των προβλημάτων που έχουν τα συστήματα ταυτοποιήσεις που χρησιμοποιούν token έχουν δημιουργηθεί διαφορά πρότυπα σχεδιασμού με ένα από τα πιο δημοφιλή να είναι το OAuth2 [12]. Η συγκεκριμένη έκδοση αυτού του προτύπου περιγράφει την έκδοση ενός access token από μια έμπιστη πηγή, το οποίο χρησιμοποιείται για την πρόσβαση στους πόρους του server και ανάλογα με τις ανάγκες της εφαρμογής μπορεί να έχει μια ημερομηνία λήξης. Θέτοντας ημερομηνία λήξης στο token μας περιορίζουμε τα προβλήματα που θα είχαμε σε περίπτωση υποκλοπής, καθώς θα ακυρωθεί μόνο του με το πέρας της. Επίσης προαιρετικά ορίζει την έκδοση ενός refresh token

το οποίο χρησιμοποιείται για την έκδοση ενός καινούργιου access token σε περίπτωση που έχει λήξει. Η διαδικασία αυτή πρέπει να πραγματοποιείτε μεταξύ client και server χωρίς να το καταλαβαίνει ο χρήστης με στόχο την καλύτερη εμπειρία του. Για την χρήση κάποιου παρόδου token μέσω OAuth2, όπως η google η το Facebook, απαιτείται αρχικά να έχουμε εγγράψει την εφαρμογή μας σε αυτούς και να έχουμε πάρει κάποια διαπιστευτήρια τα οποία θα χρησιμοποιούμε κατά την έκδοση των token. Κατά την ταυτοποίηση του χρήστη θα τον ανακατευθύνουμε στην σελίδα σύνδεσης του παρόδου που επέλεξε οπού και θα συμπληρώσει τα στοιχεία του. Ο παροχής θα μας επιστρέφεις έναν κωδικό ταυτοποίησης σε περίπτωση επιτυχούς σύνδεσης, τον οποίο θα πρέπει να στείλουμε πάλι σε αυτών μαζί με τα διαπιστευτήρια της εφαρμογής μας για να μας επιστρέψει τα tokens [21].Τα tokens αυτά μπορούν να χρησιμοποιηθούν κανονικά για την κλήση σε κάποιο API.



Εικόνα 15 - OAuth2 Authentication Flow

Μια παράλειψη του OAuth2 είναι ότι δεν περιγράφει την μορφή την οποία θα πρέπει να έχουν τα token. Σαν token μπορεί να αναφερθεί οποιοδήποτε αλφαριθμητικό περιγράφει μοναδικά έναν χρήστη . Για τον σκοπό αυτό αναπτύχθηκε το πρότυπο JSON Web Token (JWT) [18]. Το πρότυπο αυτό παρέχει μια ομάδα κανόνων για την δημιουργία ενός token με σκοπό την γενική βελτίωση όλων των απόψεων του. Μερικοί από αυτούς τους κανόνες είναι η κρυπτογραφική υπογραφή που πρέπει να φέρει κάθε token ώστε να γνωρίζουμε αν το έχει αλλάξει ο χρήστης, την κωδικοποίηση του ώστε να μην είναι αναγνώσιμο ως απλό κείμενο και την ενσωμάτωση ενός JSON αντικειμένου που θα περιέχει όλα τα στοιχεία του χρήστη που θέλουμε σε μια κανονισμένη μορφή.

3.6 Υλοποίηση συστήματος ταυτοποίησης χρηστών.

Έχοντας υπόψιν ότι αναλύσαμε παραπάνω αποφασίσαμε να δημιουργήσουμε ένα σύστημα ταυτοποίησης χρηστών βάση του πρωτοκόλλου OAuth2 ώστε να μειώσουμε την ανάγκη για υπολογιστικούς πόρους και να διασφαλίσουμε την μελλοντική επεκτασιμότητα της εφαρμογής.

3.6.1 Εφαρμογή IT API

Αρχικά έπρεπε να θέσουμε από ποιες εξωτερικές εφαρμογές θα δεχόμασταν χρήστες. Καθώς ο σκοπός της τελικής εφαρμογής είναι η διαχείριση των καθηγητών του πανεπιστήμιου δημιουργήσαμε την εφαρμογή "Οργανωμένο Σύστημα Καθηγητών" μέσω της ιστοσελίδας IT API του πανεπιστήμιου ώστε να μας παρέχει τα tokens που θα χρειαστούμε για κάθε χρήστη που θα χρησιμοποιεί το σύστημα. Επιπλέον δημιουργώντας την εφαρμογή, μας παρέχονται το client id και το client secret τα οποία θα χρησιμοποιούμε για να ανακτούμε το access token από αυτήν στην περίπτωση επιτυχούς ταυτοποίησης χρήστη. Τέλος η μονή παραμετροποίηση που χρειάζεται η εφαρμογή μας στο IT API είναι να θέσουμε το link στο οποίο επιστρέφει αφού ολοκληρώσει την ταυτοποίηση των στοιχείων.

3.6.2 Back-end

Το επόμενο βήμα είναι να παραμετροποιησουμε το back-end μας ώστε να λειτουργεί με το OAuth2. Για να το επιτύχουμε αυτό χρησιμοποιήσαμε το πακέτο Socialite της Laravel που μας προσφέρει έναν εύκολο τρόπο υλοποίησης του προτύπου. Το Socialite παρέχει ετοίμους providers για τις περισσότερες από της μεγαλύτερες εφαρμογές που παρέχουν ταυτοποίηση μέσω OAuth2, όπως η Google η το GitHub, οπού το μόνο που πρέπει να κάνουμε είναι να θέσουμε τις παραμέτρους τους στο αρχείο config/services.php. Στην δικιά μας περίπτωση όμως χρησιμοποιούμε το IT API για ταυτοποίηση και κατ' επέκταση πρέπει να δημιουργήσουμε έναν δικό μας provider ο οποίος θα υλοποιεί την αντίστοιχη λειτουργικότητα. Για τον σκοπό αυτό δημιουργήσαμε την κλαση IeeProvider ,η οποια κανει extend την κλαση Abstract Provider που μας παρέχει της μεθόδους που πρέπει να υλοποιήσουμε ώστε να λειτουργεί σωστά ο provider μας. Οι μέθοδοι που απαιτούνται θα χρησιμοποιούν στην συνέχεια από τον controller μας χωρίς να της καλούμε άμεσα εμείς, μέσω άλλων μεθόδων του socialite, και έχουν ως σκοπό την υλοποίηση βασικών βημάτων του Authorization flow του προτύπου OAuth2, όπως την ανακατεύθυνση του χρήστη στην σελίδα σύνδεσης και την ανάκτηση του access token από τον provider όταν ταυτοποιησει τον χρήστη επιτυχώς.

Αφού έχουμε πλέον ότι χρειαζόμαστε για να υλοποιήσουμε το πρότυπο OAuth2 πρέπει να δημιουργήσουμε το controller και κατ' επέκταση της μεθόδους που θα διαχειρίζονται την σύνδεση του χρήστη στην εφαρμογή μας. Το controller που δημιουργήσαμε ονομάζεται PassportAuthController και οι δυο σημαντικές μέθοδοι για την υλοποίηση του προτύπου είναι η signIn() και η redirect(). Μέσω των μεθόδων αυτών θα λάβουμε το request από την front-end εφαρμογή για την ταυτοποίηση κάποιου χρήστη, θα τον ανακατευθύνουμε στην εφαρμογή μας στο IT API για να συμπληρώσει τα στοιχεία του (χρησιμοποιώντας έμμεσα τον provider που δημιουργήσαμε για την εφαρμογή της σχολής και περιγράψαμε παραπάνω), θα λάβουμε το access token σε περίπτωση επιτυχημένης ταυτοποίησης και τελικά θα επιστρέψουμε το access token στο front-end για να χρησιμοποιηθεί σε επόμενα request.

Τέλος, για το back-end, ένα από τα πιο σημαντικά βήματα που πρέπει να υλοποιήσουμε είναι το να ασφαλίσουμε τα endpoint της εφαρμογής μας ώστε να έχουν πρόσβαση σε αυτά μόνο χρήστες που έχουν ταυτοποιηθει. Αυτό επιτυγχάνεται μέσο του middleware και πιο συγκεκριμένα μέσω του αρχείου kernel.php. οπού προσθέτουμε στην εφαρμογή μας το middleware που εμποδίζει την πρόσβαση σε πόρους αν ο χρήστης δεν είναι ταυτοποιημενος.

3.6.3 Front-end

Από την πλευρά του front-end αρχικά πρέπει να ασφαλίσουμε της σελίδες που θα προβάλουν τους προστατευμένους πόρους ώστε να μην μπορούν χρήστες που δεν έχουν ταυτοποιηθεί να τις δουν. Με την μέχρι τώρα υλοποίηση της εφαρμογής μας δεν χρειάζεται να ανησυχούμε για την προβολή

δεδομένων σε κάποιον χρήστη που δεν έχει τα ανάλογα δικαιώματα καθώς το back-end δεν θα επιστρέψει δεδομένα αν δεν του παρέχουμε ένα access token. Παρ ολ αυτά για λόγους ασφάλειας καλό θα ήταν να μην προβάλουμε την διαμόρφωση της ιστοσελίδας μας καθώς κακόβουλα άτομα μπορούν να την αντιγράψουν και να προσπαθήσουν να εξαπατήσουν χρήστες για να ανακτήσουν προσωπικά δεδομένα τους. Για τον λόγο αυτό δημιουργήσαμε το guard AuthenticationGuardService που περιέχει την μέθοδο canActivate. Η μέθοδος αυτή θα τρέχει κάθε φορά πριν αρχικοποιηθει ένα component, στο οποίο έχουμε ορίσει το guard, και θα ελέγχει μέσο της υλοποίησης μας αν υπάρχει στο local storage του προγράμματος περιήγησης το access token. Σε περίπτωση που υπάρχει θα επιτρέπει την αρχικοποίηση και προβολή του component διαφορετικά θα ανακατευθύνει τον χρήστη στην σελίδα σύνδεσης.

Επιπλέον πρέπει να υλοποιήσουμε την λειτουργικότητα που απαιτείτε για να λάβουμε ένα access token από το back-end. Για τον σκοπό αυτό δημιουργήσαμε το AuthService το οποίο μέσω ενός HttpClient καλεί την μέθοδο sign-in του backend που αναλύσαμε παραπάνω και ανακτά το access token το οποίο στην συνέχεια τοποθετούμε στο local storage του προγράμματος περιήγησης. Το AuthService καλείτε από το login component που αντιπροσωπεύει την σελίδα σύνδεσης της front-end εφαρμογής μας όταν ο χρήστης πατήσει το κουμπί σύνδεση.

Τέλος το front-end είναι υπεύθυνο για την προσθήκη του access token στα request στο back-end που επιστρέφουν προστατευμένους πόρους. Αυτό επιτυγχάνετε προσθέτοντας σε κάθε request που απαιτεί ταυτοποίηση ένα header με το όνομα Authorization και το token με την λέξη Bearer μπροστά και ένα κενό όπως υποδεικνύει το πρότυπο σχεδιασμού. Όταν το back-end δεχτεί το request αυτό θα ξέρει να ψάξει για το header αυτό, από το οποίο θα βγάλει το token και θα πραγματοποιήσει τους ελέγχους που αναλυθήκαν παραπάνω.

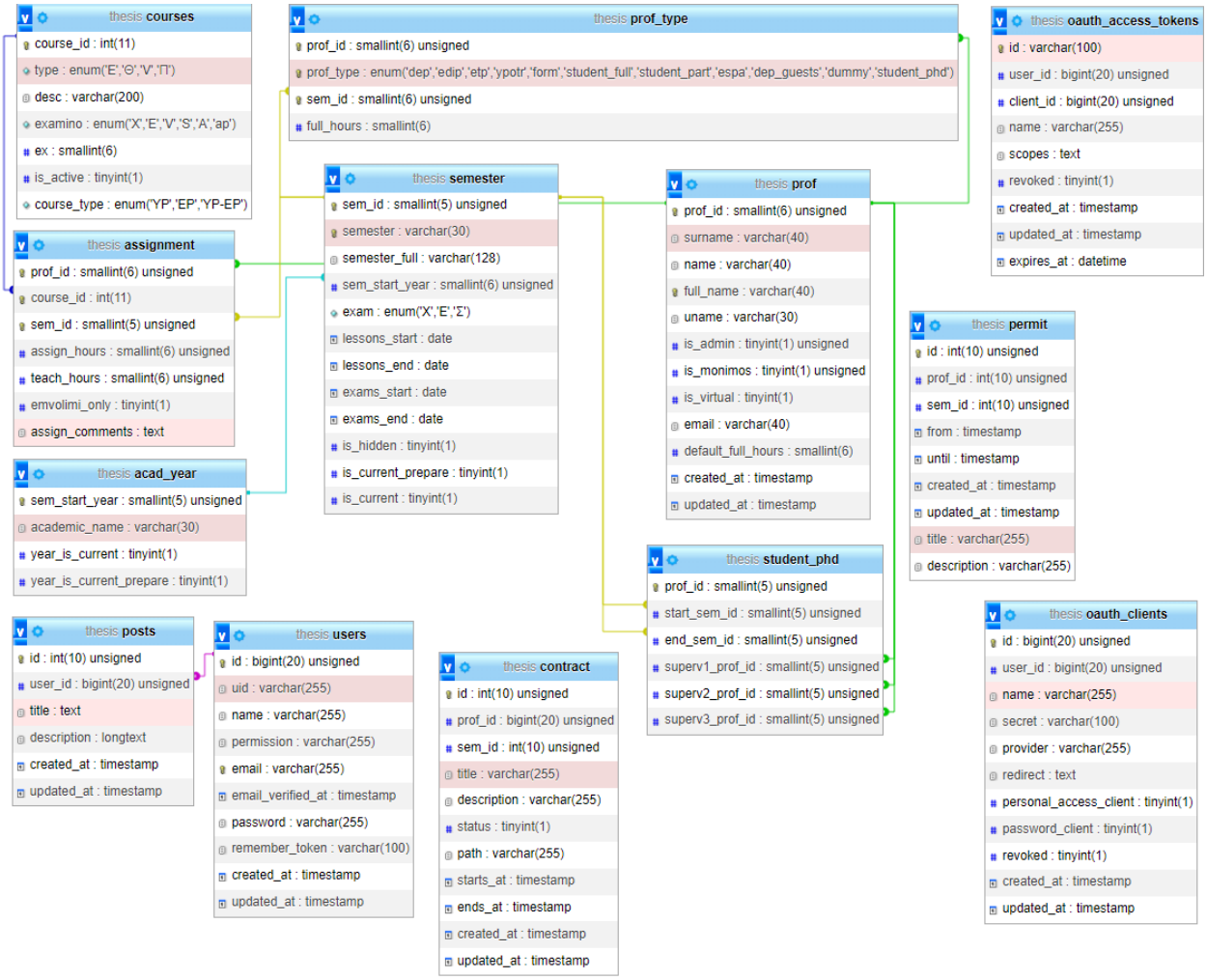
Κεφάλαιο 4ο: Βάση δεδομένων

4.1 Εισαγωγή

Οι σχεσιακές βάσεις δεδομένων οργανώνουν τα δεδομένα σε σειρές και γραμμές που μαζί αποτελούν έναν πίνακα. Τα δεδομένα είναι συνήθως αποθηκευμένα σε πόλους πίνακες οι οποίοι ενώνονται μεταξύ τους με πρωτεύων και ξένα κλειδιά. Με την χρήση αυτών τον μοναδικών χαρακτηριστικών μπορούμε να υλοποιήσουμε όλες τις σχέσεις μεταξύ των οντοτήτων.

4.2 Σχεδιάγραμμα σχέσης οντοτήτων της βάσης

Ένα διάγραμμα σχέσης οντοτήτων (ERD) είναι μια γραφική αναπαράσταση που απεικονίζει τις σχέσεις μεταξύ αντικειμένων, εννοιών ακόμα και ατόμων μέσα σε ένα σύστημα. Τα διαγράμματα αυτά είναι πολύ χρήσιμα στις βάσεις δεδομένων καθώς απεικονίζουν το πως συνδέονται οι πίνακες και το πως αποθηκεύονται τα δεδομένα πράγμα που κάνει την διαχείριση της βάσης ευκολότερη. Το παρακάτω διάγραμμα απεικονίζει τις σχέσεις των πινάκων στην βάση δεδομένων της εφαρμογής.



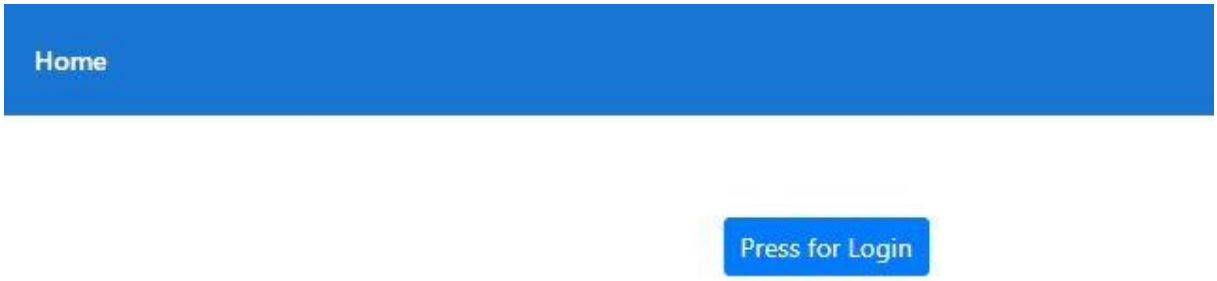
Εικόνα 16 - Σχεδιάγραμμα σχέσης οντοτήτων (ERD)

Κεφάλαιο 5ο: Παρουσίαση Εφαρμογής

5.1 Χρήστης της εφαρμογής

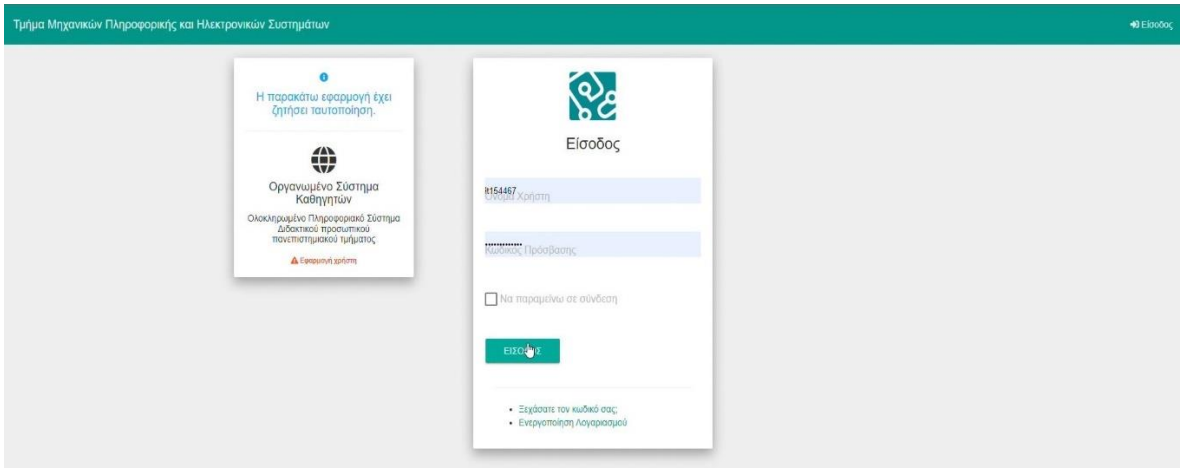
Ο χρήστης της εφαρμογής είναι ο διευθυντής του τμήματος Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων. Αυτός είναι που μπορεί να διαχειριστεί όλες τις λειτουργίες που περιγράφονται παρακάτω.

5.2 Παρουσίαση εφαρμογής client



Εικόνα 17 - Είσοδος χρήστη στην εφαρμογή

Πατώντας το κουμπί ‘Press for Login’ ο χρήστης μεταφέρετε στην επίσημη σελίδα της σχολής για να συνδεθεί με τα στοιχεία του.

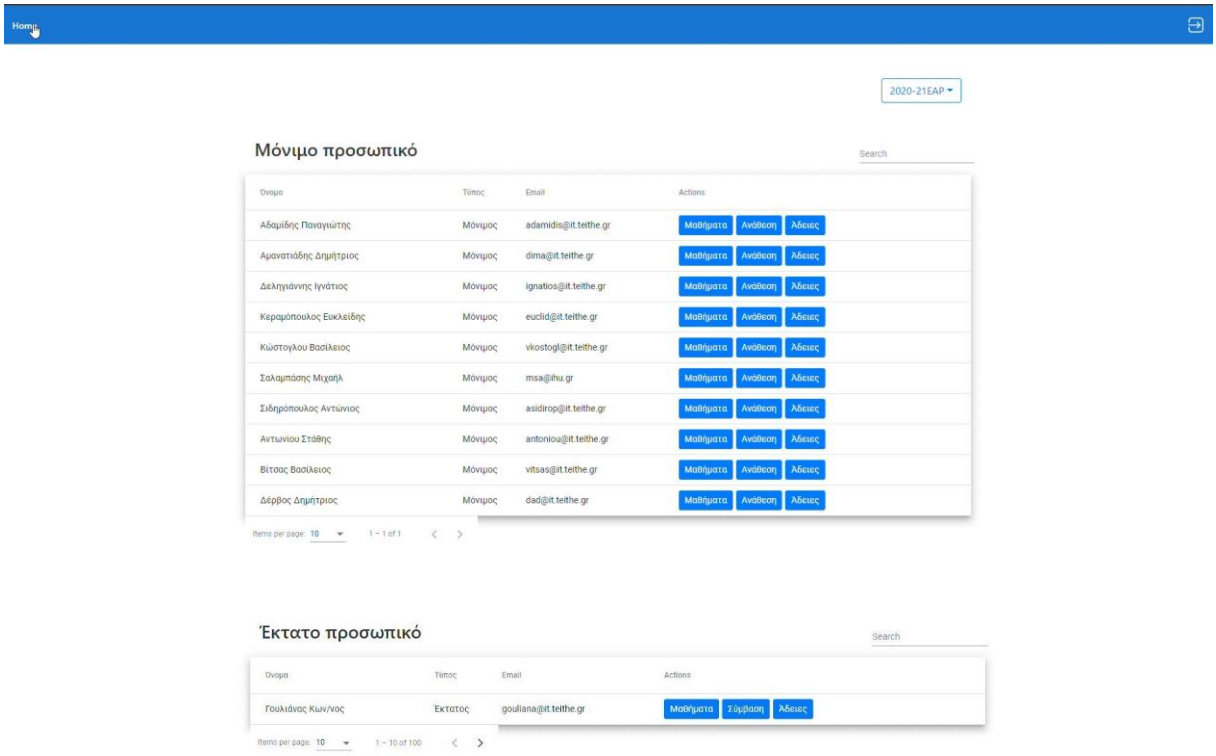


Εικόνα 18 - Ανακατεύθυνση χρήστη στην εφαρμογή "Οργανωμένο Σύστημα Καθηγητών"

Αφού συμπληρώσει τα στοιχεία του και πατήσει το κουμπί ‘ΕΙΣΟΔΟΣ’ γίνεται η ταυτοποίηση του χρήστη. Μετά από επιτυχή ταυτοποίηση γυρνάει πίσω στη σελίδα του login και πλέον μπορεί να μεταφερθεί στην κεντρική σελίδα της εφαρμογής ‘HOME’.

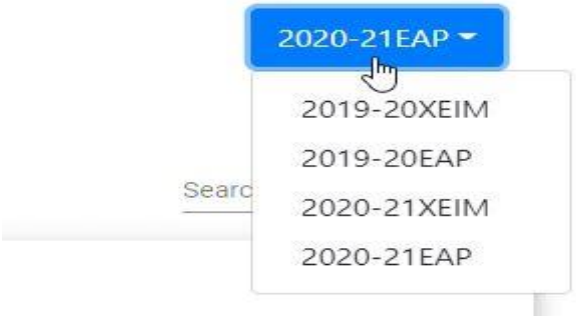
5.2.1 Κεντρική σελίδα

Στην κεντρική σελίδα της εφαρμογής ο χρήστης μπορεί να δει τους καθηγητές του τμήματος. Οι καθηγητές είναι χωρισμένοι σε δύο πίνακες. Ο πρώτος πίνακας αποτελείται απ' το μόνιμο προσωπικό του τμήματος και ο δεύτερος απ' το έκτακτο προσωπικό. Ο χρήστης μπορεί να επεξεργαστεί τα στοιχεία τους από τα κουμπιά που βρίσκονται δεξιά στον πίνακα για τον κάθε καθηγητή. Επίσης μπορεί να φιλτράρει τα αποτελέσματα βάσει του εξαμήνου, ή βάσει των στοιχείων του κάθε καθηγητή. Οι καθηγητές είναι οργανωμένοι σε σελίδες και προβάλλονται ανά 10, τιμή η οποία μπορεί να αλλάξει από τον χρήστη.



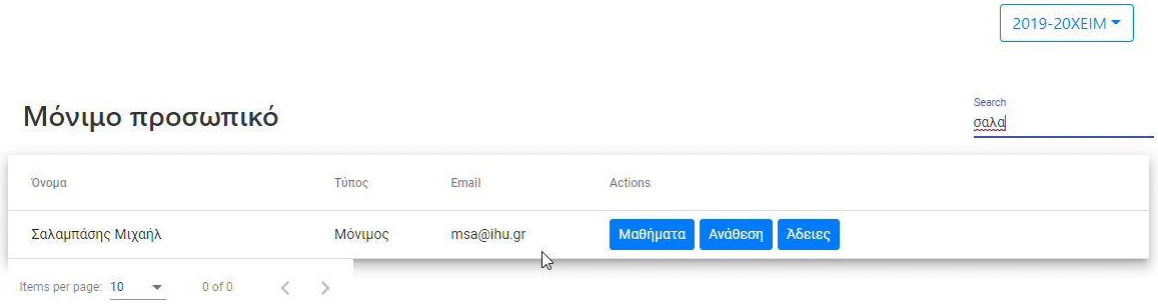
Εικόνα 19 - Κεντρική σελίδα εφαρμογής

Το φιλτράρισμα του εξαμήνου γίνεται μέσω της αναπτυσσόμενης λίστας που βρίσκεται πάνω δεξιά στην σελίδα. Έτσι, στον χρήστη προβάλλονται οι καθηγητές που αποτελούσαν το προσωπικό (Μόνιμο η Έκτακτο) εκείνο το εξάμηνο.



Εικόνα 6. Εξάμηνα

Η αναζήτηση ενός (η και πολλών) καθηγητών γίνεται από την μπάρα αναζήτησης δεξιά και πάνω από το κάθε πίνακα προσωπικού. Για παράδειγμα, αν πληκτρολογήσει ο χρήστης το γράμμα 'Κ' στην αναζήτηση κάποιου από των δύο πινάκων θα προβληθούν όλοι οι καθηγητές που τα στοιχεία τους περιέχουν το γράμμα 'Κ'.



Εικόνα 20 - Αναζήτηση καθηγητή

Για την διευκόλυνση του χρήστη οι καθηγητές προβάλλονται ανά σελίδες. Με την χρήση της μπάρας σελίδων που βρίσκεται κάτω αριστερά από τον κάθε πίνακα προσωπικού ο χρήστης μπορεί να περιηγηθεί σε όλες τις σελίδες και να δει όλους τους καθηγητές.

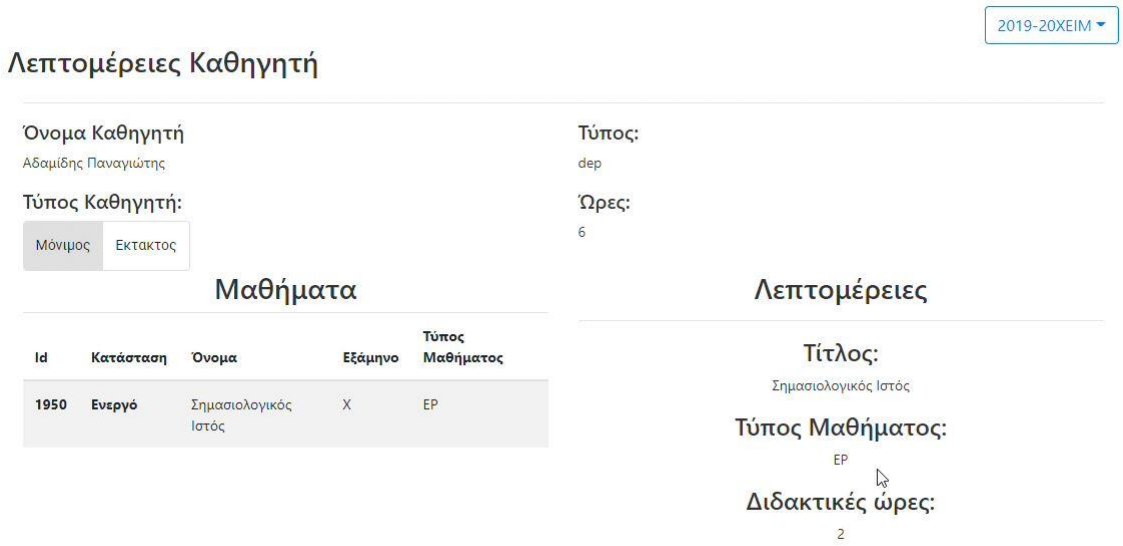
Εικόνα 21 - Paging

Το πλήθος των καθηγητών που προβάλλονται ανά σελίδα μπορεί να αλλάξει. Μέσω του αναπτυσσόμενης λίστας 'items per page'.

Εικόνα 22 - Καθηγητές ανά σελίδα

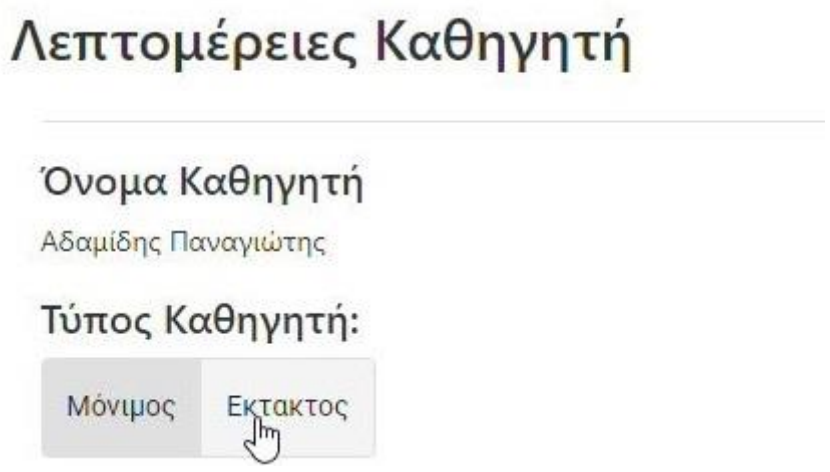
5.2.2 Λειτουργία Μαθήματα

Με το πάτημα του κουμπιού ‘Μαθήματα’ από το κεντρικό μενού ο χρήστης μεταφέρεται στην σελίδα των στοιχείων του καθηγητή που επέλεξε. Στο πάνω μέρος της σελίδας αναγράφονται τα στοιχεία του καθηγητή. Μέσω του αναπτυσσόμενου μενού που βρίσκεται πάνω δεξιά, προβάλλονται όλα τα μαθήματα που διδάσκει (η δίδαξε) εκείνο το εξάμηνο .Επιλέγοντας ένα μάθημα από την λίστα που προβάλλεται κάτω αριστερά ο χρήστης μπορεί να δει τις λεπτομέρειες του οι οποίες προβάλλονται κάτω δεξιά στην σελίδα.



Εικόνα 23 - Λεπτομερείς καθηγητή/μαθήματα

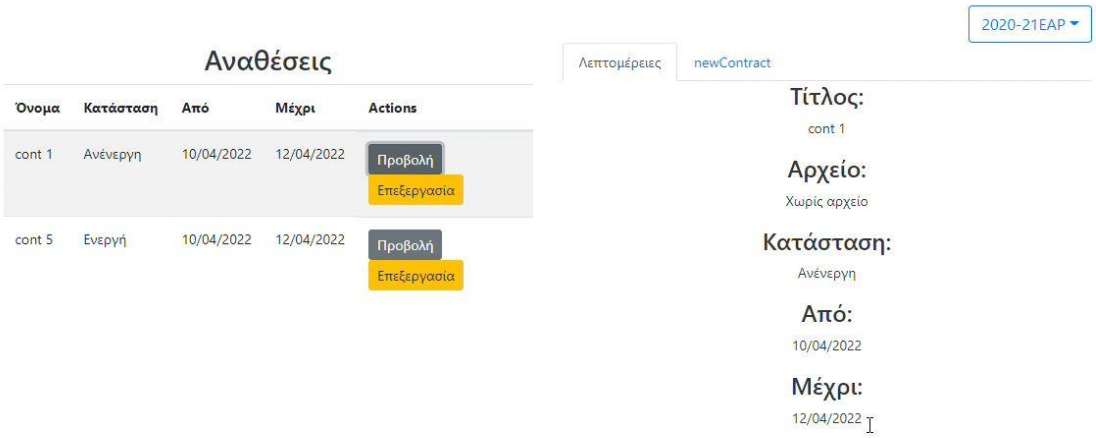
Επιπλέον δίνεται η δυνατότητα να αλλαχτεί ο τύπος του κάθε καθηγητή ανά εξάμηνο από μόνιμος σε έκτακτος και αντίστοιχα.



Εικόνα 24 - Αλλαγή είδους καθηγητή

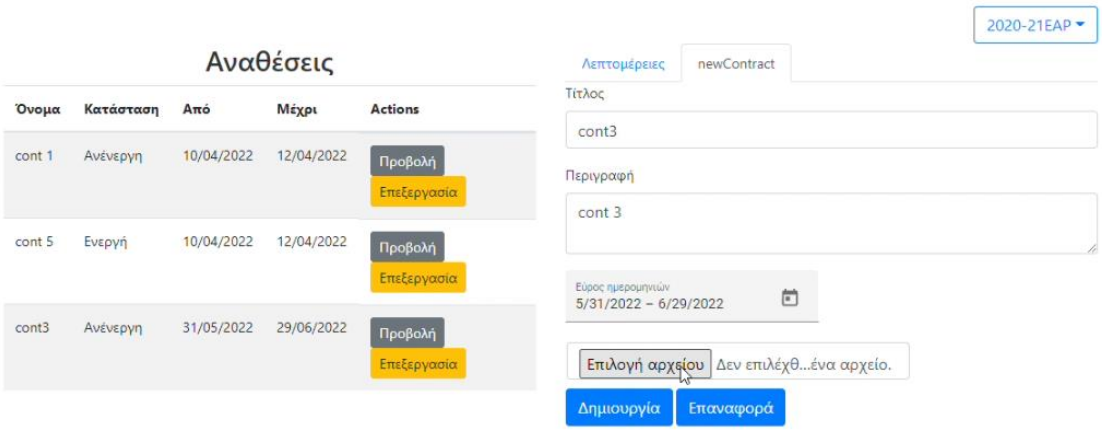
5.2.3 Λειτουργία Ανάθεση

Με το πάτημα του κουμπιού ‘Ανάθεση’ σε έναν καθηγητή από την κεντρική σελίδα εμφανίζονται στον χρήστη όλες οι αναθέσεις του σε μια λίστα αριστερά. Πατώντας προβολή σε μία από αυτές τις αναθέσεις εμφανίζονται οι λεπτομέρειες της στα δεξιά της οθόνης.



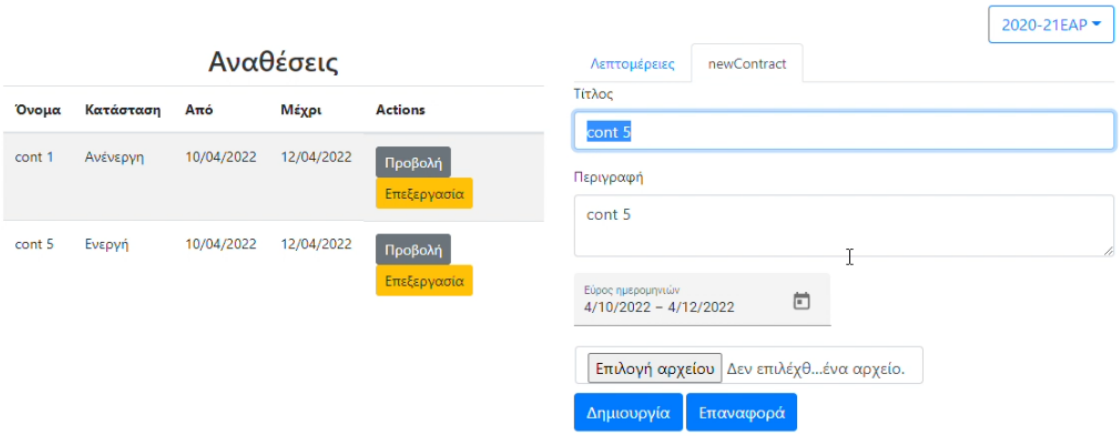
Εικόνα 25 - Αναθέσεις ενός καθηγητή

Δεξιά στην οθόνη υπάρχει η επιλογή ‘newContract’ μέσω της οποίας ο χρήστης μπορεί να κάνει μια καινούργια ανάθεση στον καθηγητή. Όταν ο χρήστης την επιλέξει θα του εμφανιστούν αρχικά τα στοιχεία της τελευταίας ενεργής του συμβάσης ειδή συμπληρωμένα στα πεδία, καθώς η πλειονότητα αυτών θα παραμείνει ιδιά και στην καινούργια σύμβαση.



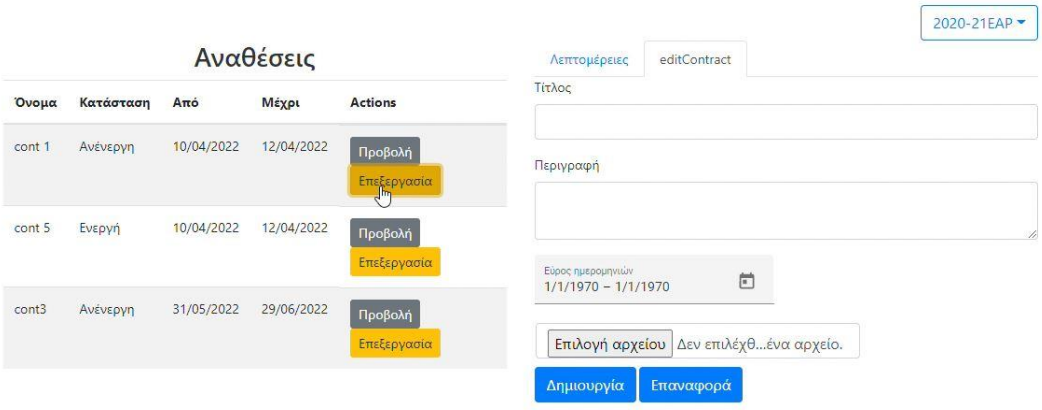
Εικόνα 26 - Αυτόματη συμπλήρωση στοιχείων συμβάσης

Σε περίπτωση εντελώς καινούργιας συμβάσης ο χρήστης πρέπει να συμπληρώσει τον τίτλο, την περιγραφή και την ημερομηνίας για την οποία θα ισχύει η ανάθεση. Επιπλέον μπορεί να ανεβάσει και ένα αρχείο σχετικό με την ανάθεση. Για την οριστικοποίηση της ανάθεσης ο χρήστης πατάει το κουμπί ‘Δημιουργία’ και σε περίπτωση που θέλει να επιστρέψει στις λεπτομέρειες πατάει το κουμπί ‘Επαναφορά’.



Εικόνα 27 - Νέα ανάθεση

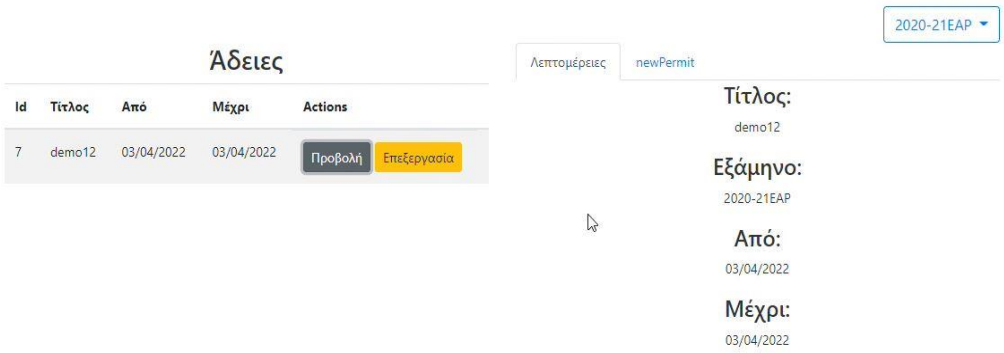
Μετά την επιτυχής δημιουργία ανάθεσης στον καθηγητή, το αποτέλεσμα εμφανίζεται μαζί με όλες τις αναθέσεις στην λίστα αριστερά στην οθόνη. Πατώντας το κουμπί ‘Επεξεργασία’ σε μία ανάθεση ανοίγει δεξιά μια σελίδα παρόμοια με αυτήν της δημιουργίας ανάθεσης και μπορεί ο χρήστης να αλλάξει τα στοιχεία της.



Εικόνα 28 - Επεξεργασία ανάθεσης

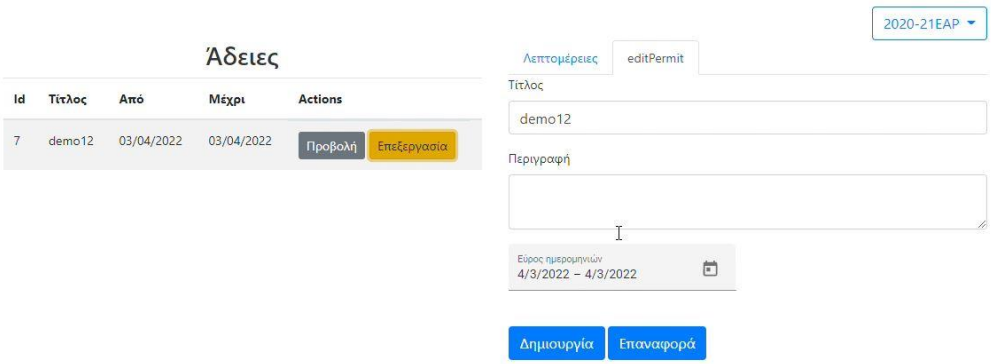
5.2.4 Λειτουργία Άδειες

Στην επιλογή ‘ΑΔΕΙΕΣ’ για έναν καθηγητή από τη κεντρική σελίδα προβάλλεται η σελίδα των αδειών του ανά εξάμηνο. Πατώντας το κουμπί ‘Προβολή’ ο χρήστης μπορεί να δει τις λεπτομέρειες της άδειας του καθηγητή ανά εξάμηνο στα δεξιά της οθόνης



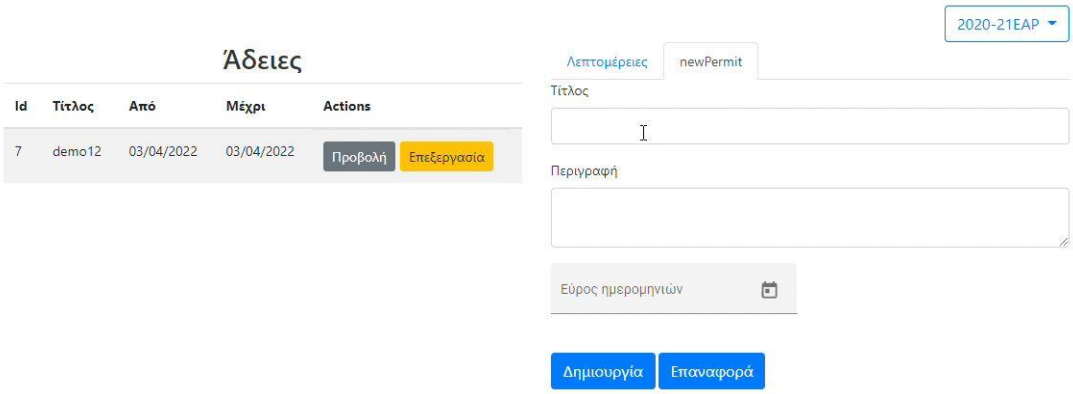
Εικόνα 29 - Αδεείς καθηγητή

Πατώντας το κουμπί επεξεργασία στην άδεια του καθηγητή ο χρήστης μπορεί να αλλάξει τα στοιχεία της άδειας του.



Εικόνα 30 - Επεξεργασία αδείας

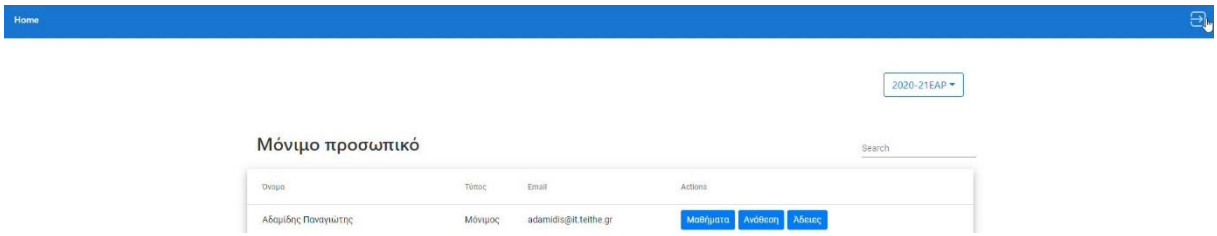
Επίσης, υπάρχει και η λειτουργία προσθήκη άδειας στα δεξιά της οθόνης. Ο χρήστης καταχωρεί τα στοιχεία της άδειας και της ημερομηνίες έναρξης και λήξης. Πατώντας το κουμπί ‘Δημιουργία’ η άδεια προστίθεται στην λίστα αριστερά.



Εικόνα 31 - Δημιουργία αδείας

5.2.5 Αποσύνδεση από το σύστημα

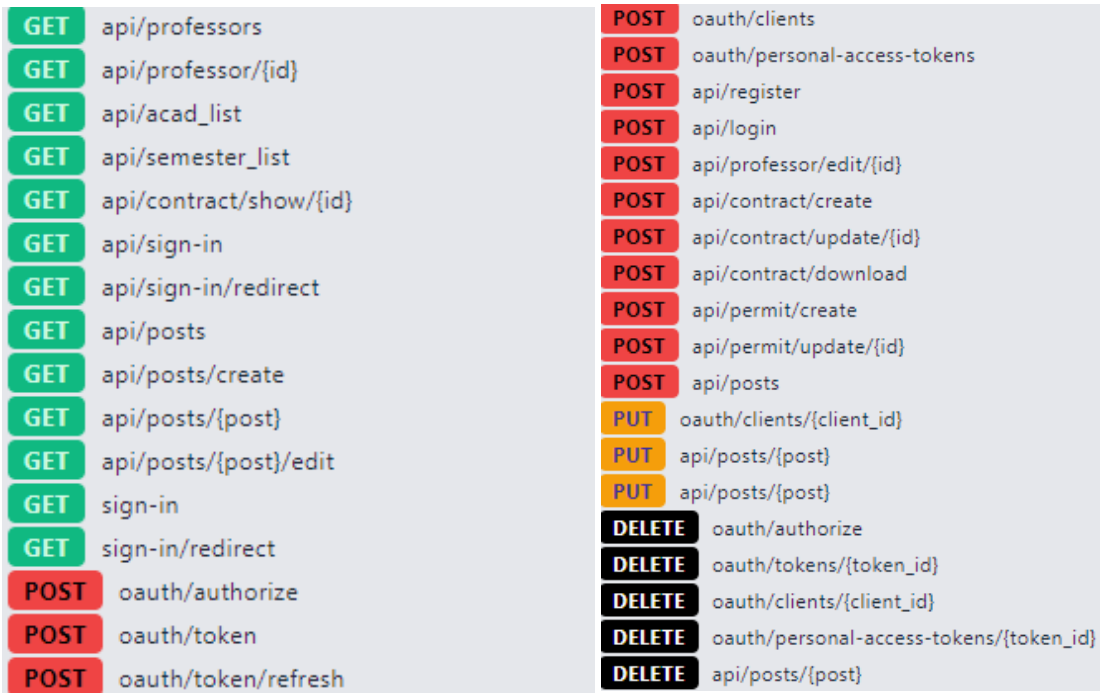
Τέλος, ο χρήστης μπορεί να αποσυνδεθεί πατώντας το κουμπί Αποσύνδεσης που βρίσκεται πάνω δεξιά στην οθόνη.



Εικόνα 32 - Αποσύνδεση από το σύστημα

5.3 Παρουσίαση των endpoint του API

Όπως αναφέραμε στο κεφαλαίο 2, είναι σημαντικό για ένα API να παρέχει ένα πλήρες και εμπεριστατωμένο έγγραφο που θα περιγράφει τις παροχές και τις δυνατότητες του. Με αυτόν τον τρόπο μπορεί κάποια άλλη εφαρμογή πελάτη που χρειάζεται τα ιδιά δεδομένα , για παράδειγμα μια εφαρμογή για κινητά τηλεφώνα, να ξεκινήσει άμεσα την υλοποίηση του client γνωρίζοντας ότι έχει τα δεδομένα που χρειάζεται έτοιμα μέσω του API μας. Το έγγραφο αυτό θα πρέπει να παρουσιάζει το κάθε endpoint μας ξεχωριστά εξηγώντας τα ζητούμενα που μπορεί να έχει, όπως κάποιο parameter η ότι απαιτεί ταυτοποίηση, αλλά και το είδος της Http μεθόδου στην οποία λειτουργεί όπως GET,POST κτλ. Στην μέρες όμως ένα έγγραφο επεξήγησης API δεν καταναλώνονται μόνο από ανθρώπους αλλά και από άλλες οι εφαρμογές οι οποίες μπορούν να το αξιολογήσουν και στην συνέχεια να χρησιμοποιήσουν το API αυτόματα. Για τον σκοπό αυτό έχει δημιουργηθεί η προδιαγραφή OpenAPI, που περιγράφει τον τρόπο με τον οποίο ένα API πρέπει να δημιουργηθεί, να καταναλωθεί και να προβληθεί. Για την εξαγωγή των endpoint της εφαρμογής μας χρησιμοποιήσαμε το πακέτο laravel-request-docs [15] το οποίο ακολουθεί την προδιαγραφή OpenAPI και μας παρέχει μια Html σελίδα με όλα τα endpoint μας και την δυνατότητα να τα τρέξουμε αλλά και ένα json με όλα τα endpoint για κατανάλωση από άλλες εφαρμογές.



Εικόνα 33 - Τα endpoint την εφαρμογής

Η πλήρης λίστα μαζί με το json μπορούν να βρεθούν στα παραδοτέα της εργασίας μας.

Κεφάλαιο 6ο: Συμπεράσματα και προτάσεις βελτίωσης

Όπως μπορούμε να δούμε, η δημιουργία μιας εφαρμογής που υποστηρίζει και βελτιώνει τις υπάρχον ανάγκες ενός πανεπιστημίου είναι δύσκολη υπόθεση. Πρέπει να λύσουμε τα υπάρχον προβλήματα με έναν αποτελεσματικό τρόπο προσφέροντας παράλληλα μια πλατφόρμα πάνω στην οποία μπορούν προστεθούν υπηρεσίες για μελλοντικές ανάγκες. Τα ήδη υπάρχον προβλήματα όπως της ανάθεσης μαθημάτων και της διαχείρισης αδειών μπορούν να λυθούν γρήγορα, οργανωμένα και πιο εύκολα μέσω της πλατφόρμας που δημιουργήσαμε. Πλέον ο πρόεδρος ενός τμήματος μπορεί να διαχειριστεί εύκολα και γρήγορα τους καθηγητές, τα μαθήματα και τις άδειες τους από τον υπολογιστή του.

Όσον αφορά τις μελλοντικές επεκτάσεις, ποικίλουν. Από την υλοποίηση παραπάνω ψηφιακών υπηρεσιών για τον ίδιο τον πρόεδρο του τμήματος μέχρι και την αλληλεπίδραση φοιτητών με τους καθηγητές. Ο πρόεδρος του τμήματος μπορεί να έχει παραπάνω υπηρεσίες όπως timesheet, πρόγραμμα για ραντεβού με καθηγητές, γραμματεία και φοιτητές. Τις ίδιες ακριβώς λειτουργίες μπορεί να έχει και ένας καθηγητής, έχοντας είσοδο στην πλατφόρμα με πιο περιορισμένα δικαιώματα από τον πρόεδρο του τμήματος όπως είναι λογικό. Αντίστοιχα, ένας φοιτητής μπορεί να γίνει χρήστης ενός τέτοιου συστήματος βλέποντας τους καθηγητές ανά εξάμηνο και κλείνοντας κάποιο ραντεβού μαζί τους. Επεκτάσεις και περιθώρια βελτίωσης πάντα θα υπάρχουν. Για αυτό και δημιουργήσαμε αυτή την web εφαρμογή στην οποία μπορούν να εφαρμοστούν και νέες ηλεκτρονικές υπηρεσίες. Επίσης διάφοροι τύποι χρηστών πέρα από τον πρόεδρο του τμήματος θα μπορούν να συνδεθούν και να βλέπουν ο καθένας ότι τους αφορά και ότι τους είναι χρήσιμο.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] C. Musciano, B. Kennedy, *HTML & XHTML: The Definitive Guide 5th Edition*. Sebastopol, CA: O'Reilly Media, 2002

[2] E. A. Meyer, *CSS: The Definitive Guide 3rd Edition.* Sebastopol. CA: O'Reilly Media, 2007

[3] D. Crockford, *JavaScript: The Good Parts*. Sebastopol CA: O'Reilly Media, 2008

[4] J. R. Groff, P. N. Weinberg, *SQL The Complete Reference*. New York: Osborne/McGraw-Hill, 1999

[5] W. J. Gilmore, *Beginning PHP and MySQL: From Novice to Professional*. New York: Springer Science & Business Media, 2010

[6] L. Richardson & S. Ruby, *RESTfull Web Services*. Sebastopol CA: O'Reilly Media, 2007

[7] S. McCool, *Laravel Starter The definitive introduction to the Laravel PHP web development framework*. Birmingham: Packt Publishing Ltd, 2012. Available

https://data.dammio.com/laravel_starter.pdf

[8] G. Bierman, M. Abadi & M. Torgersen, *Understanding TypeScript*. Berlin: Springer-Verlag, 2014

[9] S. Goyal, D. Batra, "ANGULAR JS". International Research Journal of Modernization in Engineering Technology and Science 2022, vol.04, no. 6, June-2022. Available

https://www.irjmets.com/uploadedfiles/paper/issue_6_june_2022/26286/final/fin_irjmets1655469952.pdf

[10] Dr. S. Singh, "Comparative Study of MVC (Model View Controller) Architecture". International Journal of Computer Science Engineering (IJCSE) 2016, vol 5, no. 3, May 2016. Available

http://www.ijcse.net/docs/IJCSE16-05-03-090.pdf

[11] Y. Balaj, "Token-Based vs Session-Based Authentication" University of Prishtina "Hasan Prishtina" Faculty of Electrical and Computer Engineering Prishtina, Kosovo, 2017. Available

https://www.researchgate.net/publication/320068250_Token-Based_vs_Session-Based_Authentication_A_survey

[12] OAuth 2.0

https://oauth.net/2/?fbclid=IwAR3jkpcIBn0Qx--24QPCeqVjDhKD33i0urkv083Y_nuT3SgxG3x104rpYao [Accessed: 16-Ιουνίου-2022]

[13] Composer Documentation

https://getcomposer.org/doc/ [Accessed: 20-Μάιου-2022]

[14] Angular Documentation.

https://angular.io/docs [Accessed: 10-Ιουνίου-2022]

[15] Automated API documentation.

https://github.com/rakutentech/laravel-request-docs [Accessed: 29-Αυγούστου-2022]

[16] Laravel ERD diagram generator.

https://github.com/beyondcode/laravel-er-diagram-generator [Accessed: 30-Αυγούστου-2022]

[17] JSON

https://www.rfc-editor.org/rfc/rfc4627.html [Accessed: 02-Ιουνίου-2022]

[18] JWT

https://www.rfc-editor.org/rfc/rfc7519 [Accessed: 20-Ιουνίου-2022]

[19] Laravel

https://laravel.com/ [Accessed: 07-Μάιου-2022]

[20] Integrated Development Environment

https://press.rebus.community/programmingfundamentals/chapter/integrated-development-environment/ [Accessed: 06 Μάιου 2022]

[21] https://auth0.com/docs/get-started/authentication-and-authorization-flow/authorization-code-flow [Accessed: 17-Ιουνίου-2022]

ΠΑΡΑΡΤΗΜΑ Α : Κώδικας

Η μέθοδος ngOnInit(), της οποία τον σκοπό περιγράφουμε στο κεφ. 3.2, μέσο διεργασιών παίρνει τα δεδομένα από το back-end και τα φέρνει στην επιθυμητή μορφή για να προβληθούν στον χρήστη. Η διαδικασία αυτή γίνεται κατά την δημιουργία του component.

Κώδικας 1 – Μέθοδος ngOnInit

```
ngOnInit(): void {

    this.dataSourceMonimoi.paginator = this.paginator;
    this.dataSourceEktaktoi.paginator = this.paginator;

    this._professorsService.getProfessors()
      .subscribe((res) => {
        res.forEach(item => {
          let sem = item.semesters
          sem.forEach(sem_item => {
            let sem_id = sem_item['sem_id']
            if (this.semester_list[sem_id] === undefined) {
              this.semester_list[sem_id] = sem_item['semester'];
            }
          })
        })
        this.semester_list.forEach((val, index) => {
          let arr = [];
          arr['sem_id'] = index;
          arr['sem'] = val;
          this.sem_list.push(arr);
        })

        this.selectedAcadYear = this.sem_list[this.sem_list.length - 1].sem_id
        this.selectedAcadYearName = this.sem_list[this.sem_list.length - 1].sem;

        res.forEach(elem => {
          let sems = elem.semesters
          sems.forEach(elem1 => {
            if (elem1['sem_id'] == this.selectedAcadYear && elem.is_monimos == 1) {
              this.responseDataMonimoi.push(elem);
            } else if (elem1['sem_id'] == this.selectedAcadYear && elem.is_monimos == 0) {
              this.responseDataEktaktoi.push(elem);
            }
          })
        })
        this.dataSourceMonimoi.data = this.responseDataMonimoi;
        this.dataSourceEktaktoi.data = this.responseDataEktaktoi
      })
```

Αναφέρεται στο κεφ. 3.2 οπού εξηγούμε της διαφορετικές μορφές με τις οποίες γίνεται το data-binding μεταξύ html και component όπως με “[όνομα HTML χαρακτηριστικού] = όνομα μεταβλητής” και {{όνομα μεταβλητής}}

Κώδικας 2 - mat-table

```
<table mat-table class="mat-elevation-z8" [dataSource]="dataSourceMonimoi" style="width:100%">

   <ng-container matColumnDef="name">
       <th mat-header-cell *matHeaderCellDef>Όνομα</th>
       <td mat-cell *matCellDef="let prof">{{prof.full_name}}</td>
   </ng-container>

   <ng-container matColumnDef="monimos">
       <th mat-header-cell *matHeaderCellDef>Τύπος</th>
       <td mat-cell *matCellDef="let prof">{{prof.is_monimos == 1 ? 'Μόνιμος' : 'Έκτατος'}}</td>
   </ng-container>
```

Αναφέρεται στο κεφ. 3.2 οπού αναλύουμε τον τρόπο με το οποίο γίνεται το two-way-data-binding “[(όνομα χαρακτηριστικού)] = όνομα μεταβλητής”

Κώδικας 3 - Two way data binding

```
<form #contractForm="ngForm" (ngSubmit)="onSubmit(contractForm);" novalidate>
            <div class="form-group d-none">
            <input type="text" name="prof_id" class="form-control"
[(ngModel)]="contractModel.prof_id">
            </div>
          <div class="form-group">
              <label>Τίτλος</label>
              <input type="text" name="title" class="form-control"
[(ngModel)]="contractModel.title">
            </div>
```

Αναφέρεται στο κεφ. 3.2 οπού περιγράφουμε πως τα component της εφαρμογής μας αντιστοιχίζονται με URL και τον σκοπό του router στην angular.

Κώδικας 4 – Routes της εφαρμογής μας

```
const routes: Routes = [
  {
    path: 'professors',
    loadChildren: () => import('./professors/professors.module').then(m => m.ProfessorsModule)
  },
  {
    path: 'login',
```

```
    loadChildren: () => import('./login/login.module').then(m => m.LoginModule)
  },
  { path: 'professor/:id', component: ProfessorDetailComponent },
  { path: 'contracts/:id', component: ContractDetailsComponent },
  { path: 'permits/:id', component: PermitDetailsComponent },
  { path: 'courses', component: CoursesComponent },
  { path: "**", component: PageNotFoundComponent }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
export const routingComponents = [
  PageNotFoundComponent,
  ProfessorDetailComponent,
  ContractDetailsComponent
]
```

Αναφέρεται στο κεφ. 3.3 οπού περιγράφουμε τον τρόπο με τον οποίο αντιστοιχούμε τις μεθόδους των controller μας με τα URL στα οποία θα ακούνε για request.

Κώδικας 5 – api.php

```
<?php

use Illuminate\Http\Request;
use Illuminate\Support\Facades\Route;
use App\Http\Controllers\PassportAuthController;
use App\Http\Controllers\ProfessorController;
use App\Http\Controllers\AcademyYearController;
use App\Http\Controllers\ContractController;
use App\Http\Controllers\PermitController;
use App\Http\Controllers\SemesterController;


Route::post('register', [PassportAuthController::class, 'register']);
Route::post('login', [PassportAuthController::class, 'login']);
Route::middleware('auth:api')->get('professors', [ProfessorController::class, 'index']);
Route::get('professor/{id}', [ProfessorController::class, 'fetchById']);
Route::middleware('auth:api')->post('professor/edit/{id}', [ProfessorController::class,
'editType']);
Route::get('acad_list', [AcademyYearController::class, 'index']);
Route::middleware('auth:api')->get('semester_list', [SemesterController::class, 'index']);
Route::post('contract/create',[ContractController::class, 'store']);
Route::middleware('auth:api')->get('contract/show/{id}',[ContractController::class, 'show']);
Route::post('contract/update/{id}',[ContractController::class, 'update']);
```

```
Route::post('contract/download',[ContractController::class, 'download']);
Route::post('permit/create',[PermitController::class, 'store']);
Route::post('permit/update/{id}',[PermitController::class, 'update']);
Route::get('sign-in', [PassportAuthController::class, 'signIn'])->name('login');
Route::get('/sign-in/redirect', 'Auth\AuthController@redirect');
Route::middleware('auth:api')->group(function () {
  Route::resource('posts', PostController::class);
});
```

Αναφέρεται στο κεφ. 3.3 οπού περιγράφουμε το middleware του back-end. Σκοπός αυτής της μεθόδου είναι να ελέγξει αν ένας χρήστης είναι είδη ταυτοποιημένος όπου και θα τον ανακατευθύνει στην κεντρική σελίδα του front-end.

Κώδικας 6 – RedirectIfAuthenticated.php

```
<?php

namespace App\Http\Middleware;

use App\Providers\RouteServiceProvider;
use Closure;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;

class RedirectIfAuthenticated
{
    public function handle(Request $request, Closure $next, ...$guards)
    {
        $guards = empty($guards) ? [null] : $guards;

        foreach ($guards as $guard) {
            if (Auth::guard($guard)->check()) {
                return redirect(RouteServiceProvider::HOME);
            }
        }

        return $next($request);
    }
}
```

Αναφέρεται στο κεφ. 3.3 και δείχνει την μορφή του κώδικα που χρησιμοποιείται ώστε να μετατραπεί από το ORM σε SQL

Κώδικας 7 – Query χωρίς SQL

```
$professor = Professor::where('prof_id','=' ,$id)->update([
            "is_monimos" => $request->state
```

```
        ]);
```

Αναφέρεται στο κεφ. 3.3 και δείχνει τον τρόπο με τον οποίο θέτουμε την σχέση hasMany μεταξύ των model μας .

Κώδικας 8 – Παράδειγμα hasMany()

```
return $this->hasMany(Permit::class, 'prof_id', 'prof_id');
```

Αναφέρεται στο κεφ. 3.6.2 οπού περιγράφουμε την διαδικασία που χρειάζεται να γίνει για να δουλέψει το πακέτο Socialite με τον OAuth2 provider που δημιουργήσαμε μέσω του site της σχολής μας. Συγκεκριμένα στην κλάση αυτή υλοποιούμε της μεθόδους που ορίζει η AbstractProvider

Κώδικας 9 – IeeProvider.php

```
<?php

namespace App\Auth\Social\Two;

use Laravel\Socialite\Two\AbstractProvider;
use Laravel\Socialite\Two\ProviderInterface;
use Laravel\Socialite\Two\User;
use Illuminate\Support\Str;


class IeeProvider extends AbstractProvider implements ProviderInterface {

    protected $scopes = ['profile'];

    public function getAuthUrl($state)
    {
        return $this->buildAuthUrlFromBase('https://login.iee.ihu.gr/authorization', $state);
    }

    public function getTokenUrl()
    {
        return 'https://login.iee.ihu.gr/token';
    }

    public function getUserByToken($token)
    {
        $userUrl = 'https://api.iee.ihu.gr/profile?access_token=' . $token;

        $response = $this->getHttpClient()->get(
            $userUrl, ['headers' => [
                'Accept' => 'application/json',
            ],
        ]);
```

```
        $user = json_decode($response->getBody(), true);

        return $user;
    }

    public function mapUserToObject(array $user)
    {
        //dd('map');
        $group = $user['eduPersonAffiliation'];
        $name_gr = $group === "staff" ? $user['cn;lang-el'] : Str::upper($user['cn;lang-el']);
        $name_eng = !empty(Str::title($user['cn'])) ? Str::title($user['cn']) :
Str::ascii($user['cn;lang-el']);
        $is_author = $group === "staff";
        $email = $user['mail'];

        return (new User)->setRaw($user)->map([
            'uid' => $user['uid'],
            'name' => $name_gr,
            'name_eng' => $name_eng,
            'email' => $email,
            'is_author' => $is_author
        ]);
    }

}
```

Αναφέρεται στο κεφ. 3.6.3 οπού περιγράφουμε τον τρόπο που διασφαλίζουμε της ιστοσελίδες μας από μη τακτοποιημένους χρήστες.

Κώδικας 10 – AuthGuardService.ts

```
import { Injectable } from '@angular/core';
import { TestBed } from '@angular/core/testing';
import { Router, ActivatedRouteSnapshot, RouterStateSnapshot } from '@angular/router';

@Injectable({
  providedIn: 'root'
})
export class AuthGuardService {

  constructor(
    private router: Router
  ) { }

  canActivate(
    next: ActivatedRouteSnapshot,
```

```
    state: RouterStateSnapshot
  ) {

    if (localStorage.getItem('access_token')) {
      return true;
    }

    localStorage.removeItem('access_token');
    this.router.navigateByUrl('/login');
    return false;
  }
}
```

Αναφέρεται στο κεφ. 3.4.1 οπού περιγράφουμε αναλυτικά τις λειτουργίες κάθε μεθόδου και τα URL στα οποία αντιστοιχούν.

Κώδικας 11 – ProfessorController.php

```
class ProfessorController extends Controller
{
    public function index()
    {
        $list = Professor::with('semesters', 'academy')->get();

        return response()->json($list);
    }

    public function fetchById($id)
    {
        $professor = Professor::with('semesters', 'courses', 'contracts', 'permits')-
>where('prof_id', $id)->first();

        return response()->json($professor);
    }

    public function editType(Request $request, $id){

        $professor = Professor::where('prof_id','=' ,$id)->update([
            "is_monimos" => $request->state
        ]);
        return response()->json($professor);
    }

}
```

Αναφέρεται στο κεφ. 3.4.2 οπού περιγράφουμε αναλυτικά τις λειτουργίες κάθε μεθόδου και τα URL στα οποία αντιστοιχούν.

Κώδικας 12 - AcademyYearController.php

```
class AcademyYearController extends Controller
{

    public function index()
    {
        $list = AcademyYear::all();

        return response()->json($list);
    }

}
```

Αναφέρεται στο κεφ. 3.4.3 οπού περιγράφουμε αναλυτικά τις λειτουργίες κάθε μεθόδου και τα URL στα οποία αντιστοιχούν.

Κώδικας 13 - ContractController.php

```
class ContractController extends Controller
{

    public function store(Request $request)
    {
        if(isset($request->prof_id)){
            $contracts = new Contract;
            $contracts->prof_id = $request->prof_id;
            $contracts->sem_id = $request->sem_id;
            $contracts->title = (isset($request->title) ? $request->title : null);
            $contracts->description = (isset($request->description) ? $request->description : null);
            $contracts->status = (isset($request->status) ? $request->status : false);
            $contracts->path = (isset($request->path) ? $request->path : '');
            $contracts->starts_at = (isset($request->starts_at) ? date('Y-m-d',strtotime($request-
>starts_at)) : null);
            $contracts->ends_at = (isset($request->ends_at) ? date('Y-m-d',strtotime($request-
>ends_at)) : null);

            $start = $request->starts_at;
            $end = $request->ends_at;
            $now = date('Y-m-d');

            if ($now >= $start && $now <= $end){
                $contracts->status = 1;
            }else{
                $contracts->status = 0;
```

```
            }

            $contracts->save();
            return response()->json($contracts);
        }

    }

    public function show($id)
    {
        $contract = Contract::find($id);

        $contract->starts_at = date('Y-m-d',strtotime($contract->starts_at));
        $contract->ends_at = date('Y-m-d',strtotime($contract->ends_at));


        return response()->json($contract);
    }

    public function update(Request $request, $id)
    {
        $contract = Contract::find($id);

        $contract->starts_at = date('Y-m-d',strtotime($request->starts_at));
        $contract->ends_at = isset($request->ends_at) ? date('Y-m-d',strtotime($request->ends_at)) :
null;

        $start = date('Y-m-d',strtotime($request->starts_at));
        $end = isset($request->ends_at) ? date('Y-m-d',strtotime($request->ends_at)) : null;

        $now = date('Y-m-d');

        if ($now >= $start && $now <= $end){
            $contract->status = 1;
        }else{
            $contract->status = 0;
        }
        $image = $request->path;  // your base64 encoded

        if($image != null) {

            if($contract->path != '')
                Storage::delete($contract->path);

            $extension = explode('/',explode(',',$image)[0]);
            $extension = str_replace(';base64','',$extension[1]);

            $allowedfileExtension=['pdf','jpg','png', 'PNG'];
```

```
            //$files = $request->file('fileName');
            $errors = [];


            $check = in_array($extension,$allowedfileExtension);

            if($check) {
                // $path = $request->fileName->store('public/contract');
                // $name = $request->fileName->getClientOriginalName();
                $image = str_replace(explode(',',$image)[0].',', '', $image);
                $image = str_replace(' ', '+', $image);
                $imageName = uniqid() . '.png';
                Storage::disk('contract')->put($imageName, base64_decode($image));
                $contract->path = 'public/contract/'.$imageName;



                $contract->update((array) $contract);
            } else  {
                return response()->json(['invalid_file_format'], 422);
            }
        }
        //$input = $request->all()->except(['path', 'region_id']);
        unset($contract['path']);
        $contract->update((array) $contract);

        return response()->json($contract);

    }

    public function download(Request $request){
        $file = substr($request->all()['path'], strrpos($request->all()['path'], '/') + 1);
        $path = storage_path('app/' . $request->all()['path']);
        return response()->download($path, $file);
    }
}
```

Αναφέρεται στο κεφ. 3.4.4 οπού περιγράφουμε αναλυτικά τις λειτουργίες κάθε μεθόδου και τα URL στα οποία αντιστοιχούν.

Κωδικας 14 - PermitController.php

```
class PermitController extends Controller
{

    public function store(Request $request)
    {
```

```
        if(isset($request->prof_id)){
            $permit = new Permit;
            $permit->prof_id = $request->prof_id;
            $permit->title = $request->title;
            $permit->sem_id = $request->sem_id;

            if(isset($request->description)){
                $permit->description = $request->description;
            }

            $permit->from = date('Y-m-d',strtotime($request->from));

            if(isset($request->until)){
                $permit->until = date('Y-m-d',strtotime($request->until));
            }


            $permit->save();

            return response()->json($permit);
        }
    }

    public function update(Request $request, $id)
    {
        $permit = Permit::find($id);

        $permit->title = $request->title;
        $permit->description = isset($request->description) ? $request->description : null;
        $permit->from = date('Y-m-d',strtotime($request->from));
        $permit->until = isset($request->until) ? date('Y-m-d',strtotime($request->until)) : null;

        $permit->save();
        return response()->json($permit);
    }

}
```

Αναφέρεται στο κεφ. 3.4.5 οπού περιγράφουμε αναλυτικά τις λειτουργίες κάθε μεθόδου και τα URL στα οποία αντιστοιχούν.

Κωδικας 15 - SemesterController.php

```
class SemesterController extends Controller
{
    public function index()
    {
        $list = Semester::all();
```

```
        return response()->json($list);
    }
}
```

Αναφέρεται στο κεφ. 3.6.2 και στο κεφ. 3.4.6 οπού περιγράφουμε αναλυτικά τις λειτουργίες κάθε μεθόδου και τα URL στα οποία αντιστοιχούν. Αναλυτικότερα ο συγκεκριμένος controller είναι υπεύθυνος για την ταυτοποίηση ενός χρήστη μέσω της εφαρμογής που δημιουργήθηκε από το site της σχολής.

Κωδικας 16 - PassportAuthController.php

```
class PassportAuthController extends Controller
{
    public function register(Request $request)
    {
        $this->validate($request, [
            'name' => 'required|min:4',
            'email' => 'required|email',
            'password' => 'required|min:8',
        ]);

        $user = User::create([
            'name' => $request->name,
            'email' => $request->email,
            'password' => bcrypt($request->password)
        ]);

        $token = $user->createToken('LaravelAuthApp')->accessToken;

        return response()->json(['token' => $token], 200);
    }


    public function login($socialiteUser)
    {
        $user = User::where('uid', $socialiteUser->uid)->first();
        if ($user === null) {
            $user = User::create(
                [
                    'name' => $socialiteUser->name,
                    'permission'=>$socialiteUser->user['eduPersonAffiliation'],
                    'email' => $socialiteUser->email,
                    'uid' => $socialiteUser->uid,
                ]
            );
        } else {
            $user = User::where('uid', $socialiteUser->uid)->update(
                [
                    'name' => $socialiteUser->name,
```

```
                    'permission'=>$socialiteUser->user['eduPersonAffiliation'],
                    'email' => $socialiteUser->email,
                    'uid' => $socialiteUser->uid,
                ]
            );
        }

        try {
            $user = User::where('uid', $socialiteUser->uid)->first();
            $attributes = ['id' => $user->id];
            Auth::login($user);
            $token = $user->createToken('MyApp')-> accessToken;
            return response()->json(['token' => $token]);
        } catch (\GuzzleHttp\Exception\BadResponseException $e) {
            Auth('web')->logout();
            Session::flush();
            if ($e->getCode() === 400) {
                return response()->json('Invalid request', $e->getCode());
            } else if ($e->getCode() === 401) {
                return response()->json('Invalid credentials', 401);
            }

            return response()->json('Something went wrong on the server.', $e->getCode());
        }
    }

    public function user(Request $request)
    {
        return new UserResource($request->user());
    }

    public function signIn(Request $request)
    {
        return Socialite::driver('iee')->redirect();
    }

    public function redirect(){
        $user = Socialite::driver('iee')->user();
        $token = $this->login($user);
        $query = http_build_query([
            'response_type' => 'code',
            'scope' => '*',
            'access_token'=> json_encode($token)
        ]);

        return redirect('http://localhost:4200/login?'.$query);
    }
}
```