ΠΑΝΕΠΙΣΤΗΜΙΟ
ΘΕΣΣΑΛΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΡΟΗΓΜΕΝΕΣ ΑΣΦΑΛΕΙΣ ΔΙΕΠΑΦΕΣ ΓΙΑ ΔΙΑΔΙΚΤΥΑΚΕΣ ΕΦΑΡΜΟΓΕΣ

ΔΗΜΗΤΡΙΟΣ ΘΑΝΟΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΥΠΕΥΘΥΝΟΣ

Κωνσταντίνος Κολομβάτσος
Επίκουρος Καθηγητής Πανεπιστημίου Θεσσαλίας

ΣΥΝΕΠΙΒΛΕΠΩΝ

Γεώργιος Σταμούλης
Καθηγητής Πανεπιστημίου Θεσσαλίας

Λαμία ..... έτος .....

ΠΑΝΕΠΙΣΤΗΜΙΟ
ΘΕΣΣΑΛΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΡΟΗΓΜΕΝΕΣ ΑΣΦΑΛΕΙΣ ΔΙΕΠΑΦΕΣ ΓΙΑ ΔΙΑΔΙΚΤΥΑΚΕΣ ΕΦΑΡΜΟΓΕΣ

ΔΗΜΗΤΡΙΟΣ ΘΑΝΟΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΥΠΕΥΘΥΝΟΣ

Κωνσταντίνος Κολομβάτσος
Επίκουρος Καθηγητής Πανεπιστημίου Θεσσαλίας

ΣΥΝΕΠΙΒΛΕΠΩΝ

Γεώργιος Σταμούλης
Καθηγητής Πανεπιστημίου Θεσσαλίας

Λαμία ..... έτος .....

UNIVERSITY OF
THESSALY

SCHOOL OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE & TELECOMMUNICATIONS

ADVANCED SECURE INTERFACES FOR WEB APPLICATIONS

DIMITRIOS THANOS

FINAL THESIS

ADVISOR

Konstantinos Kolomvatsos
Assistant Professor

CO ADVISOR

George Stamoulis
Professor

Lamia ..... year .....

«Με ατομική μου ευθύνη και γνωρίζοντας τις κυρώσεις [1], που προβλέπονται από της διατάξεις της παρ. 6 του άρθρου 22 του Ν. 1599/1986, δηλώνω ότι:

1. Δεν παραθέτω κομμάτια βιβλίων ή άρθρων ή εργασιών άλλων αυτολεξεί ***χωρίς να τα περικλείω σε εισαγωγικά*** *και χωρίς να αναφέρω το συγγραφέα, τη χρονολογία, τη σελίδα. Η αυτολεξεί παράθεση χωρίς εισαγωγικά χωρίς αναφορά στην πηγή, είναι λογοκλοπή. Πέραν της αυτολεξεί παράθεσης, λογοκλοπή θεωρείται και η παράφραση εδαφίων από έργα άλλων, συμπεριλαμβανομένων και έργων συμφοιτητών μου, καθώς και η παράθεση στοιχείων που άλλοι συνέλεξαν ή επεξεργάσθηκαν, χωρίς αναφορά στην πηγή. Αναφέρω πάντοτε με πληρότητα την πηγή κάτω από τον πίνακα ή σχέδιο, όπως στα παραθέματα.*

2. Δέχομαι ότι η αυτολεξεί ***παράθεση χωρίς εισαγωγικά****, ακόμα κι αν συνοδεύεται από αναφορά στην πηγή σε κάποιο άλλο σημείο του κειμένου ή στο τέλος του, είναι αντιγραφή. Η αναφορά στην πηγή στο τέλος π.χ. μιας παραγράφου ή μιας σελίδας, δεν δικαιολογεί συρραφή εδαφίων έργου άλλου συγγραφέα, έστω και παραφρασμένων, και παρουσίασή τους ως δική μου εργασία.*

3. Δέχομαι ότι υπάρχει επίσης περιορισμός στο μέγεθος και στη συχνότητα των παραθεμάτων που μπορώ να εντάξω στην εργασία μου εντός εισαγωγικών. Κάθε μεγάλο παράθεμα (π.χ. σε πίνακα ή πλαίσιο, κλπ.), προϋποθέτει ειδικές ρυθμίσεις, και όταν δημοσιεύεται προϋποθέτει την άδεια του συγγραφέα ή του εκδότη. Το ίδιο και οι πίνακες και τα σχέδια

4. Δέχομαι όλες τις συνέπειες σε περίπτωση λογοκλοπής ή αντιγραφής.

Ημερομηνία: ...../...../20.....

Ο – Η Δηλ.

(1) «Όποιος εν γνώσει του δηλώνει ψευδή γεγονότα ή αρνείται ή αποκρύπτει τα αληθινά με έγγραφη υπεύθυνη δήλωση
του άρθρου 8 παρ. 4 Ν. 1599/1986 τιμωρείται με φυλάκιση τουλάχιστον τριών μηνών. Εάν ο υπαίτιος αυτών των πράξεων
σκόπευε να προσπορίσει στον εαυτόν του ή σε άλλον περιουσιακό όφελος βλάπτοντας τρίτον ή σκόπευε να βλάψει άλλον, τιμωρείται με κάθειρξη μέχρι 10 ετών.»

ΠΕΡΙΛΗΨΗ

Μια διαδικτυακή εφαρμογή είναι ένα υπολογιστικό πρόγραμμα το οποίο σε αντίθεση με τα κοινά προγράμματα του υπολογιστή είναι προσβάσιμο από τους χρήστες μέσω του φυλλομετρητή και του διαδικτύου. Μια διαδικτυακή εφαρμογή χρησιμοποιεί το μοντέλο πελάτη – εξυπηρετητή και τις τεχνολογίες διαδικτύου με σκοπό να παρέχει υπηρεσίες στους χρήστες μέσω του διαδικτύου. Κάθε διαδικτυακή εφαρμογή αποτελείται από δύο μέρη, το back – end και το front – end. Σε ένα μοντέλο πελάτη – εξυπηρετητή το back – end αναφέρεται στον εξυπηρετητή και όλη την λειτουργικότητα που περιορίζεται στο παρασκήνιο για αποφόρτωση του πελάτη και αποθήκευση των δεδομένων, ενώ το front – end αναφέρεται στον πελάτη και τις λειτουργίες που εκτελούνται στο προσκήνιο με σκοπό τη βέλτιστη διεπαφή χρήστη. Για παράδειγμα ηλεκτρονικό ταχυδρομείο, e – shop και διαδικτυακές τραπεζικές συναλλαγές είναι μερικές από τις χρήσεις μιας διαδικτυακής εφαρμογής.

Η εργασία αυτή έχει θέμα τις διάφορες τεχνικές που μπορούν να επιστρατευτούν για τη βελτιστοποίηση της ασφάλειας, αλλά και της ταχύτητας προσπέλασης των δεδομένων μιας διαδικτυακής εφαρμογής χρησιμοποιώντας τα κατάλληλα εργαλεία, αλλά και τις τελευταίες τεχνολογικές μεθόδους για την επίτευξη αυτού. Στρέφουμε την προσοχή μας προς την υλοποίηση του JWT (Json Web Token) στο back – end και τη χρήση του για την αυθεντικοποίηση των requests ενός χρήστη, αλλά και των μεθόδων διασφάλισης αποκλειστικής πρόσβασης μελών και διαχειριστή στα routes της διαδικτυακής εφαρμογής που αντιστοιχούν στις ανάλογες ενέργειες, τόσο στο front – end όσο και στο back – end. Η διαδικτυακή αυτή εφαρμογή περιλαμβάνει επίσης τεχνικές στο front – end για την επίτευξη του βέλτιστου UI/UX (User Interface/User Experience) για ένα χρήστη.

Τέλος, η εργασία περιλαμβάνει παραδείγματα σε μια πρότυπη διαδικτυακή εφαρμογή. Πιο συγκεκριμένα, παρουσιάζουμε τις αναφερόμενες τεχνικές αναλύοντας τη χρήση τους σε μια διαδικτυακή εφαρμογή με επίπεδα ιεραρχίας χρήστη – διαχειριστή και με σκοπό τη δημιουργία και τροποποίηση τόσο των χρηστών όσο και του περιεχομένου της εφαρμογής.

ABSTRACT

A web application is a computer program that in addition to common computer programs it is accessible from users through the browser and the internet. A web application uses the client – server model and internet technologies in order to provide services to users through the internet. Every web application is composed of two parts, the back – end and the front – end. In a client – server model, the back – end is referred to the server and all the functionality that is contained in the background in order to reduce loading times and for saving purposes, where the front – end is referred to the client and all the functionality that is executed in the foreground in order to provide a better user interface. Some examples of web application usages are web mails, e–shops, online banking transactions.

This thesis focuses on the several techniques that can be used to optimize security, the speed of accessing the data in a web application using the proper tools and the most advanced methods to achieve that. We also focus on the implementation of the JWT (Json Web Token) in the back – end and its usage to authenticate user requests, but also the methods we use to ensure exclusive user and administrator access to the web application's routes that correspond to these actions, in the front – end as well as the back – end. This web application also includes techniques in the front – end to achieve the best UI/UX for a user.

Concluding, this thesis includes examples in an example web application. More specifically, we present the referred methods and techniques by analyzing their usage in a web application with user – admin hierarchy layers and with the purpose of creating and editing the users as well as the application's content.

Table of Contents

ΚΕΦΑΛΑΙΟ 1 ΕΙΣΑΓΩΓΗ

1.1 Εισαγωγή

Οι τεχνολογίες διαδικτύου αποτελούν τα εργαλεία τα οποία χρησιμοποιούμε για τη δημιουργία μιας διαδικτυακής εφαρμογής. Μέχρι τις αρχές του 21ου αιώνα οι τεχνολογίες διαδικτύου δεν είχαν γνωρίσει ιδιαίτερη ανάπτυξη καθώς μέχρι και τις αρχές της δεκαετίας του '90 και η ίδια η χρήση του διαδικτύου δεν ήταν ευρέως διαδεδομένη. Έπειτα, κατά τη δεκαετία του '90 έγιναν τα πρώτα βήματα για τη κατασκευή εργαλείων για τη δημιουργία ιστοσελίδων και διαδικτυακών εφαρμογών. Στις αρχές της δεκαετίας του '90 δημιουργήθηκε η πιο βασική γλώσσα προγραμματισμού διαδικτυακών εφαρμογών HTML και στη συνέχεια ακολούθησε η JAVASCRIPT για προσθήκη περισσότερης λειτουργικότητας και η CSS για καλύτερη εμφάνιση στις ιστοσελίδες. Αυτά υπήρξαν τα πρώτα εργαλεία για τη δημιουργία των διαδικτυακών εφαρμογών.

Στην αρχή οι ιστοσελίδες ήταν ιδιαίτερα απλές, οι περισσότερες ως ενημερωτικές ιστοσελίδες και σελίδες παρουσίασης. Με το πέρασμα των χρόνων η χρήση του διαδικτύου αυξήθηκε και μαζί της έφερε την ανάγκη για περισσότερη διαδικτυακή λειτουργικότητα και την ανάπτυξη των ιστοσελίδων και την εξέλιξή τους σε διαδικτυακές εφαρμογές.

Η ασφάλεια είναι ένα θέμα που απασχολεί τόσο τους προγραμματιστές όσο και τους χρήστες των διαδικτυακών εφαρμογών από την αρχή της χρήσης τους μέχρι και σήμερα. Η ανάγκη που οδηγεί στην συστηματική εξέλιξη των μεθόδων διασφάλισης των δεδομένων των χρηστών και κατ' επέκταση των ίδιων των διαδικτυακών εφαρμογών είναι η κακόβουλη υποκλοπή των δεδομένων των χρηστών.

Με τις διαδικτυακές εφαρμογές να αποτελούν αναπόσπαστο κομμάτι της καθημερινότητας μας σε συνδυασμό με την ψηφιοποίηση όλων των δεδομένων η ασφάλεια των διαδικτυακών εφαρμογών καθίσταται αναγκαία.

Την τελευταία δεκαετία, με την ευρεία χρήση των web framework έχουν αυξηθεί οι εφαρμογές τεχνικών ασφαλείας στις διαδικτυακές εφαρμογές, όμως αυτό έχει προκαλέσει και την αύξηση αναζήτησης των αντίστοιχων μεθόδων παραβίασης τους. Επιπλέον, η ραγδαία ανάπτυξη της τεχνολογίας και των διαδικτυακών εφαρμογών, είναι ένας ακόμα λόγος για την βελτιστοποίηση της ασφάλειας σε αυτές καθώς τα μέτρα ασφάλειας καθίστανται ανεπαρκή. Με τη λογική αυτή λοιπόν, καμία διαδικτυακή εφαρμογή δεν παραμένει ασφαλής για μεγάλο διάστημα, και είναι απαραίτητη η συστηματική ανανέωση των μέτρων και τεχνικών ασφαλείας σε αυτή.

Σε αυτή την εργασία λοιπόν θα δοκιμάσουμε την εφαρμογή όλων των διαθέσιμων τεχνικών ασφαλείας, τόσο στο front – end όσο και στο back – end. Αυτό θα επιτευχθεί με τη χρήση δυο από τα πιο δημοφιλή και χρησιμοποιημένα framework που υπάρχουν τα τελευταία

χρόνια. Ο λόγος στα Angular και Laravel frameworks, τα οποία μας παρέχουν μια μεγάλη ποικιλία από τεχνικές και εργαλεία για την βέλτιστη επίτευξη του στόχου αυτού.

Τέλος, θα παρουσιάσουμε μερικά παραδείγματα και εφαρμογές των αναλυόμενων τεχνικών πάνω σε μια εφαρμογή με επίπεδα εξουσιοδότησης χρήστη - διαχειριστή. Αλλά και μερικές μελλοντικές προεκτάσεις και προσθήκες που θα μπορούσαν να γίνουν για περαιτέρω εξέλιξη των διαφόρων τμημάτων της διαδικτυακής εφαρμογής μας.

ΚΕΦΑΛΑΙΟ 2 ΤΕΧΝΟΛΟΓΙΕΣ ΔΙΑΔΙΚΤΥΟΥ & WEB FRAMEWORKS

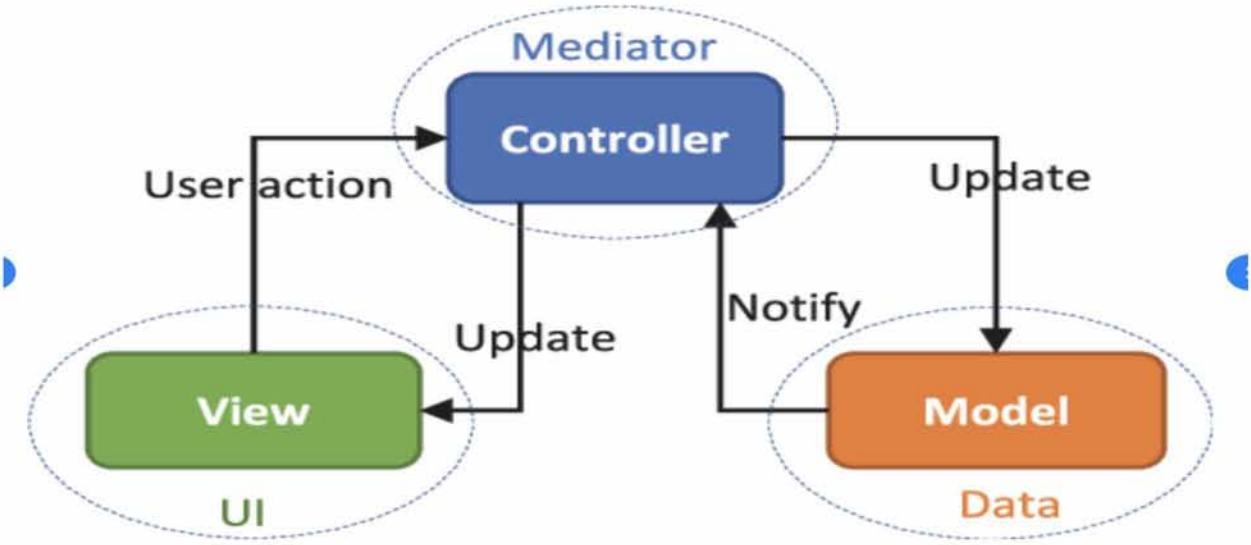
2.1 Τεχνολογίες διαδικτύου

Στις μέρες μας οι τεχνολογίες διαδικτύου έχουν μεταφερθεί στο «επόμενο επίπεδο» με τη χρήση των web frameworks. Τα web frameworks έχουν σχεδιαστεί για την ανάπτυξη διαδικτυακών εφαρμογών, συμπεριλαμβανομένων των υπηρεσιών διαδικτύου, πόρων διαδικτύου και των web APIs. Τα web frameworks στοχεύουν στην αυτοματοποίηση των συχνών ενεργειών που εκτελούνται στον διαδικτυακό προγραμματισμό. Για παράδειγμα, πολλά web frameworks παρέχουν βιβλιοθήκες για ενέργειες πρόσβασης βάσης δεδομένων, διαχείριση sessions και πολύ συχνά προωθούν την επαναχρησιμοποίηση τμημάτων κώδικα. Ενώ συχνά στοχεύουν στην ανάπτυξη δυναμικών ιστοσελίδων και διαδικτυακών εφαρμογών, ωστόσο εφαρμόζονται και στην ανάπτυξη στατικών ιστοσελίδων.

Παρόλο που η πλειονότητα των γλωσσών προγραμματισμού που χρησιμοποιούνται για τη δημιουργία δυναμικών ιστοσελίδων περιλαμβάνουν βιβλιοθήκες που βοηθούν με τις περισσότερες ενέργειες, οι διαδικτυακές εφαρμογές συχνά απαιτούν τη χρήση συγκεκριμένων «third-party» βιβλιοθηκών για ορισμένες ενέργειες, όπως για παράδειγμα την δημιουργία HTML κώδικα ή τη σύνδεση μιας διαδικτυακής εφαρμογής με μία άλλη για ανταλλαγή πληροφοριών και δεδομένων.

Στα τέλη της δεκαετίας του ’90 τα πρώτα full-stack frameworks άρχισαν να εμφανίζονται, τα οποία παρείχαν πολλές χρήσιμες βιβλιοθήκες για διαδικτυακό προγραμματισμό σε ένα συνεκτικό «πακέτο» λογισμικού για προγραμματιστές διαδικτυακών εφαρμογών. Μερικά παραδείγματα είναι τα Django, Laravel, Symfony, Ruby on Rails.

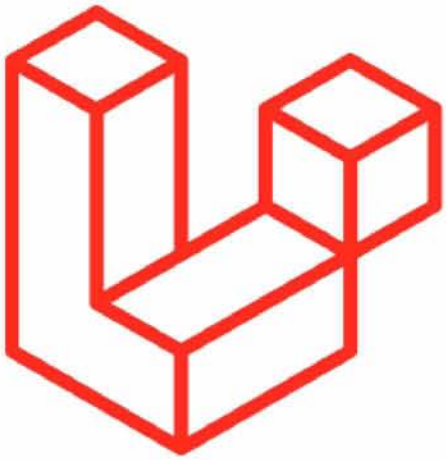
Καθένα από τα παραπάνω web frameworks χρησιμοποιεί ως βάση του μια συγκεκριμένη γλώσσα προγραμματισμού για την επίτευξη των στόχων του. Τα περισσότερα από αυτά βασίζονται στο μοντέλο MVC (Model View Controller). Το μοντέλο MVC αποτελείται από το model, το στοιχείο που περιλαμβάνει τα δεδομένα, π.χ. χρήστη, δημοσίευση, από τον controller, που αποτελεί τον διαμεσολαβητή για την πρόσβαση σε αυτά τα δεδομένα, καθώς και το view, που είναι το στοιχείο που περιλαμβάνει τη διεπαφή με τον χρήστη. Έτσι διαχωρίζεται το data model από τη διεπαφή χρήστη. Αυτό θεωρείται γενικότερα καλή πρακτική καθώς τμηματοποιεί τον κώδικα, προωθεί την επαναχρησιμοποίηση των κατάλληλων τμημάτων του και επιτρέπει την εύκολη δημιουργία και χρήση πολλαπλών διεπαφών χρήστη. Με άλλα λόγια, σε μία διαδικτυακή εφαρμογή αυτό επιτρέπει την παρουσίαση πολλαπλών views.



Model-View-Control(MVC) design pattern.

2.2 LARAVEL FRAMEWORK

Το Laravel είναι ένα open-source web framework γραμμένο στη γλώσσα προγραμματισμού PHP, που δημιουργήθηκε το 2011 και χρησιμοποιείται έκτοτε για την ανάπτυξη διαδικτυακών εφαρμογών βάση του μοντέλου MVC (Model View Controller). Το Laravel χρησιμοποιείται κυρίως ως back-end framework και παρέχει ένα προεγκατεστημένο πακέτο υποστήριξης σχεσιακών βάσεων δεδομένων, email templates, αλλά και ένα ευρύ φάσμα μεθόδων αυθεντικοποίησης και εξουσιοδότησης χρηστών. Το τελευταίο είναι και ένας από τους πιο σημαντικούς λόγους που το χρησιμοποιούμε και σε αυτή την εργασία. Το Laravel όπως και άλλα web frameworks παρέχουν επίσης εντολές CLI (command-line interface). Μέσω του artisan CLI, όπως ονομάζεται αυτό το χαρακτηριστικό του Laravel έχουμε τη δυνατότητα της προσθήκης έτοιμων αρχείων templates του Laravel και αυτό βοηθά τον προγραμματιστή στο να μην επαναλαμβάνει τη διαδικασία αρχικοποίησης αρχείων, αλλά και στην επανάληψη βασικών λειτουργιών στην εφαρμογή του.



2.3 ANGULAR FRAMEWORK

Το Angular framework είναι ένα open-source web framework βασισμένο στην TypeScript και γραμμένο από την Angular Team της Google. Το Angular ξεκίνησε το 2010 ως AngularJS, ένα web framework βασισμένο σε JavaScript. Στη συνέχεια το 2016 βγήκε η έκδοση Angular 2+ και έκτοτε το Angular είναι το πιο διαδεδομένο front-end framework για δημιουργία διαδικτυακών εφαρμογών. Στοχεύει στην απλοποίηση της ανάπτυξης αυτών των εφαρμογών με τη χρήση modules και components. Με τον τρόπο αυτό μεγιστοποιείται η επαναχρησιμοποίηση του κώδικα και γίνεται ευκολότερη η εισαγωγή third-party πακέτων. Η «λογική» των module και των component κυριαρχεί σε όλα τα front-end frameworks καθώς είναι και αυτή που μας δίνει τα στοιχεία της επαναχρησιμοποίησης κώδικα και της απλοποίησης των περισσότερων front – end λειτουργιών.

ΚΕΦΑΛΑΙΟ 3 ΑΣΦΑΛΕΙΑ ΣΤΟ BACK-END

3.1 Εισαγωγή

Η αυθεντικοποίηση και η εξουσιοδότηση ενός χρήστη μέσα σε μια διαδικτυακή εφαρμογή είναι οι κύριοι παράγοντες για την ασφάλεια της εφαρμογής αυτής. Οι περιορισμοί πρόσβασης σε μια εφαρμογή κάνουν τη διαφορά μεταξύ των διαφορετικών επιπέδων ιεραρχίας των χρηστών, π.χ. Admin, Member κ.ά. Επιπλέον, η αυθεντικοποίηση ενός χρήστη στην διαδικτυακή εφαρμογή είναι ο μοναδικός τρόπος εξασφάλισης της πιστότητας των στοιχείων που παρέχει. Αυτά αποτελούν τα προπύργια της ασφάλειας σε μια διαδικτυακή εφαρμογή.

Το Laravel framework μας προσφέρει ένα μεγάλο εύρος dependencies – πακέτων που αφορούν τόσο την αυθεντικοποίηση όσο και την εξουσιοδότηση σε μια διαδικτυακή εφαρμογή. Στην αυθεντικοποίηση ενός χρήστη έχουμε τη χρήση του JWT (Json Web Token), για την πρόσβαση της εισόδου στην διαδικτυακή εφαρμογή, ενώ στον έλεγχο εξουσιοδότησης του χρήστη για την πρόσβαση στα διάφορα views της διαδικτυακής εφαρμογής χρησιμοποιούμε τα Gates.

Τέλος, τόσο η χρήση του Json Web Token όσο και η χρήση των Gates καθίσταται αναγκαία σε μια διαδικτυακή εφαρμογή που ακολουθεί το μοντέλο Model View Controller (MVC) καθώς ο συνδυασμός τους αποτελεί το προπύργιο της ασφάλειας και του περιορισμού κακόβουλης πρόσβασης σε ένα back-end API.

3.2 Route Middleware

3.2.α Χρήσεις

Τα route middleware έχουν πολλαπλές χρήσεις. Μπορούν να χρησιμοποιηθούν για την ελεγχόμενη εξουσιοδότηση ενός χρήστη σε μια διαδικτυακή εφαρμογή ή για τον έλεγχο και προσθήκη δεδομένων στο HTTP request και response.

Πιο συγκεκριμένα, ένα route middleware ενεργοποιείται κατά την προσπέλαση του route στο οποίο έχει τεθεί και ένας από τους σκοπούς του είναι να επιτρέψει την πρόσβαση σε αυτό το route μόνο στους χρήστες που διαθέτουν την κατάλληλη εξουσιοδότηση στην διαδικτυακή εφαρμογή.

Επιπλέον, τα middleware παρέχουν έναν βολικό μηχανισμό για το «φιλτράρισμα» των HTTP requests που εισέρχονται στην διαδικτυακή μας εφαρμογή και τα responses που στέλνονται πίσω, από αυτήν. Για τον λόγο αυτό υπάρχουν τα before και τα after middleware. Εάν ένα middleware εκτελείται πριν ή μετά από ένα request εξαρτάται από το ίδιο το middleware.

Για παράδειγμα:

```
class BeforeMiddleware
{
    public function handle($request, Closure $next)
    {
        // Perform action

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

Εικόνα 3.1 Εκτέλεση ενός middleware πριν τον χειρισμό του request

```php
class AfterMiddleware
{
    public function handle($request, Closure $next)
    {
        $response = $next($request);

        // Perform action

        return $response;
    }
}
```

Εικόνα 3.2 Εκτέλεση ενός middleware μετά τον χειρισμό του request

Στο Laravel, για τη δημιουργία ενός middleware μπορούμε να χρησιμοποιήσουμε μία από τις έτοιμες εντολές του Artisan CLI, ως εξής:

```
php artisan make:middleware CheckAge
```

Εικόνα 3.2 Εκτέλεση μίας Artisan εντολής για τη δημιουργία ενός middleware template

Το Laravel «έρχεται» μαζί με ένα εύρος από middleware, ένα από αυτά είναι το auth middleware και βρίσκεται στο πακέτο "Illuminate\Auth\Middleware\Authenticate".

Καθώς αυτό το middleware είναι ήδη καταχωρημένο στον πυρήνα – kernel του Laravel, το μόνο που χρειάζεται να κάνουμε είναι να το συνδέσουμε σε ένα route,

Για παράδειγμα:

```
Route::get('profile', function () {
    // Only authenticated users may enter...
})->middleware('auth');
```

Εικόνα 3.1 Παράδειγμα χρήσης ενός middleware σε ένα route

Εάν και δεν είναι η συνιστώμενη πρακτική, στην περίπτωση που χρησιμοποιούμε controllers στη διαδικτυακή εφαρμογή μας μπορούμε επίσης να καλέσουμε το middleware μέσα από τη μέθοδο δημιουργό – constructor της κλάσης του controller μας, αντί να το συνδέσουμε με ένα route.

Για παράδειγμα:

```
public function __construct()
{
    $this->middleware('auth');
}
```

Εικόνα 3.2 Παράδειγμα χρήσης ενός middleware σε έναν controller

3.3 JWT – Json Web Token

3.3.α Τι είναι το Json Web Token

Το Json Web Token ανήκει στο πρότυπο RFC7519 και αποτελεί ένα συμπαγή και αυτοδύναμο τρόπο για ασφαλή μετάδοση πληροφοριών ως Json αντικειμένου. Αυτές οι πληροφορίες είναι πιστοποιημένες καθώς φέρουν ψηφιακή υπογραφή. Το Json Web Token μπορεί να πιστοποιηθεί ψηφιακά χρησιμοποιώντας μια συνάρτηση κατακερματισμού μέσω του HMAC αλγόριθμου ή ένα δημόσιο/ιδιωτικό ζεύγος κλειδιών μέσω RSA ή ECDSA αλγορίθμου.

Παρόλο που το Json Web Token μπορεί να κρυπτογραφηθεί για να προσφέρει περαιτέρω ασφάλεια μεταξύ των μελών όπου ανταλλάσσεται εμείς θα επικεντρωθούμε στα ψηφιακά υπογεγραμμένα tokens. Τα ψηφιακά υπογεγραμμένα tokens μπορούν να επικυρώσουν την «ακεραιότητα» των claims – περιεχομένων που αυτά περιλαμβάνουν, ενώ τα κρυπτογραφημένα tokens «αποκρύπτουν» τα claims αυτά από άλλα μέλη.

Τέλος, όταν τα tokens «υπογράφονται» χρησιμοποιώντας ένα δημόσιο/ιδιωτικό ζεύγος κλειδιών, η ψηφιακή υπογραφή πιστοποιεί επίσης ότι μόνο η πλευρά που διαθέτει το ιδιωτικό κλειδί είναι και αυτή που το «υπέγραψε».

3.3.β Λόγοι χρήσης

Σενάρια στα οποία το Json Web Token είναι χρήσιμο:

- Ανταλλαγή δεδομένων - Αυθεντικοποίηση:

  Το Json Web Token είναι ένας πολύ καλός τρόπος για ασφαλή ανταλλαγή δεδομένων μεταξύ δύο πλευρών. Επειδή τα Json Web Tokens φέρουν ψηφιακή υπογραφή, π.χ. μέσω δημόσιου/ιδιωτικού ζεύγους κλειδιών, μπορούμε να είμαστε σίγουροι πως ο αποστολέας είναι έμπιστος. Επιπλέον, καθώς η ψηφιακή υπογραφή υπολογίζεται χρησιμοποιώντας το header και το payload κομμάτι του token, μπορούμε επίσης να πιστοποιήσουμε ότι το περιεχόμενο του token δεν έχει αλλοιωθεί. Με αυτό τον τρόπο επιτυγχάνεται η ασφαλής πιστοποίηση των στοιχείων του χρήστη από την πλευρά της διαδικτυακής εφαρμογής.

- Εξουσιοδότηση:

Αυτό είναι εξίσου ένα από τα πιο συνήθη σενάρια χρήσης του Json Web Token. Μόλις ο χρήστης αυθεντικοποιηθεί και εισέλθει στην διαδικτυακή εφαρμογή, κάθε του request θα περιέχει ένα Json Web Token, επιτρέποντας στο χρήστη να προσπελαύνει συγκεκριμένα routes, services και views για τα οποία απαιτείται εξουσιοδότηση και του επιτρέπεται η προσπέλαση χρησιμοποιώντας αυτό το token. Το Single Sign On – Σύστημα Ενιαίας Πρόσβασης είναι ένα χαρακτηριστικό που χρησιμοποιεί το Json Web Token στις μέρες μας, λόγω του μικρού overhead και της ικανότητας του να χρησιμοποιείται εύκολα μεταξύ διαφορετικών domains.

3.3.γ Δομή

Στην συμπαγή του μορφή ένα Json Web Token αποτελείται από τρία τμήματα διαχωρισμένα από τελείες (.), τα οποία είναι:

- Header
- Payload
- Signature

Επομένως, ένα Json Web Token φαίνεται ως εξής:

xxxxx.yyyyy.zzzzz

Ας αναλύσουμε τα διαφορετικά μέρη.

1. **Header**

Το header – επικεφαλίδα αποτελείται από δύο μέρη: τον τύπο του token, δηλαδή JWT και τον αλγόριθμο ψηφιακής υπογραφής που χρησιμοποιήθηκε, όπως για παράδειγμα HMAC, SHA256 ή RSA.

Εικόνα 3.3 Παράδειγμα του header – επικεφαλίδας ενός JWT

2. Payload

Το δεύτερο μέρος του token είναι το payload – φορτίο, το οποίο περιλαμβάνει και τα claims – περιεχόμενα. Τα claims είναι συνήθως δηλώσεις για μία «οντότητα» (κυρίως για το χρήστη) και επιπλέον δεδομένα. Υπάρχουν τρείς τύποι από claims: registered, public και private.

- Registered claims:

Αυτά είναι ένα σετ από προκαθορισμένα claims, τα οποία δεν είναι υποχρεωτικά, αλλά συνιστώνται, ώστε να παρέχουν χρήσιμα, διαλειτουργικά claims. Μερικά από αυτά είναι: iss (issuer), exp(expiration time), sub (subject), aud (audience) και others. Όπως παρατηρούμε τα ονόματα των claims αποτελούνται από μόνο τρείς χαρακτήρες καθώς το Json Web Token πρέπει να είναι συμπαγές.

- Public Claims:

Αυτά μπορούν να οριστούν κατά βούληση από το χρήστη του JWT. Αλλά για την αποφυγή συγκρούσεων πρέπει να είναι σύμφωνα με το IANA JSON Web Token Registry.

- Private Claims:

Αυτά είναι τα claims που δημιουργούνται για την ανταλλαγή πληροφοριών μεταξύ των δυο πλευρών και δεν αποτελούν ούτε registered ούτε public claims.

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
```

Εικόνα 3.4 Παράδειγμα του payload – φορτίου ενός JWT

Πρέπει να σημειωθεί πως για τα υπογεγραμμένα token αυτές οι πληροφορίες καθώς προστατευμένες κατά της αλλοίωσης, παραμένουν αναγνώσιμες από οποιονδήποτε. Επομένως, δεν πρέπει να τοποθετούμε ευαίσθητες πληροφορίες στα header και payload στοιχεία ενός JWT εκτός και εάν είναι κρυπτογραφημένες.

3.3.δ Υπογραφή

Για να δημιουργήσουμε το μέρος της ψηφιακής υπογραφής πρέπει να πάρουμε το κωδικοποιημένο header, το κωδικοποιημένο payload, το κλειδί και τον αλγόριθμο που αναφέρεται στο header και να τα περάσουμε από τη συνάρτηση κατακερματισμού.

Για παράδειγμα εάν θέλουμε να χρησιμοποιήσουμε τον αλγόριθμο HMAC SHA256, η ψηφιακή υπογραφή θα δημιουργηθεί με τον ακόλουθο τρόπο:

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret)
```

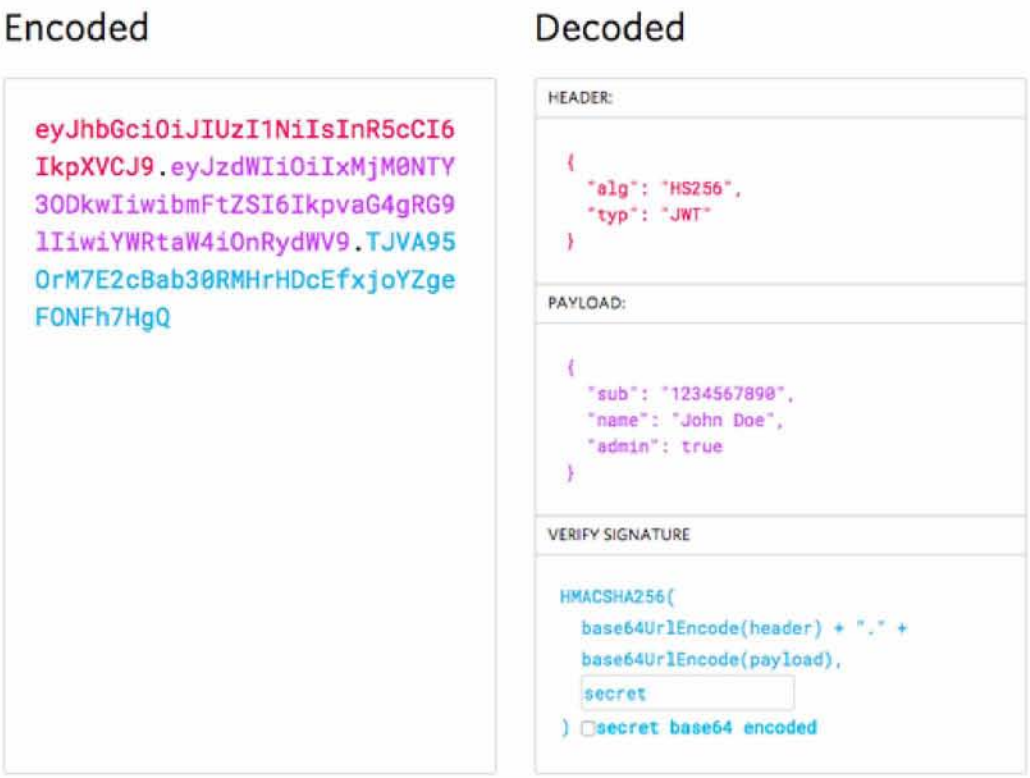
Εικόνα 3.5 Παράδειγμα δημιουργίας υπογεγραμμένου token

Η ψηφιακή υπογραφή χρησιμοποιείται για να πιστοποιήσει ότι το περιεχόμενο δεν έχει αλλάξει και στην περίπτωση των token υπογεγραμμένων με το ιδιωτικό κλειδί μπορεί επίσης να πιστοποιήσει ότι ο αποστολέας του JWT είναι έμπιστος.

Συνοψίζοντας, το αποτέλεσμα είναι τρείς γραμματοσειρές τύπου Base64-URL διαχωρισμένες από τελείες, όπου μπορούν εύκολα να περαστούν σε περιβάλλοντα HTML και HTTP καθώς και είναι πιο συμπαγή, σε σύγκριση με παλαιότερα XML-based πρότυπα. Παρακάτω βλέπουμε ένα JWT που έχει τα header και payload κωδικοποιημένα και είναι υπογεγραμμένο χρησιμοποιώντας το κλειδί.

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4
gRG9lIiwiaXNTb2NpYWwiOnRydWV9.
4pcPyMD09olPSyXnrXCjTwXyr4BsezdI1AVTmud2fU4
```

Εικόνα 3.6 Παράδειγμα ενός κωδικοποιημένου JWT



Εικόνα 3.7 Παράδειγμα ανάλυσης ενός κωδικοποιημένου JWT

3.3.ε Τρόπος λειτουργίας και διαδικασία αυθεντικοποίησης

Στην εργασία αυτή η αυθεντικοποίηση των χρηστών στηρίζεται αποκλειστικά στη χρήση του JWT token. Με τον τρόπο αυτό κατά την διαδικασία της αυθεντικοποίησης, όταν ένας χρήστης εισέρχεται επιτυχώς έχοντας εισάγει τα σωστά στοιχεία στη διαδικτυακή εφαρμογή, ένα Json Web Token επιστρέφεται συμπεριλαμβανόμενο στο response. Έπειτα στο header κάθε επόμενου request συμπεριλαμβάνουμε το JWT που ελήφθη στο προηγούμενο response. Και μέσω της συνεχής αυτής ανταλλαγής επιτυγχάνεται η πιστοποίηση του χρήστη σε πραγματικό χρόνο από την διαδικτυακή εφαρμογή.

Πιο συγκεκριμένα, κατά την είσοδο του χρήστη με σωστά στοιχεία στην εφαρμογή, επιστρέφεται από το back-end της διαδικτυακής εφαρμογής ένα http response, στο header του οποίου περιλαμβάνεται το Authorization field. Μέσα σε αυτό έχουμε το JWT token το οποίο ο χρήστης στέλνει με τον ίδιο τρόπο πίσω στο back-end της εφαρμογής συμπεριλαμβανόμενο στο header του επόμενου request και λαμβάνει πίσω ένα ανανεωμένο στο response του request αυτού.

Με τον τρόπο αυτό επιτυγχάνεται η αυθεντικοποίηση ενός χρήστη της διαδικτυακής εφαρμογής σε πραγματικό χρόνο χωρίς τον κίνδυνο υποκλοπής των στοιχείων του.

```
Authorization: Bearer <token>
```

Εικόνα 3.8 Παράδειγμα του JWT στο Authorization header

Σε μια τέτοια περίπτωση το back-end της διαδικτυακής εφαρμογής θα πρέπει να ελέγχει την ύπαρξη ενός έγκυρου Json Web Token στο Authorization header σε του κάθε request και αναλόγως να παρέχει πρόσβαση στα «προστατευμένα» routes του. Εάν το JWT περιλαμβάνει τα απαραίτητα δεδομένα, η ανάγκη για την προσπέλαση της βάσης δεδομένων μειώνεται αν και αυτό είναι πιο σπάνιο.

3.3.στ Γιατί προτιμούμε το Json Web Token

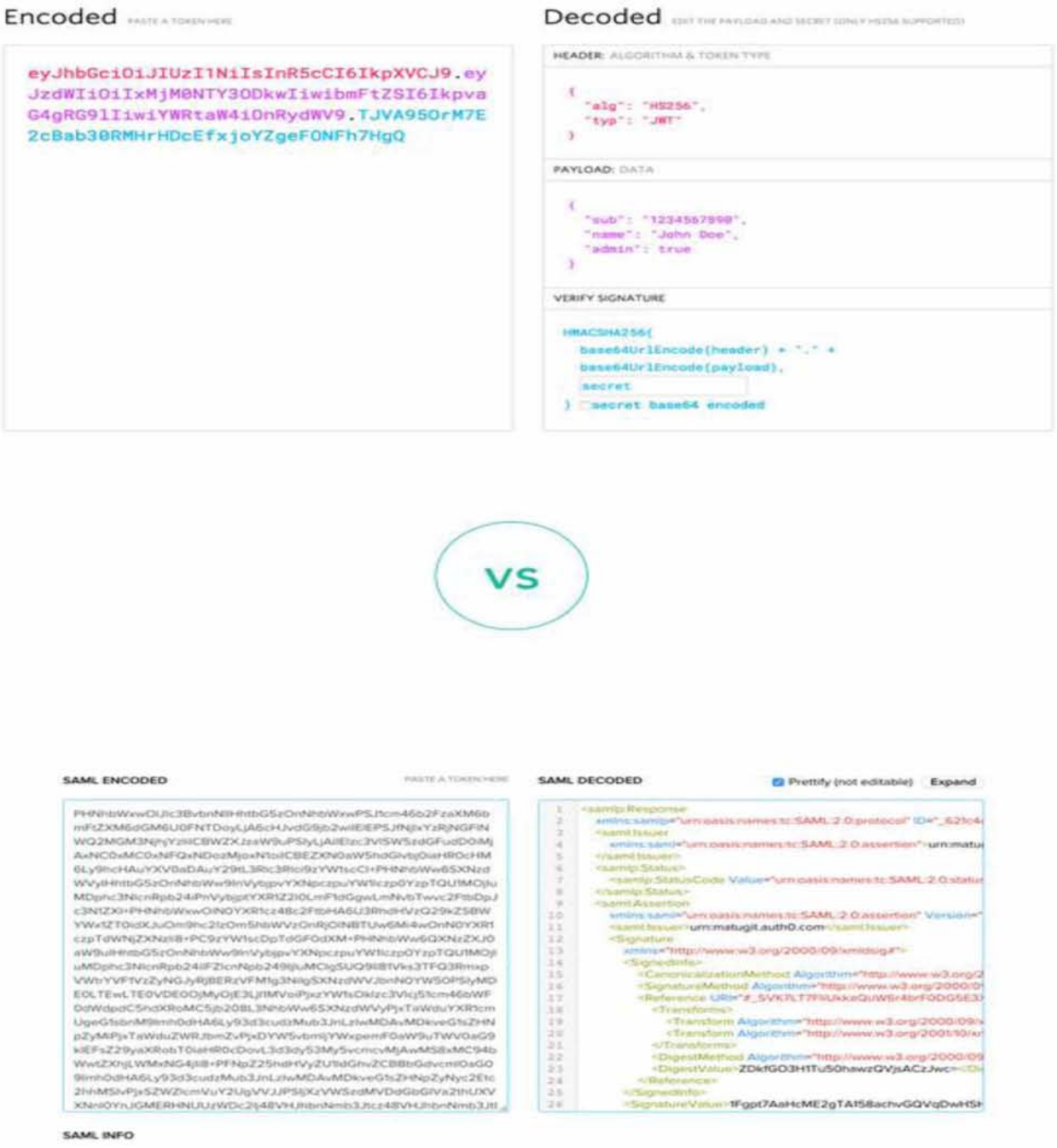
Τα οφέλη ενός Json Web Token είναι πολλά όταν το συγκρίνουμε με άλλα όπως Simple Web Tokens (SWT) ή Security Assertion Markup Language Tokens (SAML).

Ως Json το JWT είναι πιο περιεκτικό και «ολιγόλογο» από ένα XML, όταν είναι κωδικοποιημένο το μέγεθός του είναι μικρότερο, καθιστώντας το JWT πιο συμπαγές από ένα SAML. Αυτό κάνει το JWT μια καλή επιλογή για να χρησιμοποιηθεί σε περιβάλλοντα HTML και HTTP.

Από πλευράς ασφάλειας, ένα SWT μπορεί να είναι μόνο συμμετρικά υπογεγραμμένο από ένα κοινό κλειδί χρησιμοποιώντας τον αλγόριθμο HMAC. Ωστόσο, τα JWT και SAML tokens μπορούν να χρησιμοποιούν ένα δημόσιο/ιδιωτικό ζεύγος κλειδιών στη μορφή ενός X.509 πιστοποιητικού για υπογραφή. Το να υπογράψουμε ένα XML με μια ψηφιακή υπογραφή χωρίς να υπάρχουν κενά ασφαλείας είναι δύσκολο συγκριτικά με την απλότητα της υπογραφής ενός Json.

Οι αναλυτές Json είναι συνήθεις στις περισσότερες γλώσσες προγραμματισμού επειδή «χαρτογραφούν» το Json απευθείας σε τύπο αντικειμένου. Αντιθέτως για ένα XML δεν υπάρχει άμεσος τρόπος αντιστοίχισής του από κείμενο σε αντικείμενο. Αυτό καθιστά το JWT πιο εύχρηστο σε σύγκριση με το SAML (Security Assertion Markup Language).

Όσον αφορά τη χρήση, το JWT χρησιμοποιείται στην κλίμακα του διαδικτύου, αυτό αναδεικνύει την ευκολία χρήσης του Json Web Token σε πολλαπλές πλατφόρμες κυρίως κινητά.



Εικόνα 3.9 Σύγκριση του μεγέθους μεταξύ ενός κωδικοποιημένου JWT και ενός κωδικοποιημένου SAML

3.4 Gates

3.4.α Εισαγωγή

Συνάμα με την παροχή υπηρεσιών αυθεντικοποίησης, το Laravel προσφέρει επιπλέον έναν απλό τρόπο για την επιβολή περιορισμένης εξουσιοδότησης των χρηστών σε ορισμένα κομμάτια της διαδικτυακής εφαρμογής. Όπως και στην αυθεντικοποίηση, η προσέγγιση του Laravel ως προς την εξουσιοδότηση είναι απλή και υπάρχουν δυο βασικοί τρόποι για την εξουσιοδότηση των ενεργειών των χρηστών: τα Gates και τα Policies.

Ας σκεφτούμε τα gates και τα policies σαν routes και controllers σε μια διαδικτυακή εφαρμογή που ακολουθεί το MVC μοντέλο. Τα Gates παρέχουν μια απλή προσέγγιση ως προς την εξουσιοδότηση καθώς έχουν την απλή λογική της είτε επιτροπής, είτε αποτροπής της πρόσβασης π.χ. σε ένα route. Ενώ τα Policies, όπως οι controllers, έχουν την λογική τους «χτισμένη» γύρω από ένα συγκεκριμένο model – μοντέλο.

Χτίζοντας μια διαδικτυακή εφαρμογή, δεν χρειάζεται να επιλέξουμε αποκλειστικά μεταξύ της χρήσης των Gates ή των Policies. Οι περισσότερες εφαρμογές περιέχουν κατά κύριο λόγο ένα συνονθύλευμα από Gates και Policies. Τα Gates είναι πιο εύκολα εφαρμόσιμες σε ενέργειες που δεν σχετίζονται με κάποιο model, όπως για παράδειγμα η πρόσβαση στο dashboard – ταμπλό του διαχειριστή. Αντιθέτως, τα Policies πρέπει να χρησιμοποιούνται όταν επιθυμούμε να εξουσιοδοτήσουμε μια ενέργεια για ένα συγκεκριμένο model της εφαρμογής. Σε αυτό το κεφάλαιο ωστόσο θα αναλύσουμε περαιτέρω τα Gates.

```
Gate::define('edit-settings', function ($user) {
    return $user->isAdmin;
});
```

Εικόνα 3.9 Παράδειγμα ενός Gate

```
public function update(User $user, Post $post)
{
    return $user->id === $post->user_id;
}
```

Εικόνα 3.9 Παράδειγμα ενός Policy

3.4.β Σύνταξη

Τα Gates καθορίζουν εάν ένας χρήστης είναι εξουσιοδοτημένος να εκτελέσει κάποια ενέργεια και ορίζονται στην κλάση "App\Providers\AuthServiceProvider" χρησιμοποιώντας το facade – πρόθεμα "Gate". Τα Gates λαμβάνουν πάντοτε ένα αντικείμενο – object χρήστη ως πρώτη παράμετρο και μπορούν να λάβουν επιπλέον προαιρετικές παραμέτρους οποιαδήποτε models – μοντέλα της διαδικτυακής μας εφαρμογής.

```
public function boot()
{
    $this->registerPolicies();

    Gate::define('edit-settings', function ($user) {
        return $user->isAdmin;
    });

    Gate::define('update-post', function ($user, $post) {
        return $user->id === $post->user_id;
    });
}
```

Εικόνα 3.9 Παράδειγμα από δυο Gates

α) Με χρήση μόνο του user object ως παράμετρο

β) Με επιπλέον χρήση ενός άλλου model ως προαιρετική παράμετρο

3.4.γ Χρήσεις

Για την εξουσιοδότηση ενεργειών με τη χρήση Gates, πρέπει να χρησιμοποιήσουμε τις μεθόδους "allows" και "denies". Πρέπει επίσης να σημειωθεί πως δεν είναι απαραίτητο να δώσουμε ως παράμετρο το object του παρόντος χρήστη όταν χρησιμοποιούμε αυτές τις μεθόδους. Το Laravel το κάνει αυτόματα.

```php
if (Gate::allows('update-post', $post)) {
    // The current user can update the post...
}

if (Gate::denies('update-post', $post)) {
    // The current user can't update the post...
}
```

Εικόνα 3.9 Παράδειγμα από δυο Gates των μεθόδων allows και denies

Επιπλέον, το Laravel μας δίνει τη δυνατότητα της χρήσης της μεθόδου "forUser" στο Gate facade για να καθορίσουμε εάν ένας συγκεκριμένος χρήστης είναι εξουσιοδοτημένος να εκτελέσει κάποια ενέργεια.

```php
if (Gate::forUser($user)->allows('update-post', $post)) {
    // The user can update the post...
}

if (Gate::forUser($user)->denies('update-post', $post)) {
    // The user can't update the post...
}
```

Εικόνα 3.9 Παράδειγμα χρήσης της μεθόδου forUser σε ένα Gate

Ένας πολύ συνήθης τρόπος χρήσης των Gates είναι το "can" middleware. Το Laravel περιλαμβάνει ένα middleware που μπορεί να εξουσιοδοτήσει ενέργειες πριν ένας χρήστης προσπελάσει ένα συγκεκριμένο route ή controller. Το can middleware περιλαμβάνεται στο πακέτο "Illuminate\Auth\Middleware\Authorize" του Laravel, το οποίο είναι προεπιλεγμένο με τη χρήση του Laravel. Έπειτα, μετά τη δήλωση του can middleware που επιθυμούμε να χρησιμοποιήσουμε μπορούμε να προσθέσουμε, διαχωρισμένες από « , » επιπλέον παραμέτρους. Στο παρακάτω παράδειγμα δίνουμε στο can middleware δυο παραμέτρους. Η πρώτη είναι το όνομα της ενέργειας που θέλουμε να επιτραπεί στο χρήστη και η δεύτερη είναι η παράμετρος που λαμβάνεται από το ίδιο το route και επιθυμούμε να την περάσουμε στη μέθοδο του Gate η οποία θα εκτελεστεί. Στην περίπτωση αυτή καθώς η μέθοδος που χρησιμοποιούμε είναι τύπου model-binding ένα Post model θα δοθεί ως όρισμα στην μέθοδο του Gate. Στην περίπτωση που ο χρήστης δεν είναι εξουσιοδοτημένος να εκτελέσει την ενέργεια αυτή, ένα HTTP response κωδικού σφάλματος "403" θα δημιουργηθεί από το ίδιο το middleware.

```php
use App\Models\Post;

Route::put('/post/{post}', function (Post $post) {
    // The current user may update the post...
})->middleware('can:update,post');
```

Εικόνα 3.9 Παράδειγμα χρήσης του can middleware σε ένα route

ΚΕΦΑΛΑΙΟ 4 FRONT-END ΚΑΙ ΑΣΦΑΛΕΙΑ

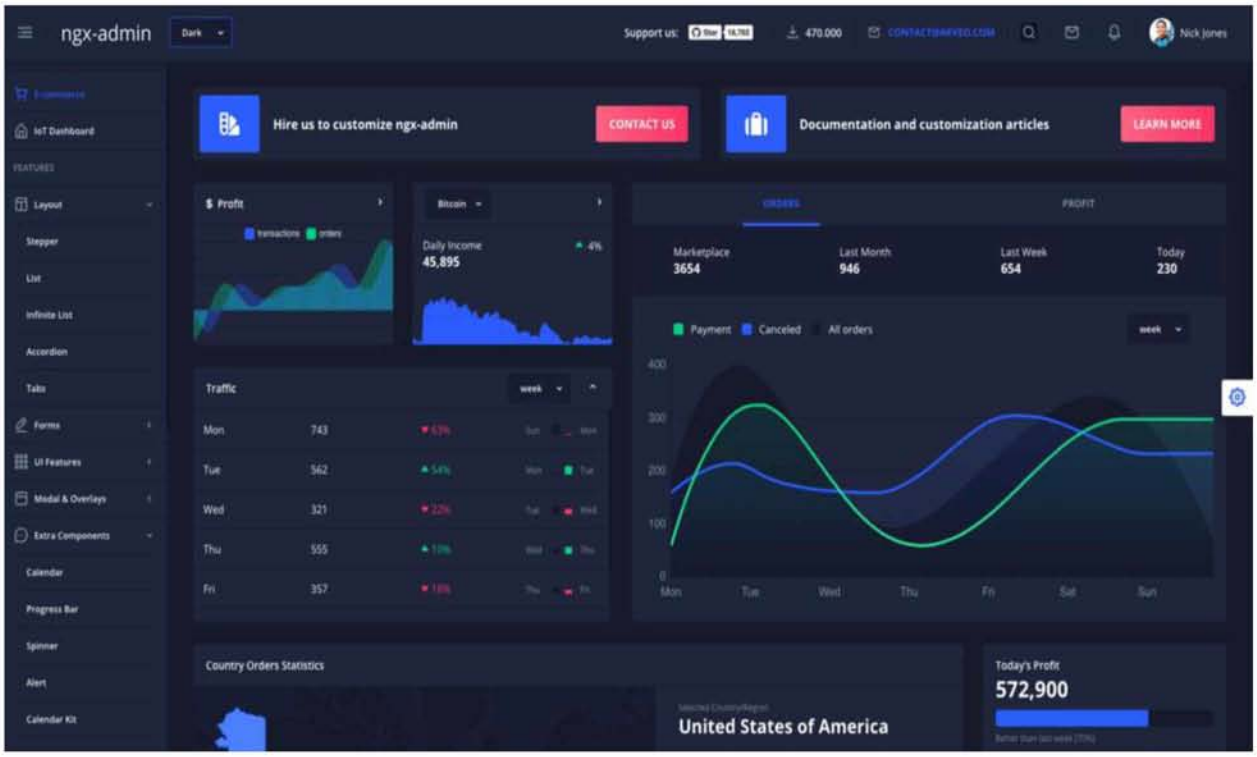
4.1 Πολλαπλές διεπαφές χρήστη

Στις μέρες μας η ύπαρξη πολλαπλών διεπαφών χρήστη σε μία διαδικτυακή εφαρμογή καθιστά την εφαρμογή αυτή στην κατηγορία της πλατφόρμας με πολλαπλές χρήσεις. Οι χρήστες αποκτούν διαφορετικά δικαιώματα και προσβάσεις στην διαδικτυακή εφαρμογή αυτή. Ενώ παλαιότερα η ανάγκη για ύπαρξη διαφορετικών επιπέδων πρόσβασης και λειτουργιών για διαφορετικούς χρήστες θα σήμαινε και την ύπαρξη διαφορετικών εφαρμογών, τώρα συνυπάρχουν στην ίδια διαδικτυακή εφαρμογή.

Με αυτό τον τρόπο, ένας διαχειριστής μπορεί σε πραγματικό χρόνο να εκτελεί ενέργειες που αποσκοπούν στη διαχείριση των μελών της διαδικτυακής εφαρμογής. Έτσι, επιτυγχάνεται καλύτερη και πιο άμεση διαχείριση της ιστοσελίδας.

Παράδειγμα διαδικτυακής εφαρμογής πολλαπλών διεπαφών:

Εικόνα 3.9 Διεπαφή user

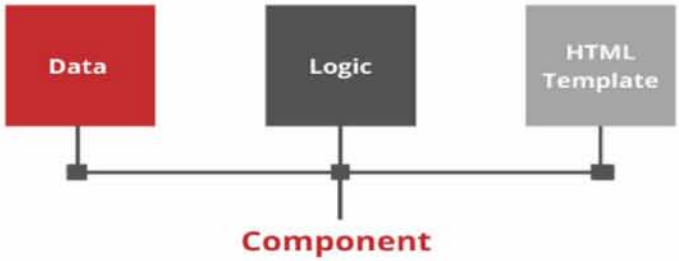


Εικόνα 3.9 Διεπαφή admin

4.2 Components and Modules

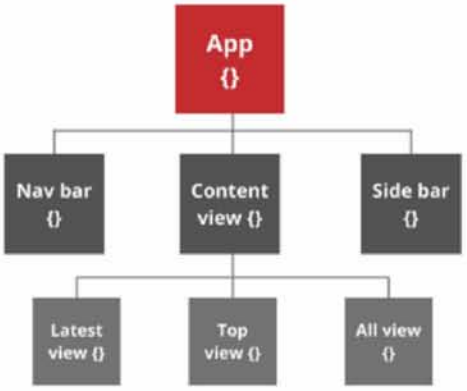
4.2.α Components

Τα Components είναι το πιο βασικό κομμάτι δημιουργίας UI (User interface) σε μια διαδικτυακή εφαρμογή που έχει αναπτυχθεί χρησιμοποιώντας το Angular framework. Μπορούμε να έχουμε ένα ή περισσότερα Components, μια εφαρμογή Angular περιλαμβάνει ένα «δέντρο» από Components. Ένα Component ενθυλακώνει τα δεδομένα και τη λογική σε όλα όσα ο χρήστης βλέπει.



Εικόνα 3.9 Λογική ενός Component

Μπορούμε να τμηματοποιήσουμε την διαδικτυακή εφαρμογή μας σε πολλαπλά Components και να τα επαναχρησιμοποιούμε μέσα στην εφαρμογή ή και σε τρίτες εφαρμογές. Σε κάθε εφαρμογή έχουμε τουλάχιστον ένα Component το οποίο είναι το App Component ή Route Component. Παρακάτω βλέπουμε ένα παράδειγμα της τμηματοποίησης αυτής.



Εικόνα 3.9 Δέντρο από Components ξεκινώντας από το App Component

Όπως παρατηρούμε υπάρχει μια σχέση πατέρα – παιδιού μεταξύ των Components μιας Angular διαδικτυακής εφαρμογής. Και μέσω του variable binding μπορούμε να έχουμε αμφίδρομη ανταλλαγή δεδομένων μεταξύ δυο Components που έχουν τη σχέση αυτή. Αυτός είναι και ένας από τους πιο σημαντικούς λόγους για τη χρήση των Components, καθώς μας επιτρέπει να μεταφέρουμε δεδομένα σε οποιοδήποτε view της διαδικτυακής εφαρμογής μας.

Ένα Component αποτελείται από τρία μέρη: την κλάση, το template και τα μεταδεδομένα. Η κλάση είναι όπως μια κλάση οποιασδήποτε άλλης γλώσσας και περιλαμβάνει μεταβλητές και συναρτήσεις. Αυτή περιλαμβάνει τον κώδικα που ακολουθεί – υποστηρίζει το view και είναι γραμμένο σε TypeScript. Το template είναι αυτό που καθιστά το view κατάλληλο για προβολή στο χρήστη. Αυτό περιλαμβάνει τον HTML κώδικα και το styling (CSS, SCSS, κ.ά.) αρχείο που το συνοδεύει. Αυτό το μέρος περιλαμβάνει επίσης το binding και τα directives. Τα μεταδεδομένα περιέχουν τα επιπλέον δεδομένα που έχουν οριστεί για την κλάση.

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

@Component({
  selector: 'app-root',
  template: '<h1>My Angular App</h1>'
})
export class AppComponent {}
```

Εικόνα 3.9 Παράδειγμα ενός Component

4.2.β Modules

Στο Angular framework ένα Module είναι ένας μηχανισμός ομαδοποίησης των components, directives, pipes και services με τέτοιο τρόπο που να μπορεί να συνδεθεί με άλλα Modules και να αποτελέσει μια διαδικτυακή εφαρμογή. Μια Angular εφαρμογή μπορεί να χαρακτηριστεί ως ένα «πάζλ» που το κάθε κομμάτι του είναι ένα Module.

Ένας ακόμα τρόπος να κατανοήσουμε τα Angular Modules είναι οι κλάσεις. Στις κλάσεις, μπορούμε να ορίσουμε public ή private συναρτήσεις. Οι public συναρτήσεις αποτελούν τμήματα του κώδικα που μπορούν να αλληλοεπιδράσουν με την υπόλοιπη εφαρμογή ενώ οι private συναρτήσεις είναι «κρυφές». Με τον ίδιο τρόπο, ένα Module μπορεί να εξάγει ή να κρύβει components, directives, pipes και services στην υπόλοιπη εφαρμογή. Τα εξαγόμενα στοιχεία μπορούν να χρησιμοποιηθούν από άλλα Modules, ενώ τα μη εξαγόμενα απλά χρησιμοποιούνται μέσα στο ίδιο το Module και δεν είναι άμεσα προσβάσιμα από άλλα Modules της εφαρμογής μας.

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

@NgModule({
  imports: [ ... ],
  declarations: [ ... ],
  bootstrap: [ ... ]
})
export class AppModule { }
```

Εικόνα 3.9 Παράδειγμα ενός Module

Για να ορίσουμε ένα Module πρέπει να χρησιμοποιήσουμε το decorator “NgModule”. Στο πιο πάνω παράδειγμα έχουμε μετατρέψει την κλάση AppModule σε ένα Angular Module απλά χρησιμοποιώντας αυτό το decorator. Αυτό το decorator απαιτεί τουλάχιστον δυο properties: imports και declarations.

Το property imports χρειάζεται ένα πίνακα από Modules. Σε αυτό το σημείο είναι που ορίζουμε τα κομμάτια του «παζλ» που αποτελεί την Angular διαδικτυακή εφαρμογή μας. Το property declarations χρειάζεται ένα πίνακα από Components, directives και pipes που αποτελούν κομμάτι του module.

4.3 Resolvers

Ένα από τα πολλά πλεονεκτήματα του να χρησιμοποιούμε το Angular framework είναι και η χρήση των resolvers. Οι resolvers αποτελούν μία σύγχρονη λύση ομαλής εμφάνισης αλλά και προκαταβολικής μεταφόρτωσης των δεδομένων μίας διαδικτυακής εφαρμογής.

Αναπτύσσοντας μια διαδικτυακή εφαρμογή με πολλαπλές κλήσεις στον εξυπηρετητή/back – end είναι πιθανό να προκύψουν προβλήματα. Μια «κλήση» API στο back – end μιας διαδικτυακής εφαρμογής έχει αρκετές καθυστερήσεις. Αυτές οι καθυστερήσεις συντελούν σε ένα αρνητικό UX

Ένας resolver είναι μία κλάση που υλοποιεί το Resolve Interface του πακέτου Angular Router της Angular. Πιο συγκεκριμένα, ένας Resolver είναι ένα service το οποίο λειτουργεί ως ενδιάμεσο λογισμικό και εκτελείται πριν τη μεταφόρτωση ενός component.

Στην περίπτωση που θέλουμε να εμφανίσουμε μια λίστα – πίνακα από δεδομένα π.χ. χρήστες, μέσα σε ένα component, δημιουργείται πρόβλημα καθώς τα δεδομένα λαμβάνονται από το back – end αφότου γίνει η μεταφόρτωση και εμφάνιση του component και η λίστα – πίνακας είναι κενή. Εδώ έρχεται να δώσει τη λύση ένας Route Resolver ο οποίος θα φορτώσει τα εν λόγω δεδομένα προκαταβολικά με την ενεργοποίηση του συγκεκριμένου Route με αποτέλεσμα κατά την μεταφόρτωση του component τα δεδομένα εμφανίζονται με τη δομή που θα έπρεπε.

```
@Injectable({ providedIn: 'root' })
export class HeroResolver implements Resolve<Hero> {
  constructor(private service: HeroService) {}

  resolve(
    route: ActivatedRouteSnapshot,
    state: RouterStateSnapshot
  ): Observable<any>|Promise<any>|any {
    return this.service.getHero(route.paramMap.get('id'));
  }
}
```

Εικόνα 3.9 Παράδειγμα μίας κλάσης Resolver

Στην παραπάνω κλάση παρατηρούμε πως η συνάρτηση "resolve" δέχεται δύο ορίσματα, ένα τύπου "ActivatedRouteSnapshot" και ένα τύπου "RouterStateSnapshot". Μέσω των ορισμάτων αυτών μπορούμε να έχουμε πρόσβαση στα Route parameters που ίσως μας είναι χρήσιμα στην επερχόμενη κλήση API. Στη συνέχεια μέσω της κλήσης API λαμβάνουμε τα δεδομένα από το back – end και τα χρησιμοποιούμε στο εκάστοτε component.

4.4 Interceptors

4.4.α Τι είναι

Οι Interceptors, είναι ένας απλός τρόπος που μας παρέχει το Angular framework να τροποποιούμε τα HTTP requests σε όλο το εύρος μιας διαδικτυακής εφαρμογής, πριν αυτά σταλούν στον εξυπηρετητή/back – end. Αυτό είναι ένα πολύ χρήσιμο εργαλείο που μας επιτρέπει να διαμορφώσουμε token αυθεντικοποίησης, να κρατήσουμε logs των requests που στέλνονται και γενικότερα να προσθέτουμε headers στα requests που στέλνει η διαδικτυακή εφαρμογή μας. Χωρίς τους interceptors θα έπρεπε να υλοποιούμε αυτές τις ενέργειες ξεχωριστά για κάθε HTTP κλήση της εφαρμογής μας.

4.4.β Υλοποίηση

Για να υλοποιήσουμε έναν Interceptor, πρέπει να δημιουργήσουμε μία κλάση που να υλοποιεί τη μέθοδο Intercept του πακέτου HttpInterceptor της Angular.

Για παράδειγμα έχουμε ένα Interceptor που εμφανίζει κάθε request που γίνεται από την διαδικτυακή εφαρμογή:

```
@Injectable()
export class RequestLogInterceptor implements HttpInterceptor {

  intercept(
    request: HttpRequest<any>, next: HttpHandler
  ) : Observable<HttpEvent<any>> {

    console.log(request.url);
    return next.handle(request);

  }
}
```

Εικόνα 3.9 Παράδειγμα μίας κλάσης Interceptor

Για να εφαρμόσουμε τα οφέλη του Interceptor σε όλη την διαδικτυακή εφαρμογή, πρέπει να δηλώσουμε την κλάση του Interceptor μέσα στο module της εφαρμογής αυτής. Αυτό ισχύει ακόμα και για τη χρήση πολλαπλών Interceptors. Επίσης, είναι σημαντικό να αναφέρουμε πως το Angular εφαρμόζει τους Interceptors με τη σειρά που τους δηλώνουμε στο module.

4.5 Guards

Το Angular framework έρχεται με ένα μεγάλο εύρος από προεγκατεστημένα πακέτα τα οποία είναι εξαιρετικά χρήσιμα για το χειρισμό της ασφάλειας μιας διαδικτυακής εφαρμογής. Ένα από αυτά είναι και τα Route Guards. Τα Route Guards της Angular είναι interfaces τα οποία ορίζουν εάν ο Angular Router θα πρέπει να επιτρέψει την πλοήγηση στο απαιτούμενο route. Αυτό κρίνεται από την τιμή true ή false που επιστρέφει η κλάση η οποία υλοποιεί το Guard interface.

Υπάρχουν πέντε διαφορετικά είδη από Guards και καθένα από αυτά καλείται σε συγκεκριμένη σειρά. Ο Router τροποποιείται αναλόγως ποιο Guard χρησιμοποιείται. Τα είδη των Guards είναι:

- canActivate
- canActivateChild
- canDeactivate
- canLoad
- resolve

```
@Injectable()
class CanActivateTeam implements CanActivate {
  constructor(private permissions: Permissions, private currentUser: UserToken) {}

  canActivate(
    route: ActivatedRouteSnapshot,
    state: RouterStateSnapshot
  ): Observable<boolean|UrlTree>|Promise<boolean|UrlTree>|boolean|UrlTree {
    return this.permissions.canActivate(this.currentUser, route.params.id);
  }
}
```

Εικόνα 3.9 Παράδειγμα μίας κλάσης Guard που υλοποιεί τη συνάρτηση canActivate

Το canActivate είναι ένα interface που μια κλάση μπορεί να υλοποιήσει για να μετατραπεί σε Guard και να έχει το ρόλο του να αποφασίζει εάν ένα Route μπορεί να

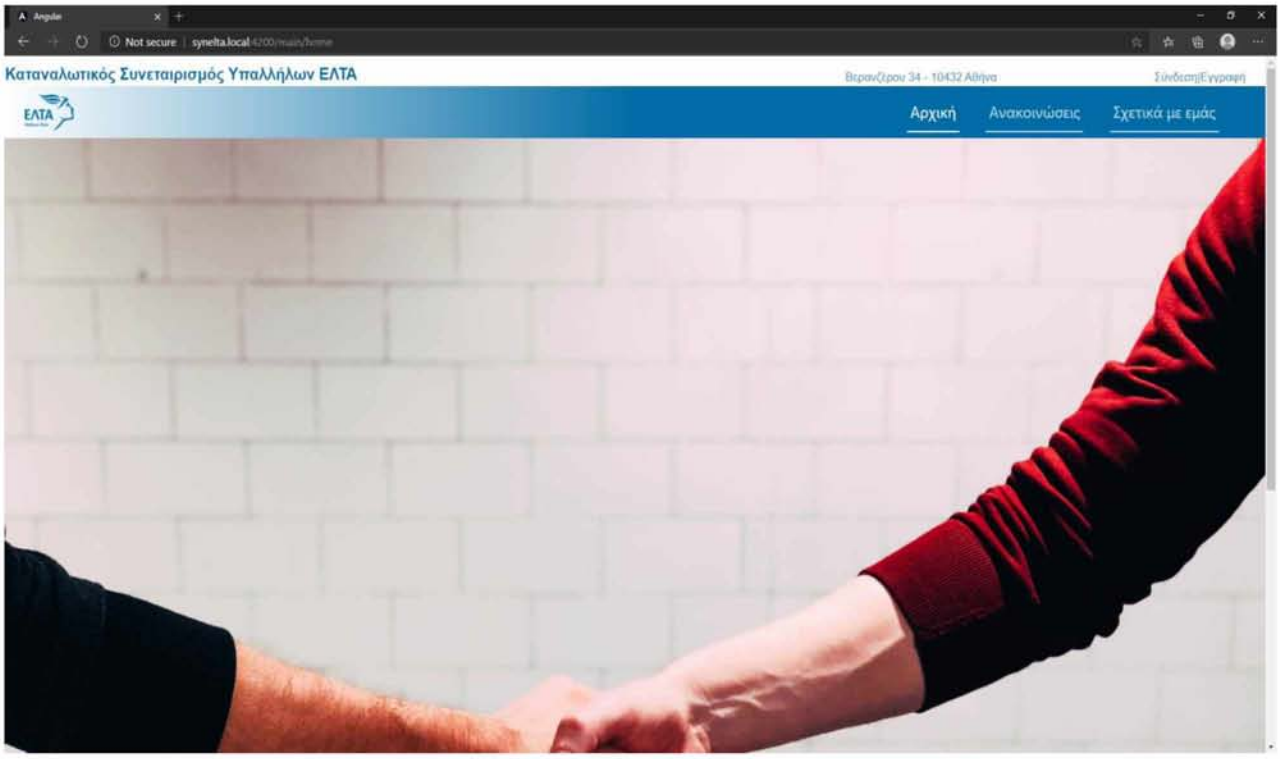
ενεργοποιηθεί ή όχι. Εάν όλα τα Guards επιστρέφουν true τότε η πλοήγηση συνεχίζεται κανονικά. Εάν οποιοδήποτε Guard συνδεδεμένο με Routes τα οποία οδηγούν στο view που θέλουμε να πλοηγηθούμε, επιστρέψουν false, τότε η πλοήγηση ακυρώνεται.

Ομοίως και τα υπόλοιπα interfaces, με τη διαφορά ότι, το canActivateChild αναφέρεται στην πρόσβαση ενός child Route, το canDeactivate είναι το αντίστροφο του canActivate, το canLoad αναφέρεται στην ικανότητα ενός χρήστη να μεταφορτώσει child Routes.

ΚΕΦΑΛΑΙΟ 5 USE CASE ΚΑΙ ΠΑΡΑΔΕΙΓΜΑΤΑ

5.1 Εισαγωγή

Στα πλαίσια της ανάπτυξης μιας διαδικτυακής εφαρμογής με όλα τα παραπάνω χαρακτηριστικά, στην εργασία αυτή έχουμε αναπτύξει μια εφαρμογή που έχει τη χρήση της διαδικτυακής πλατφόρμας ενημέρωσης, αλλά και διαχείρισης των μελών του καταναλωτικού συνεταιρισμού των Ελληνικών ταχυδρομείων.



Εικόνα 3.9 Παράδειγμα home page διαδικτυακής εφαρμογής

Στο παραπάνω παράδειγμα παρατηρούμε τη χρήση της μπάρας πλοήγησης, με την οποία οι χρήστες παντός επιπέδου εξουσιοδότησης μπορούν να πλοηγηθούν στις διάφορες υπηρεσίες που παρέχονται από τη διαδικτυακή εφαρμογή.

Στην διαδικτυακή εφαρμογή αυτή «χτισμένη» με Angular στο front – end, έχουμε τρία διαφορετικά επίπεδα εξουσιοδότησης και κατ' επέκταση πρόσβασης. Τα επίπεδα αυτά είναι:

- Guest
- Member

- Admin

Στο πρώτο επίπεδο (Guest) που ο χρήστης δεν είναι μέλος και δεν διαθέτει στοιχεία πρόσβασης στην πλατφόρμα, μπορεί να εξερευνήσει μόνο βασικές ενημερωτικές πληροφορίες για τον καταναλωτικό συνεταιρισμό. Αυτές είναι η ενημέρωση για:

- Τις ανακοινώσεις
- Την ίδρυση, τα στελέχη και την τοποθεσία

του καταναλωτικού συνεταιρισμού των Ελληνικών ταχυδρομείων.

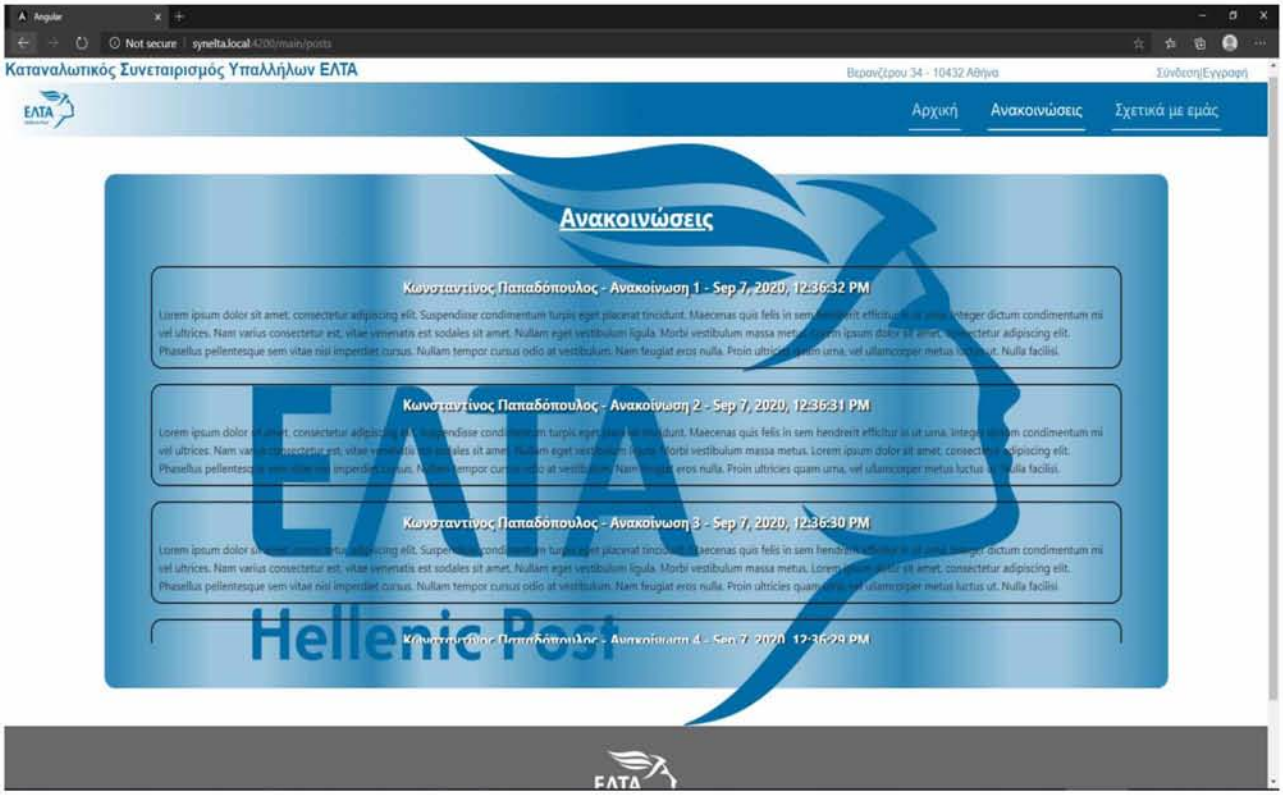
Στο δεύτερο επίπεδο(Member) που ο χρήστης είναι επικυρωμένο μέλος της πλατφόρμας με στοιχεία πρόσβασης σε αυτήν, έχει επιπλέον τη δυνατότητα της προβολής των στοιχείων εγγραφής του στην πλατφόρμα, αλλά και της επεξεργασίας ορισμένων από αυτά.

Στο τρίτο επίπεδο (Admin) του διαχειριστή, ο χρήστης έχει πλήρη εξουσιοδότηση σε όλα τα επίπεδα και views της πλατφόρμας. Πέραν λοιπόν από τις δυνατότητες ενός μέλους, ένας διαχειριστής έχει τις εξής δυνατότητες:

- Δημιουργία, τροποποίηση, απενεργοποίηση και διαγραφή ανακοινώσεων
- Δημιουργία, τροποποίηση, απενεργοποίηση και διαγραφή χρηστών
- Αλλαγή των πληροφοριών επικοινωνίας της πλατφόρμας

Οι ενέργειες σχετικά με τους χρήστες εκτελούνται από το view «Διαχείριση Χρηστών» στο οποίο έχει πρόσβαση μόνο ένας χρήστης με επίπεδο εξουσιοδότησης Admin. Το view αυτό ελέγχει την δυνατότητα πρόσβασης με τη χρήση Angular Route Guards.

5.2 Front – end παραδείγματα



Εικόνα 3.9 Παράδειγμα σελίδας ανακοινώσεων διαδικτυακής εφαρμογής

Εικόνα 3.9 Παράδειγμα σελίδας επικοινωνίας διαδικτυακής εφαρμογής

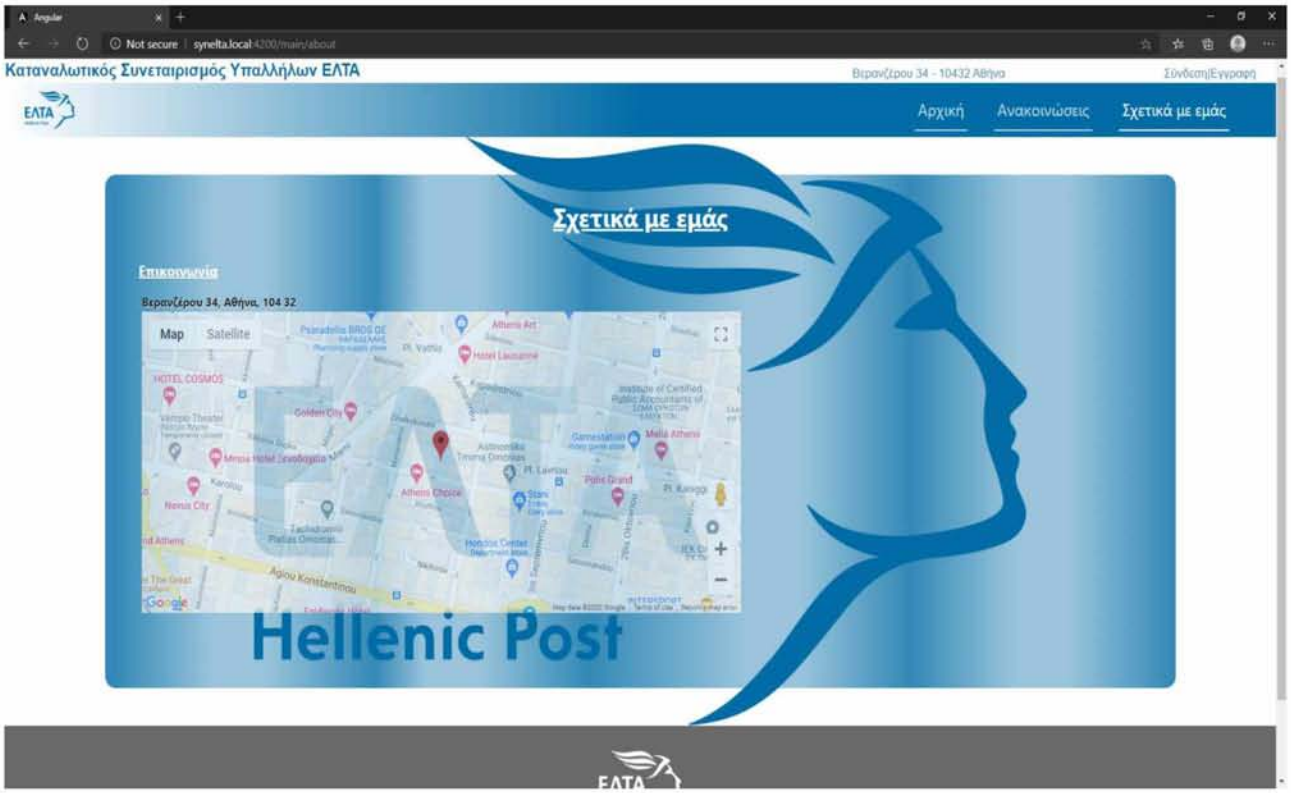
Όπως είναι εμφανές από τις παραπάνω εικόνες, οι πληροφορίες εμφανίζονται στο χρήστη σε ένα πλαίσιο. Στα πλαίσια αυτά καθόλη την έκταση της διαδικτυακής αυτής εφαρμογής, έχει εφαρμοστεί CSS τεχνική για hidden scrolling, ώστε να έχουμε ένα πιο ικανοποιητικό αισθητικό αποτέλεσμα όσον αφορά views με δυναμικό περιεχόμενο το οποίο μπορεί έπειτα από εκτεταμένη χρήση της πλατφόρμας να οδηγήσει σε πολλά δεδομένα και κατ' επέκταση μεγάλα scrollbars στα εν λόγω views.

Επιπλέον, είναι σημαντικό να αναφέρουμε πως στο view – σελίδα των ανακοινώσεων έχουμε χρησιμοποιήσει την τεχνική του Angular framework για το InfiniteScroll και το ομώνυμο εξωτερικό πακέτο. Με τον τρόπο αυτό δεν υπερφορτώνουμε το front – end κομμάτι της διαδικτυακής εφαρμογής και κατ' επέκταση τον browser – φυλλομετρητή στον οποίο αυτή εκτελείται με υπερβολική πληροφορία – δεδομένα, εκτός αν πρόκειται για την κατά βούληση ζήτησή τους. Αυτό συμβαίνει καθώς το front – end σε συνεργασία με το κατάλληλο endpoint του back – end εμφανίζει στο πλαίσιο των ανακοινώσεων τα κατάλληλα δεδομένα χρησιμοποιώντας σελιδοποίηση – pagination. Με τον τρόπο αυτό, τα δεδομένα εμφανίζονται στο χρήστη με χρονική διάταξη και ανά σελίδα, η οποία περιέχει ένα συγκεκριμένο αριθμό ανακοινώσεων. Κάθε φορά που εκτελείται scroll προς τα κάτω εντός του πλαισίου, γίνεται αίτηση στο back – end της εφαρμογής μας για μια ακόμα σελίδα. Έτσι επιτυγχάνεται η αποφόρτωση του όγκου των δεδομένων από το view των ανακοινώσεων, κρατώντας την διαδικτυακή μας εφαρμογή «ελαφριά».

```
1.  <div class="posts-
    contain-
    er" infiniteScroll [scrollWindow]="false" [infiniteScrollDisabled]="false" [infiniteScrollDistanc
    e]="2" [infiniteScrollThrottle]="50" (scrolled)="onScroll()">
2.    <div class="posts" *ngFor="let post of posts">
3.      <div class="post" [ngClass]="{'disabled-post' : !!post?.deleted_at}">
4.        <div class="post-header">
5.          {{post?.user?.first_name}} {{post?.user?.last_name}} - {{post?.title}} -
     {{post?.created_at | date: 'medium': 'UTC+3'}}
6.        </div>
7.        <div class="post-content">
8.          {{post?.content}}
9.        </div>
```

Απόσπασμα κώδικα χρήσης του InfiniteScroll

Ένα ακόμα θετικό της χρήσης του Angular framework είναι η ευκολία σύνδεσης της διαδικτυακής μας εφαρμογής με APIs άλλων εφαρμογών. Στη δική μας εφαρμογή παράδειγμα αποτελεί η σύνδεση με το API των Google Maps. Η σύνδεση γίνεται πιο εύκολη με τη χρήση του πακέτου GoogleMaps του Angular.



Εικόνα 3.9 Παράδειγμα πλαισίου Google Maps

Όπως παρατηρούμε στην παραπάνω εικόνα, το πακέτο GoogleMaps μετά την απόκτηση API κλειδιού από την Google για τη χρήση του, μας δίνει τη δυνατότητα της "out of the box" χρήσης του στην διαδικτυακή εφαρμογή μας, χωρίς καμία άλλη παραμετροποίηση πέραν του ορισμού των κατάλληλων συντεταγμένων τοποθεσίας και διαστάσεων εμφάνισης του πλαισίου.

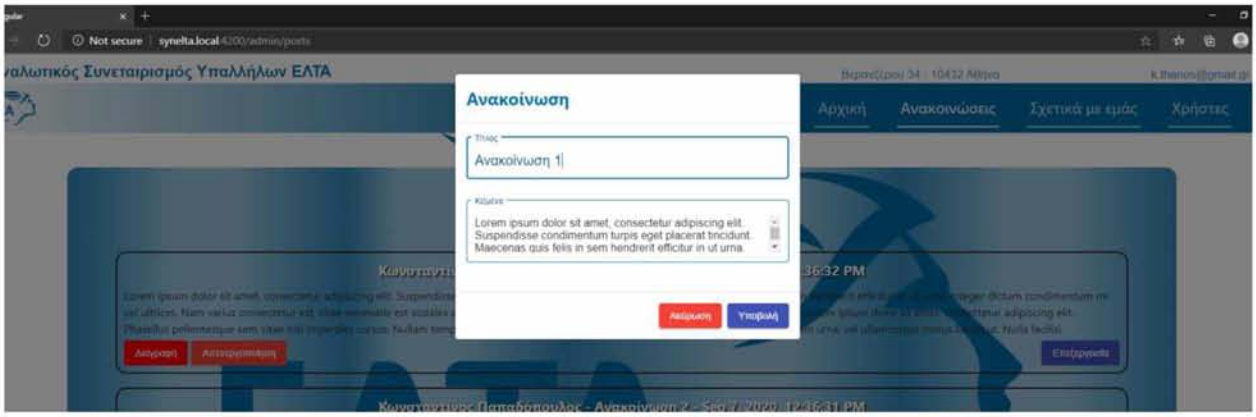
```
1.  <div class="map-container" [ngClass]="{'map-container-activated' : opacityActivate}">
2.    <google-map id="googleMap" height="400px" width="100%"
3.              [center]="center"
4.              (mapClick)="opacityActivate = true"
5.              (mapDragstart)="opacityActivate = true">
6.      <map-
    mark-
    er [position]="center" [title]="marker.getTitle()" [options]="{animation: marker.getAnimation()}" (mapC
    lick)="openMaps($event)"></map-marker>
7.    </google-map>
8.  </div>
```

Απόσπασμα κώδικα χρήσης του Google Maps

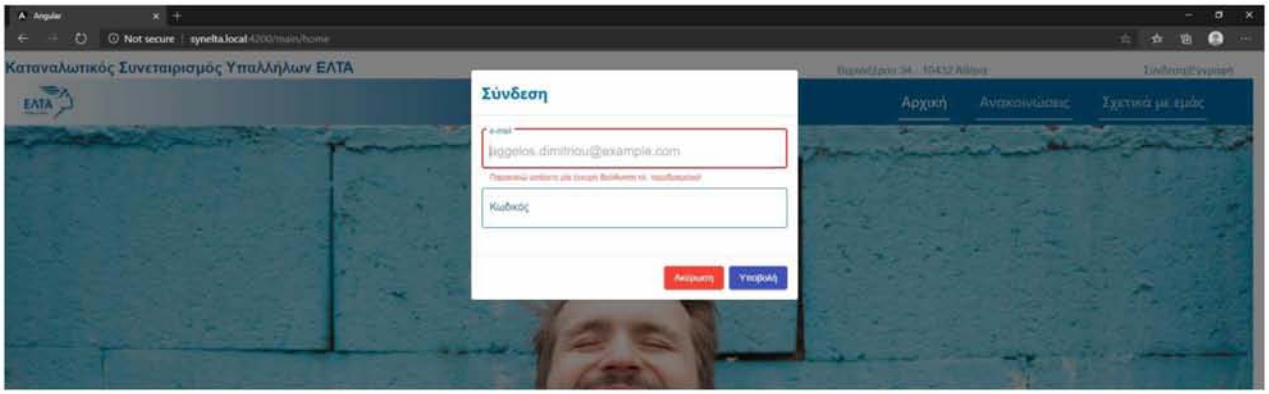
Για να πετύχουμε ακόμα καλύτερη εμφάνιση των views του front – end αλλά και για την πιο εύκολη εισαγωγή των δεδομένων από τους χρήστες για τη δημιουργία και τροποποίηση των διαφόρων στοιχείων της διαδικτυακής εφαρμογής χρησιμοποιούμε τα

NgbModals. Τα NgbModals είναι ένα εξωτερικό χαρακτηριστικό που περιλαμβάνεται στο εξωτερικό πακέτο "ng-bootstrap" του Angular.

Όπως παρατηρούμε στις παρακάτω εικόνες, τα NgbModals εμφανίζονται ως pop – up παράθυρα με φόρμες προς συμπλήρωση των κατάλληλων στοιχείων. Επομένως η χρήση τους στην εργασία μας είναι εκτεταμένη. Τα χρησιμοποιούμε για τη σύνδεση και την εγγραφή ενός χρήστη στην πλατφόρμα και για τη δημιουργία και τροποποίηση ανακοινώσεων και χρηστών από τον διαχειριστή. Με τον τρόπο αυτό τα NgbModals μας βοηθούν να διατηρήσουμε την διαδικτυακή εφαρμογή μας απλή και να μην επεκταθούμε σε χρήση επιπλέον views που θα είχαν ως σκοπό την εμφάνιση forms για εισαγωγή στοιχείων. Αντιθέτως, όλες οι φόρμες εισαγωγής υπάρχουν μέσα στα NgbModals και τα στοιχεία αυτά μετά το επιτυχές κλείσιμο του NgbModal μεταφέρονται στο ανάλογο component που έχει ως ρόλο το χειρισμό τους και την αποστολή τους στο back – end μέσω κλήσης API.



Εικόνα 3.9 Παράδειγμα NgbModal για επεξεργασία ανακοίνωσης



Εικόνα 3.9 Παράδειγμα NgbModal για είσοδο χρήστη

```
1.  createPost(){
2.    const modalRef = this.modalService.open(PostModalComponent);
3.    modalRef.result.then(
4.      result => {
5.        this.postsService.createPost(result).subscribe(response => {
6.          this.posts.unshift(response?.data);
```

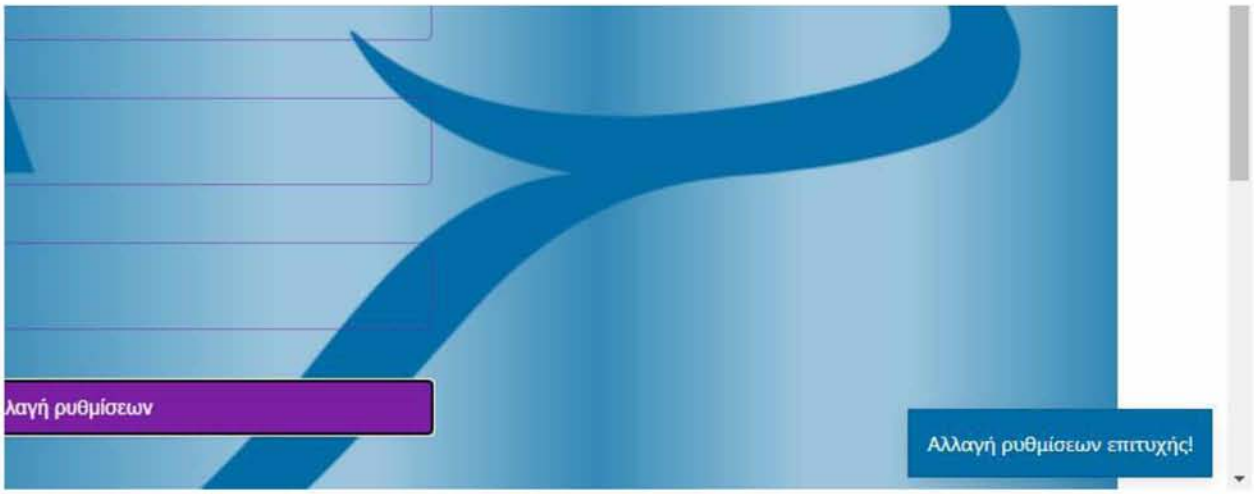
```
7.          this.posts.pop();
8.          this.toastService.show({message: response?.message, type: 'success'});
9.        });
10.     },
11.     (cancelled) => { }
12.   );
13. }
```

Απόσπασμα κώδικα με συνάρτηση που περιλαμβάνει τη δημιουργία NgbModal

Στο παραπάνω απόσπασμα κώδικα παρατηρούμε τη χρήση μιας συνάρτησης που περιλαμβάνει τη δημιουργία NgbModal για μια ανακοίνωση. Και με την υποβολή των στοιχείων αυτών και το επιτυχές κλείσιμο του NgbModal γίνεται κλήση API στο back – end για την υποβολή αυτών των στοιχείων στη βάση δεδομένων.

Επίσης, κάτι ακόμα που παρατηρούμε στις εικόνες που αφορούν τα NgbModals είναι η χρήση των Material Forms. Τα Material Forms, Inputs και Buttons συμπεριλαμβάνονται στο πακέτο Material του Angular. Με τη χρήση αυτών καταφέρνουμε ένα καλύτερο UI/UX (User Interface/User Experience). Και επιπλέον επιτυγχάνουμε την πιο εύκολη εμφάνιση των errors που εμφανίζονται σε αυτά τα forms από ελλιπή ή λανθασμένα στοιχεία.

Τέλος, μια ακόμα αισθητική βελτίωση του UI/UX στην διαδικτυακή εφαρμογή μας όπως παρατηρούμε στην παρακάτω εικόνα, είναι η χρήση των Toast notifications. Με τη χρήση των Toast notifications επιτυγχάνουμε την άμεση ενημέρωση του χρήστη για το αποτέλεσμα μιας ενέργειας που εκτέλεσε μέσα στην εφαρμογή. Μερικά παραδείγματα θα μπορούσαν να είναι η επιτυχής είσοδος του χρήστη στην εφαρμογή, η επιτυχής διαγραφή ενός χρήστη από το διαχειριστή ή ακόμα και η αποτυχία ενός χρήστη να τροποποιήσει τα στοιχεία του. Τα Toast notifications περιλαμβάνονται στο εξωτερικό πακέτο "Toast" του "ng-bootstrap". Το μόνο που χρειάζεται να κάνουμε πέρα από την εισαγωγή του πακέτου είναι να εισάγουμε μέσα στο αρχικό μας component το component που έχουμε δημιουργήσει και περιλαμβάνει το template και τον HTML κώδικα που αφορά τα Toast notifications.



Εικόνα 3.9 Παράδειγμα ενός Toast Notification

Καλή πρακτική αποτελεί η εμφάνιση των back – end response messages απευθείας μέσω των Toast notifications για αποφυγή περαιτέρω παραμετροποίησης τους αλλά και για τη συνοχή του κώδικα.

5.3 Back – end παραδείγματα

5.3.α Εισαγωγή

Το back – end της εργασίας μας χρησιμοποιεί όλα τα προαναφερθέντα εργαλεία που μας παρέχει το Laravel framework για την επίτευξη της μέγιστης ασφάλειας και ταχύτητας της διαδικτυακής εφαρμογής μας. Καθώς το back – end κομμάτι μιας διαδικτυακής εφαρμογής είναι και αυτό που τροφοδοτεί το front – end με όλα τα δεδομένα, ευαίσθητα και μη της εφαρμογής αυτής, είναι και αυτό στο οποίο πρέπει να εφαρμοστούν τα περισσότερα μέτρα ασφαλείας.

5.3.β Enumerators

Ξεκινώντας, μπορούμε να αναφέρουμε πως ένας καλός τρόπος για να επιτευχθεί ο διαχωρισμός των χρηστών και κατ' επέκταση των διαφόρων επιπέδων εξουσιοδότησης, είναι τα Enumerators. Τα Enumerators αποτελούν αρχεία τα οποία περιλαμβάνουν μια αντιστοιχία γραμματοσειράς και ακεραίου αριθμού και έχουν ως σκοπό την ευκολία χαρακτηρισμού ενός property από μια λέξη. Πιο συγκεκριμένα μπορούμε να αντιστοιχίσουμε την ιδιότητα του "Admin" σε όλη την έκταση της εφαρμογής μας με τον αριθμό 2. Αυτό όπως βλέπουμε στο παρακάτω παράδειγμα μας διευκολύνει στον έλεγχο του επιπέδου πρόσβασης σε διάφορα σημεία της εφαρμογής.

```
1.  /** Allowing admin users to access soft deleted users/posts through route model binding */
2.  Route::bind('user', function ($id){
3.      return auth()->user()->type == UserType::Admin
4.          ? User::withTrashed()->findOrFail($id)
5.          : User::withoutTrashed()->findOrFail($id);
6.  });
```

Απόσπασμα κώδικα με παράδειγμα χρήσης του UserType Enumerator

5.3.γ Sessions – Authentication

Στην εφαρμογή αυτή χρησιμοποιούμε sessions για να παρακολουθούμε τη δραστηριότητα των χρηστών. Πιο συγκεκριμένα μέσω των sessions μπορούμε να διατηρούμε χρήσιμες πληροφορίες όπως την τελευταία είσοδο του χρήστη αλλά και κρυπτογραφημένο, το JWT authentication token που του αντιστοιχεί. Παρακάτω βλέπουμε ένα απόσπασμα κώδικα που περιλαμβάνει τη συνάρτηση μέσω της οποίας πραγματοποιείται μια εγγραφή χρήστη.

```
1.  public function register(RegisterRequest $request)
2.  {
3.      $user = $this->user_service->create($request, UserType::Member);
4.      $user->sendEmailVerificationNotification();
5.      if (!auth()->validate($request->only(['email', 'password']))) {
6.          return response()->json(['message' => 'Unauthorized'], 401);
7.      }
8.      $token = auth()->attempt($request->only(['email', 'password']));
9.      $session_id = JWTAuth::setToken($token)->getPayload()['session_id'];
10.     $session = Session::findOrFail($session_id);
11.     $this->session_service->update($session, ['token' => $token]);
12.     JsonResource::wrap('user');
13.     return AuthenticatedResource::make($user)-
    >additional(['token' => $token, 'message' => 'Εγγραφή επιτυχής!'])->response();
14. }
```

Όπως παρατηρούμε μετά τη δημιουργία του χρήστη στη βάση δεδομένων στη γραμμή 3, έχουμε την αποστολή του επιβεβαιωτικού email στο email που έχει καταχωρήσει ο χρήστης κατά την εγγραφή του. Στη συνέχεια μετά τον έλεγχο των στοιχείων του χρήστη έχουμε στις γραμμές 8 – 11 τη δημιουργία του JWT token και της ανάκτησης από τη βάση δεδομένων του session που αντιστοιχεί στο χρήστη και της ανανέωσης του πεδίου token αυτού με το νέο JWT token που δημιουργήθηκε.

5.3.8 Route Middleware

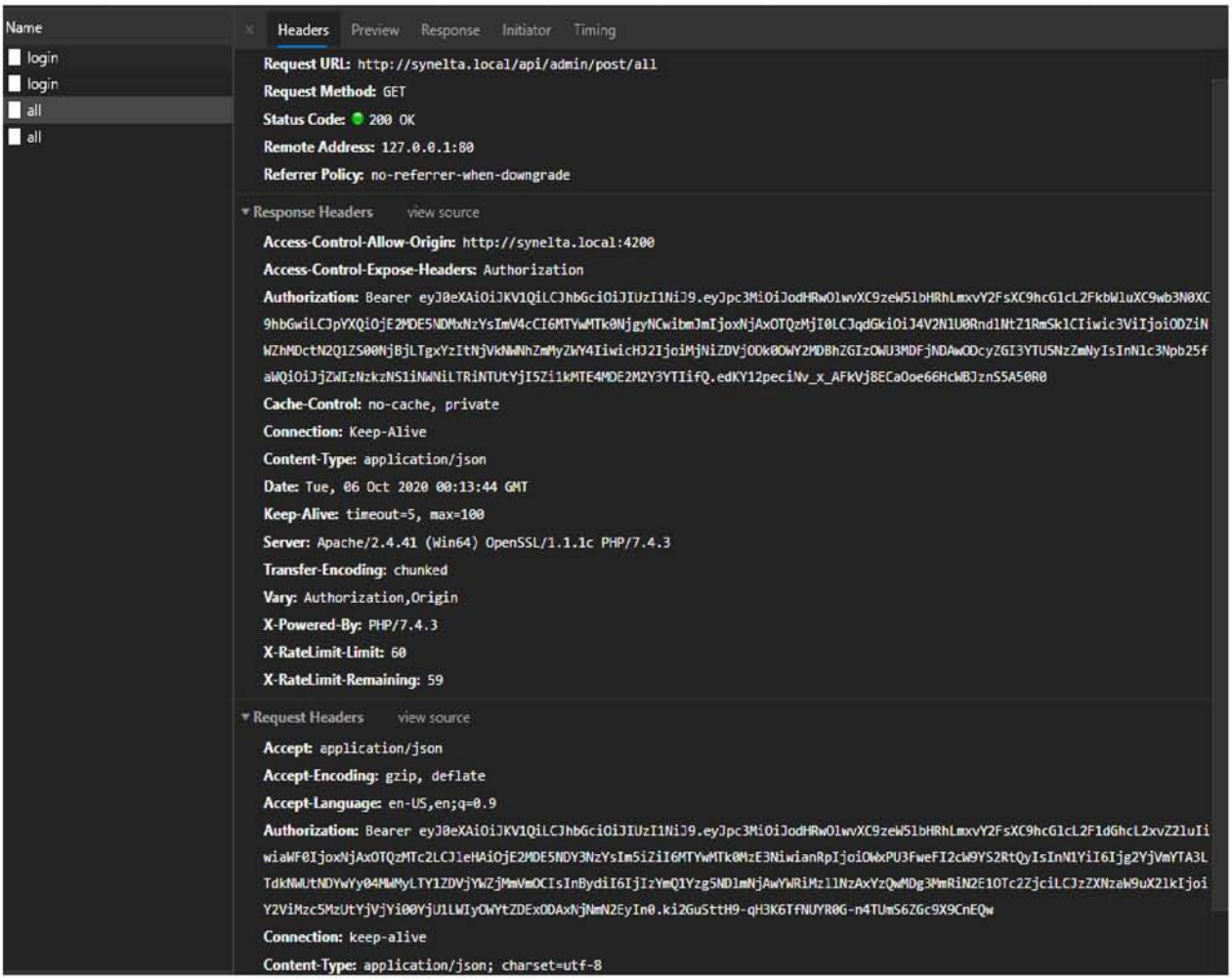
Μετά την επιτυχή εγγραφή ενός χρήστη όπως είδαμε στο παραπάνω παράδειγμα ή τη σύνδεσή του στην πλατφόρμα, κάθε request που πραγματοποιείται σε Routes του back – end τα οποία είναι προστατευμένα και απαιτούν εξουσιοδότηση για την πρόσβαση σε αυτά πρέπει να αυθεντικοποιείται με βάση το JWT token που φέρει το παρόν session του χρήστη. Επιπλέον, για να παρέχουμε περισσότερη ασφάλεια στη διαδικτυακή εφαρμογή μας πρέπει να ανανεώνουμε το JWT token του χρήστη σε κάθε request και να επιστρέφουμε στο front – end και στη βάση δεδομένων το ανανεωμένο. Σε αυτά τα μέτρα ασφαλείας έρχονται να δώσουν τη λύση τα Route Middleware του Laravel framework.

```
1.  Route::middleware(['auth:api', 'jwt.refresh_token', 'verified'])->group(function () {
2.
3.      Route::namespace('Admin')->prefix('admin')->middleware('can:isAdmin')->group(function (){
4.          Route::namespace('User')->prefix('user')->group(function () {
5.              Route::get('/all', [AdminController::class, 'getAll']);
6.              Route::get('/{user?}', [AdminController::class, 'get']);
7.              Route::post('/', [AdminController::class, 'create']);
8.              Route::put('/{user}', [AdminController::class, 'update']);
9.              Route::put('/{user}/enable', [AdminController::class, 'enable']);
10.             Route::put('/{user}/verify', [AdminController::class, 'verify']);
11.             Route::delete('/{user}', [AdminController::class, 'delete']);
12.             Route::delete('/{user}/disable', [AdminController::class, 'disable']);
13.         });
14. ..............
```

Στο παραπάνω απόσπασμα κώδικα παρατηρούμε τα Route Middleware τα οποία έχουν τοποθετηθεί σε ένα Group από Routes. Πρέπει να αναφέρουμε πως σημαντικό ρόλο έχει και η σειρά με την οποία είναι γραμμένα τα Route Middleware αυτά, καθώς αυτή είναι και η σειρά εκτέλεσής τους.

Πρώτο έρχεται το "auth:api", το οποίο είναι ένα Middleware που έρχεται μαζί με το Laravel framework. Έχοντας ορίσει στο "auth.php" configuration αρχείο τις κατάλληλες προϋποθέσεις για τη χρήση του JWT driver με το api authentication guard, το auth middleware δεχόμενο την παράμετρο api αυθεντικοποιεί το χρήστη χρησιμοποιώντας το JWT token που υπάρχει στον "Authorization" header του request.

Στη συνέχεια ακολουθεί το "jwt.refresh_token". Αυτό είναι ένα custom middleware το οποίο έχει ως μοναδικό σκοπό την ανανέωση και αντικατάσταση του ήδη υπάρχοντος JWT token με ένα νέο. Με τον τρόπο αυτό μειώνεται σημαντικά ο κίνδυνος υποκλοπής του token από κάποιον κακόβουλο χρήστη και η χρήση του για impersonation – πλαστοπροσωπίας άλλου χρήστη, καθώς το προηγούμενο token γίνεται invalidated – ακυρώνεται. Στην παρακάτω εικόνα παρατηρούμε την ύπαρξη διαφορετικού JWT token στο Authorization header του request και του response.



Εικόνα 3.9 Headers ενός request

```
1.  public function handle($request, Closure $next)
2.  {
3.      $response = $next($request);
4.      if (!$response->headers->has('Authorization')) {
5.          if ($this->auth && $this->auth->parser()->setRequest($request)->hasToken()) {
6.              try {
7.                  $session = Session::findOrFail($this->auth->parseToken()->getPayload()['session_id']);
8.                  $token = $this->auth->parseToken()->refresh();
9.              } catch (JWTException $e) {
10.                 throw new UnauthorizedHttpException('jwt-auth', $e->getMessage(), $e, $e->getCode());
11.             }
12.                 $this->session_service->update($session, ['token' => $token]);
13.                 $response->headers->set('Access-Control-Expose-
    Headers', 'Authorization');// header to allow f/e to access other headers
14.                 $response->headers->set('Authorization', 'Bearer '.$token);
15.         }
16.     }
17.     return $response;
18. }
```

Στο παραπάνω κομμάτι κώδικα παρατηρούμε πως πρόκειται για ένα after middleware. Αφού λοιπόν λάβουμε το request, στις γραμμές 4 – 5 γίνεται ο έλεγχος της ύπαρξης του token, στη συνέχεια στη γραμμή 8 γίνεται η ανανέωση του token και τέλος στις γραμμές 12 και 14 γίνεται η αντικατάσταση του παλιού token με το νέο στον header του request, αλλά και στη βάση δεδομένων.

Τέλος, παρατηρούμε τη χρήση του "verified" middleware, που πρόκειται επίσης για ένα ήδη υπάρχον middleware του Laravel framework. Το middleware αυτό έχει ως σκοπό τον έλεγχο στη βάση δεδομένων της επιβεβαίωσης του email του χρήστη. Σε αντίθετη περίπτωση όπως και το πρώτο middleware(auth:api) απαγορεύει την πρόσβαση στα περικλειόμενα Routes.

5.3.ε Gates

Η χρήση των Gates αποτελεί αναπόσπαστο κομμάτι ενός ασφαλούς back – end API μίας διαδικτυακής εφαρμογής και δεν θα μπορούσε να λείπει από την εργασία αυτή. Η χρήση των Gates είναι και ο τρόπος που ελέγχουμε τα επίπεδα εξουσιοδότησης στα Routes της εφαρμογής μας. Επομένως, στο παρακάτω παράδειγμα θα παρατηρήσουμε την χρήση των Gates μέσω middleware και πιο συγκεκριμένα του "can" middleware.

```
1.  Route::namespace('Member')->prefix('member')->middleware('can:isMember')->group(function (){
2.      Route::get('/{user}', [UserController::class, 'get']);
3.      Route::get('/', [UserController::class, 'getAll']);
4.      Route::put('/', [UserController::class, 'update']);
5.      Route::delete('/', [UserController::class, 'delete']);
6.  });
```

Όπως παρατηρούμε στο πάνω απόσπασμα κώδικα, η περικλειόμενη ομάδα από Routes, για να είναι προσβάσιμη, θα πρέπει να επιστραφεί τιμή true από το Gate που φέρει την ετικέτα "isMember", μέσω του "can" middleware. Παρακάτω βλέπουμε το κομμάτι κώδικα που περιλαμβάνει τα δύο υπάρχοντα Gates της εφαρμογής μας, βρισκόμενα στο αρχείο "AuthServiceProvider.php".

```
1.  public function boot()
2.  {
3.      $this->registerPolicies();
4.
5.      Gate::define('isMember', function ($user){
6.          return $user->type >= UserType::Member;
7.      });
8.      Gate::define('isAdmin', function ($user){
9.          return $user->type == UserType::Admin;
10.     });
11. }
```

ΚΕΦΑΛΑΙΟ 6 ΣΥΜΠΕΡΑΣΜΑΤΑ ΚΑΙ ΜΕΛΛΟΝΤΙΚΕΣ ΠΡΟΕΚΤΑΣΕΙΣ

6.1 Επίλογος

Συμπερασματικά, αυτή η εργασία μας δείχνει ότι η ασφάλεια στις διαδικτυακές εφαρμογές είναι πλέον ένα πολύ σημαντικό ζήτημα, καθώς πάνω σε αυτές στηρίζονται όλα τα δεδομένα των χρηστών στις μέρες μας. Αν δεν δοθεί λοιπόν η απαραίτητη προσοχή και λεπτομέρεια στην ασφάλεια, είναι εμφανές πως υπάρχουν πάντα οι κατάλληλοι τρόποι για κακόβουλη ή μη εξουσιοδοτημένη πρόσβαση σε μια διαδικτυακή εφαρμογή.

Σε γενική κλίμακα, είναι πολύ σημαντικό να αναφέρουμε πως όσον αφορά το θέμα της ασφάλειας δεν πρέπει να καθησυχαζόμαστε και πρέπει να ανακαλύπτουμε συνεχώς νέους τρόπους για τη βελτιστοποίηση της στις διαδικτυακές εφαρμογές που αναπτύσσουμε. Αυτό λοιπόν σημαίνει πως πέραν των εφαρμοσθέντων μεθόδων, υπάρχουν και άλλες τεχνικές τις οποίες δεν έχουμε αναλύσει σε αυτή την εργασία, αλλά και τεχνικές που δεν έχουν ακόμα ανακαλυφθεί. Πρέπει επομένως να δοκιμάζουμε πάντα την ασφάλεια των διαδικτυακών εφαρμογών μας και μένοντας ενημερωμένοι να εφαρμόζουμε τις πιο σύγχρονες και προηγμένες μεθόδους ασφαλείας σε αυτές.

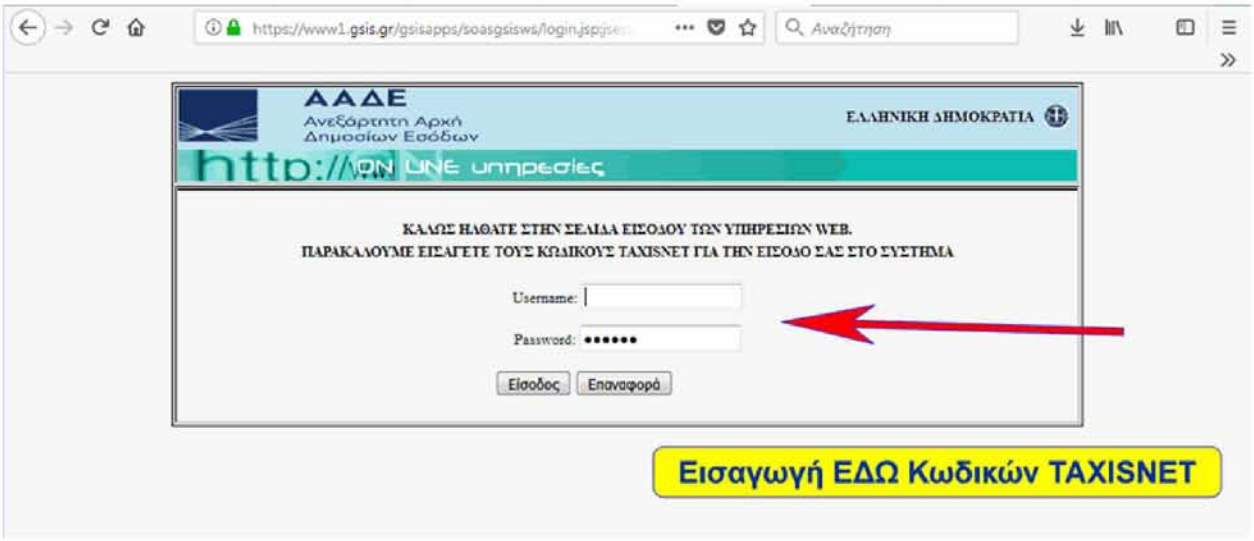
Επίσης, παρατηρούμε πως το Laravel framework είναι ένα πλέον ικανοποιητικό μέσο για την εφαρμογή προηγμένων τεχνικών ασφαλείας στο back – end API της διαδικτυακής εφαρμογής μας. Αυτό συμβαίνει καθώς περιλαμβάνει πληθώρα εσωτερικών αλλά και εξωτερικών πακέτων που βοηθούν στη διατήρηση της ασφάλειας στην εφαρμογή μας. Διαθέτει ενημερωμένο και εκτενές documentation. Επιπλέον, λαμβάνει συχνές ενημερώσεις στις εκδόσεις του με αποτέλεσμα να παραμένει σχετικό με τις εξελίξεις στην τεχνολογία. Λόγω του ότι το Laravel framework είναι ένα PHP framework, η πλειονότητα των εξωτερικών πακέτων που διαθέτει παραμένουν ενημερωμένα με τις τελευταίες εκδόσεις του.

Τέλος, θα μπορούσαμε να αναφέρουμε πως το Angular framework συνεργάζεται εύκολα με το Laravel για την ομαλή λειτουργία της εφαρμογής μας. Παρότι είναι πιο σημαντική η διατήρηση της ασφάλειας στο back – end μέρος μιας εφαρμογής, το Angular διαθέτει και αυτό το δικό του «σετ» από εργαλεία που βοηθούν στην διατήρηση της ασφάλειας και στο front – end μέρος της διαδικτυακής εφαρμογής μας.

6.2 Μελλοντικές προεκτάσεις

Παρότι η διαδικτυακή αυτή εφαρμογή διαθέτει αρκετές σύγχρονες μεθόδους και τεχνικές, τόσο σε θέματα ασφάλειας όσο και σε θέματα που αφορούν το UX. Υπάρχουν ωστόσο πολλές προσθήκες και μελλοντικές προεκτάσεις που θα μπορούσαν ακόμα να προστεθούν για ακόμα καλύτερη λειτουργεία της διαδικτυακής πλατφόρμας του καταναλωτικού συνεταιρισμού των ΕΛΤΑ.

Θα μπορούσε να προστεθεί ένα σύστημα περαιτέρω αυθεντικοποίησης του χρήστη μέσω του TAXISnet. Με τον τρόπο αυτό θα μπορούσαμε να επιτύχουμε την επιπλέον εξασφάλιση της εγκυρότητας των στοιχείων που υποβάλλουν οι χρήστες – μέλη της διαδικτυακής πλατφόρμας κατά την εγγραφή τους σε αυτή.



Εικόνα 3.9 Παράδειγμα εισόδου χρήστη σε διαδικτυακή πλατφόρμα με στοιχεία TAXISnet

Μια προσθήκη που θα μπορούσε να γίνει και θα πρόσφερε σε όλους τους χρήστες καλύτερο UX είναι το responsiveness σε όλα τα views της διαδικτυακής πλατφόρμας. Με τον τρόπο αυτό κάθε χρήστης θα μπορούσε να έχει πρόσβαση στη διαδικτυακή πλατφόρμα από οποιαδήποτε συσκευή, οποιουδήποτε μεγέθους οθόνης, π.χ. κινητό, tablet, laptop, κ.ά., χωρίς να υπάρχει ο κίνδυνος της παραμόρφωσης του περιεχομένου των views και της απόκρυψης ορισμένων δεδομένων λόγω της τοποθέτησής τους στο view. Και στα πλαίσια του responsive περιεχομένου της πλατφόρμας θα μπορούσε μελλοντικά να γίνει και ολοκληρωτική επέκταση της σε συσκευές smartphone και tablet μέσω εφαρμογής σε Google Play και App Store. Στο σενάριο αυτό η επέκταση θα ήταν εύκολη

καθώς θα μπορούσαμε να χρησιμοποιήσουμε το ίδιο back – end API και η μόνη προσθήκη θα ήταν η δημιουργία της εφαρμογής.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Angular Official Documentation - https://angular.io/docs

[2] Laravel Official Documentation - https://laravel.com/docs

[3] Json Web Token Official Website - https://jwt.io/introduction