Τεχνολογικό Εκπαιδευτικό Ίδρυμα Κρήτης

Σχολή Τεχνολογικών Εφαρμογών

Τμήμα Μηχανικών Πληροφορικής

Πτυχιακή Εργασία

Ανάπτυξη κινητής και διαδικτυακής εφαρμογής με χρήση Ionic 4

Κοσμόπουλος Γεώργιος (2949)

Επιβλέπων εκπαιδευτικός: Σπυρίδων Παναγιωτάκης

Σύνοψη

Για την παρούσα εργασία έχει δημιουργηθεί μια πλατφόρμα η οποία αφορά σε επιστημονικά συνέδρια και έχει το διττό στόχο να διευκολύνει τόσο τους διοργανωτές των συνεδρίων, όσο και τους ενδιαφερόμενους. Όσον αφορά στους διοργανωτές, αυτοί θα έχουν τη δυνατότητα να καταχωρούν και να κοινοποιούν τα συνέδριά τους εύκολα, γρήγορα και με χαμηλό κόστος. Οι ενδιαφερόμενοι από την πλευρά τους θα μπορούν να έχουν όλα τα συνέδρια στα οποία επιθυμούν να παρευρεθούν συγκεντρωμένα σε μία εφαρμογή, η οποία θα τους παρέχει εύκολη πρόσβαση σε όλες τις πληροφορίες που χρειάζονται ώστε να οργανώνουν το ημερήσιο πρόγραμμά τους όπως επιθυμούν. Για την υλοποίηση του project χρησιμοποιήθηκε Ionic Framework με Angular όσον αφορά τη μεριά της κινητής εφαρμογής και Slim PHP για τη δημιουργία του REST API.

Abstract

A platform concerning scientific conferences has been developed for this project that serves the dual purpose of facilitating both conference organizers and those interested to attend. Organizers will be able to register and share their conferences easily, quickly and at low cost. As for the attendees, they will be able to have all the conferences they wish to attend collected in one application, which will give them easy access to all the information they need in order to organize their daily schedule as they wish. To implement the project, Ionic Framework and Angular was used in terms of the mobile application and Slim PHP for the creation of the REST API.

Περιεχόμενα

1 Εισαγωγή

1.1 Περίληψη

Η παρούσα εργασία αφορά την δημιουργία/κατασκευή μιας πλατφόρμας που εξυπηρετεί δύο διαφορετικές ομάδες χρηστών, αυτούς που επιθυμούν να διοργανώσουν κάποιο συνέδριο, αλλά και αυτούς που επιθυμούν απλά να παρευρεθούν και να παρακολουθήσουν.

Αποτελείται στον πυρήνα της από μια σχεσιακή βάση δεδομένων MySql η οποία έχει στηθεί σε έναν Apache Server και περιέχει όλα τα απαραίτητα δεδομένα σχετικά με τους λογαριασμούς των χρηστών, όπως προσωπικές πληροφορίες και τις αγαπημένες τους ομιλίες ανά συνέδριο, αλλά φυσικά και πληροφορίες για τα ίδια τα συνέδρια, όπως τα προγράμματα, τις αίθουσες, συντεταγμένες κλπ. Τα δεδομένα της βάσης δεδομένων εκτίθενται από ένα API (Application Programming Interface) που έχει προγραμματιστεί με τη γλώσσα php και με βάση την αρχιτεκτονική ανάπτυξης διαδικτυακών εφαρμογών REST (REpresentational State Transfer). Το κυριότερο ίσως μέρος της πλατφόρμας αποτελείται από μια υβριδική εφαρμογή για "έξυπνα" κινητά τηλέφωνα η οποία λειτουργεί ως διεπαφή των χρηστών με τα δεδομένα των συνεδρίων.

Σαν αποτέλεσμα έχουμε λοιπόν μια εφαρμογή η οποία δίνει τη δυνατότητα στους χρήστες να διαλέξουν ένα συνέδριο ανάμεσα σε αυτά που έχουν καταχωρηθεί από τους διοργανωτές τους και συμπληρώνει τις σελίδες της με τα δεδομένα αυτού. Τα δεδομένα αυτά περιλαμβάνουν, εν συντομία, το πρόγραμμα ομιλιών ανά ημέρα διεξαγωγής, τους ομιλητές και λεπτομέρειες σχετικά με αυτούς και διαδραστικό χάρτη της περιοχής διεξαγωγής. Επίσης οι χρήστες μπορούν να δημιουργήσουν λογαριασμό ώστε να αποθηκεύουν ομιλίες για τις οποίες ενδιαφέρονται περισσότερο, χωρίς όμως αυτό να είναι απαραίτητο για τη λειτουργία της εφαρμογής, καθώς σε διαφορετική περίπτωση οι ομιλίες μπορούν να αποθηκευτούν τοπικά στη συσκευή τους.

1.2 Κίνητρο/Στόχοι

Τα περισσότερα συνέδρια διαρκούν μερικές ημέρες και αν επαναληφθούν, αυτό θα γίνει συνήθως ετησίως. Για αυτό το λόγο η μίσθωση προγραμματιστών για τη δημιουργία μιας εφαρμογής η οποία θα χρησιμοποιηθεί μόνο στο διάστημα διεξαγωγής δεν είναι ιδιαίτερα συμφέρουσα τόσο οικονομικά όσο και λογιστικά, καθώς όλες οι πληροφορίες θα πρέπει να επικοινωνούνται με τρίτους για την ενσωμάτωση σε μια μεμονωμένη εφαρμογή. Οι χρήστες της εφαρμογής ωφελούνται με το γεγονός οτι δεν χρειάζεται να εγκαθιστούν νέα εφαρμογή για κάθε συνέδριο που επιθυμούν να παρακολουθήσουν, η οποία πολύ σύντομα δεν θα έχει πια χρησιμότητα. Επίσης γίνεται ευκολότερος για αυτούς ο προγραμματισμός παρακολούθησης πολλαπλών συνεδρίων, έχοντας σε μια εφαρμογή όλες τους τις πληροφορίες.

Η πλατφόρμα που παρουσιάζεται εδώ αποσκοπεί στο να παρέχει μια ενιαία λύση για αυτές τις καταστάσεις, ιδιαίτερα αυτές που αντιμετωπίζει ο παρευρισκόμενος στα συνέδρια.

1.2 **Δομή**

Συνοπτική αναφορά στο περιεχόμενο των κεφαλαίων που θα ακολουθήσουν:

- Στο δεύτερο Κεφάλαιο (Κεφάλαιο 2) θα γίνει μια περιγραφή των διάφορων μεθόδων ανάπτυξης και τα είδη εφαρμογών για κινητές συσκευές.
- Στο τρίτο Κεφάλαιο (Κεφάλαιο 3) θα εμβαθύνουμε στις υβριδικές εφαρμογές και πιο συγκεκριμένα στο Ionic και το Angular framework, την φιλοσοφία πίσω από τα RESTful APIs και τις ιδιαίτερες δυνατότητες που αυτές οι τεχνολογίες προσφέρουν.
- Στη συνέχεια, στο τέταρτο Κεφάλαιο (Κεφάλαιο 4), γίνεται πλήρης αναφορά σε βάθος σχετικά με τη λειτουργία της εφαρμογής που αναπτύχθηκε στα πλαίσια αυτής της εργασίας.
- Το πέμπτο Κεφάλαιο (Κεφάλαιο 5) παρουσιάζει τα συμπεράσματα που προκύπτουν από την υλοποίηση του λογισμικού και την πιθανή ανάπτυξη και αναβάθμισή του για πραγματική χρήση σε εμπορικό περιβάλλον.

2 Φορητές συσκευές και εφαρμογές

2.1 Έξυπνες Συσκευές

Οι χρήση έξυπνων κινητών συσκευών και ειδικά τα έξυπνων κινητών τηλεφώνων είναι τόσο διαδεδομένη στις ημέρες μας που σχεδόν δεν χρειάζεται να κάνουμε αναφορά στη χρησιμότητα τους. Αποτελούν επανάσταση για την καθημερινότητα και τη ρουτίνα μιας συντριπτικής πλειοψηφίας ανθρώπων σε αναπτυγμένες χώρες, ενώ στις αναπτυσσόμενες το ποσοστό κατοχής τέτοιων συσκευών αυξάνεται συνεχώς. Σύμφωνα με την Αμερικάνικη μη κερδοσκοπική Pew Research Center, το 2015, σε χώρες όπως ο Καναδάς, η Αυστραλία, οι Η.Π.Α, το Ισραήλ, η νότια Κορέα και αρκετές ευρωπαΐκές χώρες έχουν πάνω απο 85% ποσοστό χρηστών κινητών τηλεφώνων, ενώ στον παγκόσμιο πληθυσμό φαίνεται ένα 67% να κατέχει έξυπνο κινητό τηλέφωνο.

Η αγορά των smartphones όμως κινείται σε μεγάλο βαθμό παράλληλα με την ανάπτυξη της πιο σημαντικής τεχνολογίας του αιώνα μας, η οποία είναι φυσικά το διαδίκτυο και ο παγκόσμιος ιστός. Αυτό συμβαίνει διότι οι πιο χρήσιμες και δημοφιλείς εφαρμογές των smartphones εξαρτώνται από την απομακρυσμένη επικοινωνία είτε μεταξύ τους, είτε με άλλα είδη συσκευών.

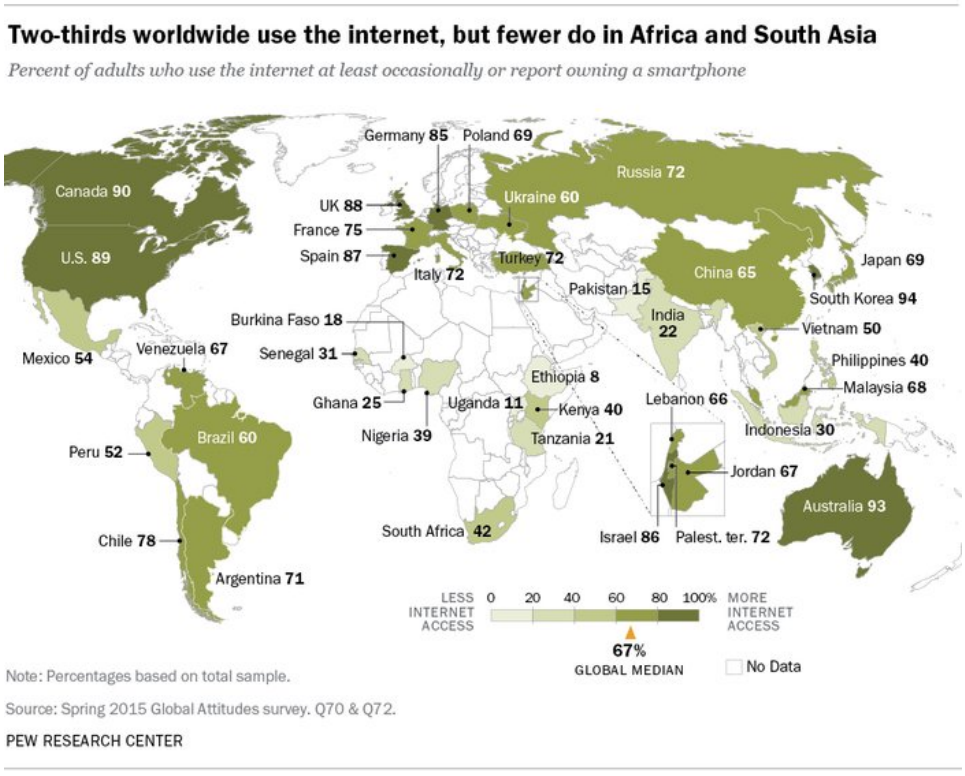


Figure 1: Ποσοστά χρήσης ίντερνετ ή/και smartphones ανα περιοχή

2.2 **Mobile Applications**

Στις αρχές της τεχνολογίας των κινητών τηλεφώνων, οι εφαρμογές που προσέφεραν ήταν περιορισμένες κυρίως σε απλά παιχνίδια, αριθμητικούς υπολογιστές, λίστα τηλεφωνικών επαφών και ημερολόγια. Στις αρχές όμως της δεκαετίας του ‘90 άρχισαν να κυκλοφορούν στην αγορά τα πρώτα smartphones και PDA’s(Personal Digital Assistants), μια πρώιμη μορφή των σημερινών tablets, τα οποία άρχισαν να προσφέρουν εφαρμογές με πολύ πιο εξελιγμένες δυνατότητες, όπως επεξεργαστές κειμένου, αποθήκευση δεδομένων, λογιστικά φύλλα κ.ά. Οι συσκευές συνέχισαν να εξελίσσονται και να προσφέρουν εφαρμογές όλο και πιο κοντά σε προσωπικούς υπολογιστές, όπως αναπαραγωγή μουσικής και βίντεο, χρήση αισθητήρων για παρακολούθηση διάφορων καταστάσεων, φωτογραφία κλπ.

Με αυτή την εξέλιξη καταλήγουμε στη δυνατότητα των smartphones να χρησιμοποιούν το διαδίκτυο, το οποίο επίσης μόλις βλέπει μεγάλη ανάπτυξη και κατα συνέπεια στις διαδικτυακές εφαρμογές για κινητά τηλέφωνα. Οι πιο γνωστές ασχολούνται με κοινωνικά δίκτυα, όμως πλέον οι περισσότερες εφαρμογές κάνουν σε κάποιο βαθμό χρήση του ιντερνετ για να προσφέρουν υπηρεσίες. Περιλαμβάνονται εφαρμογές για την υγεία, για νέα και γενικότερα οποιοδήποτε αντικείμενο χρειάζεται συχνές και δυναμικές αλλαγές στη πληροφορία που παρουσιάζει.

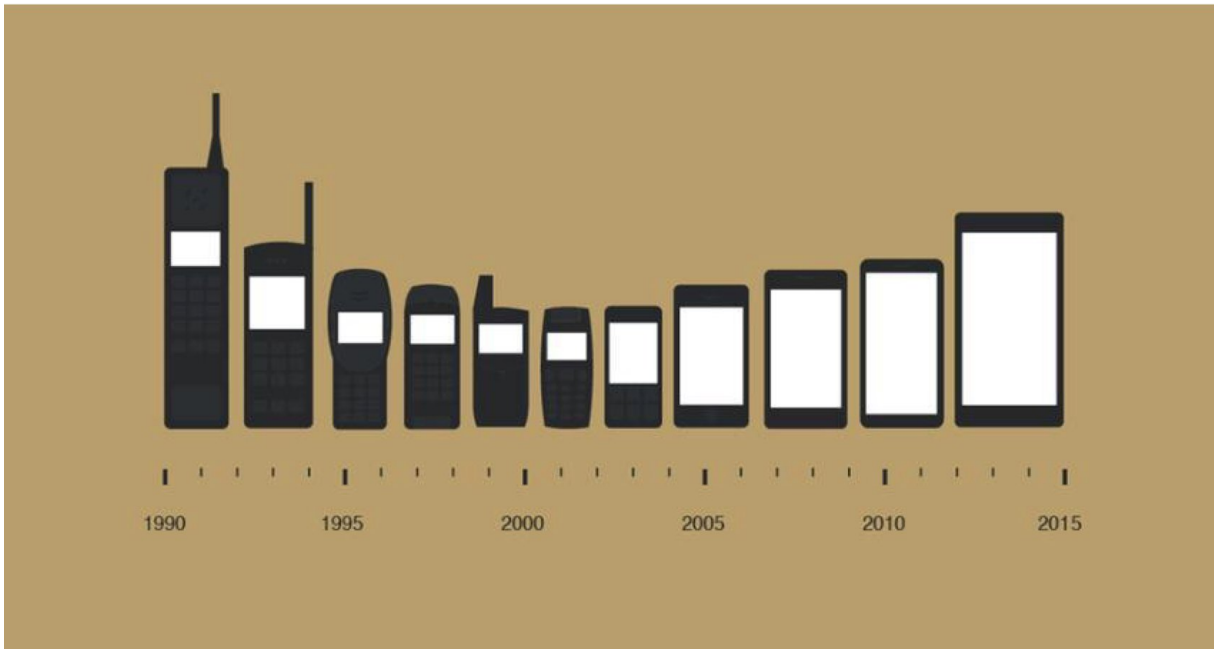


Figure 2: Εξέλιξη κινητών τηλεφώνων

2.3 Είδη εφαρμογών έξυπνων συσκευών

Τα δύο μεγαλύτερα λειτουργικά συστήματα που επικράτησαν στις κινητές συσκευές είναι το Android, του οποίου η πρώτη έκδοση κυκλοφόρησε τον Σεπτέμβριο του 2008 από την Google Inc σαν ανοιχτό λογισμικό και το iOS από την Apple το 2007 αποκλειστικά για δικές της συσκευές. Αυτά τα λειτουργικά συστήματα δημιουργήθηκαν αντίστοιχα με τις γλώσσες προγραμματισμού Java, C, C++ και Objective C, C++, Swift. Οι δύο εταιρίες παρέχουν εργαλεία ανάπτυξης εφαρμογών για το σύστημά τους, παρόλα αυτά όμως η αρχιτεκτονική, ο τρόπος λειτουργίας, οι γλώσσες προγραμματισμού, η σχεδιαστική και αισθητική φιλοσοφία απέχουν αρκετά μεταξύ τους.

Native

Οι λεγόμενες ***Native***(Εγγενείς) εφαρμογές ήταν ο πρώτος και ο ‘κανονικός’ τύπος εφαρμογών που αναπτύχθηκε για τα smartphones. Βασίζονται πάνω σε αυτά τα εργαλεία της κάθε εταιρίας για να αναπτυχθούν κάτι που συνεπάγεται σε ανάγκη μεγάλου εύρους γνώσεων, ικανοτήτων και εμπειρίας να απαιτείται από τους προγραμματιστές, αν θέλουν η εφαρμογή τους να είναι συμβατή και με τα δύο λειτουργικά συστήματα. Πρέπει να αναπτύξουν σχεδόν εξολοκλήρου δύο ξεχωριστές εφαρμογές. Το γεγονός αυτό καθιστά την ανάπτυξη μιας κοινής εφαρμογής αρκετά ακριβή καθώς ο κάθε προγραμματιστής συνήθως εξειδικεύεται σε ένα μόνο από τα δύο σετ εργαλείων. Αυτό αποτέλεσε κυρίως πρόβλημα στη δημιουργία εφαρμογών για το Android καθώς λόγω του οτι είναι ανοιχτό λογισμικό, χρησιμοποιήθηκε, συχνά τροποποιημένο, σε τεράστιο φάσμα συσκευών διασπώντας ακόμη περισσότερο την διαδικασία ακόμη και ανάμεσα σε συσκευές με το ίδιο λειτουργικό. Όσον αφορά την απόδοση, οι native εφαρμογές είναι compiled κώδικας και κοντά στο λειτουργικό σύστημα, από άποψη λειτουργικής σβελτάδας είναι πολύ καλή. Αξίζει να αναφερθεί και η άμεση προγραμματιστική πρόσβαση που προσφέρεται μέσω των εργαλείων στο υλισμικό της συσκευής αλλά και σε ιδιότητες του λειτουργικού συστήματος.

Web Applications

Αργότερα, λόγω του διαδικτύου, άρχισαν να δημιουργούνται όλο και πιο περίπλοκες και πολύ πιο διαδραστικές ιστοσελίδες, σε τέτοιο βαθμό που τώρα ονομάζονται ***διαδικτυακές εφαρμογές***(web applications). Αυτό ταιριάζει αρκετά στις κινητές συσκευές, καθώς με ένα οποιοδήποτε πρόγραμμα περιήγησης μπορούν να έχουν πρόσβαση σε εφαρμογές, οι οποίες μάλιστα δεν χρειάζεται να περάσουν από διαδικασία έγκρισης των ηλεκτρονικών καταστημάτων για εφαρμογές της κάθε εταιρίας, ούτε για να αποκτηθούν, ούτε για να αναβαθμιστούν. Παρόλα αυτά οι διαδικτυακές εφαρμογές εξακολουθούν να έχουν τους περιορισμούς που επιβάλλουν οι περιηγητές κυρίως για λόγους ασφαλείας, όπως περιορισμένη χρήση του υλικού της συσκευής και άλλων λειτουργιών του συστήματος. Επίσης, λόγω της φύσης τους απαιτούν συνεχή σύνδεση στο διαδίκτυο για να συνεχίσουν να λειτουργούν.

Hybrid Applications

Η άλλη μορφή εφαρμογών για έξυπνες συσκευές, οι Υβριδικές(***Hybrid***) αναπτύχθηκε για να ελαφρύνει τα προβλήματα των native και των web applications. Συνοπτικά, αντί να βασίζονται στα εργαλεία του κάθε λειτουργικού συστήματος, χρησιμοποιούν τις κοινές τεχνολογίες και γλώσσες προγραμματισμού του διαδικτύου html, css, javascript, όπως οι διαδικτυακές εφαρμογές, όμως 'τρέχουν' σε ένα στιγμιότυπο της πλατφόρμας περιήγησης διαδικτύου(webview) της συσκευής, όπως έκαναν έως τώρα τα προγράμματα περιήγησης(browsers) όπως το chrome και το safari, αντί μέσω αυτών. Το webview είναι ένα εγγενές κομμάτι του λειτουργικού συστήματος και μπορεί να προσφέρει πρόσβαση στις διαδικασίες του αλλά και το υλικό της συσκευής. Οι υβδριδικές εφαρμογές εξακολουθούν να χρειάζονται μια γέφυρα ανάμεσα σε αυτές και τις native δυνατότητες των λειτουργικών συστημάτων, όπως τη χρήση κάμερας, εμφάνιση ειδοποιήσεων και άλλες, την οποία και προσφέρουν τεχνολογίες με πιο γνωστή την Cordova(πρώην PhoneGap) της εταιρίας Apache. Με τη χρήση της Cordova πακετάρονται σε μορφή εγγενούς εφαρμογής και διατίθενται μέσω των ηλεκτρονικών καταστημάτων ενώ στην πραγματικότητα η κατασκευή τους μοιάζει πιο πολύ με αυτή μιας ιστοσελίδας. Η μεγαλύτερη δύναμη αυτής της τεχνικής είναι οτι με εύκολο τρόπο, οι προγραμματιστές διαδικτύου μπορούν να αναπτύξουν εφαρμογές για όλες τις υπάρχουσες συσκευές, πακεταρισμένες σαν native και με όλες τις δυνατότητες τους, ακόμα και την απόδοσή τους σε ταχύτητα λόγω της πολύ δυνατής και γρήγορα αναπτυσσόμενης javascript.

Συνοψίζοντας :

- Οι native εφαρμογές προσφέρουν την καλύτερη απόδοση και πρόσβαση στη συσκευή και το σύστημα, όμως κάθε λειτουργικό σύστημα διαφέρει αρκετά ώστε να ανεβαίνει σημαντικά το κόστος παραγωγής τους για πολλαπλές πλατφόρμες. Διατίθενται μέσω των ηλεκτρονικών καταστημάτων.
- Οι διαδικτυακές εφαρμογές λειτουργούν μέσω του προγράμματος περιήγησης(browser), είναι εξειδικευμένες ιστοσελίδες. Λειτουργούν σε οποιοδήποτε σύστημα που έχει browser και απαραίτητα πρόσβαση στο διαδίκτυο. Έχουν χαμηλή απόδοση και σβελτάδα, καθώς και πολύ περιορισμένη πρόσβαση σε λειτουργίες του συστήματος.
- Οι υβριδικές εφαρμογές μοιάζουν πολύ με τις διαδικτυακές όσον αφορά την παραγωγή τους με τις γνωστές τεχνολογίες διαδικτύου. Πακετάρονται και λειτουργούν σαν native και διατίθενται στα ηλεκτρονικα καταστήματα. Η απόδοση και οι δυνατότητες τους πλησιάζουν αυτές των native εκτός από πιο εξειδικευμένες περιπτώσεις όπως η χρήση γραφικών, παιχνίδια κλπ.

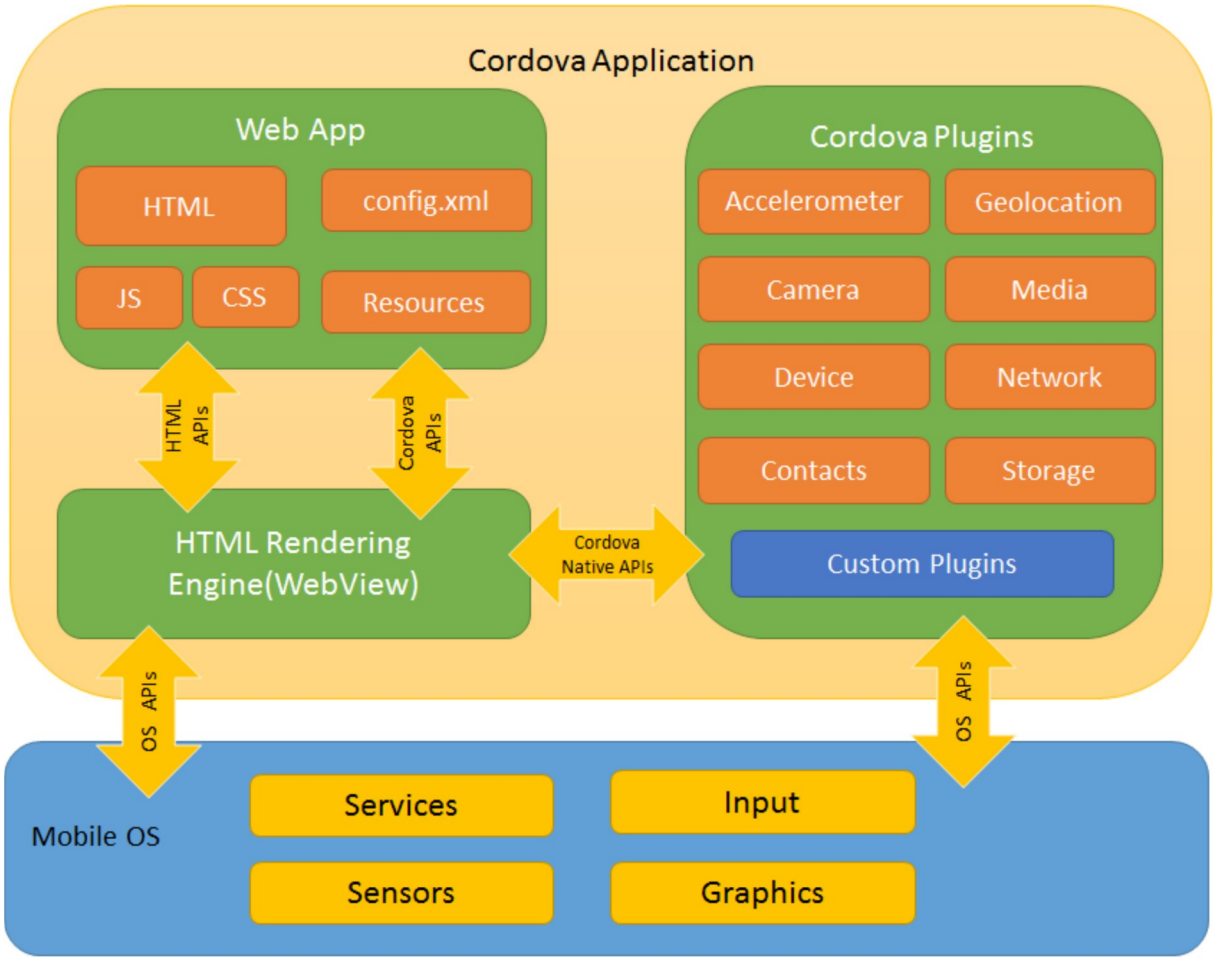


Figure 3: Υβριδικές εφαρμογές

Χαρακτηριστικά	Εγγενείς Εφαρμογές	Διαδικτυακές Εφαρμογές	Υβριδικές Εφαρμογές
Απόδοση	Υψηλή	Χαμηλή	Μέτρια
Λειτουργία Εκτός Σύνδεσης	Υποστηρίζεται	Δεν Υποστηρίζεται	Υποστηρίζεται
Διανομή	Κατάστημα Εφαρμογών	Περιηγητής Ιστού	Κατάστημα Εφαρμογών
Διαπλατφορμική Υποστήριξη	Όχι	Ναι	Ναι
Πρόσβαση Υλισμικού	Υψηλή	Χαμηλή	Μέτρια
Γλώσσα Ανάπτυξης	Εγγενής μόνο	Διαδικτύου μόνο	Εγγενής/Διαδικτύου
Χρόνος Ανάπτυξης	Υψηλός	Χαμηλός	Μέτριος
Κόστος Ανάπτυξης	Υψηλό	Χαμηλό	Μέτριο

3 Τεχνολογίες και τεχνικές υλοποίησης

Ακολουθούν περιγραφές των τεχνολογιών και τεχνικών που χρησιμοποιήθηκαν για την υλοποίηση της εφαρμογής για συνέδρια.

3.1 Ionic Framework

Σαν συνέχεια της εξέλιξης των υβριδικών εφαρμογών, προέκυψαν κάποια frameworks που ενισχύουν και διευκολύνουν τη διαδικασία ανάπτυξης τους, λειτουργώντας ένα επίπεδο ακόμα πιο πάνω, ανάμεσα στο cordova και το προγραμματιστή. Ένα από αυτά και το οποίο χρησιμοποιήθηκε στην συγκεκριμένη εφαρμογή είναι το ***Ionic Framework***. Προσφέρει βιβλιοθήκες από προκαθορισμένα κομμάτια γραφικού περιβάλλοντος, το καθένα τους με πολύ χρήσιμες λειτουργίες προγραμματισμένες εξ ορισμού. Σκοπεύει στην απαλλαγή του προγραμματιστή από το κομμάτι κυρίως του γραφικού σχεδιασμού της εφαρμογής, δίνοντας έμφαση στην ομοιομορφία της εμφάνισης των μερών της. Αυτό όμως που το κάνει τόσο ισχυρό εργαλείο για δημιουργία εφαρμογών συμβατών με πολλά συστήματα είναι η αυτόματη αλλαγή της σχεδιαστικής φιλοσοφίας να είναι ανάλογη με αυτήν των native εφαρμογών του εκάστοτε συστήματος, όπως για παράδειγμα το material design του Android. Κατά τη διάρκεια του πακεταρίσματος με το Cordova ανάλογα με την πλατφόρμα που έχει επιλέξει ο προγραμματιστής, το Ionic αναλαμβάνει να διαλέξει την αντίστοιχη εμφάνιση για το κάθε μέρος του γραφικού περιβάλλοντος ώστε να ταιριάζει με το λειτουργικό σύστημα και να δίνει στο χρήστη την αίσθηση μιας εφαρμογής native όπως αυτές που έχει συνηθίσει.

Στη βάση του το ionic framework χρησιμοποιεί τις γλώσσες HTML 5, CSS και Javascript, αν και συνήθως παραλλαγές αυτών όπως το SASS, μια συντακτική παραλλαγή για την CSS3, και για την Javascript το Vue, το React ή το Angular framework με το οποίο έχει και τη μεγαλύτερη συμβατότητα. Σε προηγούμενες εκδόσεις του, οι δυνατότητες του Ionic επεκτείνονταν αρκετά και στο λογικό μέρος των εφαρμογών. Όμως με την μεγάλη βελτίωση που έχει δει ειδικά το Angular framework, το Ionic αρχίζει να αποτελείται κυρίως από μια μεγάλη βιβλιοθήκη μερών και δυνατοτήτων γραφικού περιβάλλοντος (UI Components), ενώ το λογικό μέρος προγραμματίζεται σχεδόν εξολοκλήρου με καθαρό Angular Javascript.

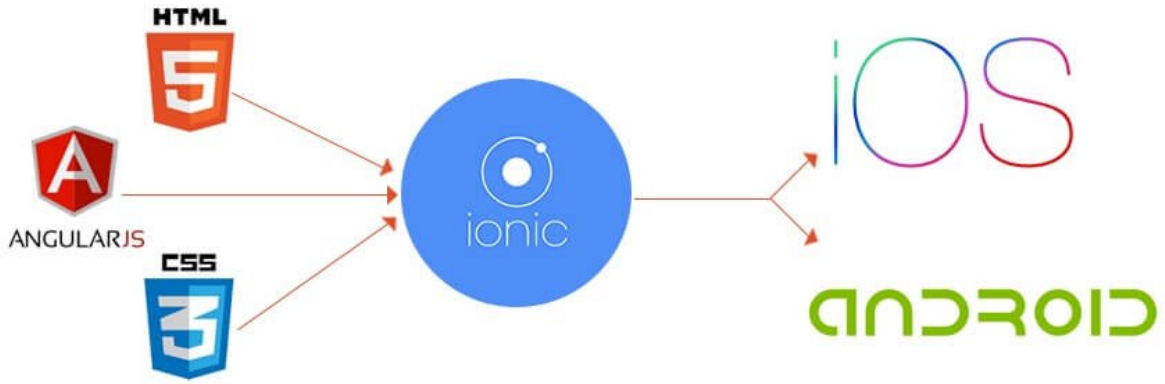


Figure 4: Η θέση του Ionic σε μια υβριδική εφαρμογή

Αξίζει να αναφερθούμε και στο ***Capacitor***, μια εναλλακτική του Cordova, αναπτυγμένη από την ίδια την Ionic, μία γέφυρα δηλαδή ανάμεσα στις τεχνολογίες διαδικτύου και τις native των συστημάτων. Είναι και αυτό ανοιχτό λογισμικό, με πληθώρα έτοιμων plugins για να προσφέρει λειτουργικότητα αλλά δεν περιορίζεται μόνο εκεί καθώς έχει και ένα βαθμό συμβατότητας με plugins του Cordova.

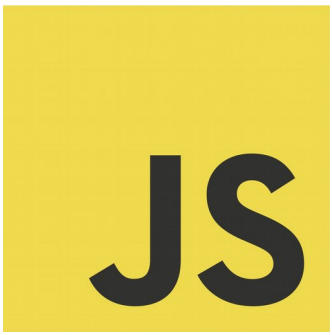
Το Ionic έχει εξάρτηση από το Node.js, ένα περιβάλλον λειτουργίας που επιτρέπει στην Javascript να τρέχει εκτός του προγράμματος περιήγησης και από το npm, το μεγαλύτερο πρόγραμμα βιβλιοθήκης, οργάνωσης και εγκατάστασης πακέτων λογισμικού. Το npm αναλαμβάνει την εγκατάσταση του Ionic CLI(Command Line Interface) αλλά και όλων των υπολοίπων εξαρτήσεων του Ionic όπως το προεγκατεστημένο Angular Framework.

Σε περιβάλλον Windows αρκούν οι εξής εντολές στη γραμμή εργαλείων για να αποκτήσει κανείς το Ionic αφού εγκαταστήσει το node.js και να εκκινήσει την πρώτη εφαρμογή :

- **npm install -g ionic**
- **ionic start myApp tabs**

Στη συνέχεια με αντίστοιχες εντολές δημιουργούνται και οι σελίδες/μέρη της εφαρμογής και προστίθενται επεκτάσεις ή αναβαθμίσεις.

3.2 Javascript



Από τις αρχές του διαδικτύου και πριν αρχίσουμε να κάνουμε λόγο για διαδικτυακές εφαρμογές, η Javascript ήταν ήδη η πιο ισχυρή γλώσσα που υπήρχε για να δίνει λειτουργικότητα στις ιστοσελίδες, η οποία περιοριζόταν κυρίως σε μικρά scripts. Η javascript είναι τώρα τόσο δημοφιλής που όλοι οι περιηγητές διαδικτύου περιέχουν μηχανές λογισμικού ειδικά για την λειτουργία της και δημιουργήθηκε βασισμένο σε αυτήν ένα διεθνές στάνταρ για τις δυνατότητες που πρέπει να υποστηρίζονται από γλώσσες διαδικτύου, το οποίο λέγεται ECMAScript. Είναι μια interpreted γλώσσα, δηλαδή αντί να μετατρέπεται ένα πρόγραμμα μέσω ενός compiler σε πακέτο γλώσσας μηχανής πριν τη χρήση, εντοπίζεται από έναν interpreter που μεταφράζει μόνο τα μέρη κώδικα που ζητούνται ανά πάσα στιγμή. Αυτό της προσδίδει ευελιξία στο περιβάλλον στο οποίο χρησιμοποιείται.

Η Javascript συντακτικά έχει λίγους περιορισμούς και αρκετές ανέσεις, όπως έλλειψη τύπων στις μεταβλητές και στις δομές της κατά τη δήλωσή τους, για να είναι εύκολο στους σχεδιαστές και προγραμματιστές να γράψουν μικρά κομμάτια κώδικα που αλληλεπιδρούν με μέρη της ιστοσελίδας. Η σχετική απλότητά της αρχικά έκανε πολλούς προγραμματιστές να μη την θεωρούν αντάξια σε άλλες καθιερωμένες γλώσσες προγραμματισμού, αλλά οι ιστοσελίδες άρχισαν να τη χρησιμοποιούν όλο και περισσότερο, ειδικά με την ανάπτυξη των τεχνικών AJAX(Asynchronous Javascript and XML).

Όταν κάνουμε όμως λόγο για ανάπτυξη διαδικτυακών εφαρμογών, η πολυπλοκότητα και η ποσότητα του κώδικα αυξάνεται, επηρεάζοντας και τη γενικότερη δομή με αντίστοιχο τρόπο και εκεί η Javascript αρχίζει να δείχνει αδυναμία κυρίως από άποψη χρηστικότητας.

Λόγω των αδυναμιών της Javascript σε μεγάλα έργα, η Microsoft δημιούργησε την γλώσσα ***Typescript***. Είναι βασισμένη πάνω στην Javascript και μάλιστα για να λειτουργήσει χρειάζεται ένας transpiler, δηλαδή ένας compiler που μετατρέπει τον κώδικα Typescript σε Javascript για να έχει την ίδια τεράστια συμβατότητα με περιηγητές. Είναι εξολοκλήρου αντικειμενοστραφής, όπως και η παραδοσιακή javascript, αλλά με σύνταξη και δομή των αντικειμένων πιο κοντά σε κλασικές γλώσσες όπως η Java, με κλάσεις, κληρονομικότητα, interfaces κλπ. Επίσης περιέχει αναγνώριση και δήλωση τύπων σε όλα τα στοιχεία της οι οποίοι ελέγχονται από τον transpiler αλλα αναγνωρίζονται και ‘ζωντανά’ από τους επεξεργαστές κώδικα για αποφυγή λαθών στην συγγραφή και στην προετοιμασία του προγράμματος, αυτόματη συμπλήρωση και πολλά άλλα.

Στην περίπτωσή μας, η Typescript είναι το πρώτο βήμα στην βελτίωση της δομής σε μια web application και κατά συνέπεια μιας υβριδικής εφαρμογής, και η γλώσσα που χρησιμοποιήθηκε στην ανάπτυξη του δεύτερου βήματος, του Angular Framework.

3.3 Angular Framework

Η συγκεκριμένη τεχνολογία δεν αφορά τόσο παραλλαγή κάποιας υπάρχουσας γλώσσας προγραμματισμού, όσο τον ορισμό μιας δομής που ακολουθούμε και δομικών στοιχείων τα οποία χρησιμοποιούμε για να την υλοποίηση διαδικτυακών εφαρμογών. Δημιουργήθηκε από μια ομάδα προγραμματιστών της Google σε μια πρώτη έκδοση που λεγόταν AngularJS η οποία μόλις το 2016 αναβαθμίστηκε σε αγνώριστο βαθμό στην τωρινή Angular. Μέσα σε ελάχιστο χρόνο έγινε ανάρπαστη ανάμεσα στους προγραμματιστές διαδικτύου γιατί ενσωματώνει όλες τις απαιτήσεις που υπάρχουν για μία εφαρμογή ή ιστοσελίδα, ιδιαίτερα σε περιπτώσεις που χρησιμοποιούνται πολλές φόρμες ή γενικότερα όταν υπάρχει έντονη διεπαφή μεταξύ της εμφάνισης δεδομένων τα οποία υπολογίζονται ή προσφέρονται δυναμικά. Αυτό το επιτυγχάνει εν μέρει ενισχύοντας την Typescript με μια βιβλιοθήκη που προσφέρει μη υπάρχουσες επιλογές σχετικά με δομικά στοιχεία. Κάνει σαφή τον διαχωρισμό ανάμεσα σε μέρη της εφαρμογής, ορίζει τις ‘αρμοδιότητες’ και τις σχέσεις που έχουν όλα μεταξύ τους. Δεν του λείπουν βέβαια και άλλες δυνατότητες μοντέρνων γλωσσών προγραμματισμού όπως το dependency injection. Μία από τις ανάγκες των σύγχρονων εφαρμογών διαδικτύου που καλύπτει η Angular είναι η δέσμευση δεδομένων διπλής κατεύθυνσης, η οποία περιλαμβάνει την ικανότητα να χρησιμοποιούμε μεταβλητές που αντικατοπτρίζουν τις αλλαγές στα δεδομένα τους άμεσα μεταξύ template και μοντέλου.

Υπάρχει μια ιεραρχία στις εφαρμογές Angular, η οποία ξεκινά με τις έννοιες του Module και του Component.

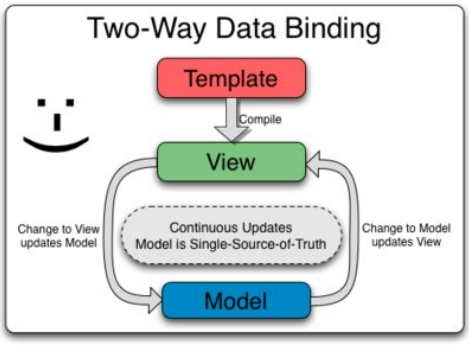
- Τα ***modules*** είναι συλλογές κώδικα που δηλώνουν μέρη του προγράμματος(components) και υπηρεσίες για χρήση από άλλα μέρη του, το εύρος στο οποίο μπορούν αυτά να χρησιμοποιηθούν, ρυθμίσεις σχετικά με τον τρόπο

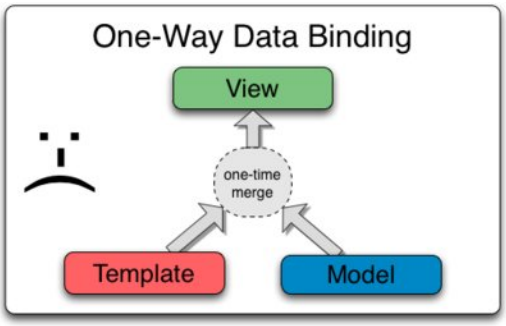
λειτουργίας τους κ.ά. Δεν περιέχουν λογική και δεν σχετίζονται άμεσα με τη λειτουργικότητα και την εμφάνιση της εφαρμογής.

- Τα ***components*** είναι βασικό δομικό στοιχείο μιας Angular εφαρμογής, ορίζει template γραμμένο σε html πάνω στο οποίο θα δρα και style σε CSS που καθορίζει την εμφάνιση του template, ενώ το ίδιο το component λειτουργεί σαν μοντέλο και μεταβιβαστής πληροφορίας. Αυτό το πακέτο τριών στοιχείων μπορεί να επαναχρησιμοποιηθεί μέσα στην εφαρμογή κατά βούληση αντί να ξαναγραφτεί για την κάθε περίπτωση. Ένα component μπορεί να αναπαρασταθεί σε ένα template ακόμα σαν html tag και να αυτό να περιέχει attributes τα οποία ορίζουμε και μεταχειριζόμαστε με τον κώδικα του component. Όπως προαναφέρθηκε δηλαδή, ενσωματώνουμε έτοιμα components μέσα σε άλλα για να κάνουμε επαναχρησιμοποίηση του κώδικα με καθαρό και δομημένο τρόπο.

- Ένα ακόμη χρήσιμο στοιχείο που επιτρέπει την επαναχρησιμοποίηση κώδικα είναι τα angular ***services*** και injectables, μέρη της λογικής του προγράμματος που αξιοποιούν το πολύ καλό dependency injection της Angular για να προσφέρουν ‘υπηρεσίες’ στα components που τις χρειάζονται και τις διαμοιράζονται. Ένα χαρακτηριστικό παράδειγμα από την εφαρμογή συνεδρίων είναι αυτό της υπηρεσίας που προσφέρει διασύνδεση με το API για άντληση ή αποθήκευση πληροφορίας από και προς τα components. Τα injectables είναι singletons, δηλαδή μόνο ένα στιγμιότυπο υπάρχει ενώ τρέχει το πρόγραμμα, το οποίο μεταχειρίζονται όλα τα components που το χρησιμοποιούν και το στιγμιότυπο αυτό δημιουργείται μόνο όταν κάποιο μέρος της εφαρμογής το χρειαστεί, όχι στην εκκίνηση.

- Τέλος, τα Angular ***Directives***, είναι λειτουργίες που συντάσσονται σαν html attributes και προσφέρουν λειτουργίες απευθείας στο template, κυρίως για την υποστήριξη δυναμικής κατασκευής του. Ένα δομικό στοιχείο της html δηλαδή μπορεί να δημιουργηθεί υπό συνθήκες με τη χρήση του ngIf ή να δημιουργηθούν περισσότερα από ένα ανάλογα με την πηγή από την οποία αντλούν με το directive ngFor. Το πιο σημαντικό ενσωματωμένο directive της Angular είναι ίσως το ngModel που προσφέρει άμεση επικοινωνία δεδομένων μεταξύ html και javascript. Τα παραπάνω είναι attribute directives και συντάσσονται με αγκύλες []. Υπάρχει άλλη μία κατηγορία η οποία συντάσσεται με παρενθέσεις (), τα event directives. Η χρήση τους είναι να ανιχνεύουν αλλαγές των στοιχείων του template όπως όταν γίνει εγγραφή σε κάποιο πεδίο φόρμας, πατηθεί ένα κουμπί κλπ.





Το Angular έχει και άλλες υπηρεσίες μέσω του εργαλείου Angular CLI(Command Line Interface) το οποίο εγκαθίσταται με το node package manager. Όπως και το ionic CLI, που βασίζεται και σε αυτό του angular, βοηθά στη δημιουργία modules, components, services και εκκίνηση του project με προκαθορισμένες ρυθμίσεις διαφόρων τύπων, ενώ επιτρέπει και την λειτουργία της εφαρμογής σε περιηγητή διαδικτύου με ζωντανή αναπαράσταση των αλλαγών του κώδικα, ενώ ακόμα βρίσκεται σε διαδικασία ανάπτυξης. Η εντολή που ενεργοποιεί αυτή τη λειτουργία είναι για το angular CLI η

- ngServe

ενώ η αντίστοιχη για το ionic CLI

- ionic serve

Μέσω αυτής της διαδικασίας ελαττώνεται κατά πολύ ο χρόνος που χρειάζεται ο προγραμματιστής για τον έλεγχο του κώδικα.

3.4 Εξυπηρετητής

Καθώς η εφαρμογή για συνέδρια είναι μια υβριδική διαδικτυακή εφαρμογή χρειαζόμαστε έναν εξυπηρετητή που θα αναλαμβάνει τόσο την βάση δεδομένων η οποία περιέχει όλα τα δεδομένα σχετικά με τα συνέδρια και τους χρήστες της, όσο και τη ‘φιλοξενία’ του API το οποίο γεφυρώνει την επικοινωνία ανάμεσα στην εφαρμογή και την βάση δεδομένων.

Για αυτό το σκοπό έχει χρησιμοποιηθεί ένας Apache HTTP Server, ο οποίος είναι ανοιχτό λογισμικό και δωρεάν, συμβατό με πολλά λειτουργικά συστήματα ενώ ταυτόχρονα και το πιο δημοφιλές λογισμικό για server παγκοσμίως. Ο Apache υποστηρίζει τις περισσότερες τεχνολογίες που περιμένει κανείς μέσω του σπονδυλωτού σχεδιασμού του, δηλαδή τη χρήση μεμονωμένων δυνατοτήτων των modules του. Σημαντικό για εμάς ήταν η υποστήριξη της γλώσσας PHP που αποτελεί τη ραχοκοκκαλιά του συστήματός μας.

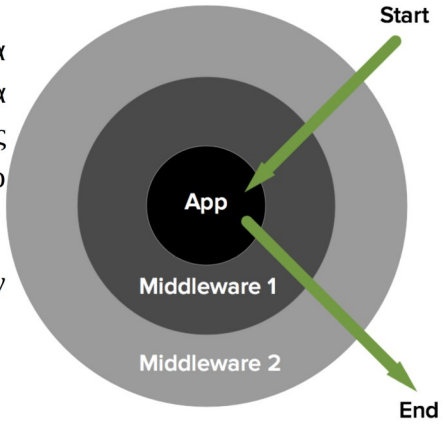
Ο Apache HTTP Server εγκαταστάθηκε σε ένα τοπικό σύστημα σαν μέρος ενός πακέτου προ-ρυθμισμένων τεχνολογιών που ονομάζεται XAMPP, το όνομα του οποίου μας δείχνει τι περιέχει, Apache MySql PHP and Perl. Το XAMPP δεν έχει απαιτήσεις συστήματος εκτός από την ύπαρξη Microsoft Visual C++ 2017 για συστήματα Windows. Η χρήση του εκτός από γρήγορη και εύκολη εγκατάσταση όλων των τεχνολογιών μας παρέχει και ένα πρόγραμμα με γραφικό περιβάλλον για τις σχετικές ρυθμίσεις, την εμφάνιση και διαχείριση logfiles κ.ά. Ένας άλλος λόγος που το XAMPP είναι μία καλή λύση για την ανάπτυξη προγραμμάτων ή ιστοσελίδων τοπικά είναι οτι κάνει τη μετακίνηση του συστήματος σε εξειδικευμένες συσκευές για εμπορική χρήση πολύ ανώδυνη, αν και εν μέρει αυτό οφείλεται στην τεράστια συμβατότητα του Apache Server.

3.5 PHP / SLIM

Σχετικά με τη γλώσσα που χρησιμοποιήθηκε για την ανάπτυξη του API μας, αυτή είναι το ανοιχτό λογισμικό PHP, ίσως η πιο γνωστή γλώσσα προγραμματισμού για ιστοσελίδες. Η PHP άρχισε να χρησιμοποιείται από τους προγραμματιστές λόγω της ομοιότητας με την τότε δημοφιλή C, ενώ στην πορεία εξελίχθηκε αρκετά αποκτώντας τεράστια συμβατότητα και ευελιξία για την ανάπτυξη προγραμμάτων και εξακολουθεί να είναι πολύ γρήγορη σε απόδοση. Σήμερα σπάνια χρησιμοποιείται στην αγνή της μορφή καθώς έχουν προκύψει με βάση αυτήν πολλά frameworks και βιβλιοθήκες τα οποία καλύπτουν ποικίλες ανάγκες, περιπτώσεις χρήση και τύπους εφαρμογών, υποστηρίζουν σχεδιαστικά πρωτόκολλα ανάλογα με τις ανάγκες και εστιάζουν τις δυνατότητες της PHP. Αυτό είναι αποτέλεσμα του τεράστιου εύρους δυνατοτήτων της PHP, την διχασμένη φιλοσοφία της ανάμεσα σε αντικειμενοστραφή και γραμμική και παράλληλα της απλότητάς της που την καθιστά δύσκολη στη διαχείριση, όταν χρησιμοποιείται σε μεγάλα και περίπλοκα προγράμματα.

Το framework που χρησιμοποιήθηκε για την εφαρμογή μας λέγεται Slim και είναι σχεδιασμένο για τη δημιουργία RESTful APIs χωρίς να ρισκάρει όμως να επιβαρύνει την ταχύτητα του συστήματος με πολυεπίπεδες λειτουργίες. Υποστηρίζει :

- τη δρομολόγηση(routing) από URI και εντολών του πρωτοκόλλου HTTP όπως οι GET, POST, DELETE και PUT προς τις μεθόδους και τους ελεγκτές της εφαρμογής.
- Dependency injection μέσω της ενσωματωμένης βιβλιοθήκης Pimple με συμβατότητα για άλλες επιλογές που να υποστηρίζουν PSR-11(στάνταρντ για βιβλιοθήκες τέτοιου τύπου).
- Κρυπτογράφηση των cookies τα οποία θέτει για μεγαλύτερη ασφάλεια.
- Caching στον περιηγητή του χρήστη για την αποφυγή επανειλημμένης μετάδοσης ίδιων δεδομένων.
- Αυτόματη εύρεση και χειρισμό λαθών αλλά και δυνατότητα τροποποίησης των μεθόδων αυτών.
- Middleware, κομμάτια κώδικα τα οποία ορίζεται να τρέχουν αυτόματα ανάμεσα στα επίπεδα επικοινωνίας του χρήστη και του κυρίως προγράμματος για τη προ-μεταχείριση και έλεγχο των δεδομένων.
- Το πρωτόκολλο PSR-7 για τη δομή των μηνυμάτων HTTP.



Το slim framework εγκαθίσταται με το Composer, έναν διαχειριστή πακέτων και εξαρτήσεων λογισμικού με την εντολή *composer require slim/slim "^3.12"* στην γραμμή εντολών. Οι απαιτήσεις του περιλαμβάνουν την PHP 5.5 και εξυπηρετητή με δυνατότητα URL rewriting.

3.6 REST

Εδώ μιλάμε πλέον όχι για γλώσσες προγραμματισμού αλλά για την αρχιτεκτονική της εφαρμογής και του API μας σε πιο αφηρημένο, μη πρακτικό επίπεδο. Το REST είναι ουσιαστικά οδηγίες και περιορισμοί για το σχεδιασμό, τη δημιουργία και τη συμπεριφορά μιας διεπαφής client – server. Ένα API λέγεται λοιπόν RESTful όταν ακολουθεί κατά βάση 6 περιορισμούς που θέτει η REST :

1. Οι ***client και server*** πρέπει να είναι ξεχωριστές οντότητες η κάθε μία εξελισσόμενη ανεξάρτητα από την άλλη. Ο client γνωρίζει για τον server μόνο τα URI(στο εξής Endpoints) τα οποία έχουν οριστεί για επικοινωνία μαζί του και ο ίδιος δεν έχει καμία επαφή με τα δεδομένα, παρά μόνο με την εμπειρία χρήστη, το περιβάλλον διεπαφής κλπ. Έτσι επιτυγχάνεται διάσπαση μεγάλων μερών ενός συστήματος/πλατφόρμας για να αναπτυχθούν ξεχωριστά ενώ να διατηρούν πλήρη επικοινωνία μεταξύ τους. Αυτό είναι σε αντίθεση με παλαιότερους τρόπους κατασκευής ιστοσελίδων όπου client και server ήταν προγραμματισμένα ταυτόχρονα και αλληλένδετα σαν μια μοναδική οντότητα περιορίζοντας έτσι την επεκτασιμότητα. Βέβαια ο τρόπος αυτός δεν χρησιμοποιείται μόνο σε συστήματα REST αλλά και στις περισσότερες σύγχρονες εφαρμογές διαδικτύου.

2. Ο από πάνω περιορισμός μας οδηγεί εν μέρει και σε έναν ακόμα του REST ο οποίος ορίζει στο σύστημα ειδικά από τη μεριά του server να σχεδιαστεί με τέτοιο τρόπο ώστε να μπορεί να είναι πολυεπίπεδο(***Layered***). Αυτό σημαίνει οτι από τη στιγμή που ο client χρειάζεται να γνωρίζει μόνο τα endpoints στα οποία θα κάνει http request και οτι θα παραλάβει πληροφορία, ο server μπορεί να διαχωρίζεται σε πολλά μέρη, να εκτείνεται με ενδιάμεσους server, όπως κάποιον cache server ή έναν ξεχωριστό database server χωρίς να επηρεάζεται η λειτουργία του συστήματος.

3. Η υποστήριξη προσωρινής αποθήκευσης ***cache***. Όλα τα δεδομένα(στο εξής resources) που απαντώνται από το API πρέπει να ορίζουν την ύπαρξη ή μη της δυνατότητας caching ώστε να γνωρίζουν όλα τα επίπεδα του συστήματος, από τον client έως του ενδιάμεσους server αν μπορούν να τα αποθηκεύσουν. Στις περιπτώσεις που τα resources είναι στατικά , δεν αλλάζουν συχνά, ή δεν χρειάζεται ο client να κατέχει την τελευταία τους έκδοση είναι συμφέρον να αποθηκεύονται σε κάποιο σημείο της διαδρομής ώστε αν ξαναζητηθούν να είναι ετοιμοπαράδοτα, ελαφρύνοντας τη χρήση των servers και των βάσεων δεδομένων μέσω διαδικτύου. Στην περίπτωση που τα resources είναι δυναμικά και είναι δημαντικό ο χρήστης να έχει την νεότερη έκδοσή τους, τότε είναι προτιμότερο να γίνεται εκ νέου ζήτηση από το api και τη βάση δεδομένων κάθε φορά.

4. Συνεχίζοντας στην ίδια φιλοσοφία του διαχωρισμού client – server ορίζεται ο περιορισμός μη ύπαρξης καταστάσεων(***Statelessness***). Όταν γίνεται επικοινωνία μεταξύ client – server δεν πρέπει να διατηρείται ειδικά από τον server καμία κατάσταση ή ιστορικό της επαφής αυτής, παρά μόνο μια ανταλλαγή HTTP requests και responses. Σαν απλό παράδειγμα μη έγκυρου RESTful συστήματος με την PHP είναι η παραδοσιακή ροή της συνεδρίας(Session) κατά την οποία ο server διατηρεί την κατάσταση κάθε χρήστη κυρίως για λόγους ταυτοποίησης μέχρι αυτός να απομακρυνθεί από την ιστοσελίδα. Σε ένα RESTful σύστημα μόνο ο client μπορεί να διατηρήσει στοιχεία ταυτοποίησης που δίνονται από τον server. Αυτό επιτυγχάνεται συνήθως με cookies ή όπως χρησιμοποιήθηκαν στην εφαρμογή συνεδρίων JWT(Json Web Tokens) στα οποία θα αναφερθούμε σε επόμενη ενότητα.
5. Ομοιομορφία στην διεπαφή(Uniform Interface). Ορίζεται οτι πρέπει τόσο τα endpoints τα οποία παρέχονται στον καταναλωτή από το API, όσο και η πληροφορία που θα επιστραφεί να έχει πάντα την ίδια και αναμενόμενη μορφή. Για παράδειγμα, όπως στην εφαρμογή για συνέδρια, όλα τα δεδομένα να παρέχονται σε μορφή JSON και με περιεχόμενα ίδιας δομής παρόλο που ο server και το API μπορεί να μην αναγνωρίζουν αυτή τη μορφή. Όσο για τα endpoints πρέπει να βγάζουν νόημα και να ακολουθούν την ίδια φιλοσοφία, για παράδειγμα το request GET *confapp.com/conferences/1* επιστρέφει πληροφορίες του συνεδρίου με id=1, το ίδιο πρέπει να ισχύει και για τον χρήστη με id=1 όταν ζητηθεί *GET confapp.com/users/1*, ενώ αν δίνεται δυνατότητα εγγραφής νέου στοιχείου θα είναι αντίστοιχα *POST confapp.com/conferences και POST confapp.com/users*
6. Τέλος, αλλά προαιρετικά μπορεί ένα RESTful API αν χρειαστεί, να επιστρέφει κώδικα.

Με αυτές τις προδιαγραφές μπορεί κανείς να δημιουργήσει ένα RESTful API αν και κανείς δεν τις επιβάλει, μπορεί ο κάθε προγραμματιστής ανάλογες με τις ανάγκες του συστήματος που δημιουργεί να χρησιμοποιήσει ένα μέρος από αυτές τις συνθήκες οπότε το σύστημά του να είναι εν μέρει RESTful. Υπάρχουν όμως και διάφορες άλλες συνθήκες για REST που όμως είναι προαιρετικές ή περιλαμβάνονται μέσα στις 6 βασικές.

Τελικά , οι λόγοι που τα πιο γνωστά APIs που υπάρχουν σήμερα όπως αυτά της Google, του Twitter, Facebook κλπ βασίζονται στο REST είναι η πολύ καλή επεκτασιμότητα λόγω της ιεραρχίας και του modularity, της απλότητας κάνοντας χρήση γνωστών και καθιερωμένων τεχνολογιών όπως το πρωτόκολλο HTTP και της ευρείας συμβατότητας με προγράμματα περιήγησης.

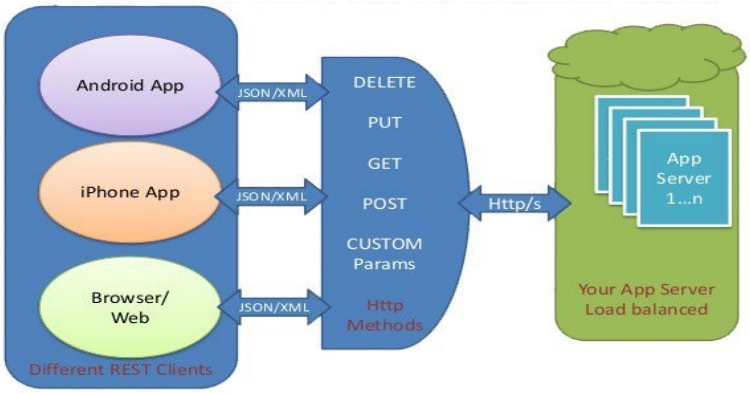


Figure 5: REST API αρχιτεκτονική

3.7 **JSON Web Tokens**

Τα JWT είναι η λύση που χρησιμοποιήθηκε στην εφαρμογή συνεδρίων για την ταυτοποίηση των χρηστών. Είναι ένα πακέτο από πληροφορίες σε μορφή αντικειμένου JSON με συγκεκριμένη δομή και στοιχεία το οποίο εκδίδει το API και αποστέλλει στον client. Περιέχει πληροφορίες σχετικά με το διάστημα της περιόδου χρήσης, στιγμή έκδοσης και άλλες χρήσιμες πληροφορίες για το σύστημα, ένα κρυπτογραφημένο από τον server πεδίο με μοναδικό κλειδί και ίσως κάποιο μικρό ωφέλιμο φορτίο. Ο client με τη σειρά του το αποθηκεύει τοπικά και με κάθε κλήση προς το API το επιστρέφει αυτούσιο ώστε αυτό να μπορεί να αποκρυπτογραφήσει το κλειδί και να ελέγξει την κατάστασή του. Για την ακρίβεια η δομή του έχει ως εξής:

- Επικεφαλίδα(***Header***): περιέχει το είδος του token και τον αλγόριθμο κρυπτογράφησης που χρησιμοποιήθηκε για την ηλεκτρονική υπογραφή.

  {
  “alg”: “HS256”,
  “typ”: “JWT”
  }

- Φορτίο(***Payload***): πληροφορίες σχετικά με το σύστημα και ωφέλιμο φορτίο

  {
  ”iss”: “issuer”,
  ”exp”: expiration time,
  ”id”: user id,
  ”admin”: true
  }

- Η επικεφαλίδα και το φορτίο κωδικοποιούνται η κάθε μία ξεχωριστά με base64 σε γραμματοσειρά και μαζί με ένα μοναδικό κλειδί εισάγονται στον αλγόριθμο κρυπτογράφησης που ορίστηκε στην επικεφαλίδα και δημιουργείται μια υπογραφή που αποτελεί και το τρίτο μέρος του JWT. Με αυτό τον τρόπο μόνο το API που το δημιούργησε μπορεί να το αποκρυπτογραφήσει αλλά και να ελέγξει για τυχούσες κακόβουλες μετατροπές του token από τρίτους καθώς η υπογραφή δημιουργήθηκε με τα αυθεντικά μέρη του.

Τέλος το τρίτο μέρος κωδικοποιείται και αυτό με base64 και ολοκληρώνεται το token, το οποίο δείχνει ως εξής:

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

Figure 6: Κωδικοποιημένο JWT token

Το JWT είναι σχετικά νέα ιδέα όμως σε σχέση με τις εναλλακτικές σε πολλές περιπτώσεις υπερτερεί λόγω της ασφάλειας, της απλότητας, της συμβατότητας και του μεγέθους του που το καθιστά πολύ

γρήγορο σε απόδοση, αν σκεφτεί κανείς οτι αποστέλλεται με το εκατό τοις εκατό των κλήσεων προς το API.

Όλες οι παραπάνω τεχνολογίες χρησιμοποιήθηκαν στην εφαρμογή που υλοποιήθηκε σαν μέρος της παρούσας εργασίας η οποία παρουσιάζεται αναλυτικά στα επόμενα κεφάλαια.

4 Λειτουργία της εφαρμογής

4.1 Εγκατάσταση και δοκιμή

Για τη δοκιμή του συστήματος χρειάζεται κατ ελάχιστο:

- Λειτουργικό σύστημα Windows
- Κινητό τηλέφωνο με λειτουργικό σύστημα Android
- XAMPP v3.2.2
- Σύνδεση στο διαδίκτυο

Τα βήματα εγκατάστασης είναι τα εξής:

- Ο φάκελος ConferenceApp που περιέχει το REST API να αντιγραφεί μέσα στον φάκελο htdocs του XAMPP ο οποίο βρίσκεται συνήθως στο C:\xampp\htdocs
- Ενεργοποιούμε το πρόγραμμα XAMPP Control Panel και από εκεί ενεργοποιούμε το Apache και το MySql.
- Από τον browser κατευθυνόμαστε στο http://127.0.0.1/phpmyadmin/ όπου μέσω του εργαλείου στην καρτέλα import δίνουμε το αρχείο της εργασίας conferenceapp_db.sql.gz ώστε να δημιουργηθεί η βάση δεδομένων με κάποια υποτυπώδη δεδομένα.
- Σε μια γραμμή εργαλείων ως administrator δίνουμε την εντολή

  ssh -o ServerAliveInterval=30 -R confapi.serveo.net:80:localhost:80 serveo.net

  ώστε να ανοίξουν πόρτες μέσω της υπηρεσίας serveo στον τοπικό μας εξυπηρετητή και να είναι διαθέσιμος στο διαδίκτυο με συγκεκριμένο όνομα το οποίο γνωρίζει η εφαρμογή μας. Τα 80 μπορεί να χρειαστεί να αλλάξουν σε 8080 ή αναλογα με τη ρύθμιση του XAMPP.
- Τέλος, μέσα στον φάκελο υπάρχει αρχείο ConfApp.apk το οποίο μπορεί να αντιγραφεί και εγκατασταθεί σε συσκευή Android. Έχει δοκιμαστεί σε περιορισμένο αριθμό συσκευών όμως δεν παρουσίασε πρόβλημα.

4.2 IONIC Conference Application

Ακολουθούν εικόνες και περιγραφή του περιβάλλοντος χρήστη της υβριδικής εφαρμογής και στη συνέχεια μια πιο αναλυτική ματιά στον κώδικα με τον οποίο υλοποιήθηκε η λειτουργικότητα και η ‘κατανάλωση’ του REST API.

4.2.1 Γραφικό περιβάλλον

Η πρώτη εικόνα(σελίδα) που συναντά ο χρήστης ανοίγοντας την εφαρμογή είναι αυτή της φόρμας εισόδου με τα στοιχεία του λογαριασμού του(figure 7). Σε περίπτωση που δεν έχει ήδη λογαριασμό, δίνεται η επιλογή μέσω ενός υπερκειμένου να μεταφερθεί σε μια νέα σελίδα η οποία περιέχει φόρμα εισαγωγής στοιχείων καθώς και επιλογή για επιστροφή στην σελίδα εισόδου. Όταν η φόρμα συμπληρωθεί και αποσταλεί, αν τα στοιχεία είναι έγκυρα και δεν υπάρξει κάποιο άλλο σφάλμα γίνεται κατευθείαν η ταυτοποίηση του χρήστη και προχωρά αυτόματα στην σελίδα επιλογής συνεδρίου. Η σελίδα εισόδου επίσης περιέχει υπερκείμενο το οποίο όταν ενεργοποιηθεί κάνει αυτόματη ταυτοποίηση σαν επισκέπτη, αποκτώντας JWT για πρόσβαση στα δεδομένα των συνεδρίων. Σε αυτή την περίπτωση όμως οι επιλογές των ‘αγαπημένων’ ομιλιών του χρήστη στα συνέδρια αποθηκεύονται τοπικά στην συσκευή και όχι στη βάση δεδομένων, με αποτέλεσμα την διαγραφή των δεδομένων κατά την απεγκατάσταση της εφαρμογής.

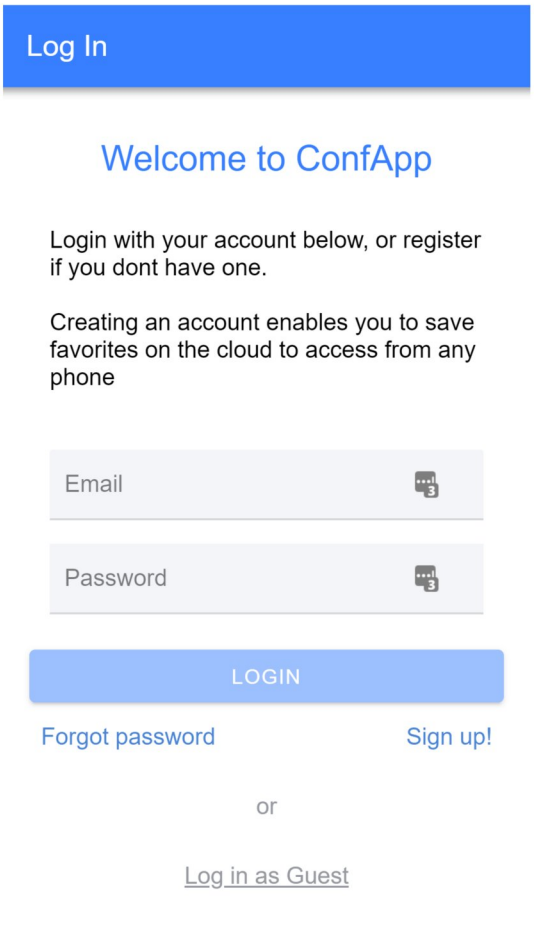


Figure 7

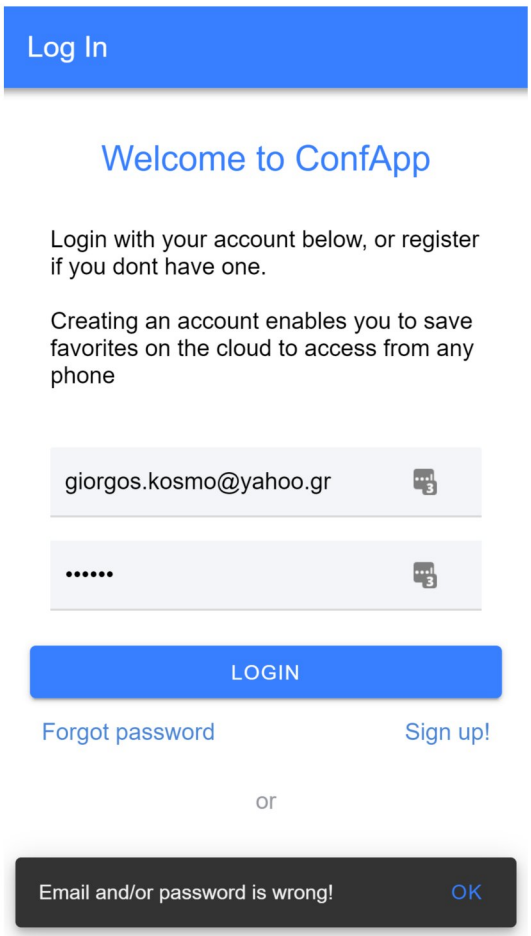


Figure 8

Το κουμπί LOGIN παραμένει ανενεργό, μέχρι να συμπληρωθεί η φόρμα με στοιχεία που τηρούν τις συνθήκες που έχουμε ορίσει, όπως για παράδειγμα το πλήθος των χαρακτήρων ή το email να είναι στην κατάλληλη μορφή. Με κάθε χαρακτήρα που εισάγεται σε πεδίο της φόρμας γίνεται δυναμικά έλεγχος και εμφανίζεται κατάλληλο μήνυμα κάτω από αυτό(figure 9). Αν είναι στη σωστή μορφή και αποσταλεί η φόρμα όμως υπάρχει κάποιο απομακρυσμένο σφάλμα όπως στην βάση δεδομένων ή το API, εμφανίζεται επίσης κατάλληλο μήνυμα σε αιωρούμενο πλαίσιο(figure 8).

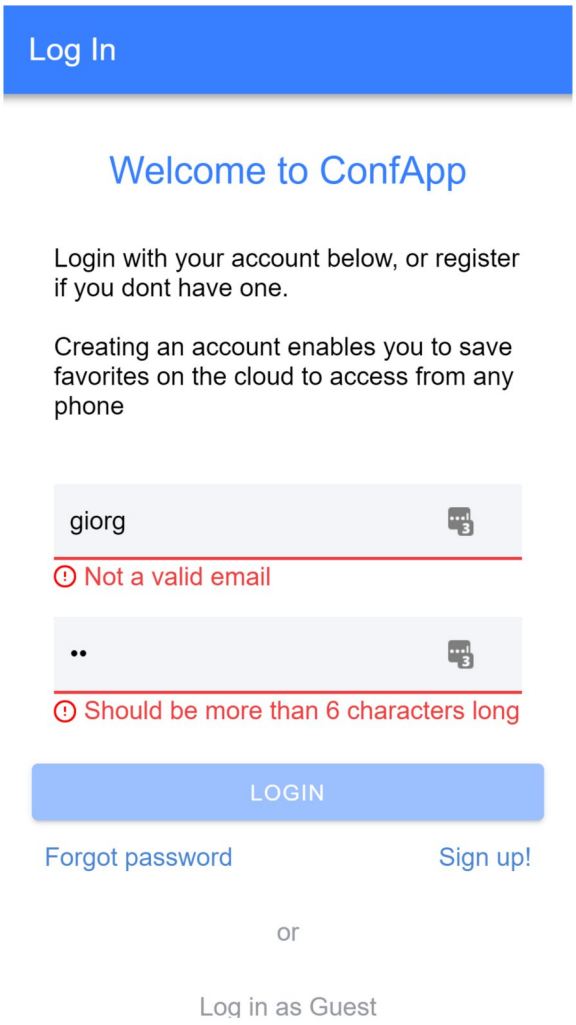


Figure 9

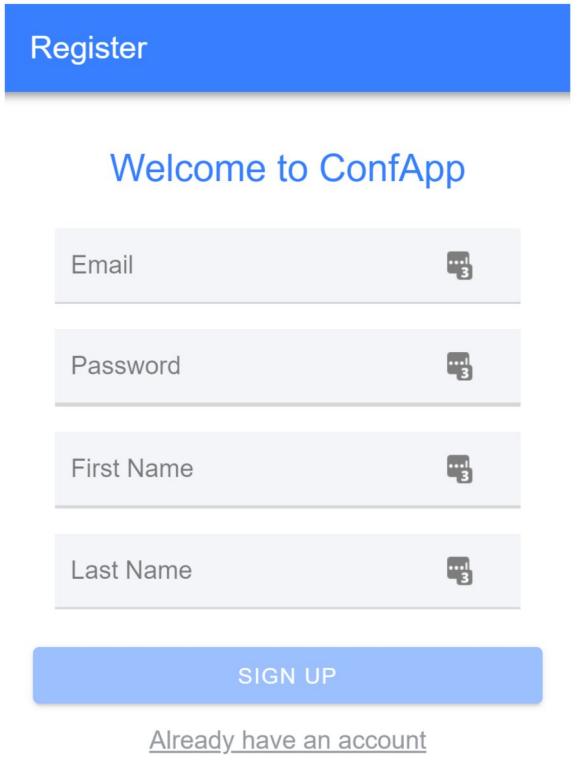
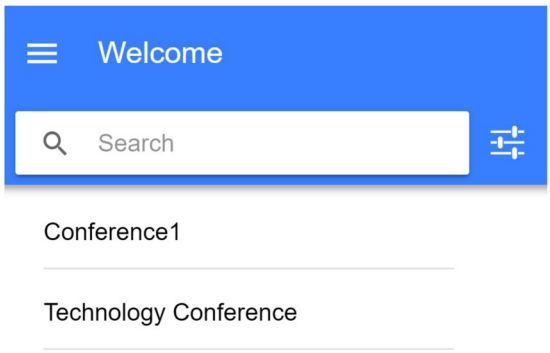


Figure 10

Ο έλεγχος γίνεται και στην φόρμα νέου λογαριασμού.

Μετά την ταυτοποίηση, ο χρήστης βρίσκεται στην σελίδα επιλογής συνεδρίου, όπου μπορεί να περιηγηθεί στη λίστα με τα συνέδρια που έχουν καταχωρηθεί ή να κάνει γραπτή αναζήτηση κατά όνομα στο κατάλληλο πλαίσιο. Στη λίστα εμφανίζονται μόνο τα συνέδρια των οποίων η ημερομηνία της τελευταίας μέρας διεξαγωγής δεν ξεπερνά την ημερομηνία του παρόντος, μετά από έλεγχο που πραγματοποιούμε.



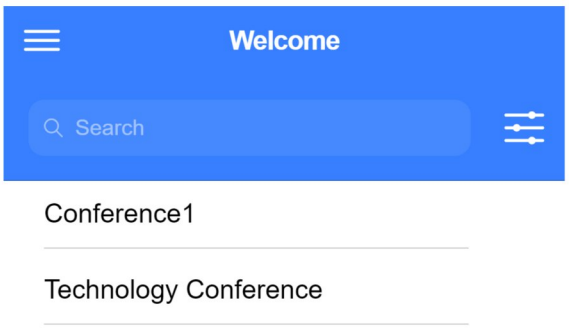


Figure 11: Android

Figure 12: IOS

Παρατηρούμε τις διαφορές ανάμεσα σε
συσκευές με λειτουργικό σύστημα Android και iOS στην εμφάνιση της εφαρμογής, τις οποίες αναλαμβάνει αυτόματα το IONIC Framework ανάλογα με την πλατφόρμα. Εμείς προγραμματίζουμε την γενικότερη δομή και στοιχεία της διεπαφής και το IONIC εκτελεί αλλαγές διατηρώντας τη δομή μας αλλά προσθέτοντας αρκετή “γεύση” χαρακτηριστική του κάθε συστήματος ώστε η εφαρμογή να μην φαίνεται ξένη στον χρήστη.

Ύστερα από την επιλογή συνεδρίου ο χρήστης βρίσκεται πλέον στο κεντρικό μέρος της εφαρμογής και συγκεκριμένα στην σελίδα προγράμματος του συνεδρίου. Στο όλες τις σελίδες του κεντρικού μέρους της εφαρμογής υπάρχουν στο κάτω μέρος της διεπαφής καρτέλες πλοήγησης ανάμεσά τους και πάνω αριστερά ένα κουμπί με τις γνώριμες τρεις οριζόντιες γραμμές, το οποίο όταν ενεργοποιηθεί εμφανίζει ένα πλευρικό μενού με επιλογές για επιλογή νέου συνεδρίου και ένα κουμπί που αποσυνδέει τον συγκεκριμένο χρήστη οδηγώντας αμέσως μετά στην σελίδα σύνδεσης(log in).

Μέσω των καρτελών επιλογής ημέρας αλλά και αναζήτησης με κείμενο μπορεί ο χρήστης να φιλτράρει τις ομιλίες, ενώ στο κάθε στοιχείο της λίστας ομιλιών φαίνονται και μερικές λεπτομέρειες σχετικά με αυτές όπως η ώρα και η αίθουσα διεξαγωγής.

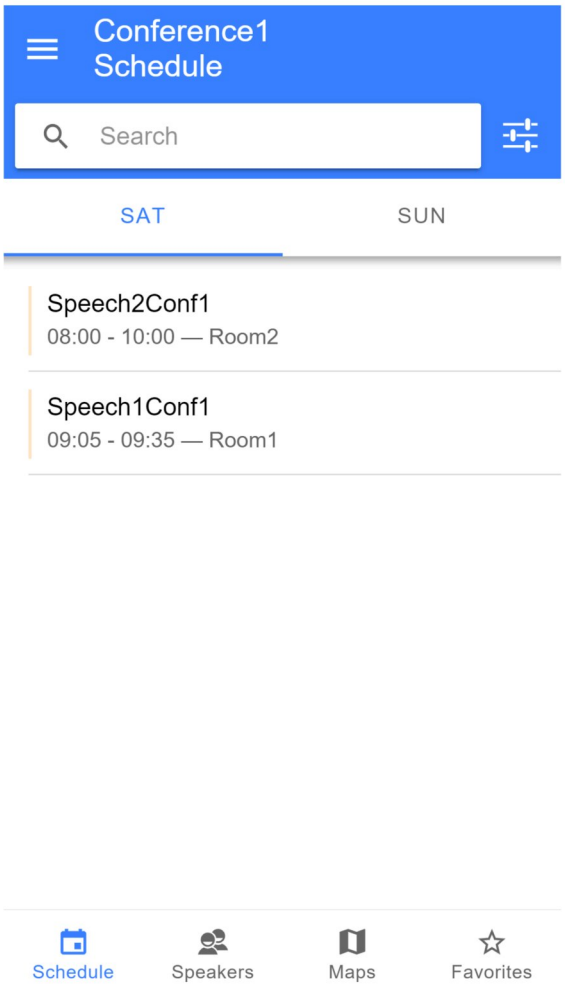


Figure 13: schedule android

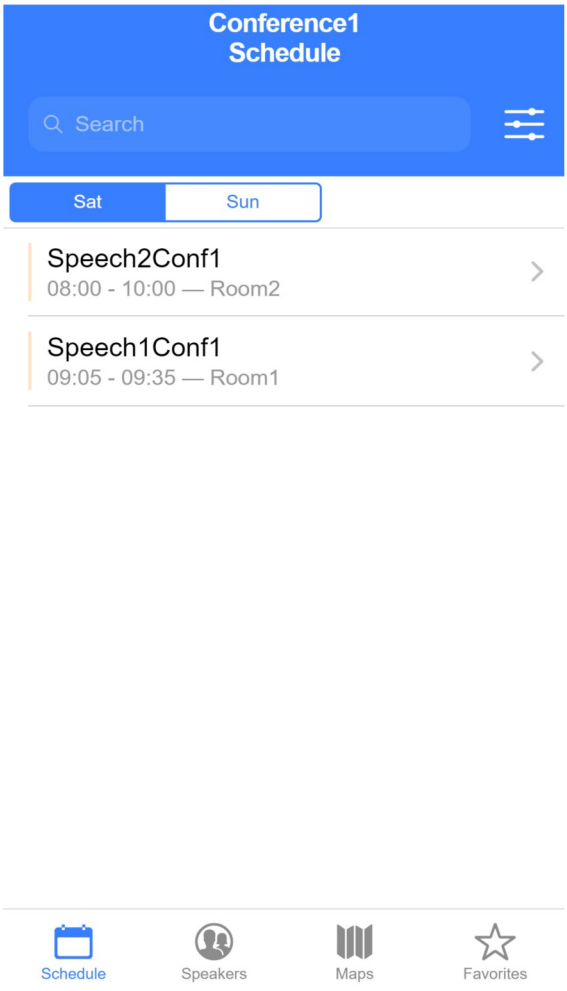


Figure 14: schedule iOS

Ένας άλλος τρόπος για την διευκόλυνση προγραμματισμού της επίσκεψης ενός χρήστη εμφανίζεται με το πάτημα του κουμπιού επιλογών και περιλαμβάνει διακόπτες με το όνομα των αιθουσών για τις οποίες θέλει να εμφανιστούν ομιλίες στο πρόγραμμα.(figure15,16)

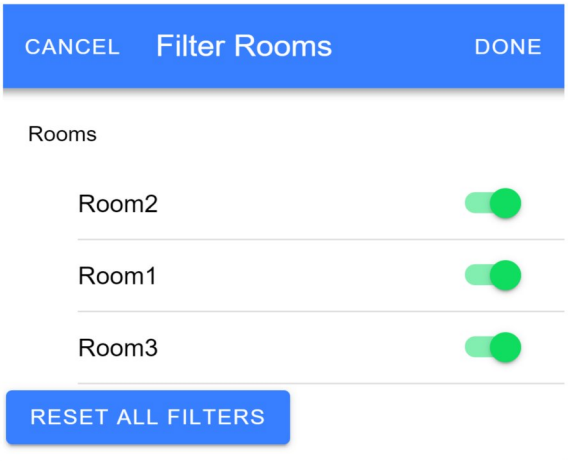


Figure 15: Φίλτρο αιθουσών Android

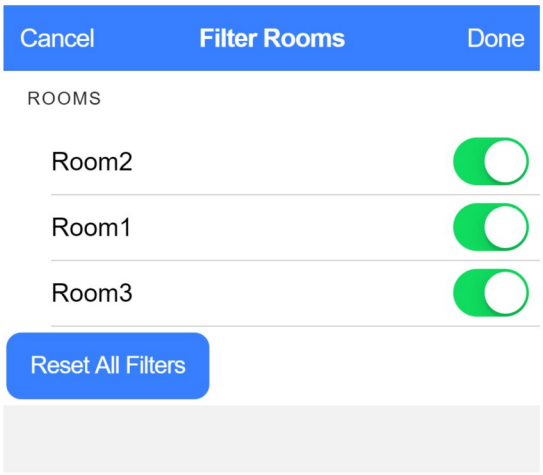


Figure 16: Φίλτρο αιθουσών iOS

Τέλος, επιλέγοντας μία από τις ομιλίες γίνεται μεταφορά σε δευτερεύουσα σελίδα η οποία περιέχει λεπτομέρειες για αυτήν, όπως ημερομηνία, ώρα, αίθουσα, περιγραφικό κείμενο και μια λίστα από όλες τους ομιλητές. Αυτοί με τη σειρά τους μπορούν να επιλεχθούν και να γίνει μετάβαση στην σελίδα λεπτομερειών ομιλητή.

Figure 18: Λεπτομέρειες ομιλίας Android 1

Figure 17: Λεπτομέρειες ομιλίας Android 2

Στη σελίδα αυτή υπάρχει και το κουμπί ***favorite*** το οποίο μόλις επιλεχθεί χρωματίζεται για να δείξει το γεγονός οτι η ομιλία αυτή αποθηκεύτηκε στις αγαπημένες του χρήστη ή αν είναι ήδη χρωματισμένο αφαιρείται από τις αγαπημένες. Τις αγαπημένες ομιλίες που έχει επιλέξει μπορεί να τις δει στην τελευταία καρτέλα πλοήγησης “Favorites” η οποία περιέχει μια σελίδα παρόμοια με αυτήν του προγράμματος, όμως εμφανίζει μόνο τις μαρκαρισμένες ως αγαπημένες ομιλίες και οι καρτέλες για ταξινόμηση κατά ημέρα δημιουργούνται δυναμικά ανάλογα μόνο για τις ημέρες που περιέχουν τουλάχιστον μία αγαπημένη ομιλία. Σε περίπτωση που δεν υπάρχουν αγαπημένες ομιλίες η μετάβαση στην καρτέλα “Favorites” δεν ολοκληρώνεται, η εφαρμογή παραμένει στην προηγούμενη σελίδα και εμφανίζει κατάλληλο μήνυμα.

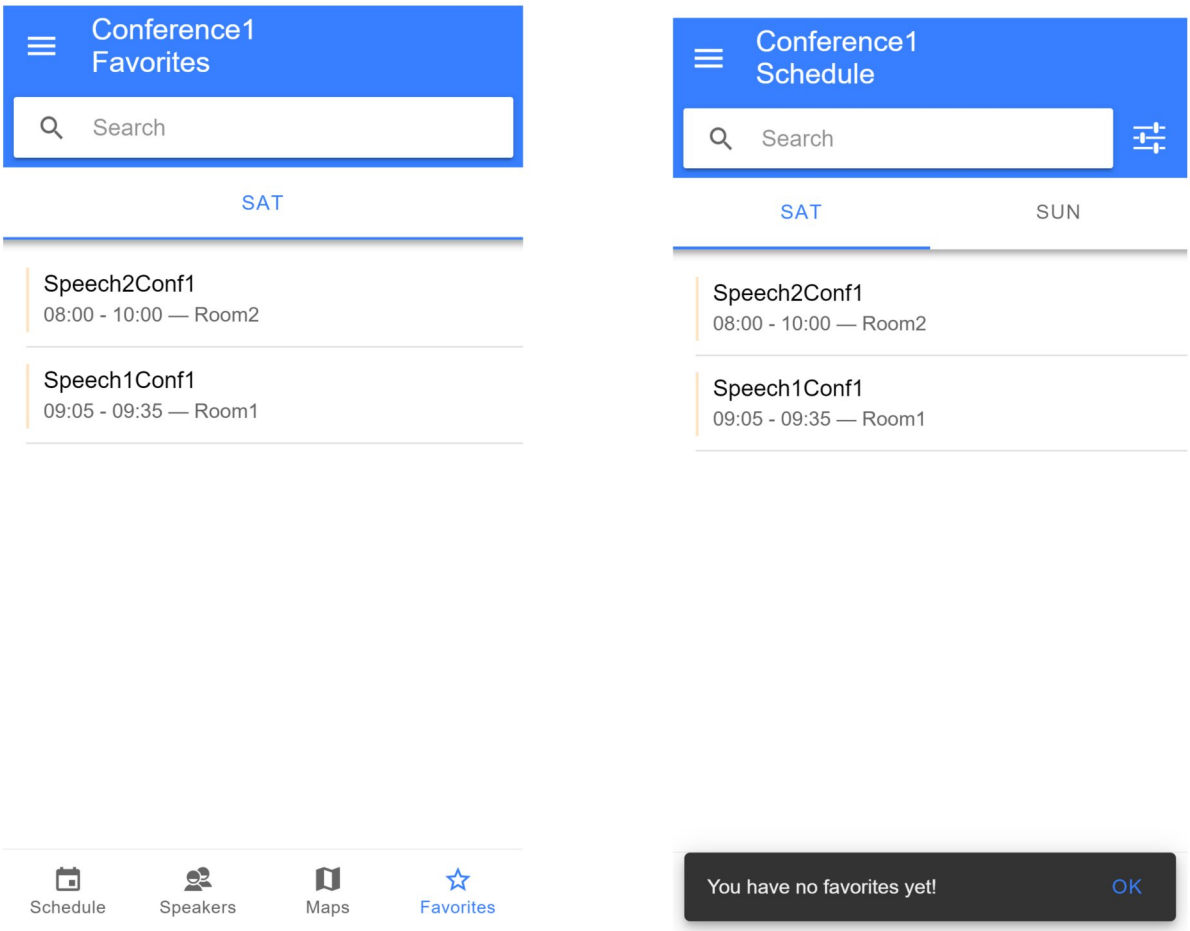


Figure 19: Σφάλμα στα Αγαπημένα

Η καρτέλα “Speakers” περιέχει μια σελίδα με μια λίστα ονομάτων όλων των ομιλητών του επιλεγμένου συνεδρίου. Όταν επιλεχθεί ένας από αυτούς γίνεται μετάβαση στη δευτερεύουσα σελίδα λεπτομερειών ομιλητή. Αυτή με τη σειρά της περιέχει εκτός των άλλων και λίστα με τις ομιλίες στις οποίες συμμετέχει ο ομιλητής που οδηγούν στην σελίδα Λεπτομερειών ομιλίας(figure 18).

Τέλος, στην καρτέλα “Maps” υπάρχει ένας διαδραστικός χάρτης κεντραρισμένος στο σημείο διεξαγωγής του συνεδρίου. Ο χάρτης εμφανίζεται με τη χρήση ενός javascript API της

εταιρίας MapBox.

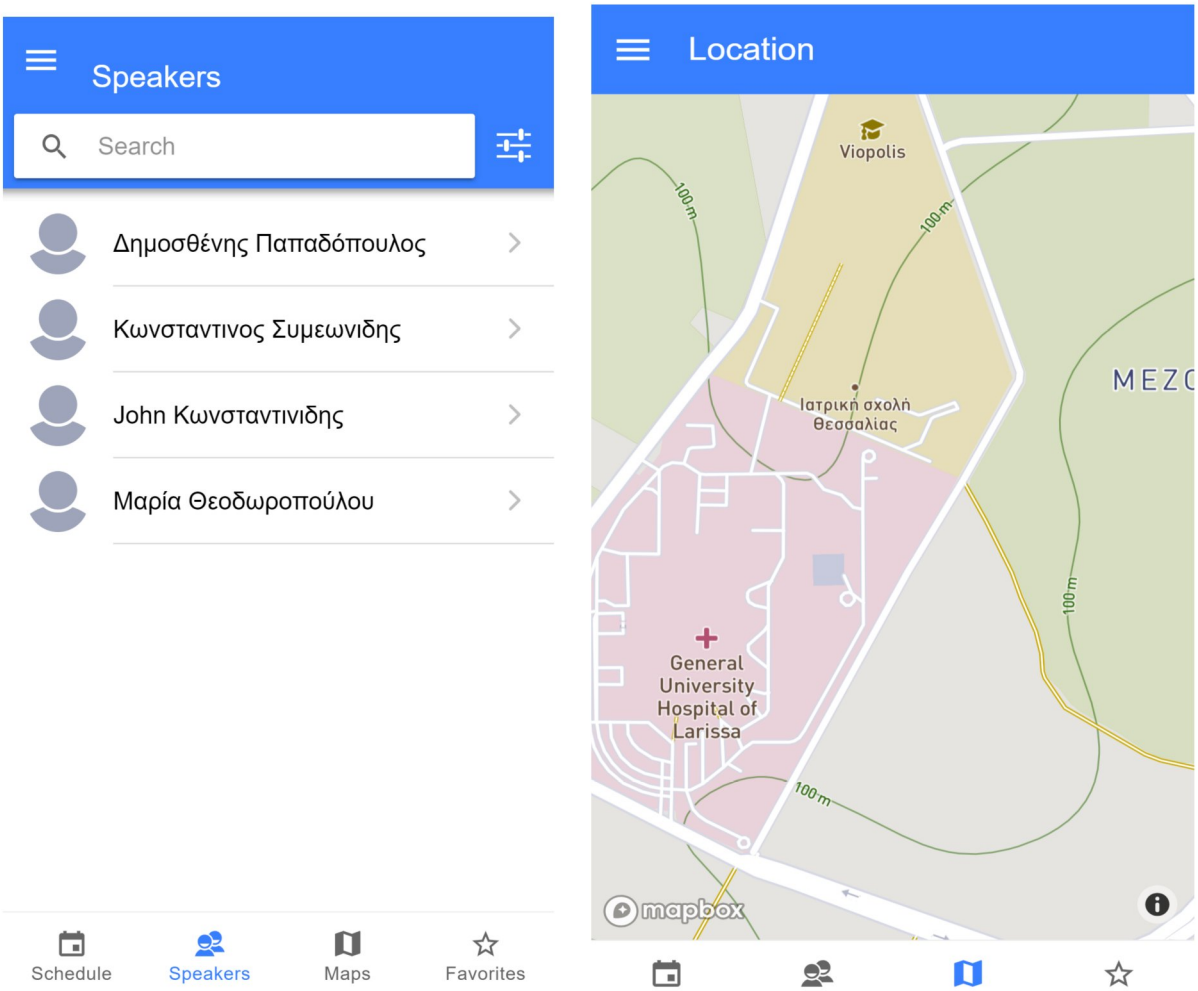


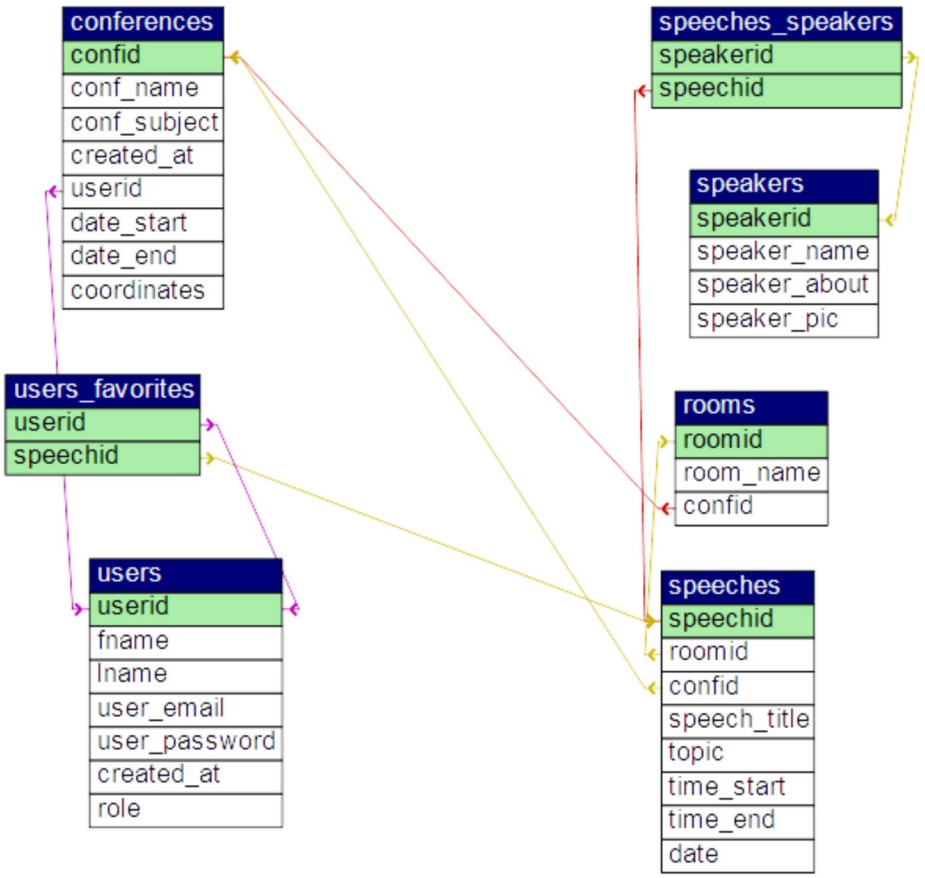
Figure 20: Λίστα ομιλητών

Ακολουθεί ανάλυση του σχεδιασμού και των λειτουργιών της εφαρμογής σε συνδυασμό με παρουσίαση του περιβάλλοντος χρήσης και της διεπαφής καθώς και επίδειξη επίμαχων σημείων του κώδικα.

5 Σχεδιασμός και υλοποίηση

5.1 Βάση δεδομένων

Η βάση δεδομένων είναι MySql, έρχεται προεγκατεστημένη με το πακέτο XAMPP μαζί με το εργαλείο διαχείρισης PhpMyAdmin. Η δομή της βάσης δεδομένων που χρησιμοποιήθηκε σε σχεδιάγραμμα πινάκων:



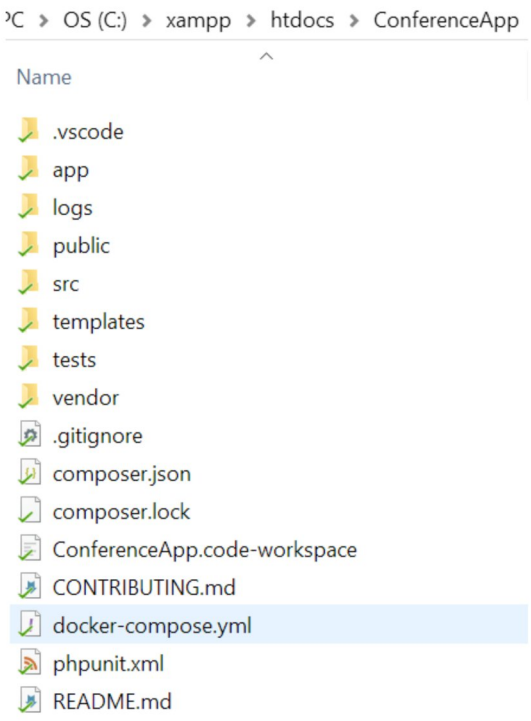
Η βάση δεδομένων δεν έχει ιδιαίτερα εκτεταμένη μορφή για το σκοπό της συγκεκριμένης εφαρμογής καθώς τα περιεχόμενά εξυπηρετούν το στόχο της, όμως μπορεί να επεκταθεί σε περίπτωση αναβάθμισης του συστήματος σε βάθος χρόνου, ανάλογα με τις ανάγκες των νέων δυνατοτήτων.

5.2 REST API

Για τη διαχείριση της διεπαφής μεταξύ δεδομένων και χρήστη αναπτύχθηκε RESTful API σε SLIM php framework και αναλύονται στη συνέχεια μερικά σημαντικά σημεία του κώδικα και της δομής. Έχουν χρησιμοποιηθεί ιδέες από την σκελετική εφαρμογή που παρέχεται από τους δημιουργούς του SLIM Framework, κάτι το οποίο προτείνεται και από τους ίδιους.

Για αρχή η εφαρμογή αυτή είναι server-side και εξαρτάται από τον Apache server. Έτσι, την εκκινούμε στον φάκελο htdocs όπου βρίσκονται τα αρχεία που σερβίρονται στο διαδίκτυο, όπως θα κάναμε και με μια παραδοσιακή ιστοσελίδα. Αυτό το κάνουμε μέσω του εργαλείου Composer και την εντολή :

- *php composer.phar create-project ConferenceApp*



Οι φάκελοι που περιέχουν την λογική του API μας είναι app και src, στον πρώτο βρίσκονται οι controllers με τις μεθόδους που επικοινωνούν με τη βάση δεδομένων, ενώ στο δεύτερο τα αρχεία που περιέχουν τα routes, το middleware και το dependency container. Η εφαρμογή όμως κατά τη χρήση της πρώτα διαβάζει το αρχείο index.php στον φάκελο public που περιέχει τις πρώτες ρυθμίσεις παραμέτρων απαραίτητων για τη λειτουργία. Στον φάκελο public υπάρχει επίσης το αρχείο .htaccess που περιέχει ρυθμίσεις σχετικά με την ενεργοποίηση δυνατοτήτων του server για την εφαρμογή μας και δήλωση ρυθμίσεων σχετικά με τα URL.

index.php

```php
<?php
require __DIR__ . '/../vendor/autoload.php';

// Instantiate the app
$settings = require __DIR__ . '/../src/settings.php';
$app = new \Slim\App($settings);

// Set up dependencies
require __DIR__ . '/../src/dependencies.php';
unset($app->getContainer()['notFoundHandler']);
$app->getContainer()['notFoundHandler'] = function ($c) {
    return function ($request, $response) use ($c) {
        return $response->withJson([
            'Error' => true,
            'Data' => '',
            'Message' => 'URL not correct!'
        ], 404);
    };
};

// Register middleware
require __DIR__ . '/../src/middleware.php';

// Register routes
require __DIR__ . '/../src/routes.php';

// Run app
$app->run();

```

Στο αρχείο index.php ορίζουμε το autoloader που δημιουργεί αυτόματα το slim το οποίο χρησιμοποιεί την τεχνική της php κατα την οποία κάθε φορά που ζητείται από τον κώδικα η δημιουργία ενός αντικειμένου μιας κλάσης ενεργοποιείται ο autoloader αυτόματα και ψάχνει το όνομα της κλάσης ή το namespace σε οποιοδήποτε αρχείο του πρότζεκτ βρίσκεται για να δημιουργήσει το αντικείμενο. Αυτό το κάνουμε ώστε να μη χρειάζεται να αναφέρουμε και να δηλώσουμε σε κάθε αρχείο ρητά με τη λέξη κλειδί include όλες τις κλάσεις και τα αρχεία τους που μπορεί να χρειαστούμε. Κυρίως δηλαδή για την αποφυγή επανάληψης κώδικα και την ‘καθαριότητα’ του. Στο αρχείο composer.json ορίσαμε και εμείς όλες τις κλάσεις μας για να είναι ορατές σε αυτή τη διεργασία. Στη συνέχεια ορίζουμε μια μεταβλητή που περιέχει ρυθμίσεις που έχουμε ήδη αποθηκεύσει στο αρχείο settings.php και το δίνουμε σαν όρισμα στον constructor της κλάσης App η οποία και δημιουργεί την βασική διεργασία που αποτελεί την εφαρμογή μας και περιέχει όλες τις ρυθμίσεις , dependencies κλπ.

```php
'settings' => [
    'displayErrorDetails' => false, // set to false in production
    'addContentLengthHeader' => false, // Allow the web server to send the content-length header

    'db' => [
        'host' => '127.0.0.1',
        'dbname' => 'conferenceapp_db',
        'user' => 'root',
        'pass' => 'root'
    ],

    "jwt" => [
        'secret' => 'EAx0R4wQFJejGcKo9SW9Lgc-r5ytADg0q9kRzFZco-cT4CZHrd7vY8nNXc4zcZ0NandkFq7nwiNcYlfEFQl9Fw',
        'algorithm' => ['HS256'],
        'attribute' => 'decoded_token_data',
        'error' => function ($response) {
            return $response->withJson(
                [
                    'Error'=>True,
                    'Data'=>NULL,
                    'Message'=>'You are not authorized to do that'
                ],401
            );
        }

    ]
],
```

Εδώ φαίνεται ένα δείγμα τους αρχείου settings.php που περιέχει μια δομή που δημιουργήσαμε σχετικά με τη χρήση του JWT, όπως τον μοναδικό κωδικό κρυπτογράφησης των tokens(η εικονιζόμενη τιμή είναι ενδεικτική και δεν είναι ασφαλής για εμπορική χρήση) και τη μορφή που επιθυμούμε να έχει το HTTP response σε περίπτωση που το JWT δεν είναι έγκυρο.

Η δομή του HTTP response είναι ίδια για όλη την εφαρμογή και επιλέξαμε όλες οι απαντήσεις να έχουν ομοιομορφία και να είναι σε μορφή αντικειμένου JSON για να εμπίπτει στα πλαίσια των οδηγιών του REST design. Η μέθοδος withJson() προσφέρεται από το SLIM και αναλαμβάνει να κωδικοποιήσει την απάντηση αποτελούμενη απο πίνακες PHP σε JSON και δέχεται σαν δεύτερο όρισμα κωδικούς λάθους ή επιτυχίας με βάση το *HTTP/1.1 standard* οι οποίοι, όπως θα αναφέρουμε στη συνέχεια, μας βοηθούν στην μεριά της υβριδικής εφαρμογής να αναγνωρίσουμε την κατάσταση της απάντησης και να εμφανίσουμε κατάλληλα μηνύματα.

Συνεχίζοντας στον φάκελο src και στο αρχείο middleware.php που περιέχει τις ενδιάμεσες ‘στρώσεις’ λογισμικού που ενεργοποιούνται με τη σειρά όταν δεχόμαστε request και όταν αποστέλλουμε response.

```
$app->add(new Tuupola\Middleware\CorsMiddleware([
    "origin" => ["*"],
    "methods" => ["GET", "POST", "PUT", "PATCH", "DELETE", "OPTIONS", "OPTION"],
    "headers.allow" => ["Authorization", "If-Match", "If-Unmodified-Since","Content-Type"],
    "headers.expose" => ["Etag"],
    "credentials" => true,
    "cache" => 86400
]));
```

Αυτό είναι το μόνο middleware που έχει οριστεί σε αυτό το αρχείο και η λειτουργία του είναι να δημιουργεί ένα αντικείμενο CorsMiddleware που παρέχεται από το ομώνυμο plugin για slim php που εγκαταστήσαμε. Το συγκεκριμένο middleware ορίζει τις πολιτικές επικοινωνίας εξωτερικής προέλευσης του server και αναλαμβάνει τα requests πριν την εφαρμογή ώστε να ελέγχει αν υπακούν στους περιορισμούς.

Στη συνέχεια το αρχείο dependencies.php. Το SLIM php δημιουργεί αυτόματα έναν πίνακα ονόματι container ο οποίος αποτελείται από μεταβλητές που δημιουργούν και επιστρέφουν στιγμιότυπα κλάσεων τη στιγμή που θα γίνει αναφορά σε αυτές. Τα αντικείμενα

```
$container = $app->getContainer();

$container['db'] = function ($container) {
    $settings = $container->get('settings')['db'];
    mysqli_report(MYSQLI_REPORT_ERROR | MYSQLI_REPORT_STRICT);
    try {
        $conn = new mysqli($settings['host'], $settings['user'], $settings['pass'], $settings['dbname']);
        $conn->set_charset("utf8");
        return $conn;
    } catch (Exception $e) {
        echo $e->getCode();
        echo " " . $e->getMessage();
    }
};

$container['validator'] = function ($container) {
    Respect\Validation\Validator::with('App\\Validation\\Rules');
    return new App\Validation\Validator();
};

$container['jsonResponse'] = function ($container) {
    return new App\Services\JsonResponseProvider;
};

$container['jwt'] = function ($c) {
    $jwt_settings = $c->get('settings')['jwt'];
    return new Tuupola\Middleware\JwtAuthentication($jwt_settings);
};
```

αυτά θεωρούνται κοινές εξαρτήσεις άλλων κλάσεων οι οποίες μπορούν να χρησιμοποιήσουν τον container με την μέθοδο getContainer() της εφαρμογής.

Σαν παράδειγμα η πρώτη εξάρτηση αφορά τη δημιουργία αντικειμένου σύνδεσης με τη βάση δεδομένων μέσω του module MySqli της PHP. Το αντικείμενο αυτό επαναχρησιμοποιούν αυτούσιο οι κλάσεις που επικοινωνούν με τη βάση δεδομένων. Το dependency injection και

τα Callable Resolvers του slim php είναι αρκετά περίπλοκες, αυτοματοποιημένες διεργασίες και ένας από τους λόγους που χρησιμοποιούμε ένα PHP framework.

Στο αρχείο routes.php ορίζουμε τα endpoints που θα είναι ο μόνος τρόπος χειρισμού του API από εξωτερικές εφαρμογές. Χωρίζονται προαιρετικά σε groups κυρίως για την καλύτερη δόμηση και καθαριότητα του κώδικα. Εδώ επισυνάπτουμε σε κάθε route ξεχωριστά ένα ακόμα middleware που έχουμε δημιουργήσει σαν εξάρτηση στο container.Μας βολεύει αυτός ο τρόπος γιατί δεν αποφεύγουμε το middleware εύκολα σε διαδρομές που δεν το χρειάζονται αλλά και βοηθά στο testing της εφαρμογής λόγω εύκολης αφαίρεσής του από κάθε διαδρομή. Το middleware αυτό ελέγχει το JWT token για εγγυρότητα όταν ζητηθεί η διαδρομή στην οποία το έχουμε προσθέσει. Η κάθε διαδρομή αντιστοιχίζεται με μια μέθοδο της εφαρμογής μας. Η μέθοδος μπορεί να δεχτεί σαν όρισμα και δυναμικά μέρη της διαδρομής, ορισμένα από τον client κατά το request.

```php
$app->group('/users', function () {
    $jwtMiddle = $this->getContainer()->get('jwt');
    $this->post('/login', UserController::class . ':loginUser');
    $this->get('/login', UserController::class . ':loginGuest');
    $this->post('/register', UserController::class . ':registerUser');
    $this->delete('', UserController::class . ':deleteUser')->add($jwtMiddle);
    $this->post('/favorites', UserController::class . ':addFavorite')->add($jwtMiddle);
    $this->delete('/{userid}/favorites/{speechid}', UserController::class . ':removeFavorite')->add($jwtMiddle);
    $this->get('/{userid}/favorites', UserController::class . ':getFavorites')->add($jwtMiddle);
});

$app->group('/conference', function () {
    $jwtMiddle = $this->getContainer()->get('jwt');
    $this->get('',ConferenceController::class . ':getAllConf');
    $this->get('/{confid}/schedule', ConferenceController::class . ':getSchedule')->add($jwtMiddle);
    $this->get('/{confid}/favorites/{userid}', ConferenceController::class . ':getFavorites')->add($jwtMiddle);
    $this->get('/{confid}', ConferenceController::class . ':getConfById')->add($jwtMiddle);
    $this->get('/{confid}/schedule/{date}', ConferenceController::class . ':getScheduleByDate')->add($jwtMiddle);
    $this->get('/{confid}/speakers', ConferenceController::class . ':getSpeakersByConf')->add($jwtMiddle);
    $this->get('/{confid}/speeches/{speechid}', ConferenceController::class . ':getSpeechById')->add($jwtMiddle);
    $this->get('/{confid}/speakers/{speakerid}', ConferenceController::class . ':getSpeakerById')->add($jwtMiddle);
    $this->post('', ConferenceController::class . ':registerConf')->add($jwtMiddle);
    $this->delete('', ConferenceController::class . ':deleteConf')->add($jwtMiddle);
});
```

Η λέξη κλειδί $this αναφέρεται στο $app που δηλώσαμε στην αρχή, το οποίο είναι διαθέσιμο σε όλα τα αρχεία εντός του project. Όσον αφορά το middleware το προσθέτουμε σε κάθε διαδρομή που θέλουμε να είναι έγκυρη μόνο αν ο χρήστης είναι ταυτοποιημένος.

Η εντολή ::class ύστερα από το όνομα κλάσης(εδώ *ConferenceController* ή *UserController*) κάνει αναφορά στο πλήρες όνομα της κλάσης μαζί με το namespace στο οποίο βρίσκεται, στο οποίο επικολλούμε σαν γραμματοσειρά την μέθοδο την οποία ζητάμε.

* ***Namespacing*** στην PHP είναι η δυνατότητα δημιουργίας ενός εικονικού συστήματος αρχείων μέσα στο project μας κατά το οποίο θα αναφερόμαστε στα αρχεία και τις κλάσεις μας. Αυτό χρησιμεύει ιδιαίτερα κυρίως για την αποφυγή λαθών σε περίπτωση που μια δική μας κλάση είναι συνονόματη με κάποια του framework, ενός plugin, ή ακόμα και άλλη δική μας μέσα στο project. Η αναφορά στην κλάση τώρα γίνεται με το πλήρες namespace της,

παράδειγμα στο παρόν API στην κλάση *ConferenceController* πρέπει να αναφερόμαστε ως *App/Controllers/Conference/ConferenceController* οπότε αν υπήρχε άλλη κλάση ConferenceController σε άλλο σημείο της εφαρμογής το πλήρες όνομα αναφοράς θα ήταν διαφορετικό. Επειδή τα ονόματα των κλάσεων έτσι γίνονται πολύ μακριά, για την διευκόλυνσή μας μπορούμε να τους δώσουμε ψευδώνυμα σε κάθε αρχείο που τις χρησιμοποιούμε με την εντολή *use <namespace/class> as <alias>*.

Στο επόμενο δείγμα κώδικα δείχνουμε μια από τις μεθόδους που περιέχει το αρχείο ConferenceController.php το οποίο αναλαμβάνει τις κλήσεις στη βάση δεδομένων σχετικά με τα συνέδρια, την επεξεργασία των δεδομένων σε κατάλληλη μορφή και επιστροφή κατάλληλης απάντησης(HTTP Response) στον client. Η κλάση ConferenceController είναι επέκταση της abstract κλάσης Controller την οποία χρησιμοποιούμε μόνο για να εισάγουμε το dependency injection container(στο εξής ***DIC***) σε όλα μας τα controllers. Η πρακτική της εισαγωγής ολόκληρου του DIC συνήθως θεωρείται κακή πρακτική καθώς η κάθε κλάση χρειάζεται μόνο ένα πολύ μικρό μέρος από τις εξαρτήσεις που αυτό περιέχει και αυξάνεται η πιθανότητα λάθους. Στην περίπτωση του συγκεκριμένου API, λόγω του μικρού σχετικά μεγέθους του μας βολεύει περισσότερο η μέθοδος με την επέκταση κοινής κλάσης χωρίς να μας επηρεάζει αρνητικά. Σε διαφορετική περίπτωση θα χρειαζόταν να εισάγουμε μεθόδους ‘εργοστάσια’ για όλες μας τις κλάσεις μέσα στο DIC, οι οποίες θα δίνουν στον constructor των κλάσεων αναδρομικά από το ίδιο το DIC μόνο τις εξαρτήσεις που χρειάζονται.

```php
public function getSpeakerById($request, $response, $args)
{
    if ($stmt = $this->c->db->prepare(
        "SELECT speakers.speakerid,speaker_name,speaker_about,speaker_pic,
        GROUP_CONCAT(speeches.speechid) AS speechid,
        GROUP_CONCAT(speech_title) AS speech_title,
        GROUP_CONCAT(time_start) AS time_start,
        GROUP_CONCAT(time_end) AS time_end,
        GROUP_CONCAT(room_name) AS room_name
        FROM speakers JOIN speeches_speakers ON speeches_speakers.speakerid = speakers.speakerid
        JOIN speeches ON speeches_speakers.speechid = speeches.speechid
        JOIN rooms on rooms.roomid = speeches.roomid
        WHERE speeches_speakers.speakerid = ?"
    )) {

        $stmt->bind_param("i", $args['speakerid']);
        $stmt->execute();
        $result = $stmt->get_result();
        $stmt->close();
        if ($result->num_rows > 0) {
            $data = $result->fetch_all(MYSQLI_ASSOC)[0];
            $data['speeches'] = array();
            $speechids = explode(',', $data['speechid']);
            $speech_titles = explode(',', $data['speech_title']);
            $time_start_arr = explode(',', $data['time_start']);
            $time_end_arr = explode(',', $data['time_end']);
            $room_arr = explode(',', $data['room_name']);

            for ($i = 0; $i < sizeof($speechids, 0); $i++) {
                array_push(
                    $data['speeches'],
                    array(
                        'speechid' => $speechids[$i],
```

(συνέχεια στην επόμενη σελίδα)Η μέθοδος δέχεται σαν ορίσματα request και response interfaces και τις δυναμικές παραμέτρους από τη διαδρομή σε έναν πίνακα ονόματι args. Χρησιμοποιεί το αντικείμενο σύνδεσης MySqli από το DIC για να δημιουργήσει και να εκτελέσει ένα query στη βάση δεδομένων με αποτέλεσμα τις λεπτομέρειες και τα στοιχεία ενός ομιλητή συνεδρίου με βάση το speakerid από τις παραμέτρους. Στη συνέχεια εκτελούνται διεργασίες επεξεργασίας των αποτελεσμάτων και ανασυγκρότησής τους σε πίνακες κλειδιών-τιμών με σκοπό τη σωστή μετάφραση σε αντικείμενο JSON το οποίο θα αποσταλεί πίσω στον χρήστη. Κατά τη διάρκεια της διαδικασίας γίνονται έλεγχοι λαθών ώστε σε όλες τις περιπτώσεις να επιστραφεί επαρκής απάντηση.

```
                    'speech_title' => $speech_titles[$i],
                    'time_start' => $time_start_arr[$i],
                    'time_end' => $time_end_arr[$i],
                    'room_name' => $room_arr[$i]
                )
            );
        }
        unset($data['speechid']);
        unset($data['speech_title']);
        unset($data['time_start']);
        unset($data['time_end']);
        unset($data['room_name']);

        /* $result = call_user_func_array('array_merge_recursive', $result);
        $result = array_map('array_unique', $result);*/

        return $response->withJson([
            'Error' => false,
            'Data' => $data,
            'Message' => ''
        ],200);
    } else {
        return $response->withJson([
            'Error' => true,
            'Data' => null,
            'Message' => 'No speech found'
        ],404);
    }
} else {
    $error = $this->c->db->errno . ' ' . $this->c->db->error;
    return $response->withJson([
        'Error' => true,
        'Data' => $error,
```

Αντίστοιχες μέθοδοι συμπληρώνουν το αρχείο αυτό καθώς και το αρχείο UserController το οποίο εκτελεί αντίστοιχες εργασίες σχετικά όμως με τους χρήστες.

Στο επόμενο δείγμα κώδικα βλέπουμε από το UserController.php και το Auth.php την διαδικασία δημιουργίας λογαριασμού χρήστη, ταυτοποίησής του και σαν συνέπεια δημιουργίας νέου JSON Web Token. Επίσης γίνεται έλεγχος εγκυρότητας των στοιχείων χρήστη που απεστάλησαν, χρησιμοποιώντας την μηχανή Respect Validation.

```php
public function registerUser($request, $response)
{
    $validation = $this->c->validator->validate($request, [
        'user_email' => v::noWhitespace()->notEmpty()->email()
                        ->EmailAvailable($this->c->get('db')),
        'user_password' => v::noWhitespace()->notEmpty(),
        'fname' => v::notEmpty()->alpha(),
        'lname' => v::notEmpty()->alpha()
    ]);

    if ($validation->failed()) {
        return $response->withJson([
            'Error' => true,
            'Data' => $validation->getErrors(),
            'Message' => 'Validation failed'
        ], 422);
    }

    $data = $request->getParsedBody();
    $token = $this->c->auth->generateToken($data);
    $fname = $data['fname'];
    $lname = $data['lname'];
    $email = $data['user_email'];
    $password = password_hash($data['user_password'], PASSWORD_DEFAULT);

    if ($stmt = $this->c->db->prepare(
        "INSERT INTO users(fname, lname, user_email, user_password)
        VALUES (?,?,?,?)"
        )) {
        $stmt->bind_param("ssss", $fname, $lname, $email, $password);
        $stmt->execute();
        $stmt->close();
        $result = $this->getUserById($this->c->db->insert_id);
```

(συνέχεια στην επόμενη σελίδα)

Η μέθοδος validate λαμβάνει τα στοιχεία του χρήστη και για κάθε ένα από αυτά ορίζουμε και συνθήκες οι οποίες πρέπει να αληθεύουν, όπως για παράδειγμα να μην είναι άδειο, να μην περιέχει κενά κλπ. Εδώ γράψαμε και μία δική μας συνθήκη,την EmailAvailable η οποία ελέγχει αν υπάρχει ήδη εγγραφή με το email του χρήστη στη βάση δεδομένων. Αν οι συνθήκες είναι αληθείς αποθηκεύονται τα στοιχεία του χρήστη στη βάση και επιστρέφονται πίσω μαζί με ένα JWT.

Τον κωδικό τον γνωρίζει μόνο ο χρήστης ενώ στη βάση αποθηκεύεται μια κρυπτογράφησή του(με την μέθοδο της PHP password_hash) την οποία ελέγχουμε με τη μέθοδο password_verify με τον κωδικό που θα σταλεί σε περίπτωση που ζητηθεί ξανά ταυτοποίηση.

```
    if (!$result['Error']) {
        $token = $this->c->auth->generateToken($result['Data']);
        $result['Data']['token'] = $token;

        return $response->withJson([
            'Error' => true,
            'Data' => $result['Data'],
            'Message' => 'Welcome'
        ], 200);
    } else {
        return $result;
    }
} else {
    $error = $this->c->db->errno . ' ' . $this->c->db->error;
    return $response->withJson([
        'Error' => true,
        'Data' => $error,
        'Message' => 'Database error occured!'
    ], 500);
}
```

```
public function generateToken($userid = '') {
    $now = new DateTime();
    $future = new DateTime("now +3 hours");
    $payload = [
        "iat" => $now->getTimeStamp(),
        "exp" => $future->getTimeStamp(),
        //"jti" => base64_encode(random_bytes(16)),
        "sub" => $userid,
        'iss' => $this->c->get('settings')['app']['url'],
    ];
    $secret = $this->c->get('settings')['jwt']['secret'];
    $token = JWT::encode($payload, $secret, "HS256");
    return $token;
}
```

Το αποτέλεσμα αυτού του REST API επιτρέπει στην υβριδική εφαρμογή μας για κινητά τηλέφωνα να επικοινωνεί με τη βάση δεδομένων. Θα μπορούσε και οποιαδήποτε άλλη διαδικτυακή εφαρμογή να επωφεληθεί από το ίδιο API όμως στην συγκεκριμένη περίπτωση έχει σχεδιαστεί κυρίως για να καλύψει τις ανάγκες της υβριδικής εφαρμογής συνεδρίων μας.

5.3 Εφαρμογή Ionic

Σε αυτό το υποκεφάλαιο θα γίνει ανάλυση των αξιοσημείωτων μερών του κώδικα της εφαρμογής, της χρήσης των δυνατοτήτων του Ionic και του Angular framework, της χρήσης του REST API και της φιλοσοφίας που ακολουθήθηκε στη δομή και το σχεδιασμό.

Στον φάκελο υπάρχουν υποφάκελοι που περιέχουν τα πηγαία αρχεία των frameworks που χρησιμοποιούμε, βιβλιοθήκες και διάφορα plugins του node ή του cordova. Στον υποφάκελο src έγινε το μεγαλύτερο μέρος της εργασίας, μέσα στον οποίο φαίνονται:

- Ο φάκελος app που περιέχει όλες τις σελίδες και τα services που έχουμε δημιουργήσει.
- Ο assets στον οποία αποθηκεύουμε αρχεία όπως εικόνες που θα παρέχονται με την εφαρμογή.
- Ο φάκελος themes αλλά και τα υπόλοιπα αρχεία στον src περιέχουν διάφορες ρυθμίσεις σχετικά με την εφαρμογή.

Στο εξής επικεντρωνόμαστε στον φάκελο app οποίος αρχικά περιέχει τα αρχεία app.module.ts, app.component.ts ,app.html και app-routing.module.ts τα οποία αποτελούν τον πυρήνα της εφαρμογής και το σημείο εκκίνησής της. Σε αυτά τα αρχεία δηλώνουμε συνολικά components και services τα οποία θα είναι διαθέσιμα σε όλα τα αρχεία της εφαρμογής.

Οι σελίδες της εφαρμογής οι οποίες βρίσκονται στον φάκελο pages, έχουν δομή παρόμοια με την από πάνω και αποτελούνται συνήθως από τέσσερα ή πέντε αρχεία μέσα σε έναν φάκελο:

- .html αρχείο για τη δομή της διεπαφής.
- .scss αρχείο για την εμφάνιση και τις παραμέτρους της δομής.
- component.ts αρχείο για τη λογική και το χειρισμό δεδομένων της σελίδας.
- module.ts για τη δήλωση της σελίδας στην εφαρμογή αλλά και άλλων δυνατοτήτων για χρήση στη σελίδα.
- Ένα αρχείο τύπου .ts που έχει το ρόλο ενός Angular Resolver.

Ακολουθούν για αρχή μερικά σημεία του κώδικα των services που δημιουργήσαμε για την υποστήριξη των σελίδων και της επικοινωνίας με το api.

Πρώτο είναι το αρχείο api.service.ts, ένα typescript αρχείο που περιέχει την κλάση ApiService η οποία αναλαμβάνει την επικοινωνία της εφαρμογής με τον “έξω κόσμο”, στην περίπτωσή μας με το REST API μέσω των μεθόδων της. Όσες σελίδες ή και άλλα services χρειάζονται δεδομένα από το API τα ζητούν μέσω του ApiService. Πρόκειται δηλαδή για μια εξάρτηση πολλών άλλων κλάσεων που στην Angular δηλώνεται ως Injectable και μόλις

δημιουργηθεί στιγμιότυπο, αυτό μοιράζεται σε όλες τις κλάσεις που την εισάγουν στον constructor τους κατά τη δημιουργία.

Οι μέθοδοι του ApiService αναλαμβάνουν να καλούν τα endpoints του API με την κατάλληλη HTTP μέθοδο, να επεξεργάζονται εν μέρη το αποτέλεσμα και να το μετατρέπουν σε μορφή αναγνωρίσιμη σε εμάς και το πρόγραμμά μας μέσω των κλάσεων Adapters που έχουμε προγραμματίσει, στις οποίες γίνεται αναφορά σε επόμενες παραγράφους.

Στο figure 21 φαίνονται οι μέθοδοι για τη ζήτηση του προγράμματος και της λίστας των ομιλητών. Το αποτέλεσμα είναι στη μορφή ενός rxjs Observable που περιέχει, όπως το αποστείλαμε από το REST API, ένα αντικείμενο JSON. Αυτό το JSON αντικείμενο γνωρίζουμε μόνο νοητά τι πληροφορία έχει σύμφωνα με τον κατασκευαστή του REST API. Κατά τον προγραμματισμό της εφαρμογής όμως πρέπει να δώσουμε σημασία σε αυτά τα δεδομένα και αυτό το πετυχαίνει το ApiService στέλνοντας όλα τα δεδομένα σε μια κλάση μετατροπέα για να μετατρέψει αυτό που γνωρίζουμε νοητά σε αντικείμενα με συγκεκριμένες μεταβλητές και συγκεκριμένου τύπου που θα είναι γνωστά σε όλη την έκταση της εφαρμογής.

```
getSchedule():Observable<Schedule>{
  return this.http.get<Schedule>(`${environment.apiUrl}/conference/
  ${this.selectedConf.confid}/schedule`)
  .pipe(
    map((schedule: any) => {
      let output: Schedule = [];
      schedule.speeches.forEach(speech => {
        output.push(this.speechLiteAdapter.adapt(speech));
      })
      this.confData.schedule.next(output);
      return output;
    })
  )
}

getSpeakersByConf(): Observable<SpeakerLite[]> {
  return this.http.get<SpeakerLite[]>(`${environment.apiUrl}/conference/
  ${this.selectedConf.confid}/speakers`)
    .pipe(
      map(speakers => {
        let adapted: SpeakerLite[] = [];
        speakers.forEach(speaker => {
          adapted.push(this.speakerLiteAdapter.adapt(speaker));
        })
        return adapted;
      }));
}
```

Figure 21: ApiService δείγμα

Σημείωση για τα rxjs Observables

rxjs είναι μια βιβλιοθήκη που υποστηρίζει την φιλοσοφία προγραμματισμού Reactive, η οποία ασχολείται με ροές δεδομένων και διάδοση αλλαγών πληροφορίας. Το πιο βασικό συστατικό της είναι το Observable, ένα αντικείμενο που περιέχει 2 σειρές(pipe) δεδομένων, μία για τη μετάδοση σφάλματος και μία για τη μετάδοση αποτελέσματος. Στα observable μπορεί να γίνει εγγραφή(subscription) από άλλα μέρη του κώδικα τα οποία μέσω της εγγραφής παρακολουθούν συνεχώς και ασύγχρονα την ροή που εκπέμπει το Observable μέχρι αυτό να ολοκληρωθεί, ή να εκπέμψει σφάλμα. Στο ***figure 21*** *φαίνεται μεταχείριση των δεδομένων αυτών πριν μεταδοθούν στους subscribers.*

```
export class Speech {
    speakers?:SpeakerLite[] = [];
    constructor(
    public id?:number,
    public roomId?:number,
    public roomName?:string,
    public title?:string,
    public topic?:string,
```

Το figure 22 είναι η κλάση – μοντέλο για τα αντικείμενα ομιλίας που περιέχουν τις πληροφορίες των ομιλιών. Αυτά τα αντικείμενα χρησιμοποιούνται μόνο στη σελίδα λεπτομερειών ομιλίας και περιέχουν πίνακα από αντικείμενα SpeakerLite που όπως είδαμε στο προηγούμενο υποκεφάλαιο χρησιμοποιούνται για τη λίστα ομιλητών που συμμετέχουν στην ομιλία.

```
@Injectable({providedIn: 'root'})
export class SpeakerLiteAdapter implements Adapter<SpeakerLite>{

    adapt(speaker):SpeakerLite {
        let output = new SpeakerLite(
            speaker.speakerid,
            speaker.speaker_name,
            speaker.speaker_about,
            speaker.speaker_pic
        )
        if (output.pic === 'N/A') {
            output.pic = "/assets/img/placeholder_profile.jpg"
        }
        return output;
    }
}
```

Figure 23: SpeakerLite Adapter

Στο figure 23 η κλάση μετατροπέας η οποία παράγει αντικείμενα SpeakerLite από αντικείμενα JSON.

Χρησιμοποιήσαμε δύο μοντέλα για τους ομιλητές, το Speaker και το SpeakerLite, το ίδιο και με τις ομιλίες Speech και SpeechLite. Αυτό για την αποφυγή κυκλικής εξάρτησης

λόγω του σχεδιασμού της εφαρμογής μας όπου η κάθε ομιλία αναφέρεται στους ομιλητές της και αυτοί με τη σειρά τους στις ομιλίες στις οποίες συμμετέχουν. Τα μοντέλα SpeakerLite και SpeechLite δεν περιέχουν εξάρτηση και χρησιμοποιούνται στα σημεία όπου η αλληλοαναφορά δε χρειάζεται, όπως στις λίστες.

Η κατασκευή αυτών των μοντέλων εξυπηρετεί το σωστό έλεγχο της εφαρμογής κατά το compilation, την σωστή δομή του project για την κατανόηση και ευκολία στην επέκταση, αλλά και την δυνατότητα του προγράμματος επεξεργασίας κώδικα, στην περίπτωση μας το VScode της Microsoft, να μας δίνει πληροφορίες με mouse-over και να μας προειδοποιεί αυτόματα σε περίπτωση κακής χρήσης. Αυτοί είναι εξάλλου και βασικοί λόγοι για τους οποίους χρησιμοποιούμε την TypeScript.

Η κλάση UserData είναι μία υπηρεσία που αναλαμβάνει οτιδήποτε έχει να κάνει με δεδομένα του χρήστη. Οι σελίδες που αναφέρονται στην ταυτοποίηση, την είσοδο και έξοδο του χρήστη, την αποθήκευση αγαπημένων κλπ επικοινωνούν με την UserData. Λόγω αυτού του κλιμακωτού σχεδιασμού, είναι και η μόνη κλάση στην εφαρμογή μας που μπορεί να ζητήσει από την ApiService να καλέσει το API για δεδομένα που αφορούν το χρήστη. Εδώ γίνονται έλεγχοι για το αν ο χρήστης είναι επισκέπτης και πώς θα διαχειριστούν τα δεδομένα του.

Όσον αφορά τις σελίδες, θα αναφερθούμε σε μερικά χαρακτηριστικά παραδείγματα χρήσης των τεχνολογιών. Αρχικά, στο ***figure 24*** φαίνεται η λίστα αγαπημένων ομιλιών από τη σελίδα προγράμματος συνεδρίου στο αρχείο favorites.component.html .

```
<ion-list>
  <ion-item-group *ngFor="let speech of schedule" [hidden]="speech.hide">
    <ion-item-sliding [hidden]="speech.hide">
      <ion-item [routerLink]="['/app/tabs/favorites',speech.id]">
        <ion-label>
          <h2>{{speech?.title}}</h2>
          <p>
            {{speech?.startTime}} - {{speech?.endTime}} &mdash; {{speech?.roomName}}
          </p>
        </ion-label>
      </ion-item>
    </ion-item-sliding>
  </ion-item-group>
</ion-list>
```

Figure 24: Λίστα αγαπημένων ομιλιών HTML

Εδώ γίνεται χρήση έτοιμων components του ionic framework και αυτά είναι όλα όσα αρχίζουν με την λέξη ion. Για τη δημιουργία των στοιχείων της λίστας αναλαμβάνει το directive της Angular *ngFor το οποίο διαβάζει τον πίνακα schedule από το αντίστοιχο αρχείο typescript της σελίδας και για κάθε στοιχείο του, δηλαδή αντικείμενα του τύπου SpeechLite, δημιουργεί και από ένα στοιχείο ion-item στη λίστα. Το κάθε στοιχείο έχει πρόσβαση στα δεδομένα του SpeechLite για το οποίο και δημιουργήθηκε. Έτσι μπορούμε να γράψουμε τον κώδικα HTML για ένα μόνο στοιχείο ion-item και η angular δημιουργεί δυναμικά τα υπόλοιπα μέσω των δεδομένων που μας δίνει εν τέλει το API σχετικά με το κάθε συνέδριο. Το κάθε στοιχείο της λίστας είναι διαδραστικό και όταν ενεργοποιηθεί,μέσω του directive routerLink, μας μεταφέρει στη σελίδα λεπτομέρειας ομιλίας για το αντίστοιχο συνέδριο.

Το ερωτηματικό (?) στις μεταβλητές του template ειδοποιεί την σελίδα να μην εμφανίσει το στοιχείο έως ότου η πληροφορία γίνει διαθέσιμη. Αυτό χρειάζεται για περιπτώσεις που η πληροφορία προέρχεται από απομακρυσμένο υπολογιστή, όπως όταν ζητείται από το REST API. Λόγω της ασύγχρονης φύσης των Observables και Promises της javascript και της καθυστέρησης δικτύου η πληροφορία θα γίνει διαθέσιμη στη σελίδα αφού αυτή επιχειρήσει να την εμφανίσει. Αυτή όμως η τεχνική μπορεί να κάνει την σελίδα να εμφανιστεί σε κομμάτια και σπασμωδικά, κάτι που δεν προσφέρει καλή εμπειρία χρήσης.

Αυτό μας φέρνει στους Resolvers του Angular Framework. Resolver είναι ένα ακόμα αρχείο το οποίο ορίζουμε να ενεργοποιηθεί πριν τη σελίδα όταν αυτή ζητηθεί και να εκτελέσει διεργασίες μετά το πέρας των οποίων η σελίδα μπορεί να ενεργοποιηθεί. Σε περίπτωση που υπάρξει σφάλμα στον resolver, η μετάβαση στη σελίδα δεν εκτελείται. Resolvers χρησιμοποιούμε σε κάποιες σελίδες της εφαρμογής και τους αναθέτουμε να φορτώσουν δεδομένα που χρειάζεται η σελίδα από το API, παγώνοντας το χειρισμό της εφαρμογής και εμφανίζοντας ένα κινούμενο στοιχείο φόρτωσης με το LoadingController που προσφέρει το Ionic, μέχρι η επεξεργασία να ολοκληρωθεί.

Σαν παράδειγμα, όταν προσπαθήσει ο χρήστης να μεταβεί στη σελίδα των αγαπημένων, αυτό θα συμβεί μόνο εάν ο Resolver (***figure 25***) καταφέρει να δεχτεί το πρόγραμμα του συνεδρίου από το API και επιβεβαιώσει οτι υπάρχουν αγαπημένα του συγκεκριμένου χρήστη για το συνέδριο. Σε διαφορετική περίπτωση παραμένει στην προηγούμενη σελίδα, στην οποία εμφανίζει κατάλληλο μήνυμα σφάλματος. Επειδή ο resolver ενεργοποιείται αυστηρά όταν ζητείται η σελίδα και δεν έχει πρόσβαση στα δεδομένα της, χρησιμοποιούμε μια μεταβλητή-σημαία η οποία γίνεται αληθής την πρώτη φορά που θα λάβει επιτυχώς το πρόγραμμα. Όσο η σημαία είναι αληθής, θεωρούμε οτι η σελίδα αγαπημένων έχει ήδη αντίγραφο του προγράμματος και ο Resolver δεν ξαναζητά τα δεδομένα.

Όλες μας οι κλάσεις που εξαρτώνται από τη γνώση του επιλεγμένου συνεδρίου, όπως και ο resolver για την ζήτηση του σωστού προγράμματος, κάνουν subscribe σε ένα αντικείμενο ReplaySubject που διατηρεί πληροφορίες σχετικά με το επιλεγμένο συνέδριο. ReplaySubject είναι μια υποκατηγορία του Observable στο οποίο ορίζουμε έναν αριθμό εκπομπών του τις οποίες θα αποθηκεύσει(εδώ μία), τις οποίες και θα ξαναεκπέμπει σε κάθε νέο subscriber. Το αντικείμενο αυτό υπάρχει στην υπηρεσία ConfData και ανανεώνεται όταν

επιλεγεί νέο συνέδριο. Όταν ανανεωθεί η τιμή της όλοι subscribers γνωρίζουν την αλλαγή και δρουν αναλόγως. Οι resolvers παράδειγμα θέτουν τις σημαίες isLoaded σε αρνητικές ώστε να ζητήσουν το νέο πρόγραμμα.

```
resolve() {
    console.log("favorites resolver sees favorites", this.userData._favorites)
    if(!this.userData._favorites.length){
        this.util.presentToast("You have no favorites yet!");
        return this.router.navigate([`app/tabs/schedule`]);
    }
    if (!this.isLoaded) {
        this.util.presentLoading('Loading schedule...');
        return this.api.getSchedule().pipe(
            map(
                schedule => {
                    this.isLoaded = true;
                    schedule.forEach((speech: SpeechLite) => {
                        speech.startTime = speech.startTime.slice(0, -3);
                        speech.endTime = speech.endTime.slice(0, -3);
                    });
                  //  this.replay.next(schedule);
                    this.util.dismissLoading();
                    return schedule;
                },
                error => {
                    this.util.dismissLoading();
                    this.util.presentToast(error);
                    throwError(error);
                }
            )
        )
    }
}
```

Figure 25: Favorites Resolver

Για τη διαχείριση των JSON Web Tokens χρησιμοποιήσαμε μια μορφή middleware της Angular που ονομάζεται ***Interceptor(Figure 26).*** Ο interceptor είναι μία κλάση με μέθοδο intercept() η οποία ενεργοποιείται αυτόματα και αυστηρά κάθε φορά που γίνεται εμφανίζεται HTTP response ή HTTP request από και προς την εφαρμογή, διακόπτοντας την εκτέλεσή της μέχρι να αποφασίσει η μέθοδος μας. Με αυτό τον τρόπο ελέγχουμε κάθε κλήση της εφαρμογής αν αποτελεί endpoint του API μας με απλή σύγκριση με regular expression στο URI του HTTP request. Αν ισχύει τότε προσθέτουμε ένα επιπλέον Authorization Header το οποίο περιέχει το JWT που διατηρήσαμε κατά την ταυτοποίηση στο HTTP request και ύστερα το εκτελούμε. Στην περίπτωση απάντησης από το API εκτελούμε μια πρώτη

μεταχείριση των δεδομένων και διαχωρισμό κάποιων ιδιαίτερων κωδικών σφαλμάτων HTTP για την εμφάνιση κατάλληλων μηνυμάτων όπως για παράδειγμα το 401 που το API μας στέλνει μόνο σε περίπτωση που το JWT έχει λήξει, οπότε ο Interceptor αυτόματα αποσυνδέει το χρήστη και τον οδηγεί στη σελίδα σύνδεσης.

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

  //check if requests and responses come from the ConferenceApi
  if (this.isApiCall(request)) {
    //intercept requests when logged in to add token
    if (this.userService.isAuthenticated.value) {
      request = request.clone({ headers: request.headers
        .set('Authorization', `Bearer ${this.userService.user.token}`) })
    }
    return next.handle(request)
      .pipe(
        map((event: HttpEvent<any>) => {
          if (event instanceof HttpResponse) {
            const clone: HttpResponse<any> = event.clone({ body: event.body['Data'] })
            return clone;
          }
        }),
        catchError((error) => {
          let errorMessage: string = 'An unknown error has occured!';
          if (error instanceof HttpErrorResponse) {
            if(error.status===401){
              errorMessage = "Token has expired, please log in again!";
              this.userService.logout()
              .then(()=>this.router.navigate([``]))
            }
            else if (error.error) {
              errorMessage = error.error['Message'];
            }
          }
          else {
            if (error instanceof ErrorEvent) {
              errorMessage = "The app has encountered an error!"
            }
          }
          return throwError(errorMessage);
        })
      )
  }
  else { return next.handle(request) }
}
```

Figure 26: API interceptor

Στη συνέχεια μερικές γραμμές κώδικα της υπηρεσίας UserData από την μέθοδο ελέγχου ύπαρξης αγαπημένων στο επιλεγμένο συνέδριο, αλλά και τη μέθοδο αφαίρεσης αγαπημένου η οποία καλείται από τη σελίδα λεπτομερειών ομιλίας με το πάτημα του κουμπιού. Όπως σε αυτή τη μέθοδο, έτσι και όλες τις άλλες που διαχειρίζονται τα αγαπημένα γίνεται έλεγχος για το αν ο χρήστης είναι επισκέπτης και ενεργοποιείται μια από τις δύο υποπεριπτώσεις, η αποθήκευση στην μνήμη της συσκευής ή η κλήση του API για αποθήκευση στη βάση δεδομένων.

```
hasFavorites():Boolean {
   return this._favorites.some(
      (favorite:Favorite)=> Number(favorite.confid) === Number(this.selectedConfId))
}

removeFavorite(speechid: number): Observable<any> {
    const favorite:Favorite = {speechid: speechid, confid: this.selectedConfId, userid: this.user.id};
    const index = this._favorites.findIndex(favorite=>favorite.speechid == speechid);

    if (this.isAuthenticated.value && !this.user.guest) {
        return this.apiService.removeFavorite(favorite).pipe(
            tap(success => {
                if (index > -1) {
                    this._favorites.splice(index, 1);
                    this.favorites.next(this._favorites)
                }
            },
            error => {
                return throwError(error);
            })
        )
    }else if (index > -1) {
        this._favorites.splice(index, 1);
        this.favorites.next(this._favorites)
        return from(this.storage.set('favorites', this._favorites));
    }
}
```

Δημιουργούμε ένα αντικείμενο με βάση το απλό Interface που δημιουργήσαμε για τα αγαπημένα(Favorite) και αναζητούμε τη θέση του id της ομιλίας από την οποία ενεργοποιήθηκε η μέθοδος αφαίρεσης αγαπημένου μέσα στον πίνακα _favorites, όπου διατηρούμε όλα τα αγαπημένα, ώστε να μπορούμε να το αφαιρέσουμε από αυτόν.

Ιδιαίτερο ενδιαφέρον στην εφαρμογή έχει ο τρόπος με τον οποίο χειριζόμαστε την ιεραρχία των σελίδων μέσω του routing module του Angular Framework. Το routing ακολουθεί παρόμοια δομή με αυτή του REST API, προσφέροντας στην εφαρμογή endpoints τα οποία αντιστοιχίζονται με μία σελίδα το καθένα. Έχουμε ακολουθήσει την τεχνική του Lazy Loading δηλαδή η φόρτωση των modules και των component στα οποία συνεπάγονται μόνο όταν ενεργοποιηθεί. Σε διαφορετική περίπτωση όλα τα modules δημιουργούνται στην εκκίνηση της εφαρμογής πιθανώς προσθέτοντας μεγαλύτερο χρόνο φόρτωσης και

δεσμεύοντας μνήμη ενώ μπορεί να μην χρειαστούν ποτέ κατά τη διάρκεια χρήσης της εφαρμογής.

```
const routes: Routes = [
  { path: '', redirectTo: '/tabs/schedule', pathMatch: 'full' },
  {
    path: 'tabs', component: TabsPage, children: [
      {
        path: 'schedule', children: [
          { path: '', pathMatch:'full', resolve:{schedule:ScheduleResolver} ,loadChildren: '../schedule/schedule.module#ScheduleModule' },
          { path: ':speechid',resolve:{speechDetails:SpeechDetailsResolver}, loadChildren: '../speech-details/speech-details.module#SpeechD
        ]
      },
      {
        path: 'speakers', children: [
          { path: '',pathMatch : 'full', resolve: {speakers:SpeakerListResolver}, loadChildren: '../speakers/speakers.module#SpeakersModule
          { path: ':speakerid', resolve:{speaker:SpeakerDetailsResolver}, loadChildren: '../speaker-details/speaker-details.module#SpeakerD
        ]
      },
      {
        path: 'maps', children: [
          {
            path: '', loadChildren: '../maps/maps.module#MapsModule'
          }
        ]
      },
      {
        path: 'favorites', children: [
          { path: '', pathMatch:'full',resolve:{schedule: FavoritesResolver},loadChildren: '../favorites/favorites.module#FavoritesModule'
          { path: ':speechid',resolve:{speechDetails:SpeechDetailsResolver}, loadChildren: '../speech-details/speech-details.module#SpeechD
        ]
      },
    ]
  }
];
```

Figure 27

Σε κάθε route ορίζεται και ο resolver ο οποίος ενεργοποιείται πριν ολοκληρωθεί η μετάβαση προς το endpoint και τα δεδομένα που αποστέλλουμε μαζί με την εντολή μετάβασης μέσω του URI. Επειδή αυτές είναι υπο-διαδρομές των καρτελών μπορούμε να ανοίξουμε σε διαφορετικές καρτέλες τα ίδια modules όπως τις λεπτομέρειες ομιλίας στην καρτέλα προγράμματος και στην καρτέλα αγαπημένων.

Τέλος στα επόμενα δείγματα κώδικα(figure 28) φαίνονται το html και το typescript που χρησιμοποιήθηκαν για τη δημιουργία της φόρμας εγγραφής νέου χρήστη με τη χρήση των βιβλιοθηκών ReactiveForm του Angular. Η Angular προσφέρει δύο διαφορετικές βιβλιοθήκες για δημιουργία φορμών, ο ένας του τύπου template driven και ο άλλος reactive ή model driven. Η βιβλιοθήκη μας βοηθά στην σύνδεση διεπαφής και λογικής χειρισμού δεδομένων.

```
<form (ngSubmit)="register()" [formGroup]="registerForm">
    <ion-list inset lines="inset">
        <ion-item color="light">
            <ion-input #emailInput type="email" placeholder="Email" formControlName="email"></ion-
        </ion-item>
        <div *ngIf="email.invalid && !email.pristine && email?.errors['email']"
            class="ion-no-padding ion-no-margin">
            <ion-icon name="alert" item-start></ion-icon>
            <ion-text color="danger">Not a valid email</ion-text>
        </div>
        <ion-item color="light" class="ion-margin-top">
            <ion-input type="password" placeholder="Password" formControlName="password"></ion-inp
        </ion-item>
        <div *ngIf="password.invalid && !password.pristine && password?.errors['minlength']"
            class="ion-no-padding ion-no-margin">
            <ion-icon name="alert" item-start></ion-icon>
            <ion-text color="danger">Should be more than 6 characters long</ion-text>
        </div>
        <ion-item color="light" class="ion-margin-top">
            <ion-input placeholder="First Name" formControlName="fname"></ion-input>
        </ion-item>
        <div *ngIf="fname.invalid && !fname.pristine && fname?.errors['minlength']"
            class="ion-no-padding ion-no-margin">
            <ion-icon name="alert" item-start></ion-icon>
            <ion-text color="danger">Should be more than 3 characters long</ion-text>
        </div>
        <ion-item color="light" class="ion-margin-top">
            <ion-input placeholder="Last Name" formControlName="lname"></ion-input>
        </ion-item>
        <div *ngIf="lname.invalid && !lname.pristine && lname?.errors['minlength']"
            class="ion-no-padding ion-no-margin">
            <ion-icon name="alert" item-start></ion-icon>
            <ion-text color="danger">Should be more than 3 characters long</ion-text>
        </div>
    </ion-list>
    <ion-button [disabled]="!registerForm.valid" type="submit" expand="block">Sign Up</ion-button>
</form>
```

Figure 28: Φόρμα Εγγραφής

Μεγάλο μέρος του κώδικα που βλέπουμε αποτελείται από τα στοιχεία που εμφανίζονται σε περίπτωση σφάλματος και οι συνθήκες με το directive *ngIf κατά τις οποίες εμφανίζονται. Μέσω του directive formGroup και των attributes formControlName αναφερόμαστε στη φόρμα και τα στοιχεία της μέσα στον typescript κώδικα του component. Μέσω του directive disabled και των συνθηκών εγκυρότητας που ορίζουμε στον κώδικα, ελέγχουμε αν το κουμπί τύπου submit θα είναι διαδραστικό ή ‘γκριζαρισμένο’.

```
async register() {
    await this.util.presentLoading();
    let user = {
        "user_email": this.registerForm.value['email'],
        "user_password": this.registerForm.value['password'],
        "fname": this.registerForm.value['fname'],
        "lname": this.registerForm.value['lname']
    }
    this.userService.register(user).subscribe(
        data => {
            this.router.navigate([`landing`]);
        },
        async error => {
            console.log("Show alert with error: ", error);
            await this.util.dismissLoading();
            this.util.presentToast(error);
        },
        () => {
            this.util.dismissLoading();
        });
}


createForm() {
    this.registerForm = this.formBuilder.group({
        email: ['', [Validators.required, Validators.email]],
        password: ['', [Validators.required, Validators.minLength(6)]],
      //  username: ['', [Validators.required, Validators.minLength(3)]],
        fname: ['', [Validators.required, Validators.minLength(3)]],
        lname: ['', [Validators.required, Validators.minLength(3)]],
    });
}

get email() { return this.registerForm.get('email'); }
get password() { return this.registerForm.get('password'); }
get fname() { return this.registerForm.get('fname'); }
get lname() { return this.registerForm.get('lname'); }
```

6 Επίλογος

6.1 Συμπεράσματα

Ο κώδικας της εφαρμογής είναι αρκετά εκτεταμένος και υπάρχουν αρκετά σημεία γραμμένα κατά περίπτωση για διάφορες λειτουργίες και υποπεριπτώσεις χρήσης που προκύπτουν λόγω του σχεδιασμού για να αναφερθούμε σε κάθε γραμμή αυτού, όμως αυτή η εργασία περιέχει κατά τη γνώμη μου τα σημαντικά σημεία που αναδεικνύουν τη χρήση των τεχνολογιών που επιλέχθηκαν, οι οποίες στη στιγμή της συγγραφής αποτελούν τεχνολογίες αιχμής. Δεν είναι στο πεδίο της εργασίας τόσο η λεπτομερής περιγραφή της λειτουργίας των τεχνολογιών όσο η χρήση και επίδειξη των δυνατοτήτων τους σε μία εφαρμογή η οποία μπορεί να είναι χρήσιμη σε πραγματικές συνθήκες.

Όσον αφορά τον έλεγχο λειτουργίας του συστήματος, έχει γίνει σε αρκετά περιορισμένο αριθμό συσκευών, όμως σε αυτά δεν βρέθηκε κάποιο σφάλμα κατά τη διάρκεια της χρήσης. Για να μπορεί κανείς να χρησιμοποιήσει την εφαρμογή ακολουθεί στο τέλος αυτού του αρχείου παράρτημα με οδηγίες για την εκκίνησή της και τις απαιτήσεις συστήματος όσον αφορά εξαρτήσεις λογισμικού.

6.2 Επέκταση και μελλοντικά σχέδια

Η εφαρμογή και το API για συνέδρια είναι μια στέρεη βάση για ένα σύστημα το οποίο στο μέλλον έχει προοπτικές εμπορικής χρήσης. Δόθηκε αρκετή σημασία στην καθαρότητα του κώδικα και χρήσης ‘βέλτιστων πρακτικών’ ώστε η πλατφόρμα να είναι επεκτάσιμη, διότι το αποδεκτό επίπεδο ποιότητας και δυνατοτήτων των εφαρμογών έχει ανέβει αρκετά, καθώς και ο αριθμός των λειτουργιών που θεωρείται δεδομένο πλέον οτι περιέχουν.

Σε αυτή τη μορφή κυρίως η υβριδική εφαρμογή εξακολουθεί να υστερείται εργαλείων διαχείρισης χρήστη, εξατομίκευσης της εμφάνισης και δομής εκ μέρους του διοργανωτή, όπως επιλογή χρωμάτων, επιλογή φίλτρων και υποστήριξη διαφορετικών δομών προγράμματος συνεδρίου. Επίσης για τη μεριά του χρήστη θα μπορούσαν να υπάρχουν λειτουργίες όπως η δυνατότητα αποθήκευσης αγαπημένων συνεδρίων και εμφάνιση ειδοποιήσεων κάποιο χρονικό διάστημα πριν την εκτέλεση τους, καθώς και σήμανση, ταξινόμηση και φιλτράρισμα των συνεδρίων με βάση το πεδίου τους(βιολογία, ιατρική κλπ.) Τέλος έχει παραληφθεί η δυνατότητα χρήσης του αισθητήρα GPS για την εμφάνιση του χρήστη στο χάρτη και η ενσωμάτωση του API MapBox των χαρτών είναι αρκετά ελλιπής, ενώ αυτό προσφέρει τεράστιο εύρος δυνατοτήτων.

Βιβλιογραφία

Charland, A. and Leroux, B., 2011. Mobile application development: web vs. native. Communications of the ACM, 54(5), pp.49-53.

Flanagan, D., 2006. JavaScript: the definitive guide. " O'Reilly Media, Inc.".

Samra, J., 2015. Comparing performance of plain php and four of its popular frameworks.

https://en.wikipedia.org/wiki/Android_(operating_system)

https://en.wikipedia.org/wiki/IOSpparchitecture.png

https://cordova.apache.org/static/img/guide/cordovaapparchitecture.png

https://cordova.apache.org/docs/en/latest/guide/overview/

https://ionicframework.com/docs/intro

https://vsavkin.com/writing-angular-2-in-typescript-1fa77c78d8e8

https://www.sitepoint.com/practical-guide-angular-directives/

https://docs.angularjs.org/guide/

https://www.apachefriends.org/index.html

https://www.diva-portal.org/smash/get/diva2:846121/FULLTEXT01.pdf

https://dzone.com/articles/differences-in-performance-apis-amp-more

https://restfulapi.net/rest-architectural-constraints/

https://auth0.com/learn/json-web-tokens/

https://tools.ietf.org/html/rfc7519

https://packagist.org/packages/tuupola/slim-jwt-auth

https://www.joshmorony.com/building-mobile-apps-with-ionic-2/

https://github.com/tuupola/cors-middleware