

A Mobile Application Aiming To Provide Selected Content And Offline Experience to Students

Georgios Parasyris

Thesis submitted in partial fulfillment of the requirements for the
Masters' of Science degree in Computer Science and Engineering

University of Crete
School of Sciences and Engineering
Computer Science Department
Voutes University Campus, 700 13 Heraklion, Crete, Greece

Thesis Advisor: Prof. *Evangelos Markatos*

This work has been performed at the University of Crete, School of Sciences and Engineering, Computer Science Department.


The work has been supported by the Foundation for Research and Technology - Hellas (FORTH), Institute of Computer Science (ICS).

UNIVERSITY OF CRETE
COMPUTER SCIENCE DEPARTMENT


**A Mobile Application Aiming To Provide Selected Content And
Offline Experience to Students**

Thesis submitted by
Georgios Parasyris
in partial fulfillment of the requirements for the
Masters' of Science degree in Computer Science

THESIS APPROVAL

Author: 

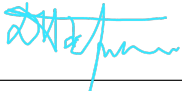
Georgios Parasyris

Committee approvals: 

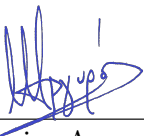
Evangelos Markatos
Professor, Thesis Supervisor



Yannis Tzitzikas
Professor, Committee Member



Dimitris Plexousakis
Professor, Committee Member

Departmental approval: 

Antonios Argyros
Professor, Director of Graduate Studies

Heraklion, March 2020

Abstract

There has been a significant change on the way users access content during the past decade, where a consistent increase on mobile device usage can be observed. The mobile device market share increased from almost 1% during 2009 [18] to 54.34% in 2019 [19]. Users nowadays have multiple channels to access information on the web since each one of them typically owns a number of devices.

In the meantime, it became apparent that device and OS tailored user experience and consistency is expected. Web and app development technologies progressed during this journey ending up providing a variety of tools to developers to accommodate the increasing need for mobile-friendly apps. While mobile apps evolved, users were found to spent significantly more time using them instead of accessing content via mobile browsers. Native device features revolutionised user experience in mobile devices, resulting in richer and more personalised interaction between users and their digital content.

This thesis proposes an end-to-end solution to deliver university-related content to students via a cross-platform mobile application. The solution includes a website scrapper that generates data sets and stores them in a publicly accessible folder within the Computer Science Department's ecosystem. The application's implementation includes interchangeable data providers, in order to both consume the generated data and account for future changes. The content delivered consists of areas that are likely to be accessed daily by a typical student. The work proposed in this thesis aims to create a new channel for students to access university content and provide to them a richer user experience.

After reviewing the plurality of development options and presenting the ones picked, the solution architecture details are provided. A high level description of the user experience per module follows, presenting the proposed mobile-friendly developed interfaces.

Εφαρμογή Για Κινητές Συσκευές Με Σκοπό Την Προσφορά Επιλεγμένου Περιεχομένου Και Εκτός-Σύνδεσης Εμπειρία Χρήστη σε Φοιτητές

Περίληψη

Κατά τη διάρκεια της τελευταίας δεκαετίας έχει αλλάξει σημαντικά ο τρόπος πρόσβασης των χρηστών όσον αφορά το περιεχόμενο στο διαδίκτυο, όπου παρατηρείται συνεχής αύξηση χρήσης κινητών συσκευών. Το μερίδιο αγοράς των κινητών συσκευών αυξήθηκε από σχεδόν 1% το [18]2009 σε 54,34% το 2019[19]. Οι χρήστες σήμερα έχουν στη διάθεσή τους πολλαπλά κανάλια πρόσβασης στο διαδίκτυο, μιας και συνήθως ο καθένας τους έχει στην κατοχή του έναν αριθμό συσκευών.

Παράλληλα, έγινε φανερό ότι η εξατομικευμένη εμπειρία και συνέπεια ανά συσκευή και λειτουργικό σύστημα είναι χαρακτηριστικά που αναμένονται από τους χρήστες. Οι τεχνολογίες ανάπτυξης ιστοσελίδων και εφαρμογών εξελίχθηκαν κατά τη διάρκεια αυτής της διαδρομής, καταλήγοντας να παρέχουν μια ποικιλία εργαλείων, συμβάλλοντας στην κάλυψη της αυξανόμενης ανάγκης για εφαρμογές φιλικές προς κινητές συσκευές. Ακολουθώντας την εξέλιξη των εφαρμογών για κινητά, παρατηρήθηκε ότι οι χρήστες αφιερώνουν πολύ περισσότερο χρόνο χρησιμοποιώντας εφαρμογές αντί των προγραμμάτων περιήγησης για να αποκτήσουν πρόσβαση στο περιεχόμενο. Τα native χαρακτηριστικά προκάλεσαν επανάσταση στην εμπειρία χρήσης σε κινητές συσκευές, με αποτέλεσμα την πιο πλούσια και πιο εξατομικευμένη αλληλεπίδραση μεταξύ χρηστών και του ψηφιακού τους περιεχομένου.

Η παρούσα εργασία προτείνει μια end-to-end λύση για την παροχή περιεχομένου σε φοιτητές μέσω εφαρμογής συμβατής με πολλαπλές πλατφόρμες. Η λύση περιλαμβάνει πρόγραμμα ανίχνευσης ιστοτόπου που εξάγει δεδομένα και τα αποθηκεύει σε έναν κοινόχρηστο φάκελο εντός του οικοσυστήματος του Τμήματος Επιστήμης Υπολογιστών. Η υλοποίηση της εφαρμογής περιλαμβάνει εναλλάξιμους παρόχους δεδομένων, προκειμένου να καταναλώσει τα παραγόμενα δεδομένα και να προβλέψει για μελλοντικές αλλαγές. Το περιεχόμενο που παραδίδεται αποτελείται από ενότητες που είναι πιθανό να προσεγγίζονται καθημερινά από έναν τυπικό φοιτητή. Η εργασία που προτείνεται σε αυτή τη διατριβή στοχεύει στην προσθήκη ενός νέου καναλιού πρόσβασης πανεπιστημιακού περιεχομένου και στη δημιουργία μιας πλουσιότερης εμπειρίας για τους χρήστες τους.

Μετά την ανασκόπηση των επιλογών όσον αφορά την ανάπτυξη εφαρμογών και την παρουσίαση εκείνων που επιλέχθηκαν, παρατίθεται η αρχιτεκτονική της προτεινόμενης λύσης. Ακολουθεί μια παρουσίαση σε υψηλό επίπεδο της εμπειρίας χρήστη ανά ενότητα, παρουσιάζοντας τις προτεινόμενες mobile-friendly διεπαφές.

Acknowledgements

First and foremost I would like to thank Prof. Evangelos Markatos for the continuous support and guidance I received through this journey.

Secondly I would like to thank Mrs Evaggelia Kosma and Prof. Georgios Georgakopoylos for their understanding and patience regarding the remote status during certain parts of my studies.

I have the utmost respect for the Computer Science Department of University of Crete, as a university and as an organisation, and I would like to state that I recognise and advertise the fact that I have been treated well during both my Undergraduate and Postgraduate studies along with being prepared for this competitive market.

Moreover, I would like to thank Marco Ottolini - Styloola CEO/CTO during my time at the company - and Nick Desjarnis - my manager in Wealth Dynamix - for never raising issues that would affect my studies.

A shout-out to Yannis Vardas and Stylianos Piperakis for never letting me get consumed by the business side, ensuring that there is (some) work-studies balance.

I would like to also thank Tilemahos Argiris for being a great designer and spending the time to advise on the UX aspect of the application as well as designing figures included in this Thesis.

Last but not least, I am more than grateful to my parents Evripidis and Lili and to my sister Vasia, they have done their very best to support me during this double-goal journey.

Contents

| | |
|---|------------|
| Table of Contents | i |
| List of Tables | iii |
| List of Figures | v |
| Listings | vii |
| 1 Introduction | 1 |
| 1.1 Overview - Identifying the gap | 1 |
| 1.2 Thesis structure | 2 |
| 2 Background | 5 |
| 2.1 Website crawler | 5 |
| 2.2 Mobile Applications | 5 |
| 2.2.1 App Types | 5 |
| 2.2.1.1 Native App | 5 |
| 2.2.1.2 Hybrid App | 6 |
| 2.2.1.3 Progressive Web App (PWA) | 7 |
| 2.2.2 Front-End Technology - Options | 7 |
| 3 System Modeling and Implementation Details | 9 |
| 3.1 Modules and Data Structures | 9 |
| 3.1.1 Announcements - News | 9 |
| 3.1.2 Contact Information | 10 |
| 3.1.3 Courses | 10 |
| 3.1.4 Documents | 11 |
| 3.1.5 Map | 11 |
| 3.1.6 Model Program | 11 |
| 3.1.7 People | 12 |
| 3.1.8 Schedule | 12 |
| 3.2 Server-side | 12 |
| 3.2.1 Python Website Crawler | 13 |
| 3.2.1.1 Main Function | 15 |

| | | |
|----------|------------------------------------|-----------|
| 3.2.1.2 | Crawl module overview | 16 |
| 3.2.2 | REST Test Server | 17 |
| 3.2.2.1 | Node.js Server Structure | 17 |
| 3.3 | Front-end | 18 |
| 3.3.1 | API Library | 19 |
| 3.3.1.1 | Module Classes | 20 |
| 3.3.1.2 | Data Providers | 21 |
| 3.3.1.3 | Data Services | 25 |
| 3.3.2 | Helper Services | 28 |
| 3.3.3 | Application Components | 29 |
| 4 | User Interface | 31 |
| 4.1 | Announcements - News | 31 |
| 4.2 | Contact Information | 33 |
| 4.3 | Courses | 34 |
| 4.4 | Documents | 35 |
| 4.5 | Map | 36 |
| 4.6 | Model Program | 37 |
| 4.7 | People | 38 |
| 4.8 | Schedule | 40 |
| 5 | Summary and Future work | 43 |
| 5.1 | Summary | 43 |
| 5.2 | Future work | 43 |
| | Bibliography | 45 |

List of Tables

| | | |
|-----|---|----|
| 3.1 | Announcement/News Data Structure | 9 |
| 3.2 | Contact Data Structure | 10 |
| 3.3 | Course Data Structure | 10 |
| 3.4 | Document Data Structure | 11 |
| 3.5 | Room Data Structure | 11 |
| 3.6 | Model Program Semester Data Structure | 11 |
| 3.7 | Person Data Structure | 12 |
| 3.8 | Schedule Data Structure | 12 |
| 3.9 | Configuration Model | 14 |

List of Figures

| | | |
|------|---------------------------------------|----|
| 2.1 | Native application flow [20] | 6 |
| 2.2 | Hybrid application flow [20] | 7 |
| 2.3 | Ionic Framework wrapping Angular | 8 |
| 3.1 | Web scrapping process overview | 13 |
| 3.2 | API library overview | 20 |
| 3.3 | File structure example | 29 |
| 3.4 | Data preparation for display purposes | 30 |
| 4.1 | Home page - Announcements tab | 32 |
| 4.2 | Home page - News tab | 32 |
| 4.3 | Contact List page - Search | 33 |
| 4.4 | Course List page - Search | 34 |
| 4.5 | Course List page - Details | 34 |
| 4.6 | Document List page - Search | 35 |
| 4.7 | Map page | 36 |
| 4.8 | Model Program page | 37 |
| 4.9 | People page - Details | 38 |
| 4.10 | People page - Filters | 39 |
| 4.11 | Schedule page - Week swipe | 40 |
| 4.12 | Schedule page - Search | 41 |
| 4.13 | Schedule page - Landscape mode | 41 |

Listings

| | | |
|------|--|----|
| 3.1 | Configuration file example | 14 |
| 3.2 | Version file example | 15 |
| 3.3 | Python - main function | 15 |
| 3.4 | Python - crawl module | 16 |
| 3.5 | Node.js - app.js | 17 |
| 3.6 | Node.js - Mock routes | 17 |
| 3.7 | Node.js - Mock functions | 18 |
| 3.8 | App configuration example | 19 |
| 3.9 | Front-end API - BaseData class | 20 |
| 3.10 | Front-end API - ISerializer interface | 20 |
| 3.11 | Front-end API - Document class | 21 |
| 3.12 | Front-end API - StaticProvider class | 22 |
| 3.13 | Front-end API - RESTProvider class | 24 |
| 3.14 | Front-end API - DataService class | 26 |
| 3.15 | Front-end API - Module service class example | 28 |
| 3.16 | Fron-tend API - CallHandler Service class | 28 |

Chapter 1

Introduction

Over the past few years, users have been found to spend more and more time on their mobile devices using apps instead of accessing websites. This increasing usage is being leveraged by organisations to keep their audience better engaged. Following the same paradigm, top universities utilize mobile apps as a means for their students to access university-related user-tailored content. This work aims to bridge this gap regarding Computer Science Department of University of Crete, to provide a new way to access educational content while leveraging mobile devices' native capabilities. The next section will describe the problem and then this chapter concludes with the thesis structure.

1.1 Overview - Identifying the gap

Nowadays there are multiple ways to deliver content to end-users, the most common one being websites. Web development has greatly progressed from static websites to fully Responsive Web Design (RWD), initially using CCS3 media queries and later utilizing libraries that have taken RWD to the next level (e.g. Bootstrap [3]). This has given the developers the means to design for a range of devices, bringing information to desktop computers, tablets and mobile phones through the same code-base.

On the other hand, users have been observed to be turning away from the browser, spending almost 86% of their time on mobile apps while using mobile devices, a number that rises almost 14% per year [26]. This has lead the industry to separate the purpose of each channel, using websites to increase the marketing share and the target audience, while apps are being used to offer a different user experience (UX) to existing users. Usually a subset of content and services is available through mobile apps, enriched with the native feel and look and the convenience of offline data access."Well-designed apps aren't just small desktop apps or sites, they focus on mobile scenarios and take full advantage of the mobile device capabilities. As a result, they may provide only a subset of the full desktop or Website functionality, or even completely different functionality" [24].

Mobile apps present a number of advantages over responsive websites:

- mobile-friendly gestures that are considered a given nowadays in mobile user experience, "swipe" and "long-press" being the key ones,
- native capabilities, where offline storage, geo-location and push notifications being the most important ones,
- the convenience of the content being "one-button-away" either in the app drawer or on the home screen of the mobile device, which bundled with offline-first approach creates the illusion of instant access to the desired content.

The question "Responsive web app vs mobile app" is a common one. Often a decision has to be made to pick one of the two where both is not an option. This popular dilemma has been investigated and the most popular mobile app disadvantage is the cost, the amount of investment required to develop an app [12] [21] [23] [27] [28].

Bringing the above to the Computer Science Department of University of Crete ecosystem, the work in this thesis aims to serve a subset of the content through a mobile app to the students. The areas that have been selected for this purpose are (i) announcements and news, (ii) general purpose contact information, (iii) courses, (iv) documents, (v) map-related information, (vi) model program, (vii) people and their contact information - where available - and (viii) current semester's schedule. These areas are to be referred to as modules in Chapter 3.

The last part of the problem resides on the data source side. Publicly available REST APIs do not exist and a constraint regarding this work was to not add to the maintenance responsibilities by deploying another server. A python crawler has been developed in order to overcome this obstacle and deliver the content to the app.

1.2 Thesis structure

The rest of the thesis is organized in four (4) sections as indicated in the table of contents:

- The second section will provide information related to the technologies this implementation is based on.
- The third section will dive into the technical implementation of the system end-to-end. It includes details related to the architecture of the three distinct components of the solution: (i) the python crawler, (ii) the front-end data API and (iii) the app implementation.

- The fourth section will present a high level description of the solution, demonstrating the algorithm to feed data to the app along with providing data and user interface(UI) details for each module included.
- Finally, the fifth section will summarize this thesis and discuss future work directions.

Chapter 2

Background

In this section, the key technologies utilized for the proposed implementation are presented.

2.1 Website crawler

As stated in section 1.1, a low-maintenance data source was included in the requirements of this solution. Hence, a website crawler was implemented.

The language of choice was Python[9] due to the range of libraries available for web scraping. Three libraries were considered regarding the web crawling functionality: (i)Scrapy [13], (ii) Selenium[14] and (iii)BeautifulSoup[8]. All of them are open-source projects.

The last one, BeautifulSoup, was picked for this solution. The library is being imported locally, along with all the python-related external dependencies, all of them being part of the submitted repository. The crawler aspect of the solution has been tested in the environment it will run in, the department's Unix machines, and has been found to be fully compatible. More details are presented in section 3.2.

2.2 Mobile Applications

This is an era of smartphones and mobile apps, as thousands of apps are launched in app stores daily [22]. But they are not all built the same way, there are at least 3 different types of apps, which are listed in section 2.2.1.

2.2.1 App Types

2.2.1.1 Native App

Native apps are developed targeting a single mobile operating system (OS) exclusively, thus they are native for that particular platform or device [7].

Main advantages of native apps is the high performance and the consistent user experience since developers use native UI. They are usually accessible through the equivalent app store.

The main disadvantage regarding native apps is the cost. Due to the need of developing apps for each OS, organisations end up facing almost doubled development effort, maintenance responsibilities and price.

An example of a native application flow can be seen in Figure 2.1:

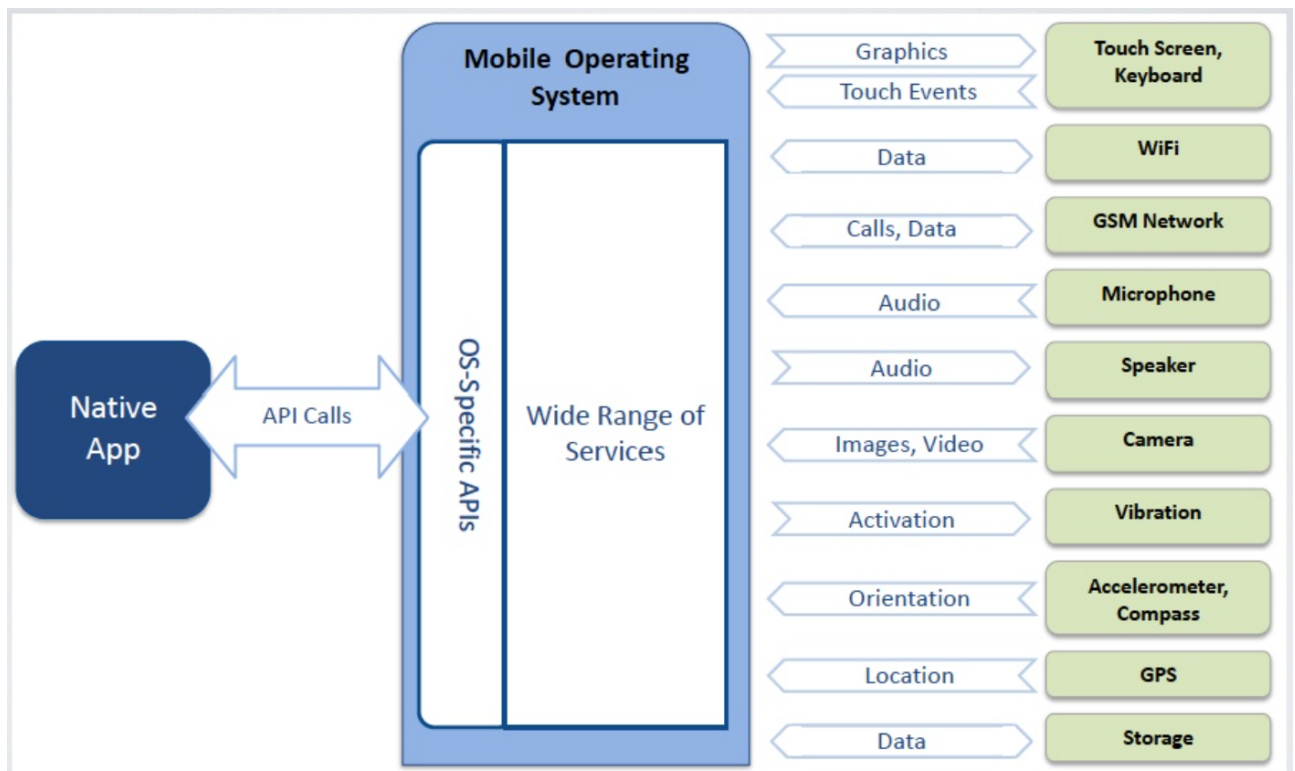


Figure 2.1: Native application flow [20]

2.2.1.2 Hybrid App

Hybrid apps are built using web technologies, with HTML5, CSS and Javascript playing a crucial role. This itself increases the development options significantly, since there are a number of Single Page Application (SPA) frameworks available, most popular of which are the following: (i) Angular [1], (ii) React [11], (iii) Ember [4] and (iv) Vue [16].

If a Hybrid is based on an SPA, usually Apache Cordova [2] is being used to provide access to native functionality.

An example of a hybrid application flow utilizing Cordova can be seen in Figure 2.2:

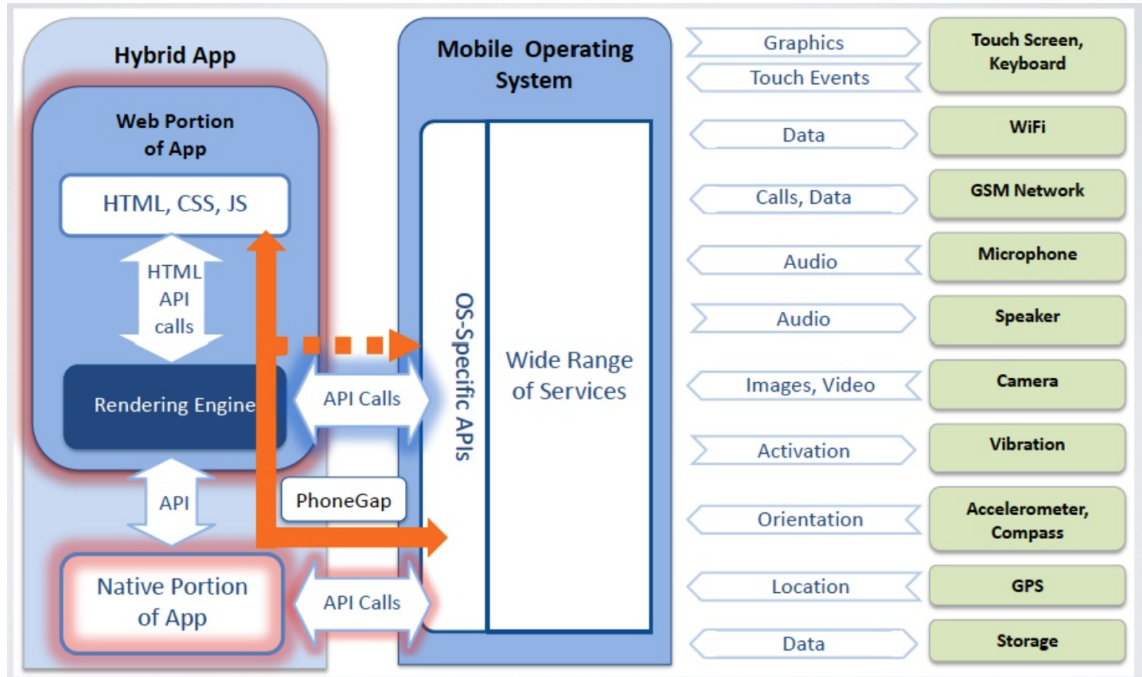


Figure 2.2: Hybrid application flow [20]

Apart from Cordova, Ionic[5], React-Native[10] and Xamarin[17] frameworks can be used for developing a hybrid app, all of them being open-source projects and providing cross-platform capabilities, meaning the app can target multiple operating systems while having one code-base.

2.2.1.3 Progressive Web App (PWA)

A PWA is positioned between a common web application and a hybrid app. Following the typical website paradigm, PWAs are hosted on a server and distributed using URLs instead of app stores.

They utilize Service Workers in order to provide offline user experience. They meet their limitation on the native part, where access is only permitted to device features supported by modern web browsers e.g. camera, audio recording and video capture.[25].

2.2.2 Front-End Technology - Options

Following the presentation of the different mobile app types, this section will present the choices that were made regarding the proposed solution.

Due to the app not being demanding in terms of performance, since it resembles more a store application following the list-details example per module rather than a graphic intensive game, the proposed app for this thesis is a hybrid one.

The cross-platform framework of choice is Ionic which, as stated above, is an open-source project. Ionic wraps Apache Cordova on the native side and provides flexibility on the framework the SPA is developed in, where the choice was the latest version of the Angular Framework. Angular was introduced initially in 2010 as AngularJS and was completely refactored to a different framework in 2016 which is usually referred to as Angular2+. Applications are developed in both versions today, which adds to the credibility and the lifespan of the framework.

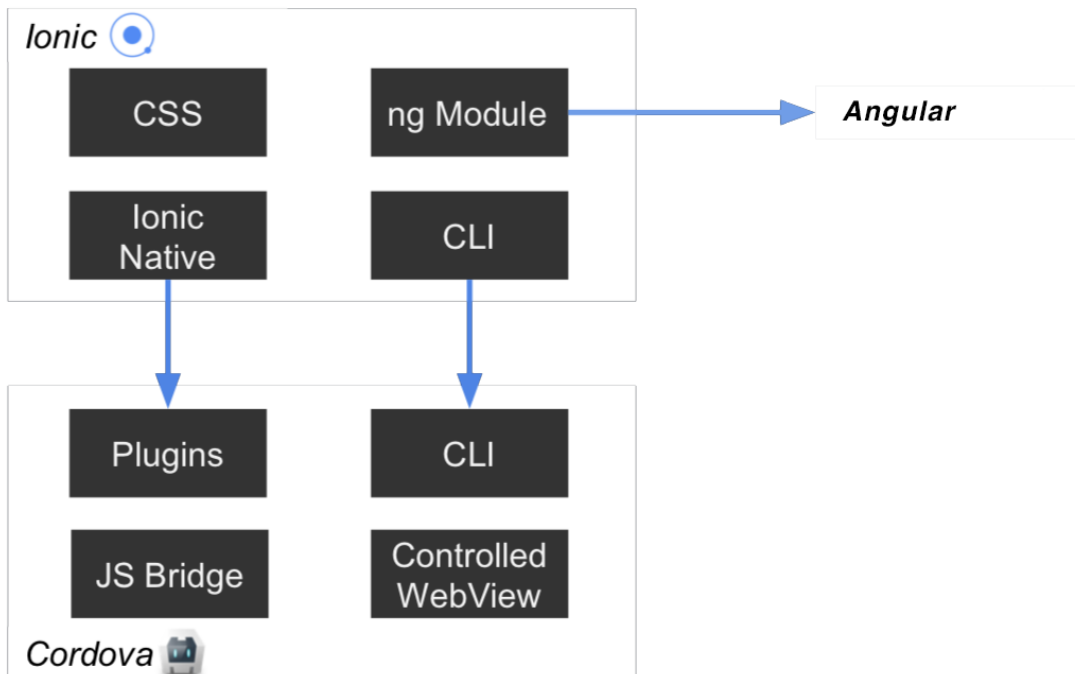


Figure 2.3: Ionic Framework wrapping Angular

Chapter 3

System Modeling and Implementation Details

This chapter provides details regarding the system implementation. Firstly, data models per module are presented, followed by the python crawler description. In Section 3.3 the front-end side is presented, providing details on the API Library implementation, the helper services and the application components developed within this proposal.

3.1 Modules and Data Structures

3.1.1 Announcements - News

Announcement and news entities share the very same structure. An announcement or a news entity can have its state set to "pinned" or "important", otherwise the item is considered to be part of the feed with state set to "stream".

| Field | Description |
|-------------------------------------|------------------------------------|
| entityId: <code>string</code> | Id defined in the HTML element |
| title_gr: <code>string</code> | Title in Greek |
| title_en: <code>string</code> | Title in English |
| description_gr: <code>string</code> | Description in Greek |
| description_en: <code>string</code> | Description in English |
| date: <code>string</code> | Date posted |
| url: <code>string</code> | Unique announcement/news URL |
| state: <code>enum</code> | Pinnied, important or stream state |

Table 3.1: Announcement/News Data Structure

3.1.2 Contact Information

The module in this section includes the department's general contact information.

| Field | Description |
|-------------------------------|--------------------------------|
| entityId: <i>string</i> | Id defined in the HTML element |
| title_gr: <i>string</i> | Title in Greek |
| title_en: <i>string</i> | Title in English |
| description_gr: <i>string</i> | Description in Greek |
| description_en: <i>string</i> | Description in English |
| date: <i>string</i> | Date posted |
| url: <i>string</i> | Unique announcement/news URL |

Table 3.2: Contact Data Structure

3.1.3 Courses

Courses module includes both undergraduate and postgraduate courses.

| Field | Description |
|----------------------------------|---------------------------------|
| area_code_gr: <i>string</i> | Course area code in Greek |
| area_code_en: <i>string</i> | Course area code in English |
| area_name_gr: <i>string</i> | Course name code in Greek |
| area_name_en: <i>string</i> | Course area code in English |
| code_gr: <i>string</i> | Course code in Greek |
| code_en: <i>string</i> | Course code in English |
| name_gr: <i>string</i> | Course name in Greek |
| name_en: <i>string</i> | Course name in English |
| program_gr: <i>string</i> | Course program in Greek |
| program_en: <i>string</i> | Course program in English |
| description_gr: <i>string</i> | Course Description in Greek |
| description_en: <i>string</i> | Course Description in English |
| prerequisites_gr: <i>complex</i> | Prerequisite courses in Greek |
| prerequisites_en: <i>complex</i> | Prerequisite courses in English |
| suggested_gr: <i>complex</i> | Suggested courses in Greek |
| suggested_en: <i>complex</i> | Suggested courses in English |
| url: <i>string</i> | Unique course URL |
| ects: <i>string</i> | ECTS assigned to the course |
| email: <i>complex[]</i> | Emails related to the course |

Table 3.3: Course Data Structure

3.1.4 Documents

The document collection consists of the items listed in "https://www.csd.uoc.gr/index.jsp?content=secretariat_services#Entypa" website - there is no English version available although the groundwork for it already exists. Each document can have different source types, "WORD" and "PDF" being the common ones.

| Field | Description |
|----------------------------|---|
| label_gr: string | Document label in Greek |
| label_en: string | Document label in English |
| sources_gr: complex | Crawled sources in the Greek version of the website |
| sources_en: complex | Crawled sources in the English version of the website |

Table 3.4: Document Data Structure

3.1.5 Map

Map module has been added in order to provide info related to the rooms in the building of the Computer Science Department. This collection is not the result of website scrapping as it consists of a room list and each room's map coordinates.

| Field | Description |
|----------------------------|---|
| title_gr: string | Room title in Greek |
| title_en: string | Room title in English |
| aliases: string [] | Other names the room is known as |
| floor: number | The floor the room is present at |
| coords: number [] | Room's exact latitude and longitude coordinates |

Table 3.5: Room Data Structure

3.1.6 Model Program

Model program collection includes the suggested route for undergraduates to complete the program in eight (8) semesters.

| Field | Description |
|--------------------------------|--|
| title_gr: string | Semester title in Greek |
| title_en: string | Semester title in English |
| courses_gr: complex [] | Greek variation of courses included in the semester. |
| courses_en: complex [] | English variation of courses included in the semester. |

Table 3.6: Model Program Semester Data Structure

3.1.7 People

”People” module pulls data from different categories, (i)academic staff, (ii)administrative personnel, (iii)emeriti faculty, (iv)honorary doctors, (v)lab personnel, (vi)special teaching staff and (vii)visiting instructor member are aggregated into one collection. The categories drive the ”position_en” and ”position_gr” fields which can then be used as filters in equivalent page, as shown in section 4.7.

| Field | Description |
|-------------------------------|--------------------------------|
| ID: <i>string</i> | Id defined in the HTML element |
| name_gr: <i>string</i> | Name in Greek |
| name_en: <i>string</i> | Name in English |
| description_gr: <i>string</i> | Description in Greek |
| description_en: <i>string</i> | Description in English |
| position_gr: <i>string</i> | Person position in Greek |
| position_en: <i>string</i> | Person position in English |
| email: <i>string</i> | Person’s email |
| img?: <i>string</i> | Person image |
| url?: <i>string</i> | Person’s website |

Table 3.7: Person Data Structure

3.1.8 Schedule

Schedule collection provides the weekly schedule for the current semester.

| Field | Description |
|-----------------------------|----------------------------|
| code_gr: <i>string</i> | Course code in Greek |
| code_en: <i>string</i> | Course code in English |
| name_gr: <i>string</i> | Course name in Greek |
| name_en: <i>string</i> | Course name in English |
| schedule_gr: <i>complex</i> | Weekly schedule in Greek |
| schedule_en: <i>complex</i> | Weekly schedule in English |

Table 3.8: Schedule Data Structure

3.2 Server-side

As mentioned in Introduction - Defining the problem, the requirements included the server-side to not require maintenance.

This resulted into a python crawler to be developed and tailored towards the current Computer Science Department website content, both in Greek and English, as described in Section 3.2.1. Aiming to accommodate future changes regarding

data sources, a REST API test server is presented in section 3.2.2, responsible for serving demo data to the front-end, demonstrating that the solution is flexible enough to consume data from different data sources.

3.2.1 Python Website Crawler

The python crawler is developed to be run within the Computer Science infrastructure, hence a dedicated account has been created for that purpose.

The crawling process, an overview of which can be presented in Figure 3.1, gets triggered by a cron job on a predetermined timeslot each day.

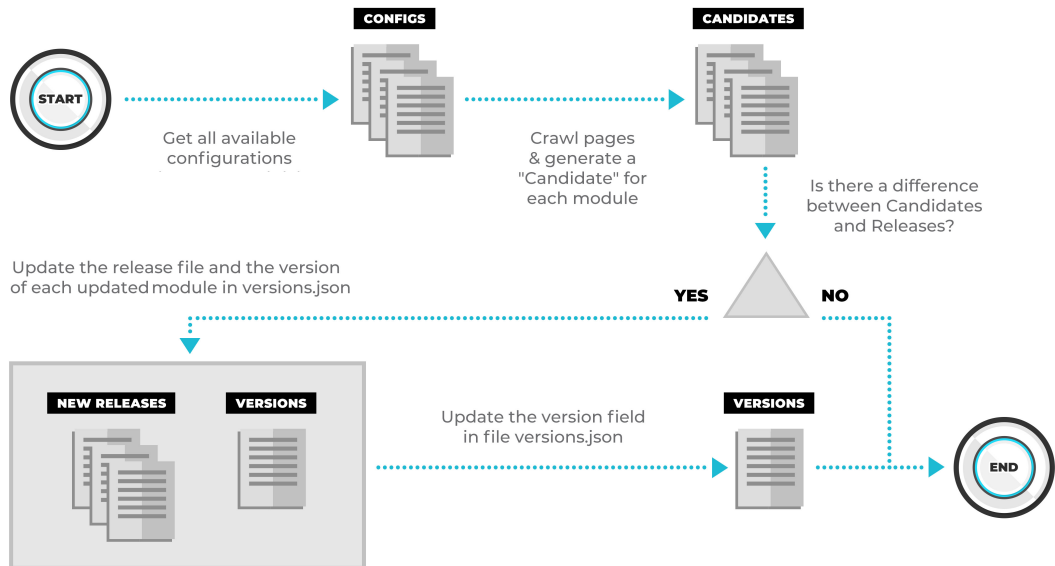


Figure 3.1: Web scrapping process overview

1. Initially all available configuration files are gathered, the structure of which can be seen in Listing 3.1.
2. Following each configuration, the python crawler gets data for every language present in the "url" field.
3. Data gathered in step 2 are being appended to the file determined as "outputFileName" which is considered a new dataset candidate. This leads to the option of different configurations updating the same file e.g. the people collection is the target of a number of configuration files.

4. Dataset candidates are being compared to the equivalent release versions (releases are candidates from a previous crawl invocation). In case they are found not to be identical, the candidate file gets moved to the release set and the module's version gets updated.
 - ⇒ There is one candidate per module.
 - ⇒ Each release is attached to a version, which is the current module version.

5. In case at least one module is updated, the version of the version.json is updated.

| Field | Description |
|--------------------------------------|--|
| module: <code>string</code> | This field identifies the module the configuration belongs to. Multiple configurations can be included in the same module. |
| url: <code>string</code> | Key-value collection where available languages are the keys and the target URLs that need to be crawled are the values |
| outputFileName: <code>string</code> | The file where the result of crawling based on the current configuration has to be appended. |
| type: <code>string</code> - optional | This field identifies the sub-module type. |

Table 3.9: Configuration Model

Configuration file example:

```

1 {
2   "module": "people",
3   "type": "administrative_personnel",
4   "url": {
5     "gr": "https://www.csd.uoc.gr/CSD/index.jsp?
           content=administrative_personnel&openmenu=demoAcc2&lang=g
           ",
6     "en": "https://www.csd.uoc.gr/CSD/index.jsp?
           content=administrative_personnel&openmenu=demoAcc2&
           lang=en"
7   },
8   "outputFileName": "people.json"
9 }

```

Listing 3.1: Configuration file example

Versions File example:

```

1 {
2     "_v": "0.0.7",
3     "modules": {
4         "announcements": "0.0.1",
5         "contacts": "0.0.2",
6         "courses": "0.0.2",
7         "documents": "0.0.1",
8         "model-program": "0.0.2",
9         "news": "0.0.1",
10        "people": "0.0.4",
11        "schedule": "0.0.2"
12    }
13 }

```

Listing 3.2: Version file example

3.2.1.1 Main Function

```

1 configs = os.listdir(DEFAULT_CONFIG_PATH)
2 newKeys = {}
3 for config in configs:
4     configOptions = json_load_byteified(open(DEFAULT_CONFIG_PATH
5         + config))
6     moduleName = configOptions['module']
7     sourceFile = DEFAULT_CANDIDATE_PATH + configOptions['module']
8         + '.json'
9     if shouldUpdateRelease(DEFAULT_CANDIDATE_PATH + configOptions
10        ['outputFileName'], DEFAULT_RELEASE_PATH + configOptions['
11        outputFileName']):
12        currentVersion = None
13        try:
14            currentVersion = increment_ver(
15                versions['modules'][configOptions['module']])
16        except (AttributeError, KeyError):
17            currentVersion = "0.0.1"
18        if os.path.isfile(sourceFile) and os.stat(sourceFile).
19            st_size != 0:
20            # print('copying...')
21            shutil.move(os.path.join(sourceFile),
22                os.path.join(DEFAULT_RELEASE_PATH +
23                    configOptions['module'] + '.json'))
24            newKeys[moduleName] = currentVersion
25 if(len(newKeys.keys()) > 0):
26     versions['_v'] = '0.0.1' if '_v' not in versions else
27         increment_ver(
28             versions['_v'])
29     for key in newKeys:
30         versions['modules'][key] = newKeys[key]
31     with open(DEFAULT_VERSION_PATH, 'w') as f:
32         json.dump(versions, f, indent=4, ensure_ascii=False,
33             sort_keys=True)

```

Listing 3.3: Python - main function

3.2.1.2 Crawl module overview

In this section an overview of the crawl module is provided. As mentioned in Section 3.2.1, the configuration files hold a set of URLs - one per language. After retrieving the HTML response for a language version, then - based on the module provided by the configuration - a module-tailored process prepares the data that end up in the pre-defined output file.

```

1 def crawlModule(configPath):
2     with open(configPath) as json_data_file:
3         config = json_load_byteified(json_data_file)
4         module = config['module']
5         outputFile = config['output']
6         retdata = []
7         firstDone = False
8         for lang in config['url']:
9             html = config['url'][lang]
10            r = requests.get(html)
11            c = r.content.decode('utf-8').replace(...).encode('
                utf-8')
12            parser = config['parser'] if 'parser' in config else
                'html.parser'
13            soup = BeautifulSoup(c, parser)
14            if module == "people":
15                ...
16            if module == "schedule":
17                ...
18            if module == "contacts":
19                ...
20            if module == "model-program":
21                ...
22            if module == "courses":
23                ...
24            if module == "documents":
25                ...
26            if module in ['announcements', 'news']
27                ...
28            # check if file exists, read it
29            if os.path.isfile(outputFile) and os.stat(outputFile).st_size
                != 0:
30                olddata = json_load_byteified(open(outputFile))
31                retdata = retdata + olddata
32            with open(outputFile, 'w') as f:
33                json.dump(retdata, f, indent=4, ensure_ascii=False,
                    sort_keys=True)

```

Listing 3.4: Python - crawl module

3.2.2 REST Test Server

A Node.JS server has been included in the implementation to ensure the flexibility and the abstraction of the data. This has been developed while believing that in the future, when REST APIs are available, a large-scale refactoring would not be required.

The server has been created through utilizing a generator library, Sventech Node.js generator[15].

3.2.2.1 Node.js Server Structure

Server.js

```
1 var express = require('express'),
2   mongoose = require('mongoose'),
3   app = express(),
4   cors = require('cors'),
5   { router } = require('./api/utilities'),
6   responseHanlder = require('./api/middleware/responseHandler');
7
8 /* setting port */
9 var port = process.env.PORT || 5000;
10 // Initialize the modules
11 const modules = require('./api/modules');
12 mongoose.connect('mongodb://localhost/demo', { useNewUrlParser:
13   true }); // connect to our database
14 app.use(cors());
15 app.use(router);
16 app.use(responseHanlder);
17
18
19 // port attached
20 app.listen(port);
21 console.log('Node JS server listening to port ' + port);
```

Listing 3.5: Node.js - app.js

Mock Routes

```
1 /**
2  * ROUTES FOR MODULE: MOCKS
3  */
4 const { router } = require('../utilities');
5 const mockFunctions = require('./functions');
6
7 ...
8
9 router.post('/mock', createMock);
10 router.get('/mock/:collection/:id', retrieveMock);
11 router.get('/mock/:collection', retrieveMock);
12 router.put('/mock/:mockId', updateMock);
```

```
13 router.delete('/mock/:mockId', deleteMock);
```

Listing 3.6: Node.js - Mock routes

Mock Module Functions

```
1  /**
2   * FUNCTIONS FOR MODULE: MOCKS
3   */
4  const create = (req, res, next) => {
5    ...
6  }
7
8  const retrieve = (req, res, next) => {
9    if (req.params.hasOwnProperty('collection')) {
10     let rawdata = require('./csdExports/' + req.params.collection
11       + '.json');
12     let collection = JSON.parse(JSON.stringify(rawdata));
13     if (req.params.hasOwnProperty('id')) {
14       res.locals.data = collection.find(i => i.id == id);
15       next();
16     }
17     else {
18       res.locals.data = collection;
19       next();
20     }
21   }
22 }
23 }
24
25 const update = (req, res, next) => {
26   ...
27 }
28
29
30 const _delete = (req, res, next) => {
31   ...
32 }
33
34 module.exports = {
35   create,
36   retrieve,
37   update,
38   delete: _delete
39 };
```

Listing 3.7: Node.js - Mock functions

3.3 Front-end

On the application side the decision was to utilize Angular and Cordova in order to target more devices, giving the application cross-platform capabilities.

This led to the Ionic framework to be selected for development reasons specifically because of its ability to wrap both Angular and Cordova, and deliver a mobile hybrid app while providing the option of deploying a PWA. Also, Ionic framework ensures that the look and feel of the app follows the OS-related styling guidelines.

The front-end implementation has been divided into three parts, the API Library - that handles the incoming data -, the helper services that have been developed to overcome certain obstacles and the UI Components that are responsible for consuming said data.

In Listing 3.8 a typical app configuration file is presented which demonstrates how modules can be enabled and point to a specific data source.

```
1  {
2    "dataSources": {
3      "rest": {
4        "enabled": true,
5        "url": "http://localhost:5000"
6      },
7      "static": {
8        "enabled": true,
9        "url": "https://www.csd.uoc.gr/~folderServingData/releases"
10     }
11   },
12   "modules": {
13     "announcements": {
14       "enabled": true,
15       "source": "static"
16     },
17     "documents": {
18       "enabled": true,
19       "source": "static"
20     },
21     // populate foreach module
22   }
23 }
```

Listing 3.8: App configuration example

3.3.1 API Library

The API Library has been developed in order to be able to implement interchangeable services and feed data arriving from different sources. This level of abstraction and flexibility is achievable through the separation between services that handle the data and the data provider types.

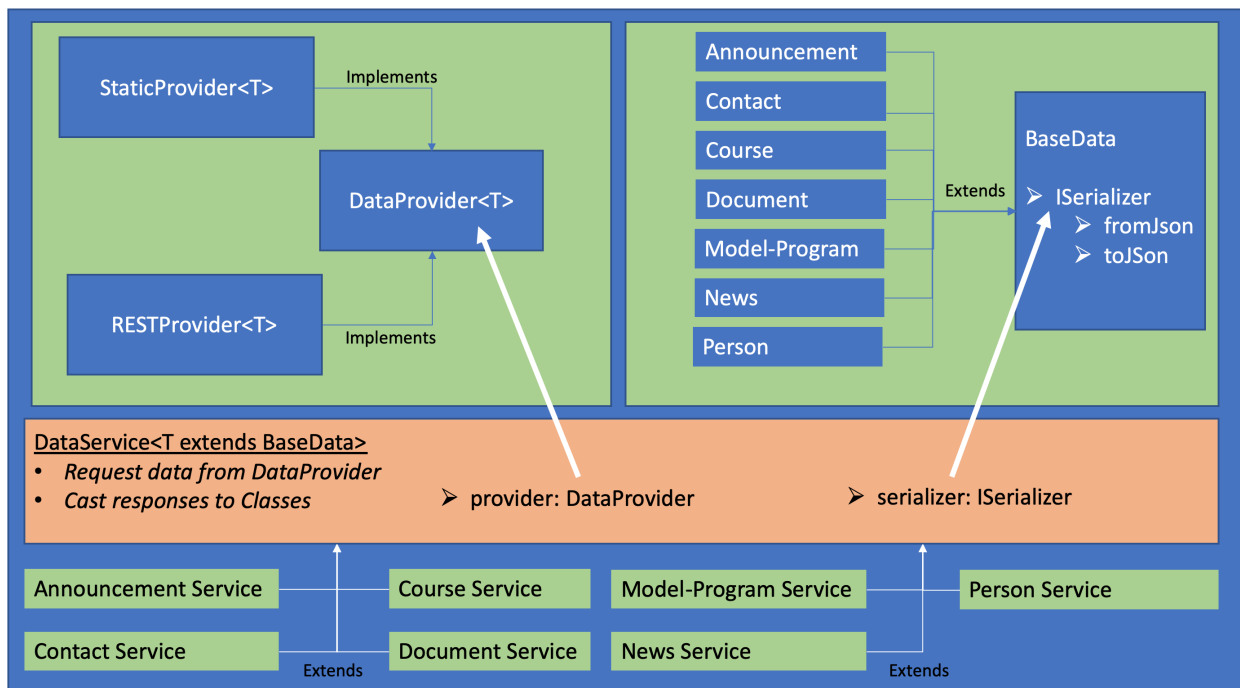


Figure 3.2: API library overview

3.3.1.1 Module Classes

Each module has an equivalent class that casts the incoming data and ensures that the structure consumed by the UI is the expected one. The classes extend the BaseData class which consists of a static ISerializer property. This interface forces each module class to implement a "fromJson" and a "toJson" method in order to properly cast data.

```

1 export class BaseData {
2   public static serializer: ISerializer<BaseData>;
3 }

```

Listing 3.9: Front-end API - BaseData class

```

1 export interface ISerializer<T> {
2   fromJson(json: any): T;
3   toJson(resource: T): any;
4 }

```

Listing 3.10: Front-end API - ISerializer interface

A module class example, the Document class, is presented in Listing 3.11:

```

1  export class Document extends BaseData {
2    constructor(
3      public _id: string,
4      public label: string,
5      public source: object,
6      public type?: ModuleType // enum
7    ) {
8      super();
9    }
10
11   public static serialiser = {
12
13     /**
14      * Cast a JSON object to Document class
15      * @param obj: JSON object
16      */
17     fromJson(obj: any): Document {
18       return new Document(
19         obj._id,
20         obj.label,
21         obj.source,
22         ModuleType.documents
23       );
24     },
25
26     /**
27      * Cast Document class to JSON object
28      */
29     toJson(item: Document): any {
30       return {
31         _id: item._id,
32         label: item.label,
33         source: item.source,
34         type: item.type
35       };
36     }
37   };
38
39   public toJson(): any {
40     Document.serialiser.toJson(this);
41   }
42 }

```

Listing 3.11: Front-end API - Document class

3.3.1.2 Data Providers

There are two (2) distinct data providers included in the solution:

- Static Data Provider
- REST Data Provider

Static Data Provider

Serving the need for no server actually included in the implementation and the python crawler being developed to bridge the gap between webpage data and the absence of REST APIs, static data provider is necessary on the front-end side.

This provider mirrors the python crawler behaviour:

1. On startup, if at least on of the modules in the app configuration is set to use the static provider, then the version file is retrieved.
2. The retrieved file is checked against the local one. If a module version is different - or non existent in the local version file - and is set to use the static provider, then this module is marked as needed to be updated. In case the local version file is inaccessible (e.g. initial run), then all the related modules are marked for update.
3. When the module service (e.g. PeopleService) is being instantiated for the first time, and the related module has been marked for update, then the data for this collection are retrieved from the remote dataset (e.g. people.json).
 - If the collection already exists, it is being fully replaced, ensuring that the application is updated with the latest data.

It's worth mentioning that StaticProvider utilizes PouchDB to store data on the device, ensuring the offline user experience.

```

1  @Injectable()
2  export class StaticProvider<T> extends DataProvider<T> {
3
4    proxyPath: string = 'releases';
5    baseKey: string = 'csd_';
6    public mandatoryUpdate: boolean = false;
7    offlineDb: PouchDBDataProvider;
8    constructor(
9      public route: string,
10     public module: string,
11     private callhandler: CallHandlerService,
12   ) {
13     super();
14     this.offlineDb = ServiceLocator.injector.get(
15       PouchDBDataProvider);
16     this.data$ = new BehaviorSubject<any>(this.data);
17     this.mandatoryUpdate = this.checkRetrieveState(this.module,
18       this.baseKey);
19   }
20   getSignle(id: string): Observable<T[]> {
21     return from(this.offlineDb.retrieve(this.module, { id }));
22   }

```

```

23   getSignleByField(field: string, value: string): Observable<T> {
24     return from(this.offlineDb.retrieveByField(this.module, field
25       , value));
26   }
27
28   getAll(): Observable<T[]> {
29     const ret = new Promise<T[]>(async (resolve, reject) => {
30       if (this.mandatoryUpdate) {
31         try {
32           const data: any = await this.sendRequest('GET', `${
33             this.proxyPath}/${this.route}`, {});
34           await this.offlineDb.updateCollection(this.module, data
35             );
36           this.updateData(data, resolve);
37         }
38         catch (err) {
39           throw Error(err);
40         }
41       }
42       else {
43         let response;
44         try {
45           response = await this.offlineDb.retrieve(this.module);
46           this.updateData(data, resolve);
47         }
48         catch (err) {
49           response = null;
50         }
51         if (response == null) {
52           const data: any = await this.sendRequest('GET', `${
53             this.proxyPath}/${this.route}`, {});
54           await this.offlineDb.updateCollection(this.module, data
55             );
56           this.updateData(data, resolve);
57         }
58       }
59     });
60     return from(ret);
61   }
62
63   private sendRequest(method, url, payload): Promise<any> {
64     // ... submit request through the callhandler
65   }
66
67   async updateCollection(module: string): void {
68     try {
69       const data = await this.getAll().toPromise();
70       this.offlineDb.updateCollection(this.module, data);
71       this.mandatoryUpdate = false;
72       localStorage.setItem(`static_retrieve_${this.module}`, '
73         false');
74     }
75     catch (err) {

```

```

71     }
72   }
73 }
74
75 private checkRetrieveState(module: string, baseKey: string):
76   boolean {
77   const retrieveState = localStorage.getItem(`static_retrieve_${
78     {module}`);
79   if (retrieveState !== null) {
80     const moduleState = JSON.parse(retrieveState);
81     return moduleState || false;
82   }
83   return false;
84 }
85
86 private updateData(data: T[], callback: any): void {
87   this.data = data;
88   this.data$.next(data);
89   callback(this.data);
90 }

```

Listing 3.12: Front-end API - StaticProvider class

REST Data Provider

In REST provider implementation, every call is handled by the callhandler instance provided. It has been included in the solution to ensure the data source flexibility.

```

1  @Injectable()
2  export class RESTProvider<T> extends DataProvider<T> {
3
4    constructor(
5      public route: string,
6      public module: string,
7      private callhandler: CallHandlerService,
8    ) {
9      super();
10     this.data$ = new BehaviorSubject<any>(this.data);
11   }
12
13   getSignle(id: string): Observable<T> {
14     return from(this.sendRequest('GET', `${this.route}/${id}`));
15   }
16
17
18   getAll(): Observable<T []> {
19     const ret = new Promise<T []>(async (resolve, reject) => {
20       try {

```



```

21         const data: any = await this.sendRequest('GET', `${
22             this.route}`);
23         this.updateData(data, resolve);
24     }
25     catch (err) {
26         reject(err);
27     }
28     });
29     return from(ret);
30 }
31
32 getSignleByField(field: string, value: string): Observable<T> {
33     if (this.data != null) {
34         const found = this.data.find(x => x[field] === value);
35         if (found != null) {
36             return of(found);
37         }
38     }
39     return this.getAll().pipe(map((data: T[]) => {
40         const found = data.find(x => x[field] === value);
41         if (found == null) {
42             throw new Error('Item not found');
43         }
44         return found;
45     }));
46 }
47
48 private sendRequest(method: string, url: string, payload = null
49 ): Promise<any> {
50     return new Promise(async (resolve, reject) => {
51         this.callhandler.sendRequest(
52             method, url, payload
53         ).then((data: any) => {
54             resolve(data);
55         }).catch((err) => {
56             reject(err);
57         });
58     });
59 }

```

Listing 3.13: Front-end API - RESTProvider class

3.3.1.3 Data Services

The implementation of the DataService is a key part of the solution. This is the decision point regarding which data provider is used for each module.

DataService wraps CRUD functions by calling the provider's equivalent action. This service is responsible for casting the data in the correct format before feeding them to the UI components.

DataService constructor requires four (4) parameters: i)route, which is the route that should be called to retrieve data, ii)module, the collection tag, iii) CallhandlerService instance, as described in Section 3.3.2, and an ISerializer structure (Listing 3.10).

```

1  {
2  @Injectable({
3    providedIn: 'root'
4  })
5  export class DataService<T extends BaseData> {
6    provider: DataProvider<T>;
7    constructor(
8      public route: string,
9      public module: string,
10     private callhandler: CallHandlerService,
11     public serializer: ISerializer<T>
12   ) {
13     const config: ConfigService = ServiceLocator.injector.get(
14       ConfigService);
15     this.provider = this.getDataProvider(config.getModuleProvider(
16       module));
17   }
18   private getDataProvider(config: any): DataProvider<T> {
19     let ret: DataProvider<T>;
20     switch (config) {
21       case 'static':
22         ret = new StaticProvider<T>(
23           this.route,
24           this.module,
25           this.callhandler
26         );
27         break;
28       case 'rest':
29         ret = new RESTProvider<T>(
30           this.route,
31           this.module,
32           this.callhandler
33         );
34         break;
35       case 'empty':
36       default:
37         throw new Error('Provider type not found');
38         break;
39     }
40     return ret;
41   }
42
43   public get data(): T[] { return this.provider.data; }
44

```

```

45     public get data$(): BehaviorSubject<T[]> { return
         this.provider.data$; }
46
47     public getSingle(id: string): Observable<T> {
48         return this.provider.getSingle(id)
49             .pipe(map((data: any) => this.serializer.fromJson(data) as
                    T));
50     }
51
52     public getAll(): Observable<T[]> {
53         return this.provider.getAll()
54             .pipe(map((data: any[]) => this.castData(data)));
55     }
56
57     public create(item: T): Observable<T> {
58         return this.provider.create(item)
59             .pipe(map((data: any) => this.serializer.fromJson(data) as
                    T));
60     }
61
62     public update(item: T): Observable<T> {
63         return this.provider.update(item)
64             .pipe(map((data: any) => this.serializer.fromJson(data) as
                    T));
65     }
66
67     public delete(id: string): Observable<T> {
68         return this.provider.delete(id)
69             .pipe(map((data: any) => this.serializer.fromJson(data) as
                    T));
70     }
71
72     public getSignleByField(field: string, value: string):
         Observable<T> {
73         return this.provider.getSignleByField(field, value)
74             .pipe(map((data: any) => this.serializer.fromJson(data) as
                    T));
75     }
76
77     protected castData(data: any): T[] {
78         return data.map(item => this.serializer.fromJson(item) as T);
79     }
80 }

```

Listing 3.14: Front-end API - DataService class

This leads services attached to the static or the REST provider to be declared in a simple compact pattern. These services are the following:

- AnnouncementsService
- ContactService
- CoursesService

- DocumentsService
- ModelProgramService
- NewsService
- PeopleService

In Listing 3.15 the ContactService declaration is presented as an example:

```

1  @Injectable()
2  export class ContactService extends DataService<Contact> {
3    constructor(
4      public callHanndler: CallHandlerService
5    ) {
6      super('contacts', 'contacts', callHanndler,
7           Contact.serializer);
8    }

```

Listing 3.15: Front-end API - Module service class example

3.3.2 Helper Services

CallHanlderService

Call handler service is responsible for preparing and dispatching HTTP requests. The request can be sent either via the Angular level or through the native one. The latter contributed in solving an issue regarding the static provider, where getting the data from a public folder within the Computer Science ecosystem raised cross-origin resource sharing (CORS) errors.

```

1  @Injectable()
2  export class CallHandlerService {
3    serverURI: string;
4    constructor(private angularHttp: HttpClient, private
5               nativeHttp: HTTP) { }
6
6  sendRequest(method: string, endpoint: string, payload: any =
7             {}) {
7    return new Promise((resolve, reject) => {
8      // ... Prepare request
9      if (this.isNative()) {
10       this.sendNativeHttpRequest(`${this.serverURI}${endpoint}
11                                `, method, params, {}, headers, payload, resolve,
12                                reject);
11     } else {
12       this.sendAngularHttpRequest(endpoint, method, params, {},
13                                  headers, payload, resolve, reject);
13     }

```

```

14     });
15   }
16
17   private sendAngularHttpRequest(url: string, method: string,
18     params: any, options: any, headers: any, body: any,
19     onSuccess, onError) {
20     // ... handle request on the browser level
21   }
22
23   private sendNativeHttpRequest(url: string, method: string,
24     params: any, options: any, headers: any, body: any,
25     onSuccess, onError) {
26     // ... handle request on the native level
27   }
28
29   private isNative() { return !!window['cordova']; }
30 }

```

Listing 3.16: Fron-tend API - CallHandler Service class

3.3.3 Application Components

Each module has it's own set of files, some of them following the single-details pattern, a pattern which brings consistency in the UX across different modules. A typical file set of a module is presented in Figure 3.3:

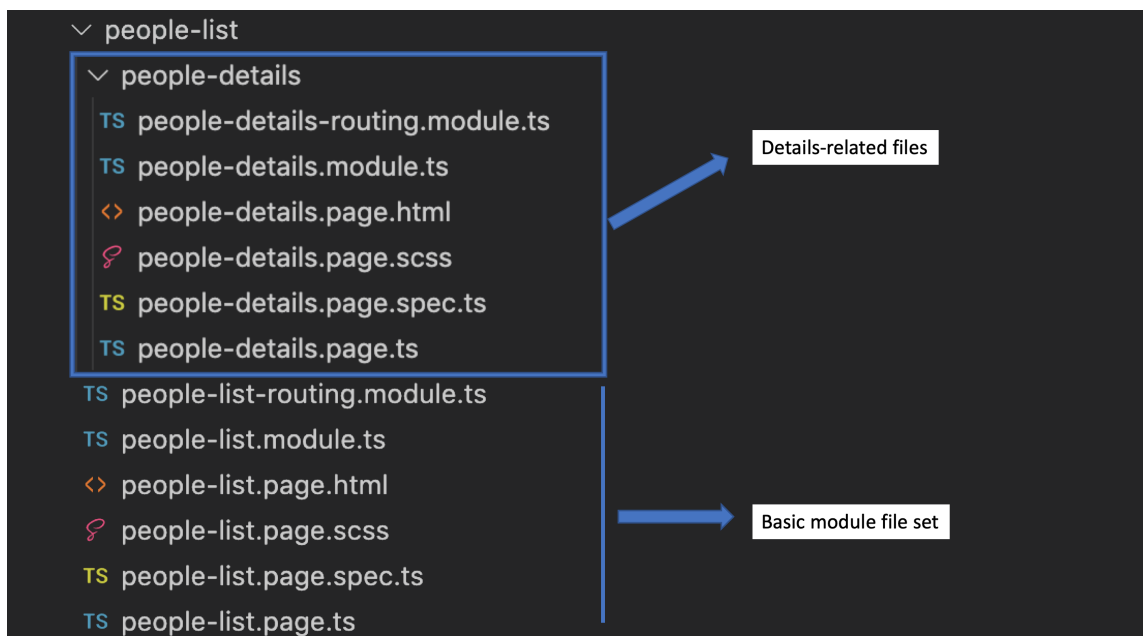


Figure 3.3: File structure example

In a module page, data are being received through the equivalent service by subscribing to the service's `getAll()` method that returns an `Observable` as shown in Figure 3.4.

There are cases where data need to be processed before fed to the template e.g. when there is a need to be sorted alphabetically.

```
constructor(  
  private peopleService: PeopleService,  
  private navCtrl: NavController,  
  private alertController: AlertController  
) {  
  this.people$ = this.peopleService.getAll()  
  .pipe(  
    map((people: Person[]) => people.sort((a, b) => {  
      if (this.getField(a, 'name', this.language) < this.getField(b, 'name', this.language)) { return -1; }  
      if (this.getField(a, 'name', this.language) > this.getField(b, 'name', this.language)) { return 1; }  
      return 0;  
    })))  
  )  
  .subscribe(people => {  
    this.people = people;  
    this.filters = _.chain(this.people)  
      .map((x: Person) => this.getField(x, 'position', this.language))  
      .uniq()  
      .reject(x => !x)  
      .sort()  
      .value();  
  });  
}
```

Figure 3.4: Data preparation for display purposes

Chapter 4

User Interface

This chapter provides details regarding the modules that have been picked to be served via the app to the end users.

Note: The content displayed in the rest of the chapter is the result of crawling the actual website, hence language incompatible content should not be considered an issue - the app serves the content available in the specified language with Greek acting as the fallback option.

4.1 Announcements - News

As stated in section 3.1.1, announcements and news share the same structure. This led to the decision to include both of them in the same screen as tabs and present the same behavior by design. Each module page consists of a list of cards that provide the title. Users can see details by tapping on the card, triggering the description area to enlarge.

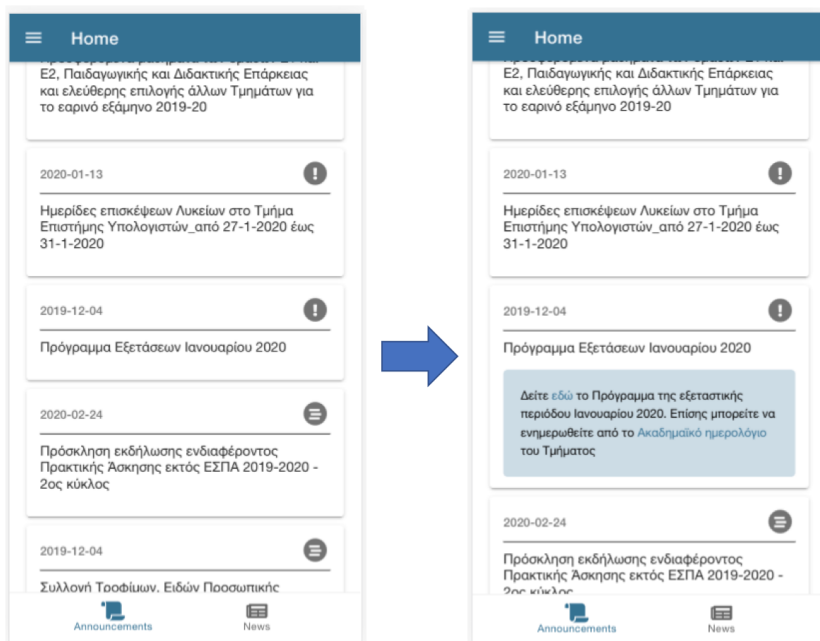


Figure 4.1: Home page - Announcements tab

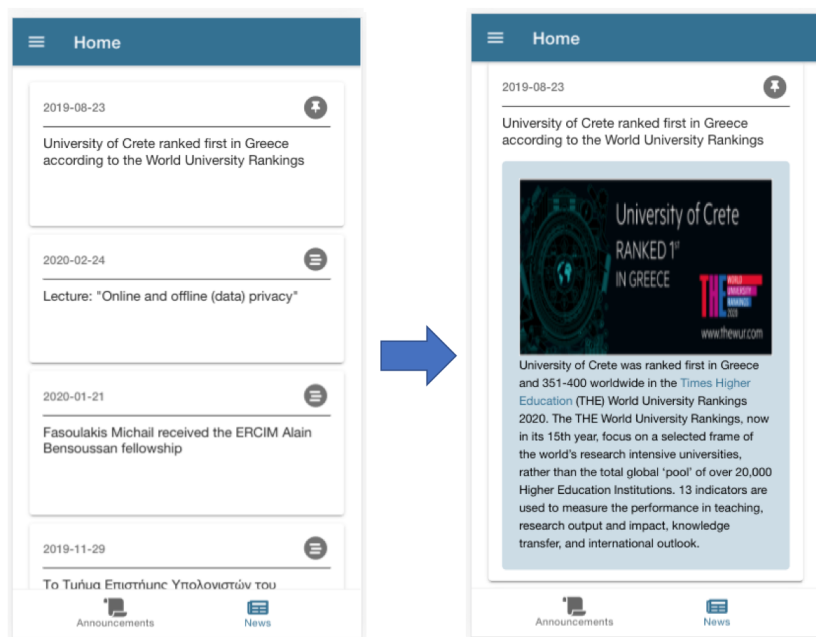


Figure 4.2: Home page - News tab

4.2 Contact Information

Contact information page consists of cards displaying for each entity the related fields such as email, phone number(s) and website links. By selecting one the equivalent functionality is triggered, e.g. the phone app, the mail app or the default browser.

There is also a search box available aiming to assist the user in reaching faster the desired information.

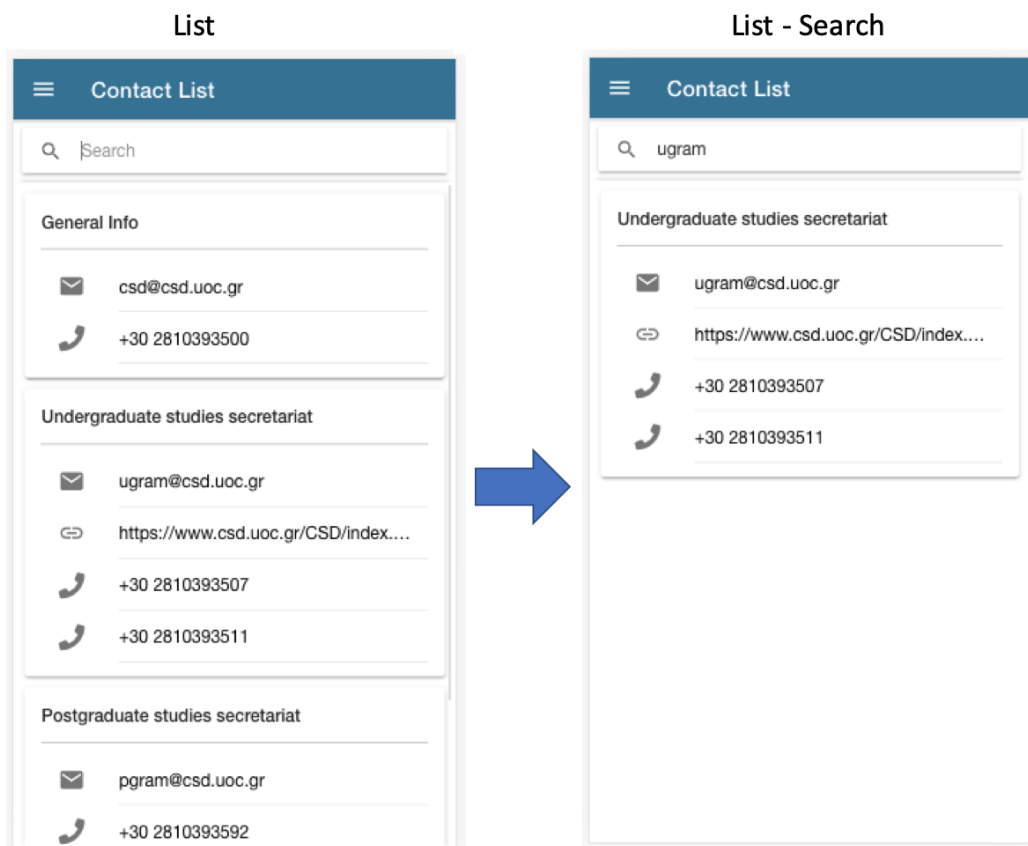


Figure 4.3: Contact List page - Search

4.3 Courses

Courses module follows the list-details paradigm, assisted by a search box.

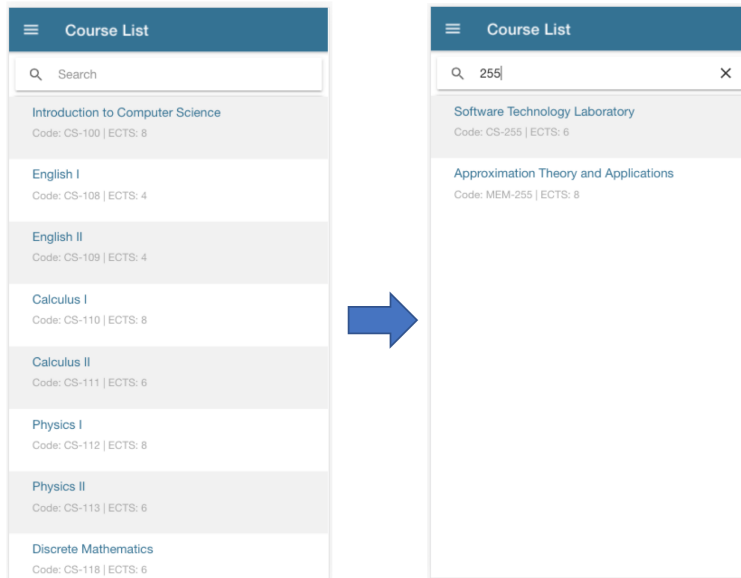


Figure 4.4: Course List page - Search

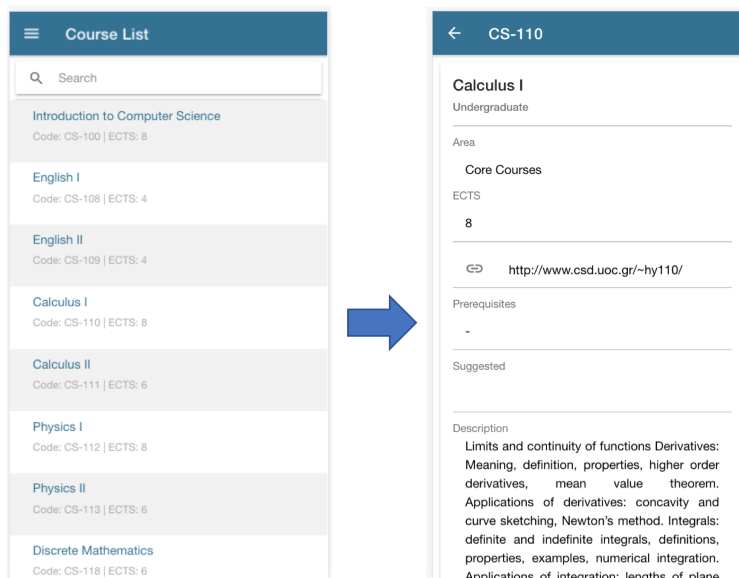


Figure 4.5: Course List page - Details

4.4 Documents

Documents page consists of cards displaying for each entity the related available formats. By clicking one the document formats, the user can download the selected document

There is also a search box available that utilizes document titles, filenames and extensions.

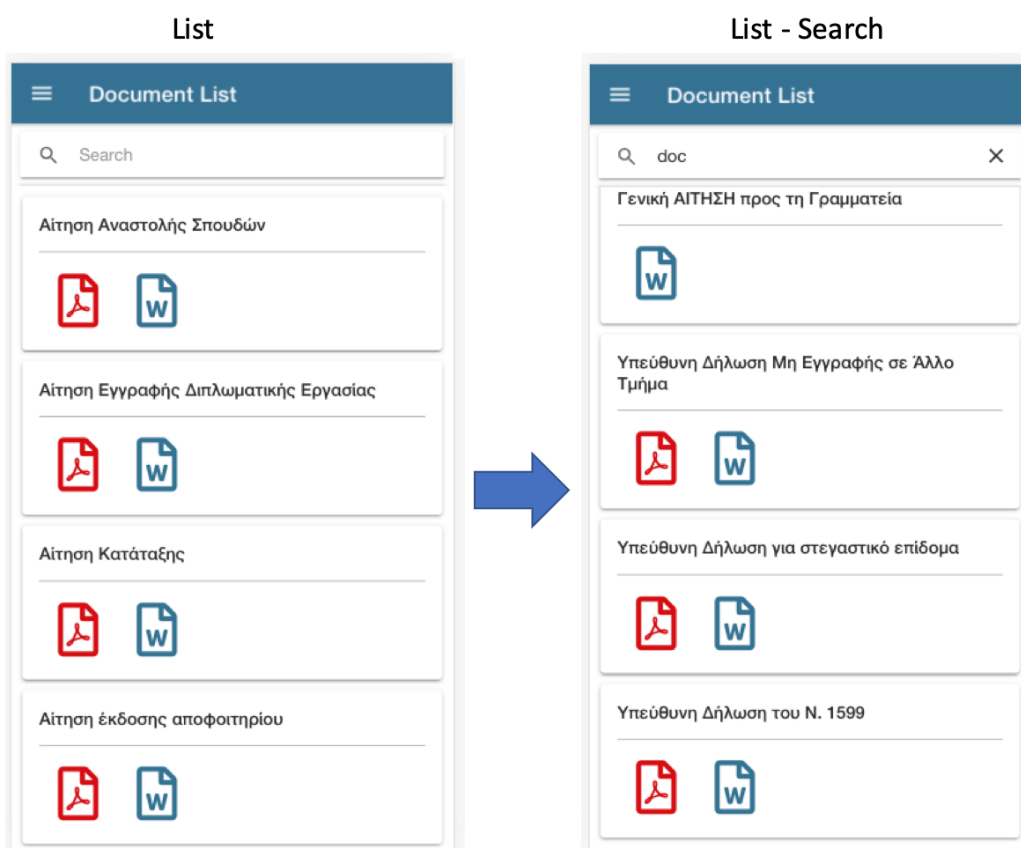


Figure 4.6: Document List page - Search

4.5 Map

Leaflet library [6] is utilized in order to implement the map module. The user is presented with a map centered to the Computer Science department location. A search box on top reveals a list of rooms when focused. When a room is selected, the equivalent floor overlay is enabled and a marker on the location of the room is revealed along with a card containing the room information.

A button present left of the search box re-centers the map to the initial coordinates.

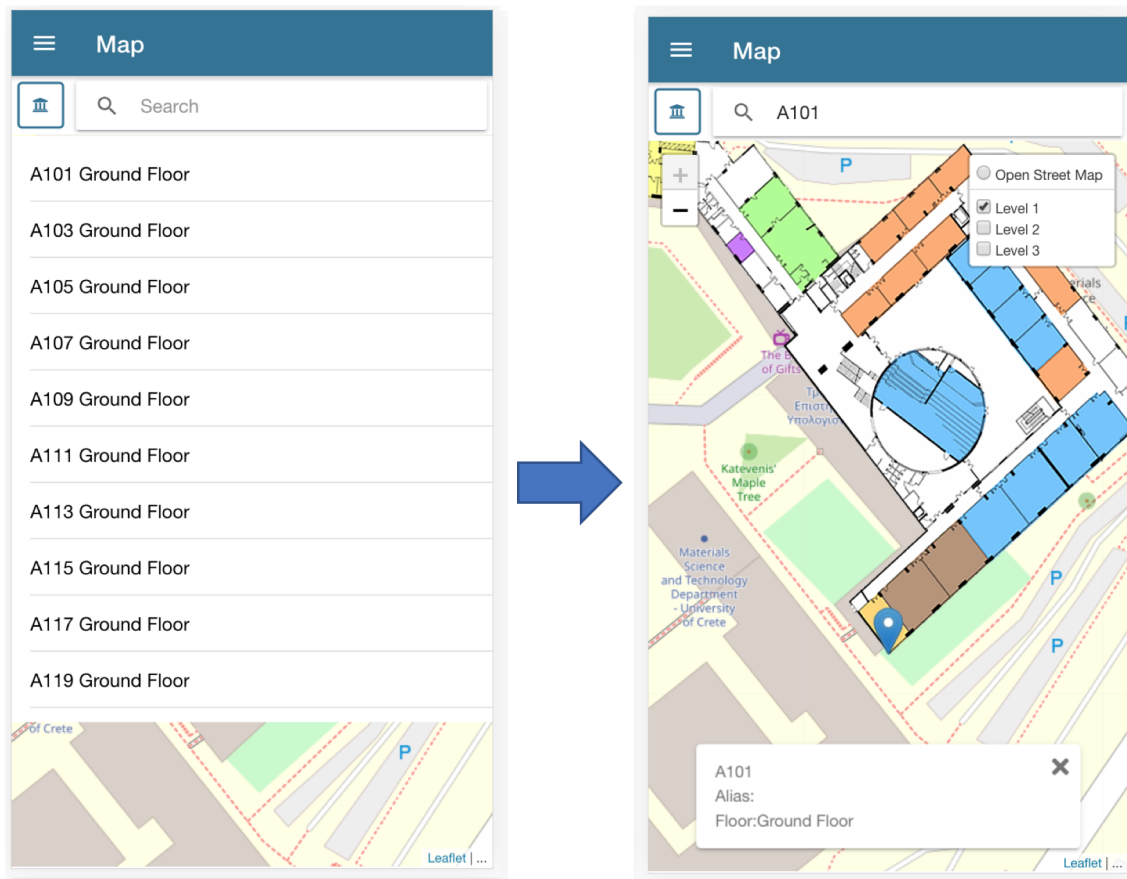
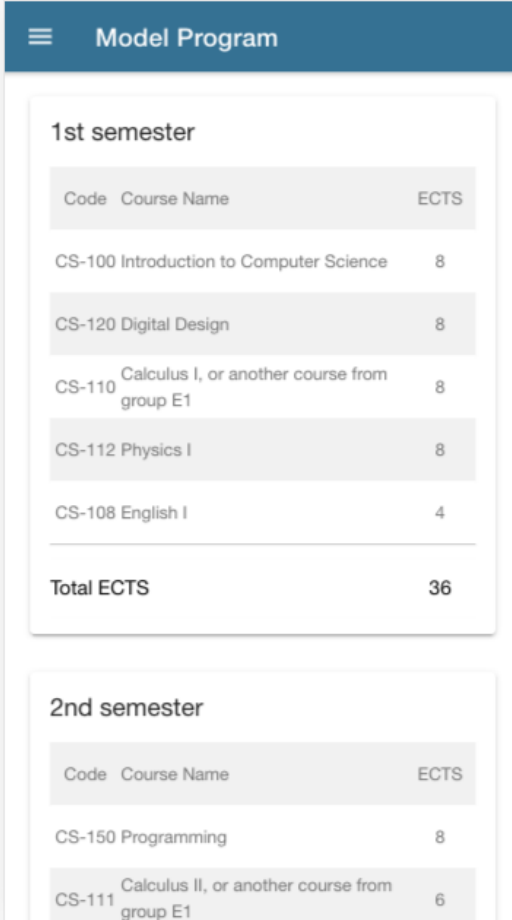


Figure 4.7: Map page

4.6 Model Program

Model program module displays the current eight-semester suggested route to satisfy the Bachelor's requirements.

Each semester holds a list of courses and the related ECTS credits. Total ECTS row exists at the bottom of each semester's card as shown in Figure 4.7:



The screenshot shows a mobile application interface titled "Model Program". It displays two semester cards. The first card, "1st semester", contains a table with the following data:

| Code | Course Name | ECTS |
|-------------------|---|-----------|
| CS-100 | Introduction to Computer Science | 8 |
| CS-120 | Digital Design | 8 |
| CS-110 | Calculus I, or another course from group E1 | 8 |
| CS-112 | Physics I | 8 |
| CS-108 | English I | 4 |
| Total ECTS | | 36 |

The second card, "2nd semester", contains a table with the following data:

| Code | Course Name | ECTS |
|--------|--|------|
| CS-150 | Programming | 8 |
| CS-111 | Calculus II, or another course from group E1 | 6 |

Figure 4.8: Model Program page

4.7 People

People module follows the list-details paradigm, assisted by a search box, as presented in Figure 4.8. On top of that, a filter button exists near the top right corner, which reveals all the person types within the current data set.

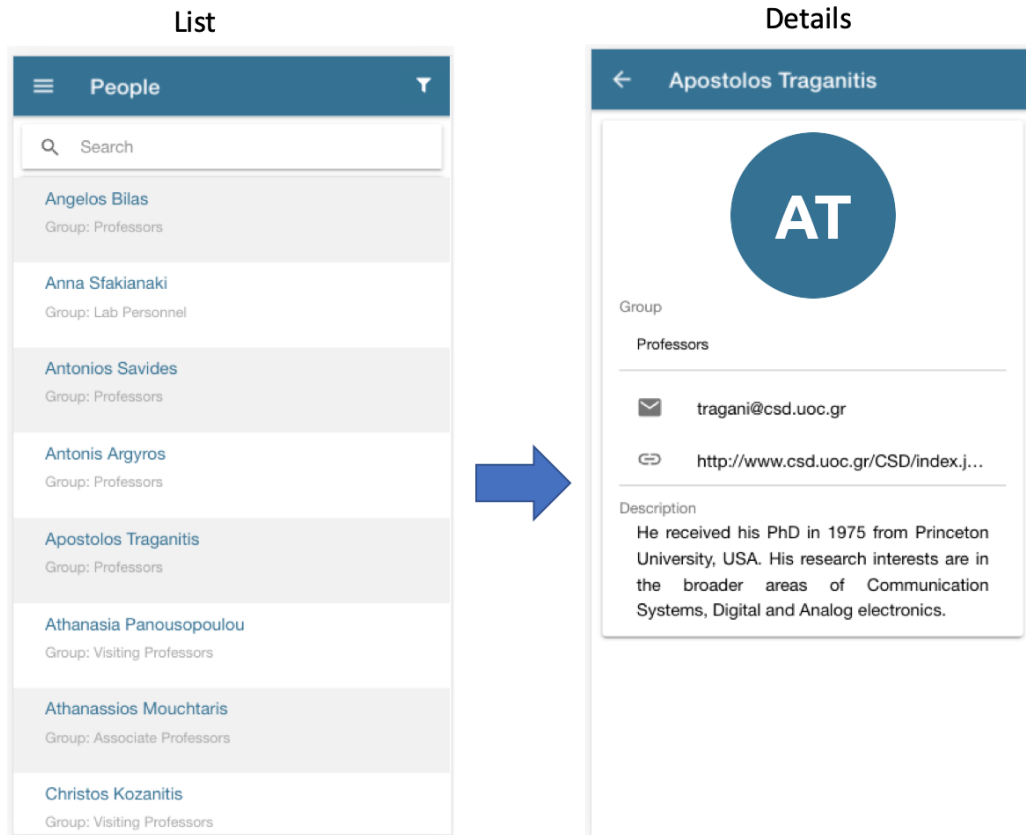


Figure 4.9: People page - Details

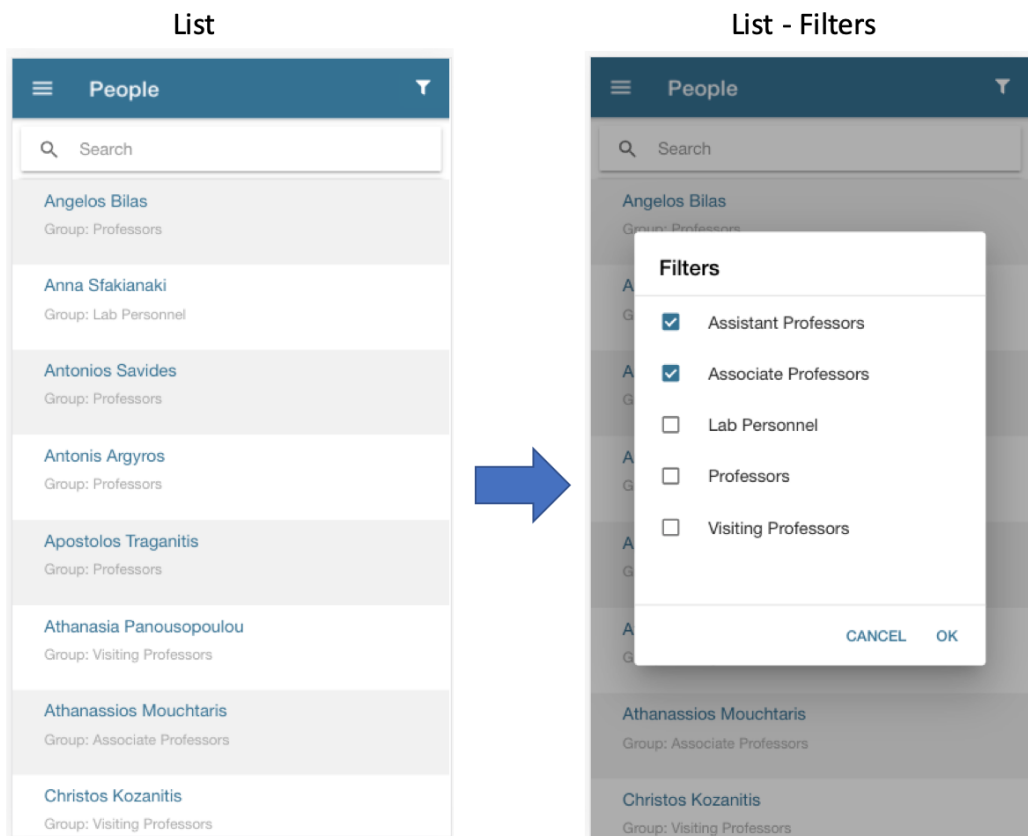


Figure 4.10: People page - Filters

4.8 Schedule

Schedule module displays information related to the current semester's classes available on a weekly basis. The right side of the UI is swipable (Figure 4.10) in order to reveal all five (5) days of the week. A searchbar has been added to provide the user with the choice to filter based on the courses name or code.

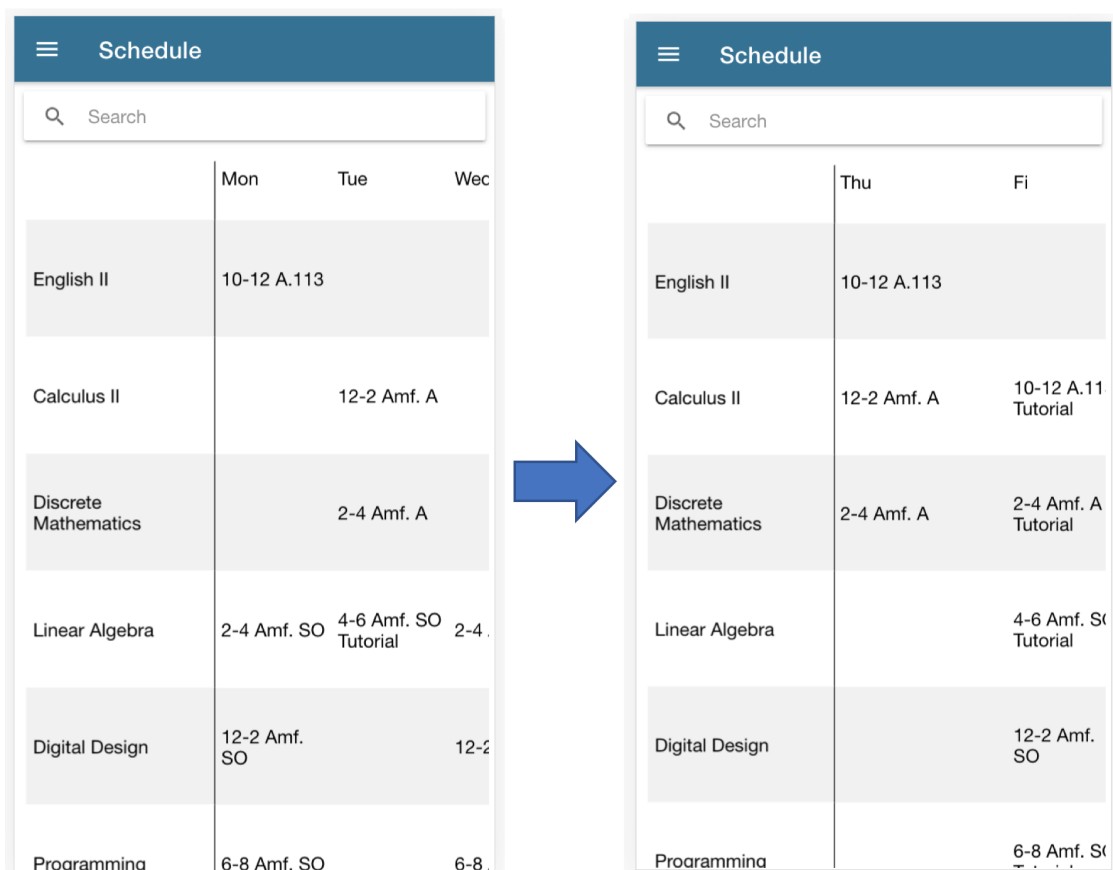


Figure 4.11: Schedule page - Week swipe

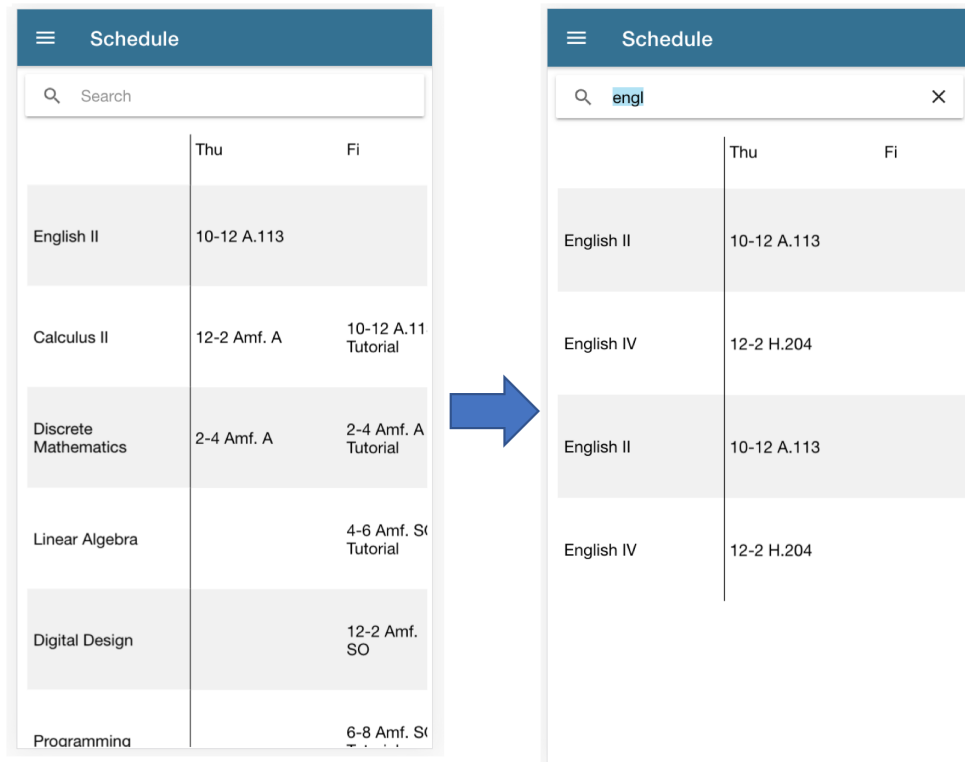


Figure 4.12: Schedule page - Search

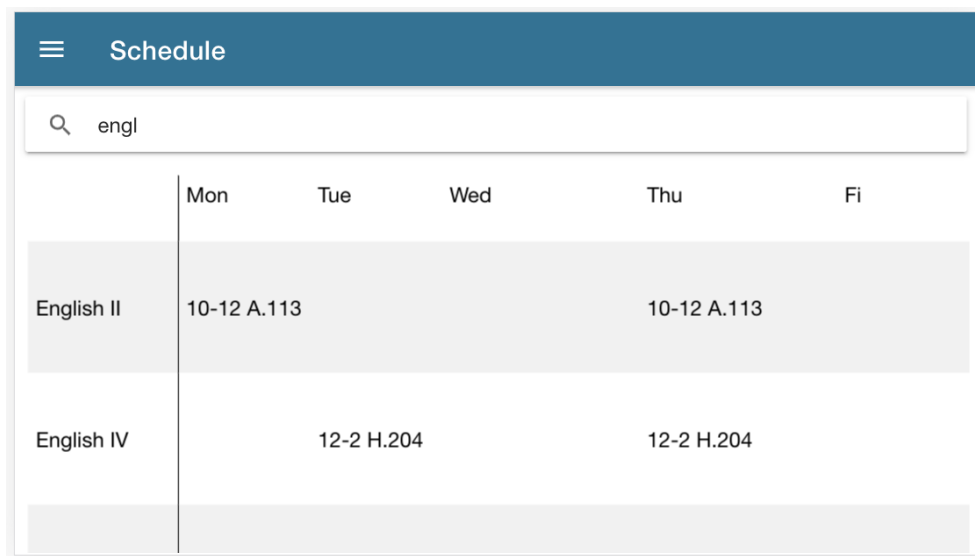


Figure 4.13: Schedule page - Landscape mode

Chapter 5

Summary and Future work

5.1 Summary

Nowadays users spend an increasing amount of time on their mobile devices. To add to that, the preference of using mobile apps over accessing content via mobile browsers has been observed. This has been leveraged by organisations in multiple occasions in order to keep the users better engaged and connected to their digital content.

The proposed work in this thesis is the adoption by the Computer Science Department of a new channel to connect with its students. It has been designed taking into consideration the core areas students tend to interact with on a daily basis and has accounted for the absence of the REST APIs with a daily-spawned website scrapper. The front-end API library has been developed following a generic pattern, aiming to accommodate future changes regarding data sources. Lastly, the versioning of the data ensures that the end-users receive new data only if a collection has been updated.

5.2 Future work

As mentioned in section 5.1, the current implementation includes a web scrapping mechanism to generate data sets. The first step regarding future work needs to be the implementation of a server to replace that mechanism and act as the main data source for the mobile app. The server should act as an intermediate between the students and their digital content that is currently scattered between at least three websites: (i) www.csd.uoc.gr, (ii) elearn.uoc.gr and (iii) www.students.cc.oc.gr. This will provide a single point of access for the students and, on top of that, personalized content since not only will they be able to access course information, but also their grades, their GPA, the courses they registered for in the current semester etc.

Moreover, since this server is suggested to be developed in-house and be part of

the technologies maintained by the department, there is a wide variety of services and features that can be added in this new relationship with the students that are currently handled via emails:

1. Push notifications, the most popular way to keep users engaged and informed on mobile devices, should be utilized to communicate to the students important messages. Some scenarios regarding this feature are the following ones:
 - (a) an important announcement needs to reach every student (emergency situation, important event etc.)
 - (b) when a class has been cancelled, students registered to the class need to be notified.
 - (c) an assignment deadline is in the near future, a situation where it would be beneficial for students that have not submitted their solutions to be reminded of that.
2. New processes can be added as in-house maintained content, where the student can be instantly informed regarding the availability (e.g. using push notifications) and track the status of the request at any time. The automation of these processes can also lessen the staff's workload. There are a number of cases regarding this feature:
 - (a) Applications for graduate studies
 - (b) Meal services requests,
 - (c) Accommodation requests,
 - (d) Suspension of studies or
 - (e) Course registration for the current semester - a case where the server has to act as an intermediate between the students and the current place this functionality exists in, students.cc.uoc.gr
3. The functionality described in (2) suggests that an administrator dashboard is included in the solution, where actions like handling the requests and digitally signing documents take place.

Bibliography

- [1] angular.io.
- [2] Apache cordova - cordova.apache.org.
- [3] Bootstrap.
- [4] emberjs.com.
- [5] Ionic framework.
- [6] Leafletjs.
- [7] Popular types of apps - thinkmobiles.com/blog/popular-types-of-apps/.
- [8] Python beautifulsoup library - <https://www.crummy.com/software/beautifulsoup/>.
- [9] Python programming language - [.python.org](https://python.org).
- [10] React native.
- [11] reactjs.org.
- [12] Responsive website vs mobile app: Comparison.
- [13] scrapy.org.
- [14] selenium-python.readthedocs.io.
- [15] Sventech node.js generator.
- [16] Vue.js.
- [17] Xamarin - dotnet.microsoft.com/apps/xamarin.
- [18] Desktop vs mobile vs tablet market share worldwide, 2009.
- [19] Desktop vs mobile vs tablet market share worldwide, 2019.
- [20] Muhammad Hakim A. Phonegap/cordova vs native application, 2012.

- [21] Chris Ciligot. Mobile app vs. mobile website: A ux comparison – which is the better option?, 2019.
- [22] Suyash Dubey. Types of mobile apps, 2019.
- [23] By Dhananjay Goel. Native mobile app or mobile web app: What is best for your business?, 2018.
- [24] Everett N McKay. *UI is Communication: How to Design Intuitive, User Centered Interfaces by Focusing on Effective Communication*. 2013.
- [25] Priyesh Patel. Pwa vs hybrid app vs native: Choosing the right mobile app, 2018.
- [26] Ewan Spence. The mobile browser is dead, long live the app, 2014.
- [27] Stelios Xinogalos Spyros Xanthopoulos. A comparative analysis of cross-platform development approaches for mobile applications. 2013.
- [28] Matt Warcholinski. App vs website – which to develop first?