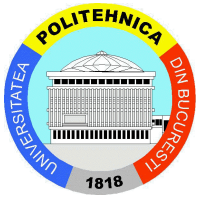
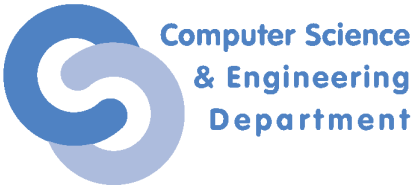
UNIVERSITY POLITEHNICA OF BUCHAREST
FACULTY OF AUTOMATIC CONTROL AND COMPUTERS





Design and Implementation of a Cross-Platform Mobile Application That Facilitates Student Collaboration

Graduate:
Ioana Alexandru

Thesis supervisor:
Lecturer, Dr. Vlad Posea

Bucharest
2020

The university is a complex environment with many creative people having as a disadvantage its sometimes chaotic development. This evolution is visible in the vast array of software applications meant to assist students. Over time, in our university, many such platforms were developed, including the university website, the faculty website, websites of different student organizations, wikis created by professors or departments, and the official course platform. This variety turned into an information overload for students who are expected to be aware of everything that is posted on these websites, to discern between relevant information and spam, while also being busy attending courses and completing assignments.

We believe that students themselves can address this issue by using the power of their community to flag and organize important information from these websites, disseminate it to relevant colleagues and help each other get through the labyrinth of information in the university.

With this goal in mind, this paper describes a collaborative app that allows students to better organize their time and schedule by collectively identifying the most useful information from the university's sources and intelligently sharing it with their peers.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

According to Kevin Kelly [1], *technology is an extension of life*. Its roots in our day to day lives go so deep that, for most of us, it is complicated or impossible to even imagine life without technology. It has made its way into our jobs [2], our commute [3], even our nutrition [4] and personal lives [5]. It should come as no surprise that it has become a huge part of students' lives, which we will be focusing on hereafter.

Due to the rapid pace with which new and better technologies replace the old ones, as well as the existence of a variety of tools that serve more or less the same purpose, students nowadays find themselves flooded with a plethora of platforms and resources provided by their faculty.

In recent years, new and better technologies tend to replace old ones at a rapid pace. This progress often implies that some people will continue to use the old tools (or even "analog" tools instead of "digital" ones) out of habit, while early adopters will immediately switch to the new ones. As a result, there exists a discrepancy in the platforms used by professors for the same purpose. This discrepancy, in turn, overwhelms students, particularly in their first year of university.

In order to cope with the massive influx of information, students often turn to each other for help and assistance. This process is informal, taking place either face to face or through social media.

This paper proposes a cross-platform mobile application that aims to act as a compendium of the university's various resources and to provide an easy-to-use, intuitive platform for collaboration.

1.2 Environment

For the purpose of this paper and the application prototype, we will be analyzing the case of one of the 15 faculties[1] of **University POLITEHNICA of Bucharest** in Romania, namely the **Faculty of Automatic Control and Computer Science**. From here on out, we will be referring to the larger engineering institution as *the university* or *UPB*, and to the computer science faculty as *the faculty* or *ACS*.

This section provides insight into the context of this application, namely the typology of the ACS student (subsection 1.2.1) - our user - as well as relevant details about the way their university life is organized. The latter includes information about how the student groups are organized within the university (subsection 1.2.2), which platforms they need to use in order to perform their duties (subsection 1.2.3) as well as what kind of activities make up their schedule (subsection1.2.4).

1.2.1 Student/user profile

The student using this app has the following characteristics:

- an ongoing **Bachelor's/Master's degree** in computer science
- average to high **digital proficiency**, common for computer science students
- proficiency in either **Romanian or English**, the two languages that the institution provides courses in
- an age between **18 and 25 years**

1.2.2 Student body structure

The faculty provides Bachelor's, Master's, and Doctoral science degrees. We are interested in the B.Sc. and M.Sc. degrees, each of which is split into a couple of *domains*. They last for four and two years, respectively.

[1] https://upb.ro/en/faculties/

For the **B.Sc.**, each domain has its separate curriculum, and the students are organized hierarchically into *series*, which are comprised of up to 5 *groups*, which in turn can be split into two *subgroups* each.

For the **M.Sc.**, each domain hosts several Master's programs, and each program corresponds to a single *series* of students, which is not split further.

Each *series* and each *group* have a student representative. This person acts as a link between the corresponding group of students and the faculty and is tasked with things like passing on important information, keeping in touch with the Student Office staff, and making sure the contracts are signed by all students each semester.

1.2.3 University platforms

The main platforms that the University staff use to provide resources to the students are:

- **the official university website**[2] for general university information such as the campus map and academic calendar
- **the official faculty website**[3] for the curriculum, certain announcements and timetable information
- **various course/wiki websites**[4] for class resources
- **a Moodle instance**[5] for class-related announcements, discussions and assignments
- **the university's administrative website**[6] for the student contract, personal information and grades
- **several automated-testing environments**[7] for coding assignments

Most of the platforms mentioned above can be accessed using a single set of credentials provided by the university upon admission.

[2] http://upb.ro/
[3] http://acs.pub.ro/
[4] https://ocw.cs.pub.ro/, http://elf.cs.pub.ro/
[5] https://acs.curs.pub.ro/
[6] https://studenti.pub.ro/
[7] https://vmchecker.cs.pub.ro/, https://v2.vmchecker.cs.pub.ro/

1.2.4 Class structure

A student's schedule contains three main types of classes:

- **lectures**: theoretical classes taught by professors or lecturers, which take part in a lecture hall and in which a whole series (or two, in some cases) of students participates
- **laboratories**: practical classes taught by professors, lecturers (rarely) or TAs (Teaching Assistants), which require a computer or other external tools (e.g., lab equipment, electrical equipment, etcetera), which take part in dedicated lab rooms and in which only a group or subgroup of students participates
- **seminars**: practical classes taught by professors or lecturers, which only require a pen, paper, and a whiteboard, which take part in classrooms and in which only a group or subgroup of students participates

The primary forms of evaluation are theoretical exams, homework, projects, tests, practical exams, and research assignments.

1.3 Proposed functionalities

Given the environment described in section 1.2, we can propose a number of important actions that the student should be able to do within our application:

- **access, modify and filter a list of platforms provided by the university** (see section 1.2.3), as well as view information about their use
- collaboratively **create and edit a timetable** which includes class information as well as various events (based on the types described in section 1.2.4)

Additionally, the application can provide additional, useful features such as:

- **campus map and navigation information**
- **news, FAQs and other relevant information** to which students can contribute
- **authentication** (ideally linked to the common credentials used for the other platforms)
- **localization**: the application should provide a UI (User Interface) in both Romanian (for local students) and English (for international students)

1.4 Goals

We believe that the students understand their own needs better than the faculty staff ever could. The university lacks the agility to supply useful technologies aiding students in managing their resources and schedule. Therefore, we believe that an application built, maintained, and managed by students has the best chance of providing a much-needed helping hand for students juggling a vast amount of information coming from the university.

This application should be intuitive and easy to use, as well as allow customization for any student. Above all, it should be particularly helpful for first-year students by offering an easy way to find out anything they want to know about the ins and outs of their new university life.

Students should be able to contribute to the application by submitting a PR (Pull Request) into the public repository[8], which means that it can easily be kept up-to-date with their needs and wants.

Ultimately, the application aims to become an integral part of students' lives and contribute to their education, all the while decreasing the feelings of anxiety caused by being bombarded with new, often disorganized information.

1.5 Outline

Chapter 2 outlines the current state of the art, in terms of existing applications with a similar or related purpose to our application.

Chapter 3 presents the methods used for our user study and focuses on what we learned from the survey we created.

Chapter 4 portrays the iterative design process we went through in order to come up with the UX and UI of our application. It includes results from another step of our user study - focus groups and feedback requests.

Chapter 5 describes our implementation - the technologies we used as well as the system architecture and automated testing/deployment pipelines.

Chapter 6 provides an overview of the conclusions we have reached throughout the paper, including discussion about possible caveats and future improvements.

[8]https://github.com/acs-upb-mobile/acs-upb-mobile/

Chapter 2

State of the Art

There are a large number of applications currently in the industry aiming to help students of all ages better organize their school lives. These range from *apps explicitly made for a particular university* to *generic, highly customizable applications* meant to tackle a student's needs regardless of their institution. Many productivity tools are *not explicitly targeted at students* but are frequently used by them for school-related activities. A 2013 study[6] shows that students generally use a large variety of different mobile resources to help with their academic life.

We will attempt to analyze the pros and cons of each category mentioned above and pick the features that would be most appropriate for our application, to reach the goals described in section 1.4.

Additionally, we will also be looking into the existing applications targeted for students at ACS/UPB, to plan a potential integration and collaboration and avoid re-inventing the wheel.

2.1 University-specific applications

2.1.1 Overview

Many top universities from around the world nowadays leverage the popularity of mobile devices by offering their students an application (or a set of applications) for their university needs, as pictured in figure 2.1. These often become indispensable, because a lot of the communication with the university is done solely through the application. They are usually meant to complement the university's official website as a mobile-friendly, feature-rich platform. They

are generally available on both iOS and Android devices so that any student can have access to them.

The most common features found within these applications are: university information and news (including syllabus, admissions, and student accommodation information), class timetables and academic calendars, campus maps including canteen, library, and shuttle information for universities that provide such amenities.

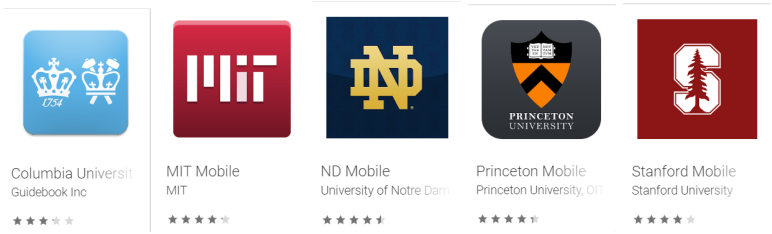


Figure 2.1: University-specific applications on Google Play

2.1.2 Case Study

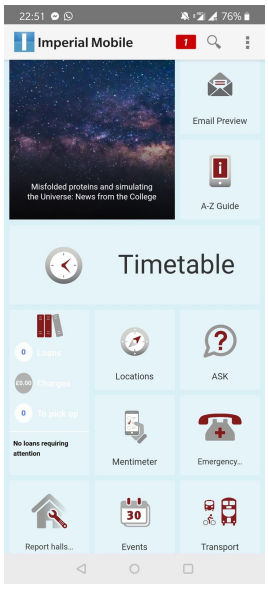


Figure 2.2

We interviewed a third-year Romanian medical engineering student at Imperial College London[1] to learn more about their experience with the official university app (fig. 2.2). They admitted that the app played an essential role in their adaptive process in the first year of university because it provided valuable information about student halls of residence as well as incoming freshman events. They also pointed out that the app's primary use for them and their peers is to find out the timetable, but they were bothered by the fact that it was a static table rather than a customizable one. Additionally, they mentioned that they wished they could import the events to any calendar application of choice (e.g., *Google Calendar*[2], *Microsoft Outlook*[3]).

[1]https://www.imperial.ac.uk/
[2]https://calendar.google.com/
[3]https://outlook.office.com/

A graduate US computer science student at the Massachusetts Institute of Technology[4] that we interviewed shared that the most used features of their university's app (fig. 2.3) are the map, shuttle schedule, and dining menus. They also pointed out that the app is most useful (and therefore used by) freshman students. In contrast, usage for older students is limited to the first day of classes if they have a class in a new building, or more regularly to check menus in dining halls. Furthermore, the student mentioned that the most significant problems they had with the university app were performance issues and outdated or inaccurate information (i.e., shuttle showing as running when it is not, or marked in the wrong location).

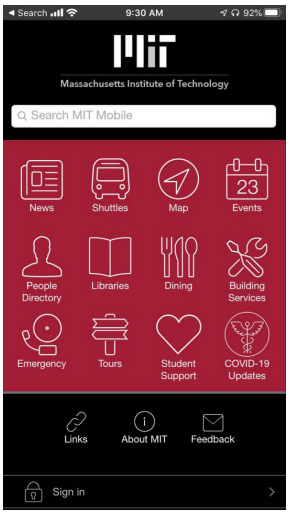


Figure 2.3

An interviewed German computer science student at the Digital Engineering Faculty within the University of Potsdam[5] described a different experience. The faculty app is currently under development by a group of students, including themselves. The application provides information about courses, faculty news as well as food options in the university's canteen and café, the last of which the student believes to be the most useful.

2.1.3 Pricing

It is worth noting that, while some universities rely on teams of students and professors to develop their app, others prefer to hire professional teams offering development solutions. The costs of developing such an app vary, since countless companies are offering specific services for universities - for example, as of May 2020, *Guidebook*[6] prices start at $3,200, while *buildfire*[7] offers different plans that range from $2,500 up to $7,500 depending on requirements. Many companies opt against listing their prices publicly in favor of setting an appropriate price based on the specific client's needs (e.g., MODO[8], Lets Nurture[9]).

[4]https://www.mit.edu/
[5]https://www.uni-potsdam.de/en/digital-engineering/
[6]https://guidebook.com/gb/pricing/
[7]https://buildfire.com/pricing/white-glove-pricing/
[8]https://www.modolabs.com/products/modo-campus/
[9]https://www.letsnurture.com/services/mobile-app-development.html

2.2 Generic education applications

The market also offers a number of highly customizable class management solutions for students of all ages, such as *School Assistant*[10], *myHomework*[11] and *ClassUp*[12] (all of which have over one million downloads on the Google Play Store[13] alone).

2.2.1 Platforms

The availability of these applications ranges from single-platform or single-OS (*School Assistant* - Android devices only) to multi-platform or multi-OS (*ClassUp* - Android and iOS devices) and up to fully cross-platform (*myHomework* has support for Android, iOS, Windows, macOS, ChromeOS and FireOS, as well as a web version that can be accessed by any web-enabled device).

2.2.2 Features

Most of these applications have a similar set of features, out of which the most notable are:

- customizable timetable that can be synced with other calendar applications, with the ability to define holidays and types of classes (fig. 2.4)
- assignment/homework/evaluation trackers with custom reminders
- tracker for class information such as professors who teach it, grades, associated events and notes (fig. 2.5)

Other features include automatically muting the phone during classes, importing/exporting data, and customizing the looks of the application.

The user needs to manually input and modify this information at the beginning of each semester and throughout the year as new events arise or changes are made.

[10] https://www.school-assistant.com/
[11] http://myhomeworkapp.com/
[12] https://classup.plokia.com/
[13] https://play.google.com/

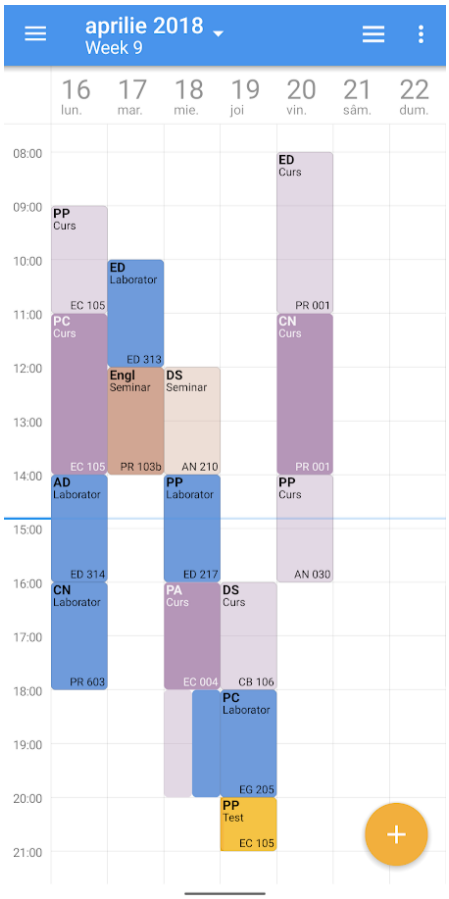


Figure 2.4: School Assistant: timetable page

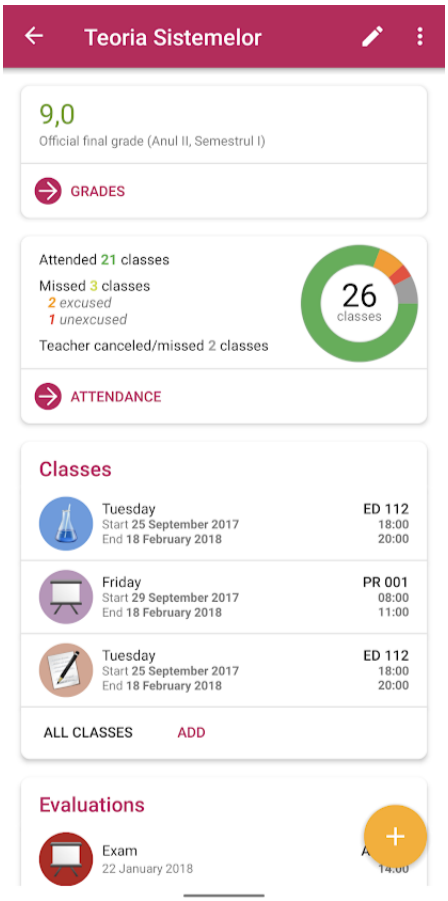


Figure 2.5: School Assistant: class details page

2.2.3 Pricing

Unlike university-specific apps described in section 2.1, where costs are supported entirely by the university, and usage is free for the students, generic applications usually require each student to pay for themselves. As is the trend with a lot of mobile applications nowadays, most of them are free initially (and might contain ads) and either offer a trial period or a limited number of features until the user upgrades to a paid version. *School Assistant*, for instance, has another version of the app with more features, *School Assistant +*, with an upfront cost of $2.75. *myHomework* has a subscription-based model: it provides a free version with limited functionality and advertisements, as well as the ability to upgrade to a *Premium* version for $4.99 a year. *ClassUp*, on the other hand, is entirely free and has no ads.

2.2.4 Caveats

For users, the biggest issue with these apps, stemming from their customizability, is that *they need to be set up to be usable*. Unlike university-specific apps described in section 2.1, which generally "just work" as soon as you log in using your university credentials, for generic applications to be useful, the students need to manually input all of their classes, assignments, professor information, and others.

Depending on the app, as well as the amount and complexity of the information that should be inputted, this process can initially take from a few minutes to a few hours and requires active updating by the student throughout the academic year (as new assignments are issued, classes change, etcetera). For most students, this is often more effort than it is worth. We observed this in our user study (described in chapter 3), in which out of over 200 students, only three said they were using a class management app (specifically, the three applications we chose as examples in this section).

2.3 Other productivity applications

To stay on track, well-organized students often use productivity applications that are not necessarily targeted for university specifically, but which get the job done. Our user study (further described in chapter 3) points out that out of over 200 students, about 60 regularly use *Google Calendar*[14] or another calendar app to track their university tasks. 14 students mention using other productivity apps such as *Trello*[15] and other note-taking or todo-list applications (e.g. *Google Keep*[16], *ColorNote*[17]). Another collaborative productivity application worth mentioning is *monday*[18]. Although marketed for students and professionals, it is most suitable for work and school team projects, and not for tracking daily tasks and classes.

The main reason why only 30% of students opt for using a productivity app is that these have the same main disadvantage as generic university apps (described in section 2.2.4), which is that they require active management and involvement.

[14]https://calendar.google.com/
[15]https://trello.com/
[16]https://keep.google.com/
[17]https://www.colornote.com/
[18]https://monday.com/lp/students/

To keep track of their tasks, the other 70% of students rely solely on their memory, reminders from colleagues and the limited resources provided by the university, such as the timetable (a simple spreadsheet) and the assignments listed in the Moodle calendar (which is only used for specific classes).

2.4 Existing applications for our university

Our goal being to "fill in the blanks" and provide students with tools that they would need to make their lives easier and that they do not already have, we looked into the existing applications (or existing attempts) for students in our target group.

In the past few years, we have seen an increasing interest in providing a mobile application for students, with more and more students implementing small applications aiming to help their peers in various ways, and presenting them as projects for various classes, conferences (notably, the UPB Students Scientific Communications Session[19]) and even their Bachelor thesis (fig. 2.6).

However, most of these projects do not pass the stage of proof of concept, due to lack of time from the initiating students and lack of support from the university (see subsection 2.4.1). To find out more, we interviewed some of these students and will be describing their work in subsections 2.4.2 - 2.4.4.

2.4.1 History

As far as UPB and ACS are concerned, new platforms have a history of either not being officially accepted, or going through an extremely long process to be adopted eventually.

2.4.1.1 Current platforms

The university's current platform for students to manage and view their enrollment information (including contracts, grades, and accommodation), *studenti.pub.ro*, started back in 2007 as a small platform developed by students for managing housing requests for the university's student halls of residence. More much-needed functionality was added as time passed, and it slowly became an

[19]https://upb.ro/sesiunea-de-comunicari-stiintifice-studentesti-2020-online/

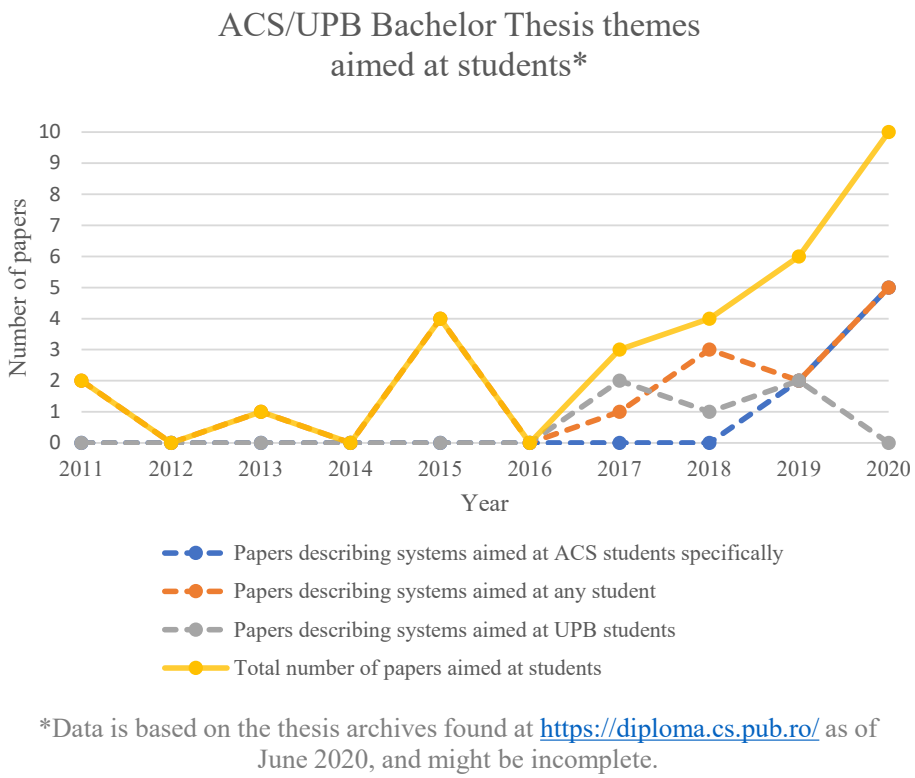


Figure 2.6: ACS/UPB Bachelor Thesis themes aimed at students

integral part of any UPB student's life. However, it has not seen a significant update in a long time, with features such as requesting proof of enrollment, graduation, or other documents being greyed out and marked as *coming soon* for the past few years.

In 2009, an ACS professor introduced the university's Moodle platform through an EU-funded project called "E-learning and e-content curriculum platform for technical higher education" (project number 154/323, SMIS code 4428). According to the project's official website[7], the percentage of university courses providing digital (e-learning) materials grew from 9% (170 courses) up to 35% (600 courses) thanks to the project.

2.4.1.2 COVID-19 response

Due to the 2020 pandemic with the new coronavirus (SARS-CoV-2), UPB moved all classes and evaluations online starting in March. This situation led to a sudden increase in usage for the university and the faculty's online platforms. Professors are now required to post materials and assignments on Moodle, as

opposed to before the pandemic when they could choose. With students having no other choice but to use the platform and attend online classes on Microsoft Teams[20], the number of users per hour soared. This spike initially led to issues, due to the platform servers not being able to process such a large number of requests.

Furthermore, some long-awaited changes came into effect thanks to the pandemic. For instance, requesting a document such as proof of enrollment could finally be done virtually by sending an e-mail, as a result of the university building being closed for students due to the pandemic. This activity previously required personally dropping off a hand-written request during the Student Office hours (9-11 am).

Ultimately, we believe that this experience led to a massive step in improving digitalization within the faculty and university, and we hope that future adoption processes will run more smoothly.

2.4.2 Navigation tools

Since UPB prides itself with the largest campus in Romania[21], navigating the tens of buildings can be difficult for new students. While the map[22] helps, students often find themselves in need of a more interactive navigation solution. It should come as no surprise that computer science students would try to handcraft such a solution.

In April 2019, a group of students attempted to create an Android app that provides indoor navigation instructions[23], as a group project for their software engineering class. The project, however, remained in the state of a simple proof of concept (fig. 2.7), since the lack of existing digital data of the room layout in the campus buildings meant that they had to manually write the information in the form of a graph in a JSON[24] file. This process proved to be extremely tedious.

Roughly a year later and entirely separately, a student from the Faculty of Engineering in Foreign Languages[25] within UPB, with a passion for programming,

[20]https://teams.microsoft.com/
[21]http://international.upb.ro/studenti-internationali/
[22]http://international.upb.ro/campus/transport-eng/
[23]https://github.com/IrinaM09/UPB
[24]https://github.com/IrinaM09/UPB/blob/master/app/src/main/res/raw/nodes.json
[25]http://ing.pub.ro/en/

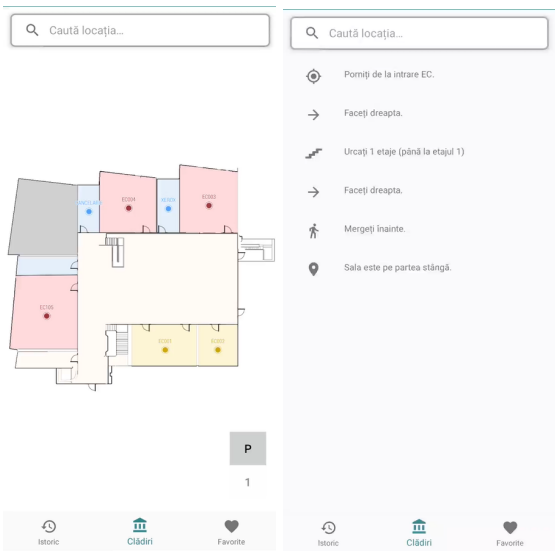


Figure 2.7: Unpublished UPB campus map application

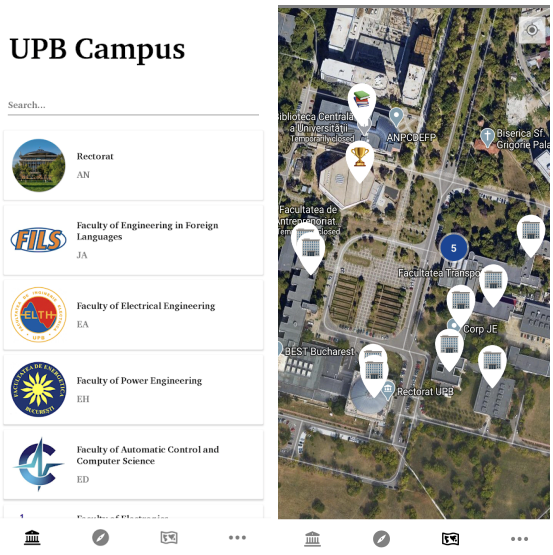


Figure 2.8: Published UPB campus map application

wrote a paper[8] for the UPB Students Scientific Communications Session. Consequently, they won a prize and published a campus map mobile application for both iOS and Android devices (fig. 2.8), using the Apple Maps[26] and the Google Maps[27] APIs respectively.

2.4.3 Timetable/event tools

Throughout the years, students have attempted to automatically extract class information from the university-provided timetable, in order to add the events to their calendar application of choice more easily. A Master's student made one such attempt in January 2019 as a project for a mobile computing class[28]. However, the main difficulty they all encountered was that the timetable files are manually-written spreadsheets that do not have a set format. As a result, there is no way to fully automate the extraction of events from these files unless the professor in charge of designing the timetables agrees to standardize the whole process.

In September 2019, a student who was tired of having to rely on colleagues or spend a significant amount of time browsing *Facebook* to find out what relevant

[26]https://developer.apple.com/maps/
[27]https://developers.google.com/maps/documentation
[28]https://github.com/laurentiustamate94/timetable-app

events are taking place in or related to the university published an application called *Politehnik*[29]. This app promised to offer an easy way for students to keep track of what events are happening, as well as get notified about relevant upcoming events. Although the app was solely promoted through faculty WhatsApp groups and that the features offered are only useful for a handful of students, the application saw much interest among students. Usage statistics are pictured in fig. 2.9, with the first steep rise being caused by the initial peer to peer promotion and subsequent spikes by in-app notifications. The app's events are added manually by the developer, with users being able to suggest events through the app.

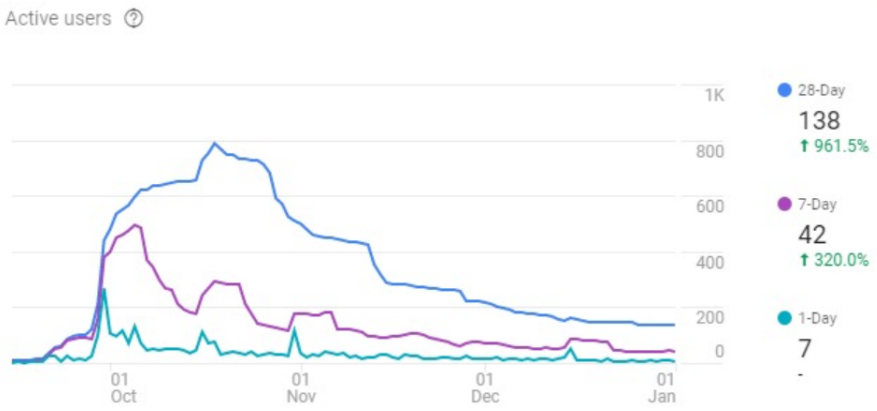


Figure 2.9: Active users on the application *Politehnik*, as reported by *Firebase*

2.4.4 Other tools

Albeit not a mobile application, a relevant platform as far as *tools made by students for students* go, is *CTI Info*[30]. It is a webpage made by a student in 2017, aiming to act as a small compendium with information that is particularly relevant for students pursuing the Computer Science domain (also known as CTI) within the faculty. This content acts as a good example of what kind of information our app should provide. It contains details about various platforms, student associations, a small FAQ section (including information about reading the timetable and a thoroughly explained campus map), as well as other useful resources.

[29]https://politehnik.ro/
[30]https://infocti.github.io/

2.5 Mixing and matching for our application

University apps (section 2.1) generally provide a vast amount of useful information (which may, however, sometimes be out of date) but lack customizability. In contrast, generic apps (section 2.2) are highly customizable but difficult to set up. Consequently, we believe that a *collaborative approach* would be the best middle ground.

ACS students have a long history of putting materials together in order to help each other, be it through *WhatsApp*[31] or *Facebook*[32] groups, or through various websites[33] or shared *Google Drive*[34] folders. Furthermore, thanks to our user study (described in chapter 3), we know that we can safely assume that there will always be at least one student who takes the time to create events for themselves as they arise regularly. We can take advantage of this and provide these students with a way to easily share the events with their colleagues. In the unlikely event that there is a group where not a single student naturally takes these notes, the task can fall to the group representative, or even better, all students in the group can contribute. As ideal as that may seem on the surface, the implications are far-reaching. For starters, as seen on social media platforms nowadays (e.g., *Facebook*[32] groups, *Reddit*[35] subreddits), any platform where multiple people can collaborate requires a content moderation system[9] to function reliably, a subject we will discuss more in section 4.3.

A collaborative system would also mean that we do not need to rely on a single person (such as the developer in the case of the app *Politehnik*, described in subsection 2.4.3) or a small group of people (the app administration staff in the case of most official university applications, as seen in subsection 2.1.2) to keep the information relevant and up to date. As benefits of collaborative learning have long been proven[10], we could assume that, if enough users without malicious intent (further discussion in section 4.3) contribute to a data set, customizability is intrinsic, and data is unlikely to be inaccurate or out of date.

[31] https://www.whatsapp.com/
[32] https://www.facebook.com/
[33] http://andrei.clubcisco.ro/, http://exams.ro/
[34] https://drive.google.com/
[35] http://reddit.com/

Chapter 3

User study

Before designing the application, we needed to find out more about the needs and wants of students enrolled at ACS and the general context in which this application would function.

3.1 Methods

The research and data collection methods we used were **observation** (throughout the four years that the main author of the paper was enrolled at our faculty of choice, ACS), **interviews** (with students, professors as well as administrative personnel of the faculty) and a **survey**, the results of which we will be focusing on in this chapter. We mention and discuss the results of the first two methods throughout the whole paper, where they offer insightful information for the specific section. Additionally, throughout every step of the whole process of designing the application, we went through a **feedback loop** with a small group of students to ensure that we were progressing in the right direction.

Partly due to the pandemic, the main method of communication with our potential users was online communication. Most of the interviews were done either via video call (through *Microsoft Teams*[1]) or via chat (*Facebook Messenger*[2], *WhatsApp*[3]) or e-mail (*Gmail*[4]). The survey was a *Google Form*[5] shared both verbally

[1]https://teams.microsoft.com/
[2]https://messenger.com/
[3]https://www.whatsapp.com/
[4]http://gmail.com/
[5]http://forms.google.com/

and through social media platforms, receiving 212 responses between October and December 2019. Since we regrettably could not gather a relevant number of responses from Master's students, with only seven responses, we will be focusing on the Bachelor's students, since 200 out of the total of about 600 enrolled B.Sc. students is statistically relevant.

To analyze the data, we imported the results from the form into a database using *Microsoft SQL Server*[6], normalized the data and created visualizations using *Microsoft Power BI*[7].

3.2 Target audience

Studies[11][12] show that demographic information in surveys helps to avoid bias towards a specific section of the target audience. Therefore, we included some **demographic questions** to ensure that the pool of respondents is relatively balanced.

In terms of **gender demographics**, our results (fig. 3.1) closely match a separate study[13] that is specifically targeting gender demographics at computer science faculties in Romania, including ACS.

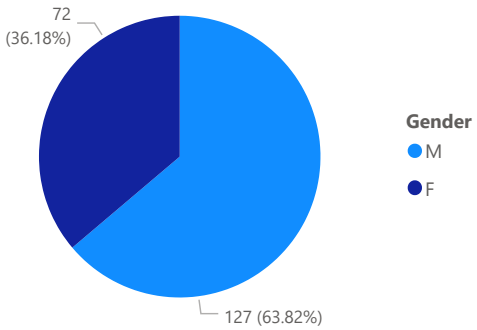


Figure 3.1: Gender of survey respondents

Concerning **academic distribution**, we have received four times more responses from students pursuing the Computer Science domain (Computers and Information Technology, or CTI) than the Automatics domain (Systems Engineering, or IS), as seen in figure 3.2. Some discrepancy is expected, considering there are twice as many enrolled CTI students than IS students, according to the same

[6]https://www.microsoft.com/en-us/sql-server/
[7]https://powerbi.microsoft.com/

study[13] mentioned above. The distribution of the respondents' year of study is relatively balanced (fig. 3.3).

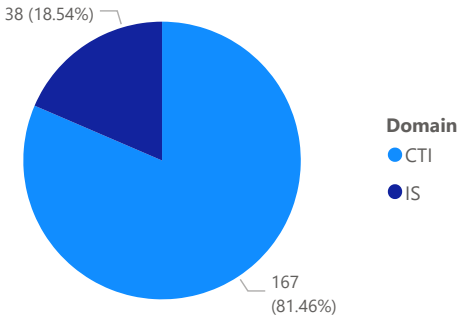


Figure 3.2: Domain of survey respondents

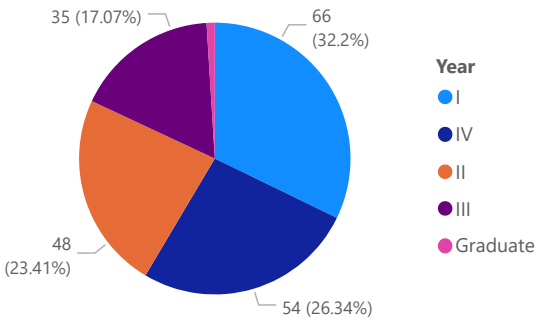


Figure 3.3: Academic year of survey respondents

Regarding **mobile operating systems**, the distribution is very similar to the Romanian mobile operating system market share, according to StatCounter GlobalStats [14], with 80% of students using Android and 20% of students using iOS.

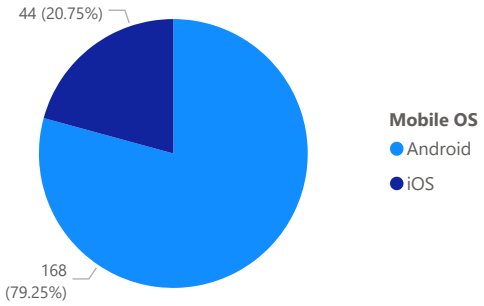


Figure 3.4: Mobile OS of survey respondents

Upon reviewing the data, we believe that, aside from the lack of representation for Master's students, the respondents' demographics are varied enough for a relevant sample group. Since the student group we expect the app to be the most useful for is first-year B.Sc. students, we will be focusing more on their needs, since the app can be further extended in the future to meet the needs of older students, where these needs differ.

3.3 Results

We used a variety of questions to learn more about the student's preferences and what they would be looking for in a university mobile application. For this purpose, we included **multiple-choice questions**, **Likert scale questions** and **open-ended questions** in the survey.

3.3.1 Features

The survey's primary purpose was to find out what kind of features would be the most useful for our students. We provided a list of potential features or types of information that could be available in the application. We asked them to give each of them a rating from 1 (not very useful) to 5 (very useful). We devised a scoring system, with points between -2 and +3 for each answer, and came up with the hierarchy seen in figure 3.5.

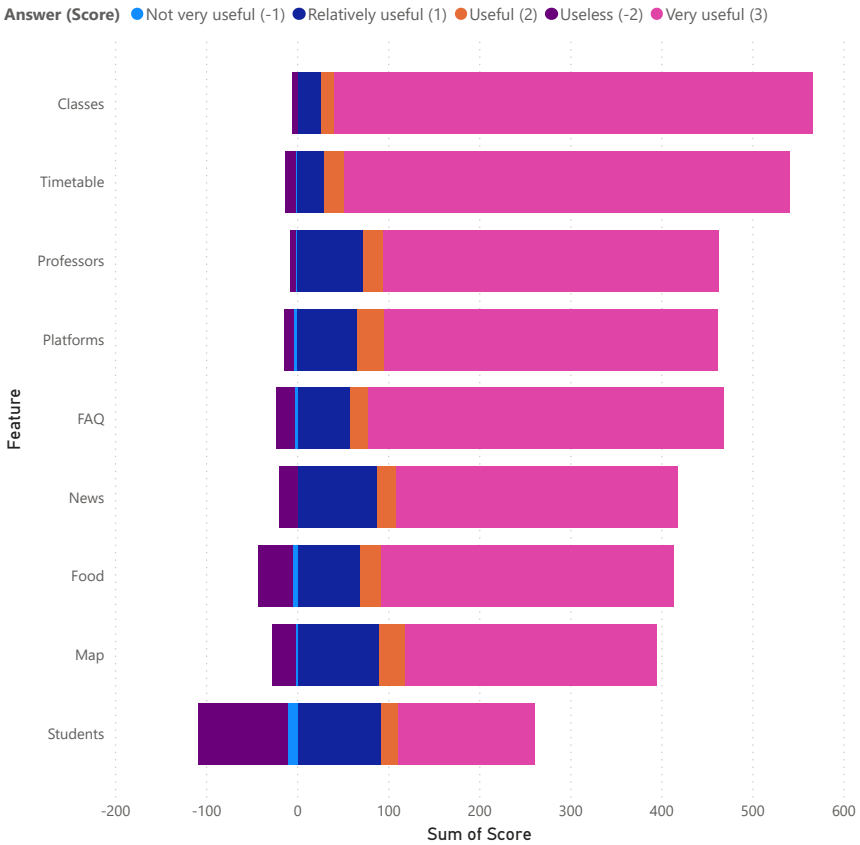


Figure 3.5: Preferred app features for survey respondents

The vast majority of students considered information and resources for classes and a timetable to be the most useful features.

In an open-ended question, students suggested other features, such as an estimated waiting time for the canteens and a way to upload files for easy access. In terms of useful information that should be available in the app, some students suggested administrative information (about scholarships, or how to request specific documents) and feedback from other students about different classes and professors.

3.3.2 Appearance

To find out how much we should prioritize coming up with an outstanding design for the application, we asked the students how important they believe the general appearance of an application to be. As seen in figure 3.6, a vast majority of students believes appearance to be important or very important. Therefore, we need to focus as much as possible on the design step of the development process, described in chapter 4.

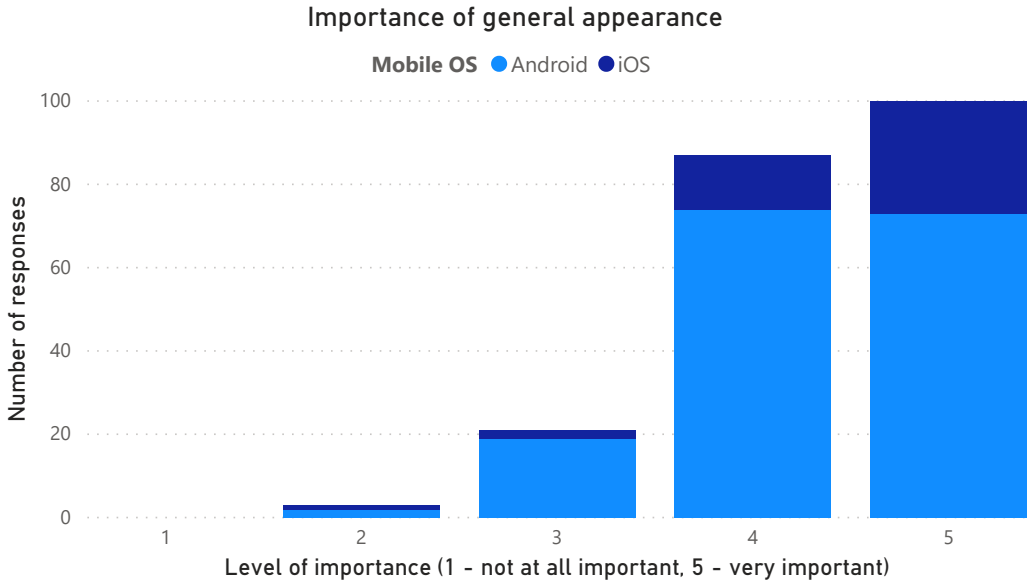


Figure 3.6: Importance of general appearance by OS for survey respondents

In an open-ended question, several students stressed that performance, as well as how smoothly an application works, are also essential factors. Additionally, some students mentioned that they prefer a simple, easy to understand menu

that does not require a user to go through more than three steps to achieve a specific goal. Through this, they were referencing the 3-click rule of web design (as defined by Catherine E. Weeks[15], "A person shouldn't have to click more than 3 times to find a piece of info"). However, we will not be focusing specifically on this arbitrary UX requirement, because multiple studies[16][17] have proved it to be nothing other than a myth.

One difficulty in designing a cross-platform application is that different platforms have different design guidelines that tend to be quite different[18]. To help decide on a common design language for the application, we asked students how important native appearance is for them in a mobile application. We split the answers based on the student's mobile OS of choice, as seen in figure 3.7. A native look and feel to an application is more important for iOS users than for Android users. However, it does not seem to be as important as the overall appearance of an application. A visual comparison between native and non-native design can be seen in appendix A.

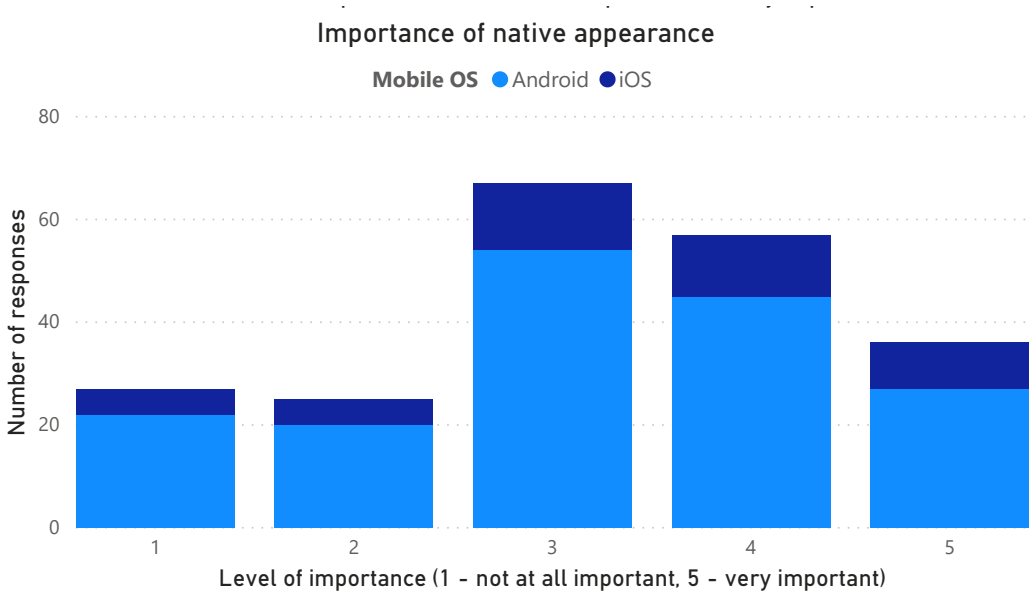


Figure 3.7: Importance of native appearance by OS for survey respondents

Based on these results, we will attempt to come up with a smooth, intuitive interface that is based on but does not strictly abide by the design rules of either iOS (Human Interface Guidelines[19]) or Android platforms (Material Design[20]). We will make it our goal to find a proper middle-ground solution for our cross-platform application.

3.3.3 Language

Finally, we asked students what language they prefer their mobile OS to be in, to find out whether we should prioritize localization for our application. Somewhat surprisingly for a faculty with almost exclusively native Romanian students, more than half of respondents said that they prefer their mobile language to be English, as shown in figure 3.8. Only 2% of respondents chose a different answer, with only one student mentioning German and the others saying that they do not prefer one language over the other.

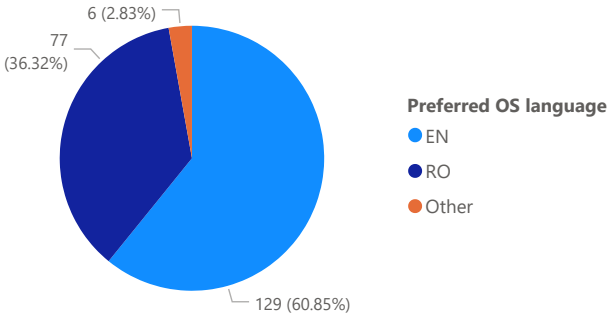


Figure 3.8: Preferred OS language of survey respondents

With this data in mind, we know we need to pay close attention to our application's quality of localization. This feature is not just for the very few international students studying at ACS, but also for the many Romanian students who prefer their mobile devices and applications to be in English rather than their native language.

Chapter 4

UX & UI design

Because our survey indicated that we need to pay special attention to designing the interface of our application (see section 3.3.2), we decided to take an iterative approach to defining the User Experience (UX), and, by extension, sketching the User Interface (UI).

4.1 Requirements

The proposed functionalities of our application are listed in section 1.3. We will aim to reach our goals, as described in section 1.4.

We will be designing a collaborative application (a decision motivated in section 2.5) with a moderation system based on permission levels, described in this chapter in section 4.3.

Based on the results of our survey (section 3.3.2), our design language of choice will be neither Google's Material Design[20], nor Apple's Human Interface Guidelines[19], but rather what we consider to be a healthy mix of both. This design aims to be pleasant for both Android and iOS users, as is required of a cross-platform application which aims to have a consistent design on all platforms. A good example of another such application is *Reddit* (available on both the App Store[1] and Google Play[2] with a very similar design that only slightly differs in the web version[3]). However, since it sports a platform-agnostic design, independent developers have come up with separate applications for viewing *Reddit* content

[1] https://apps.apple.com/us/app/reddit/id1064216828

[2] https://play.google.com/store/apps/details?id=com.reddit.frontpage

[3] http://reddit.com/

on Android and iOS devices, which claim to have a more "native" look for the more nitpicky users. A visual comparison between these applications can be seen in appendix A.

4.2 Paper prototyping

4.2.1 Initial concept

We kick-started the design process by making a rough sketch of what we imagined the app to look like, on paper, with the features we had in mind in the beginning. We used the *School Assistant* app (see section 2.2) as the main inspiration for this first design.

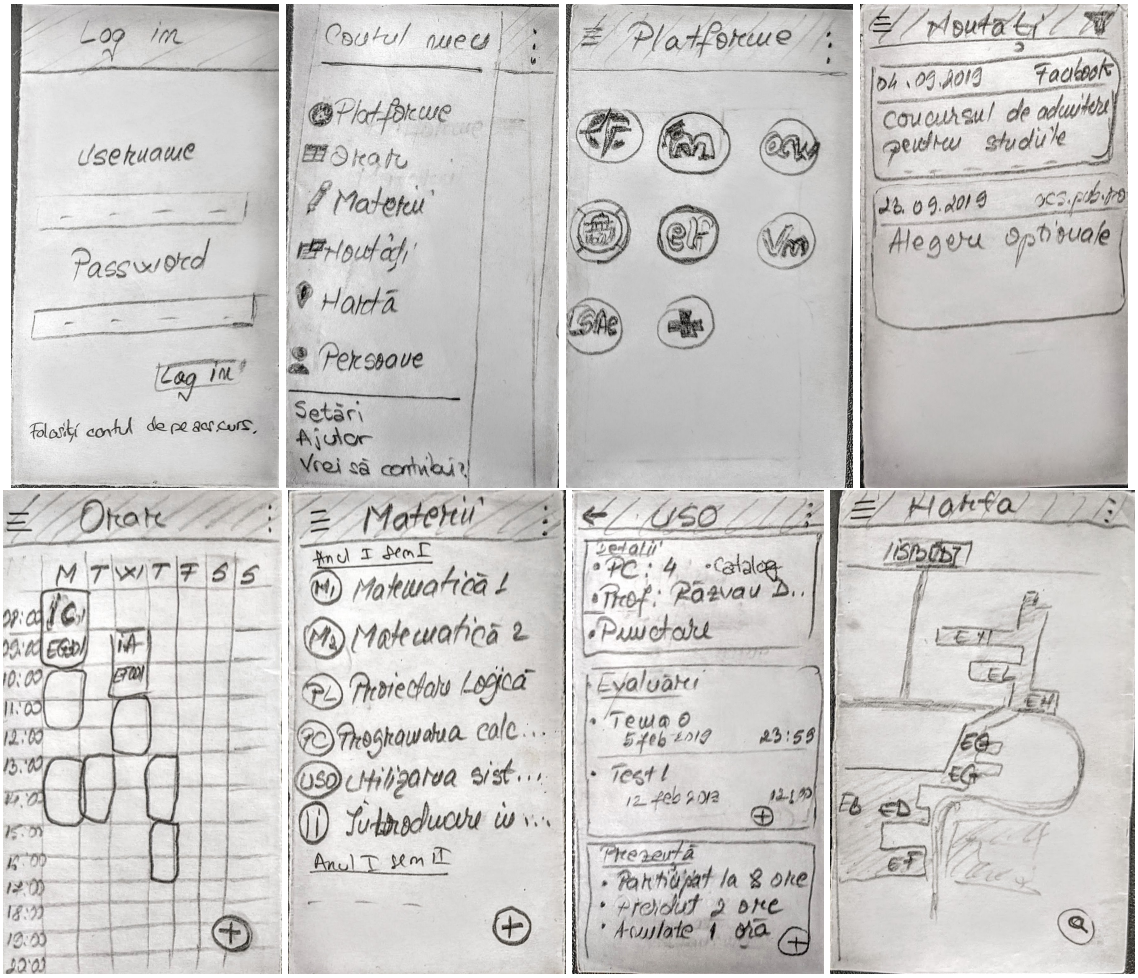


Figure 4.1: Paper prototype

Figure 4.1 shows the very first sketches of the application interface. The pages pictured, from left to right and top to bottom, are:

- the login page the user would see when first opening the app
- the Material[20]-style side drawer with the different pages in the app, available once the user logs in
- a list of the university's platforms, which would serve as the default page the user would see without accessing the drawer
- the "News" page with information from various sources, such as relevant Facebook pages and the official website
- the timetable, with a button to add a new event
- the list of classes, organized by year/semester, with a button to add a new class
- information about a class that a user would see when clicking on an event or class
- the map of the campus, with a button to search for a specific location

4.2.2 Feedback

To find out what students would think about this initial concept, we organized a focus group with about 15 students and presented each of them the workflow of our application. Regarding **authentication**, we have received feedback regarding the need for a profile page for the user. The possibility for a user to use the app without having to authenticate was also mentioned. Multiple students pointed out that having the "Platforms" page as the default was counter-intuitive, and that a **designated "Home" page** would be a better solution. When asked what they think this homepage should contain, they suggested quick actions and upcoming events; in other words, an overview at a glance of the other pages within the app.

Some students pointed out that the homework-type events should have both **a soft and a hard deadline** associated with them, as is the norm in the faculty. Additionally, they wanted to see how the process of adding a new event would work.

4.3 Permissions & moderation system

4.3.1 The need for moderation

As mentioned in section 2.5, any collaborative system involving more than a few users requires some form of moderation[9]. This system is meant to ensure that content stays accurate and up-to-date and avoid offensive/inappropriate content.

Based on the categories described by a *Bridged.co* writer[21], we will be using a mixture of **pre-moderation** and **reactive moderation**, meaning that content will be verified before appearing on the platform, and users can report or flag content that they believe to be inappropriate after it has been posted.

Since the platform should eventually be self-sustainable, without the need of constant involvement from the developer in content moderation (as is the case for the nonscalable solution of the application *Politehnik*, described in section 2.4.3), we need a way to offer students the ability to moderate content on the platform themselves. This situation begs the question - how do we decide who should have the right to moderate content, and who should not?

4.3.2 Permission levels

To establish who should have certain rights, we need to define the rights that are to be granted. We have designed a permission-level system, where each user has a permission number, which indicates what they can do within the application. Each level includes the permissions of lower levels.

- **0** - no special permissions; user can read public (already filtered) content and add private content or make suggestions
- **1** - helper level; user can view suggestions made by other people and vote up/down depending on whether they believe the suggestion to be appropriate or not
- **2** - basic moderator level; user can approve suggestions (an approved suggestion would be made public) and post content directly (without needing review); they can only delete content they posted or approved

- **3** - complete moderator level; user can dismiss suggestions (dismissed suggestions would be deleted from the suggestions page and no longer visible to anyone) and delete content added by anyone
- **4** - administrator level; user can change other users' permission level

4.3.3 Assigning permissions

By default, any new user would have the first permission level (0). Initially, only the developers would have level 4, allowing them to assign special permissions to trusted students in their circle. Additionally, students who have a special status within the faculty (group/series representatives, certain members of student associations) can request to be granted level 2 to inform their fellow students easily.

In the future, a manually-defined permission system might prove not to be scalable. In this case, one potential solution would be for users' permissions to gradually increase as they contribute quality content to the platform (e.g., make/vote up suggestions that end up being approved, vote down suggestions that end up being dismissed). This system could be implemented into the application in a **gamified** manner (e.g., unlocking achievements and earning points by doing specific actions). Game-like platforms have a history of motivating students to participate actively. One such example within the faculty is *World of USO*[4], a competitive knowledge game played throughout the first semester by first-year students for the Introduction to Operating Systems (USO) class.

4.4 Wireframe

4.4.1 Prototype

Following the focus group, we noted the feedback and created a more detailed, interactive (clickable) wireframe using Balsamiq[5]. We re-drew the login page (fig. 4.2), drawer (fig. 4.4) and websites page (fig. 4.5), and introduced a home page (fig. 4.3), as suggested by the feedback described in section 4.2.2. The latter

[4]https://wouso.cs.pub.ro/
[5]https://balsamiq.com/

includes a list of frequently accessed websites, as well as upcoming events and tasks.

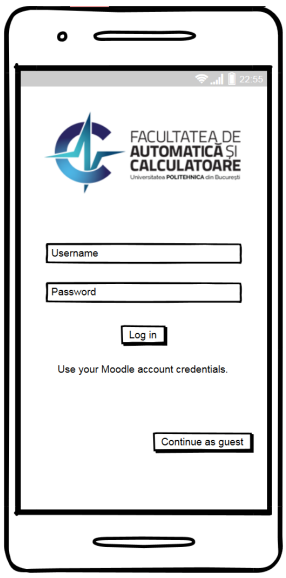


Figure 4.2: Login page wireframe

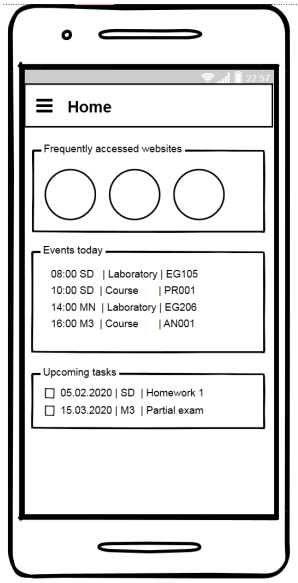


Figure 4.3: Home page wireframe

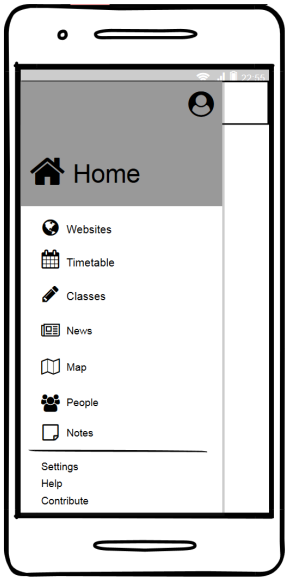


Figure 4.4: Menu wireframe

We also re-drew the news page (fig. 4.6) and added a visualization of the filtering option for this page.

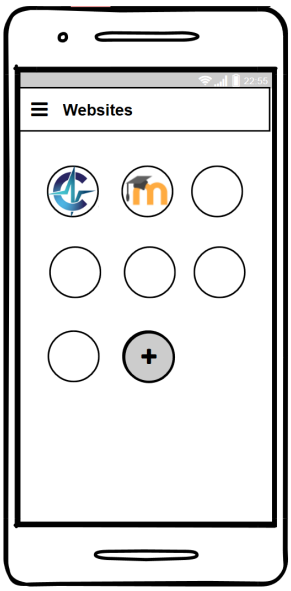


Figure 4.5: Websites page wireframe

Figure 4.6: News page wireframe

Additionally, we added the profile page (fig. 4.7) requested by the feedback. It includes a potential presentation of the gamification feature suggested in section 4.3.3. This concept displays the current level of the user as well as the requirements for progressing to the next one.

We also drew the "People" page containing professor and student information (fig. 4.8). It is pictured as a list of names and allows the user to click/tap a name to view more information about the person. It also allows filtering the list by type (student/professor).

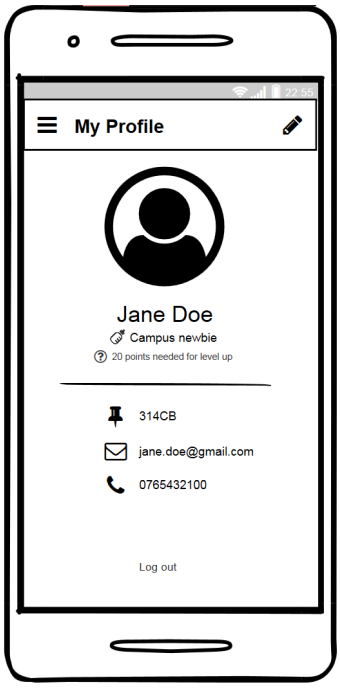


Figure 4.7: Profile page wireframe

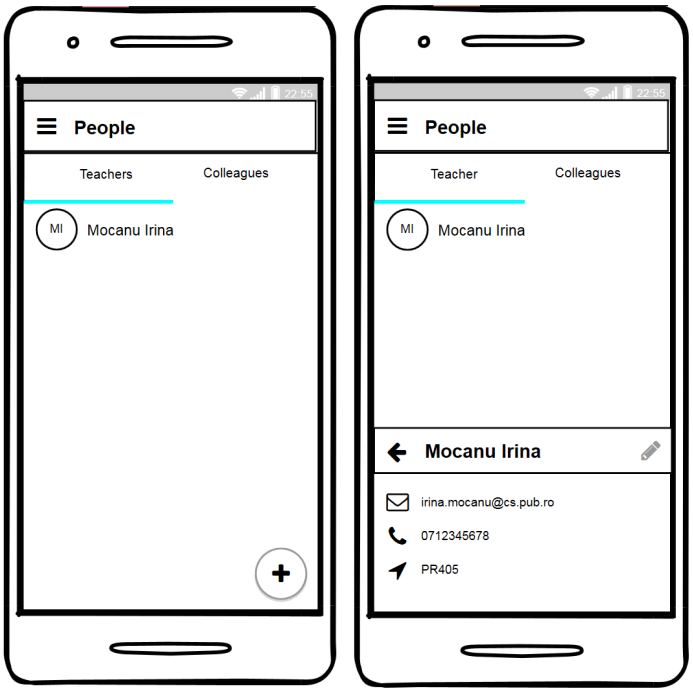


Figure 4.8: People page wireframe

Finally, we re-designed the classes and class information pages (fig. 4.9) and added visualizations for the filtering mechanism as well as the entire process of adding a new event to a class (fig. 4.10), as requested by the feedback.

Adding an event can be done by either manually inputting all data for the event (such as the name, class, and deadlines for homework) or using an event that somebody else has already created. Existing events can be voted up or down. They would be automatically voted up if a user chooses to add it to their calendar in the application.

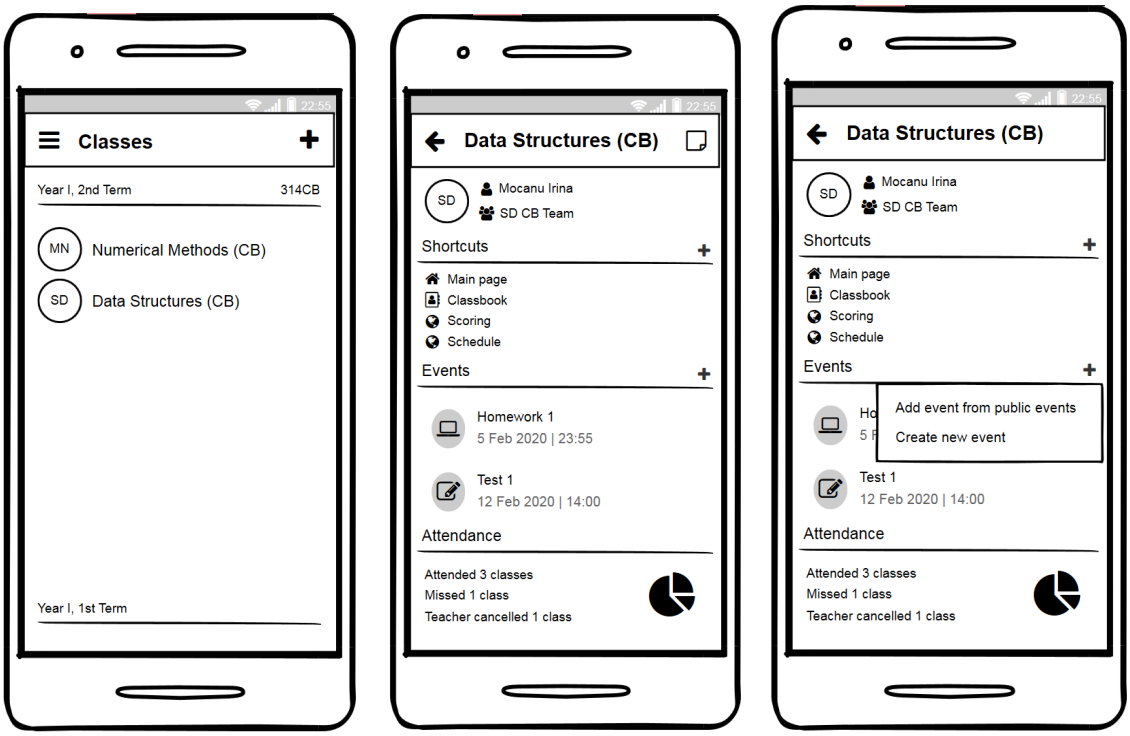


Figure 4.9: Classes page wireframe

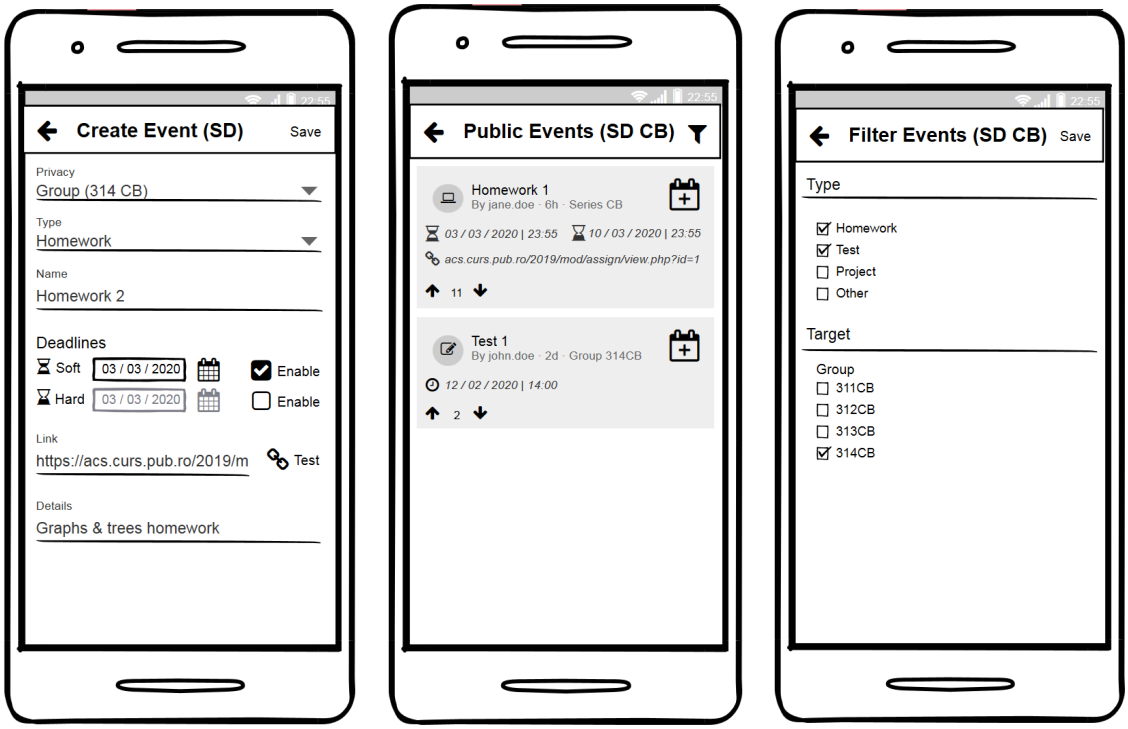


Figure 4.10: Events page wireframe

4.4.2 Feedback

We asked ten students from different years to test this prototype and share their experiences. The feedback we received was very positive, with all students showing excitement about the application's features.

The students who have iOS devices pointed out that the design style is more appropriate for Android than iOS devices, which was to be expected since the main inspiration for the prototype was an Android application which uses Material Design[20] (*School Assistant*).

Some students noted that a voting system on events could be abused since the user can ask their colleagues for votes.

4.5 Final design

4.5.1 Theme

Figure 4.11: ACS logo

For the final design, we created a dual theme (including dark & light mode) that is inspired by the faculty logo (fig. 4.11). The primary and secondary colors of the application are extracted from the logo and used for text, buttons as well as illustrations.

The font used in the application is Montserrat[6]. All icons are either Material Design icons[7] or FontAwesome icons[8]. Illustrations are from the open-source unDraw[9] gallery. All of the above are under open licenses and allow usage, modification, and redistribution.

4.5.2 Pages

For the sake of this paper, we will include only light theme / English screenshots of the most important pages in the application (figures 4.12 - 4.23).

We decided to drop the navigation drawer - hamburger menu (☰) combo (fig. 4.4) in favour of a bottom navigation bar (fig. 4.13). This decision is mostly due to the drawer not being an accepted pattern on iOS. Additionally,

[6]https://fonts.google.com/specimen/Montserrat
[7]https://material.io/resources/icons/
[8]https://fontawesome.com/icons/
[9]https://undraw.co/illustrations

in recent years, navigation drawers have sparked controversy[22] in the context of mobile gestures and many applications - including Google applications like Google Maps[10] - are removing the drawers entirely, opting for a bottom navigation bar instead.

The final design is otherwise fairly similar, UX-wise, to the wireframe concept, but adds many UI design elements such as a colour palette and illustrations on pages without a lot of content (such as the login/signup page - fig. 4.12 - and the settings page - 4.14).

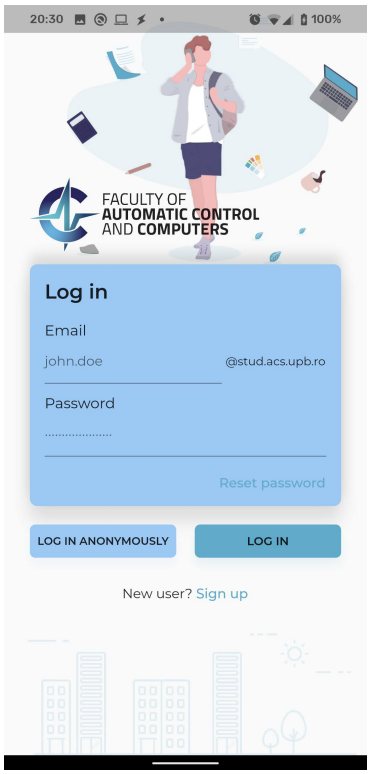


Figure 4.12: Login page

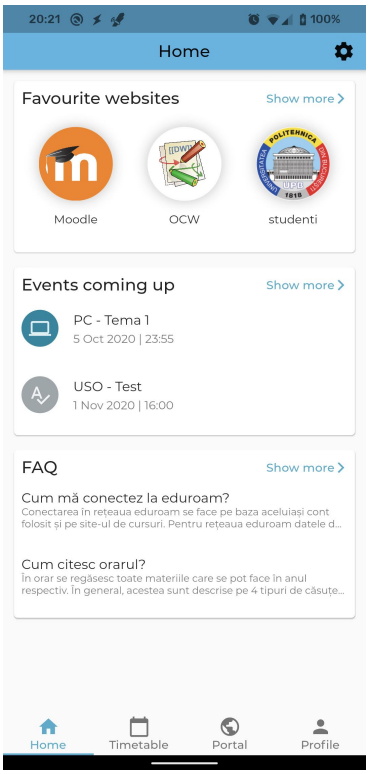


Figure 4.13: Home page

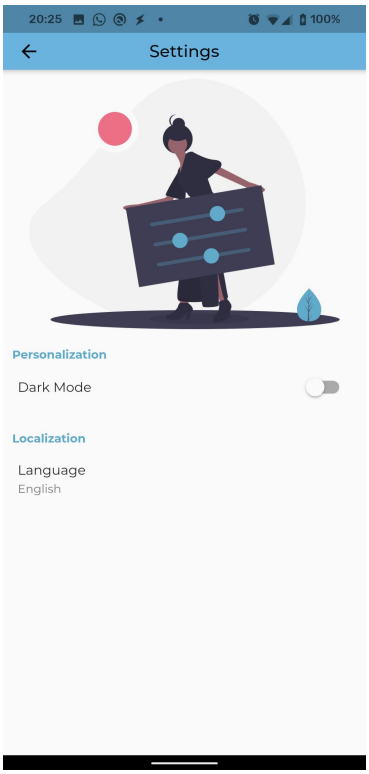


Figure 4.14: Settings page

The portal page (fig. 4.15) allows the user to tap a website to open a browser window following that link. Users can also add and edit websites (depending on their permissions, see section 4.3.2), as shown in figure 4.16, and filter them (fig. 4.17). The filter is a tree structure based on the student body structure described previously in section 1.2.2), where students can select any number of nodes they would be interested in.

[10]https://www.google.com/maps

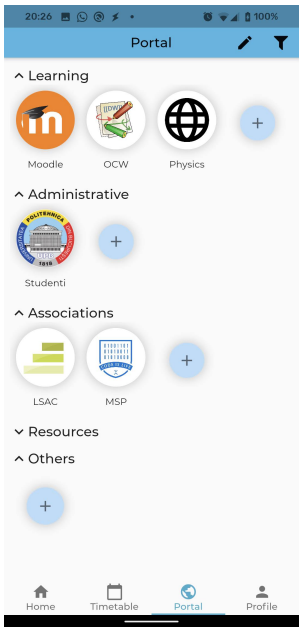


Figure 4.15: Portal page

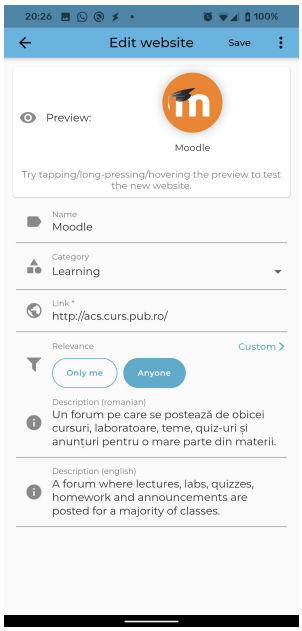


Figure 4.16: Edit website page

Figure 4.17: Filter page

The classes page (fig. 5.4) allows the user to view information about the specific class (fig. 4.19) and even the lecturer (4.20).

Figure 4.18: Classes page

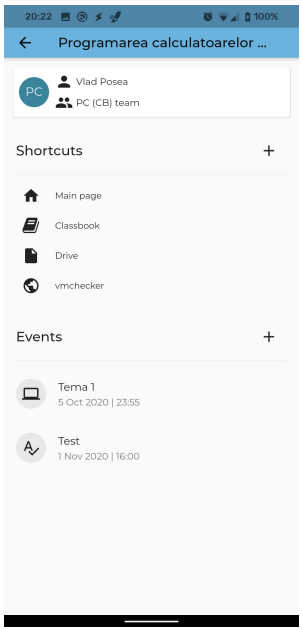


Figure 4.19: Class information page

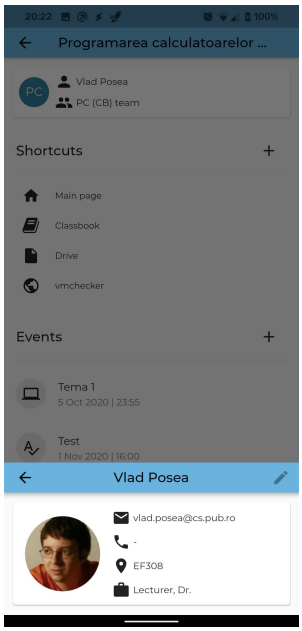


Figure 4.20: Lecturer information modal

The timetable page (fig. 4.21) acts as a specialized calendar for university activities. Users can add and edit various events (fig. 4.23) that can be linked to a specific class, as well as view details about existing events (fig. 4.22).

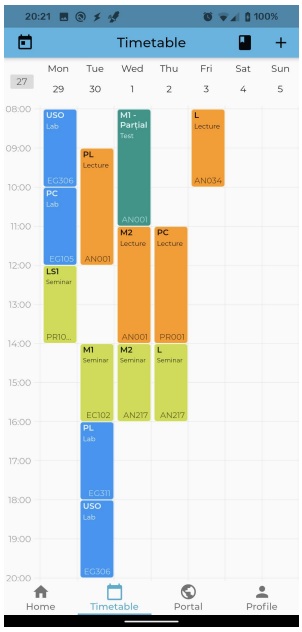


Figure 4.21: Timetable page

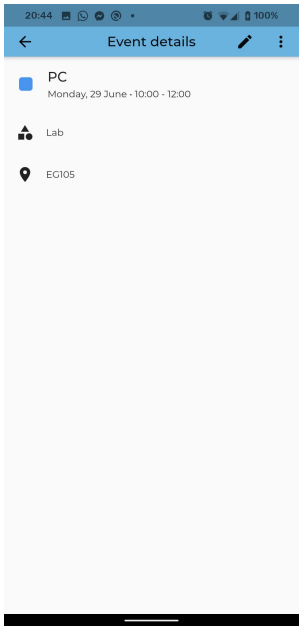


Figure 4.22: Event information page

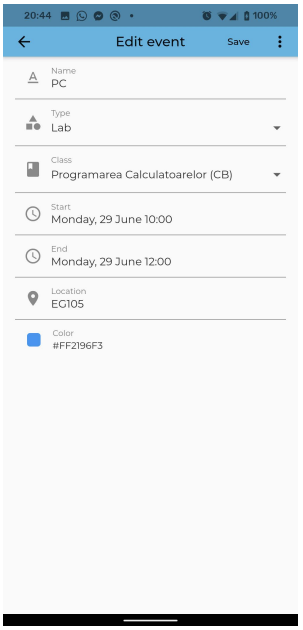


Figure 4.23: Edit event page

Chapter 5

Architecture

In this chapter, we will be discussing the technologies we chose to use for this project and our implementation.

5.1 Mobile technology

5.1.1 The native approach

In terms of the best core technology/language to use for the mobile application itself, we initially considered a native Android solution, since the vast majority of students in the faculty (80% according to our study, section 3.2) own an Android phone. However, since our application should be available to all students, and the quality of information is dependent on the number of users in a collaborative system, a pure Android solution proved not to be the best approach.

Native mobile implementations come with various benefits, from consistent, native-compliant application design (see appendix A) to outstanding performance and making the best of what the device has to offer. However, having only one supported platform would mean excluding a significant number of students from the user base. On the other hand, having two separate, native implementations for each mobile OS (one written in Java/Kotlin for Android, and one written in Objective-C/Swift for iOS) would require double the amount of time and effort. Due to the complexity of the application, the native approach is regrettably out of the question.

5.1.2 The cross-platform approach

Given the circumstances, we decided to look into cross-platform technologies for mobile applications. According to Sandeep Agarwal[23], the 5 most popular mobile app development frameworks in 2019 are React Native[1] (by Facebook), Flutter[2] (by Google), Ionic[3] (by Drifty), Xamarin[4] (by Microsoft) and PhoneGap[5] (by Adobe). Due to our developer not having any prior experience with JavaScript development, we decided to choose between Flutter and Xamarin. This decision is mainly influenced by the other three frameworks all being based on HTML5, CSS, and JavaScript. In terms of development language, Xamarin uses C# and .NET, while Flutter uses Dart.

Both Flutter and Xamarin have the significant benefit of being part of a vast development ecosystem, therefore choosing the technology also implies choosing the ecosystem. Flutter is integrated within the Google ecosystem (including support for services like Google Cloud[6] and a designated development environment: Android Studio[7]). Xamarin is part of the Microsoft ecosystem (offering alternatives such as Azure Cloud Services[8] and Visual Studio[9]).

Xamarin is a more "mature" framework with deep roots in the developer community since it was released in 2011, six years before Flutter came into the cross-platform market. However, Flutter's community is growing rapidly, and the developers' response is overwhelmingly positive. According to the 2020 StackOverflow Developer Survey[24], Flutter was rated the third most loved non-web framework and is the first mobile development framework in the list (with the second being React Native, and the third Xamarin).

Even though Xamarin provides support for more platforms (iOS, Android, Windows, macOS), it still requires platform-specific code for iOS and Android. For our use case (a mobile application), Flutter seems to be more appropriate since it provides better code reusability. The stable release of Flutter supports

[1] https://reactnative.dev/
[2] https://flutter.dev/
[3] https://ionicframework.com/
[4] https://dotnet.microsoft.com/apps/xamarin
[5] https://phonegap.com/
[6] https://cloud.google.com/
[7] https://developer.android.com/studio/
[8] https://azure.microsoft.com/en-in/services/cloud-services/
[9] https://visualstudio.microsoft.com/

iOS and Android, while the beta and alpha channels provide web support and macOS support, respectively. A web version of a Flutter application can be run on any web-enabled device (including Windows and Linux devices, for which native support is still under development). Therefore we believe that Flutter appropriately meets the need for our application to be readily available for any student.

Upon careful consideration of the leading cross-platform technologies available for developing mobile applications, we have concluded that **Flutter** is the best technology for our application.

5.2 Database

Being part of the Google ecosystem, Flutter offers seamless integration with the Google Cloud Platform and other Google services such as Firebase[10]. For mobile applications, Firebase offers features such as analytics, database, and authentication, therefore we have decided to take advantage of it for all of our cloud-based needs.

5.2.1 Firestore

Cloud Firestore[11] is a noSQL database that organises its data in *collections* and *documents*.

5.2.1.1 Data model

Collections are simply a list of documents, where each document has an ID within the collection.

Documents are similar to a JSON file, in that they contain different fields which have three important components: a *name* - what we use to refer to the field, similar to a dictionary key -, a *type* (which can be one of `string`, `number`, `boolean`, `map`, `array`, `null`, `timestamp`, `geopoint`, `reference`), and the actual *value*, the data contained in the field. In addition to fields, documents can contain

[10]https://firebase.google.com/

[11]https://firebase.google.com/docs/firestore/

collections, which in turn contain other documents, thus allowing us to create a hierarchical structure within the database.

5.2.1.2 Security

Firestore allows for defining specific security rules for each collection. Rules can be applied for each different type of transaction - `reads` (where single-document reads - `get` - and queries - `list` - can have different rules) and `writes` (where `create`, `delete` and `update` can be treated separately).

5.2.2 Authentication

Firebase provides an entire suite of back-end services and SDKs for authenticating users within an application, through FirebaseAuth[12].

For our application, we use the following features:

- account creation
- login
- password reset via e-mail
- account verification via e-mail
- account deletion

The class structure can be seen in appendix B, figure B.3. The `AuthProvider` class is responsible for communicating with FirebaseAuth.

This service automatically handles the authentication tokens and enforces security rules, which is particularly useful for an open-source application which users can fiddle with, such as ours. For example, multiple failed authentication attempts lead to a temporary timeout, and a user cannot delete their account unless they have logged in very recently (or refreshed their authentication token).

Firestore security rules (see subsection 5.2.1.2) can be enforced based on the user's UID. This method means that, even though users can access the database connection string through the public repository, they can only do a limited set of actions on the database, depending on whether they are authenticated and their rights.

[12]https://firebase.google.com/docs/auth

5.2.3 Project database structure

We opted for a database architecture where all documents within a collection can contain a subset of fields from a given list, similar to a relational database table. Some fields are required to be present (similar to `NOT NULL` columns in a *SQL* database), while others might not be present (similar to a column with the `NULL` property). This structure makes it easier to migrate to a relational database in the future, should the need arise.

The project database contains the following collections:

`users`

This collection stores per-user data. The document key is the user's `uid` (from FirebaseAuth, see subsection 5.2.2).

All the documents in the collection share the structure described in table 5.1.

Table 5.1: **users** collection structure

Field	Type	Required?	Additional info
group	`string`	☑	e.g. "314CB"
name	`map<string, string>`	☑	keys are "first" and "last"
permissionLevel	`number`	☒	a numeric value that defines what the user is allowed to do; if missing, it is treated as being equal to zero

A user can define their private events and websites, that only they can access. These will reside in the **events** and **websites** sub-collections, respectively, which have a similar structure as the root-level collections with the same names, described below in this section.

We define special ***rules*** (see subsection 5.2.1.2) for this collection and its sub-collections. Anyone can create a new user (a new document in this collection) if the *permissionLevel* of the created user is `0`, `null` or not set at all. Authenticated users can only `read`, `delete` and `update` their own document (including its sub-collections) and no one else's. However, they cannot modify the *permissionLevel* field.

`websites`

This collection stores useful websites, shown in the app under the **Portal** page. Which students they are relevant for depends on the *degree* and *relevance* fields (for more information, see the **filters** collection described below in this section).

The documents in this collection share the structure pictured in table 5.2.

Table 5.2: **websites** collection structure

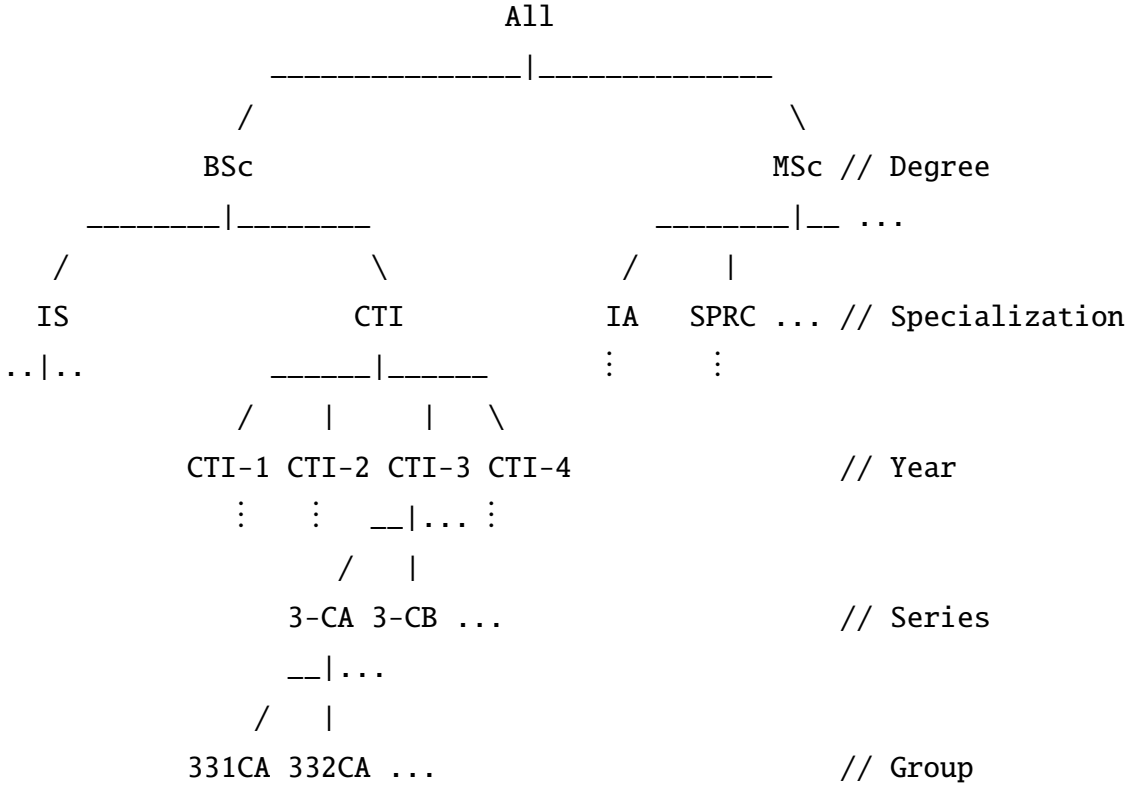
Field	Type	Required?	Additional info
category	`string`	☑	one of: "learning", "association", "administrative", "resource", "other"
degree	`string`	☐	"BSc" or "MSc", must be specified if *relevance* is not `null`
editedBy	`array<string>`	☒	list of user IDs
icon	`string`	☒	path in Firebase Storage; if missing, it defaults to "icons/websites/globe.png"
label	`string`	☑	unless specified, the app sets this to be the link without the protocol
link	`string`	☑	it needs to include the protocol (*http://*, *https://* etc.)
relevance	`null` / `list<string>`	☑	`null`[13]if relevant for everyone, otherwise a string of filter node names

This collection also has a special set of **rules** (see subsection 5.2.1.2), based on the permission levels described in section 4.3.2. Since websites in this collection are public information (anyone can `read`), altering and adding data here is a privilege and needs to be monitored, therefore anyone who wants to modify this data needs to be authenticated in the first place. Users can `create` a new public website only if their *permissionLevel* is equal to or greater than two and they sign the data by putting their *uid* in the *addedBy* field. Users can `update` a website if they do not modify the *addedBy* field and they sign the modification by adding their *uid* at the end of the *editedBy* list. Users can only delete a website if they are the ones who created it (their *uid* is equal to the addedBy field) or if their *permissionLevel* is equal to or greater than three.

[13]In Firestore, `null` is a data type; a field that is `null` is different from a field that is not present in a document.

filters

This collection stores `Filter` objects. These are essentially trees with named nodes and levels. In the case of the relevance filter, they are meant to represent the way the University organizes students:



The tree-based structure is defined using `maps`, as pictured in table 5.3.

Table 5.3: **filters** collection structure

Field	Type	Required?	Additional info
levelNames	`array<map<string, string>>`	☑	localized names for each tree level (e.g. "Year"); the map keys are the locale strings ("en", "ro")
root	`map<string, map<string, map<...>>>`	☑	nested map representing the tree structure, where the key is the name of the node and the value is a map of its children

The filter structure is public information and should never (or very rarely) need to be modified, therefore for this collection, anyone can `read`, but no one can `write`.

`classes`

This collection contains information about the classes taught in the faculty, its structure being pictured in table 5.4. Classes are taught in a specific year during a specific semester. While some are taught by the same professor (and have the same rules) for a whole generation, others are split on student series. In order to represent this, a **class** document can contain a **subclasses** collection which splits the information that can differ by series (fig. 5.5).

Table 5.4: **classes** collection structure

Field	Type	Required?	Additional info
acronym	`string`	☑	unique class acronym according to the Moodle page
class	`string`	☑	official class name
degree	`string`	☑	degree where the class is taught
domain	`string`	☑	domain where the class is taught
semester	`string`	☑	semester in which the class is taught
year	`string`	☑	academic year in which the class is taught
lecturer	`string`	☒	lecturer that teaches the class
shortcuts	`array< map<string, string>>`	☒	links to relevant resources, similar to websites on the portal page (keys are "name", "link", "addedBy" and "type" - one of "main", "classbook", "resource", "other")

Table 5.5: **classes/subclasses** subcollection structure

Field	Type	Required?	Additional info
series	`string`	☑	series where the class is taught
lecturer	`string`	☒	lecturer that teaches the class
shortcuts	`array< map<string, string>>`	☒	links to relevant resources, similar to websites on the portal page (keys are "name", "link" and "type" - one of "main", "classbook", "resource", "other")

Most of this information almost never changes from year to year, therefore students with at least *permissionLevel* 3 (see section 4.3.2) can only add or remove resource links (namely, the *shortcuts* field).

`events`

This collection contains timetable events. Depending on the event type, it will have a different set of relevant fields in its document. For example, recurring events such as laboratories and lectures will be defined by a *start* and *end* day and a recurrence rule (*rrule*) and have an associated *location*. Homeworks will not have a location, but rather have a *soft* and *hard* deadline as well as a *name* and a *link*. Most events will have a *class* associated with them (see the **classes** collection described above).

Table 5.6: **events** collection structure

Field	Type	Required?	Additional info
type	`string`	☑	one of "lab", "seminar", "lecture", "sports", "exam", "homework", "project", "test", "practical", "research"
class	`string`	☒	document ID of a class in the **classes** collection
start	`timestamp`	☒	a start date, where applicable (e.g. beginning of the semester)
end	`timestamp`	☒	an end date, where applicable (e.g. end of the semester)
rrule	`string`	☒	recurrence rule (e.g. "weekly", "bi-weekly")
location	`string`	☒	location where the event takes place, if applicable
description	`string`	☒	event description, if relevant
name	`string`	☒	event name, if applicable (labs/lectures don't have a name)
soft	`timestamp`	☒	soft deadline (for homework)
hard	`timestamp`	☒	hard deadline (for homework)

This collection has the same permissions as the **websites** collection described above.

people

Public information about people (particularly faculty members) is available in this collection. It is currently immutable and can only be read by any user since it is extracted from the faculty information website[14], but in the future, it can be modified depending on the user's permissions. The structure of this information is described in table 5.7.

Table 5.7: **people** collection structure

Field	Type	Required?	Additional info
name	string	☑	the person's full name
email	string	☒	the person's e-mail address
office	string	☒	the person's office, if applicable
photo	string	☒	a link to the person's photo
position	string	☒	the person's position (e.g. "Student", "Lecturer" etc.)

5.3 System design & implementation

The entire system is fairly complex (figure B.1 in appendix B shows a simplified UML diagram that does not include the localization class). We will not describe it in detail, but rather focus on describing the top-level components and the main design pattern used to implement them.

5.3.1 Primary components

The main components of the architecture are linked to its features:

- the **authentication** component uses FirebaseAuth (see subsection 5.2.2) to allow the student to make various account management actions
- the **page** components deal with each of the application's pages (see section 4.5.2) and handle their specific data

[14] https://cs.pub.ro/index.php/?option=com_comprofiler&task=userslist&listid=2

5.3.2 State management

`Provider` is a dependency injection framework which can be used to manage state in a Flutter application.

To understand how `Provider` works, we first need to understand the structure of a Flutter application. Flutter UI revolves around widgets, which are components controlling some part of the interface. Widgets are organized in a tree structure, with parent widgets controlling the behavior of their children. Widgets are partly similar to HTML tags - for example, a widget containing a bit of text can be centered by wrapping it inside a `Center` widget (see table 5.8).

Table 5.8: Tree-based structure of HTML and Flutter

HTML	Flutter

`<center>`
`  <h1>Hello, world!</h1>`
`</center>`
	`Center(`
	`  child: Text(`
	`  "Hello, world!",`
	`  textTheme.headline1,`
	`  );`
	`);`

`Providers` offer an easy way to encapsulate state and share it with a branch of the widget tree. It achieves that by being defined as the parent of that branch. Alternatively, if the whole application requires access to a specific state (as is the case of authentication in our application), the `Provider` needs to be defined as the parent of the entire widget tree:

```
runApp(ChangeNotifierProvider<AuthProvider>(
      create: (_) => AuthProvider(), child: MyApp()));
```

5.3.3 Separation of concerns

For each component, we use a variant of the BLoC (Business Logic Component) design pattern which utilizes `Provider` (described in the previous subsection, 5.3.2) in order to avoid boilerplate code.

Figure 5.1 shows the classic BLoC structure. Based on this pattern, each page in our application is made up of three main components:

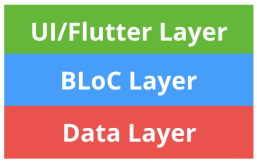


Figure 5.1: Standard BLoC architecture

- the **UI/Flutter layer**, also called the **view**, which acts as a middleman between the user and the BLoC layer
- the **BLoC layer**, which we also refer to as the **`Provider`** or **service**, contains the actual business logic and communicates with the database
- the **data layer** or the data **model** contains simple data classes which represent the data that is to be displayed in the UI, and is provided by the BLoC

For example, the portal page has three main components, which can be observed in appendix B, figure B.2. The **view** manages the UI and includes both the page listing the websites as well as the form page allowing users to add/modify a website. The **model** is the `Website` data class and contains all of the relevant fields for a website (e.g., name, link, localized information, path to the icon). The **service** is a `ChangeNotifierProvider` and is responsible for fetching the data from Firestore and converting it to a list of `Website`s. The service also checks the validity of new data and sends it to the server to be updated.

5.3.4 Localization

Thanks to our user study, we know that Romanian and English localization is essential for a large portion of our potential user base (see section 3.3.3 for the relevant survey results). Therefore, we implemented localization using Flutter-compatible tools.

The application's localized strings are defined in .ARB (App Resource Bundle)[15] files, a simple localization resource format based on JSON and introduced by Google. The *Fluter Intl*[16] Android Studio plugin automatically generates *Dart Intl*[17] boilerplate code which binds the Flutter application to the translations in the .ARB files.

[15]https://github.com/google/app-resource-bundle
[16]https://plugins.jetbrains.com/plugin/13666-flutter-intl
[17]https://pub.dev/packages/intl

5.4 CI & CD

5.4.1 Testing

A test-driven approach is necessary for any medium to large project with multiple contributors, to ensure that new changes work as intended and do not break existing features. It has been shown[25] to improve code quality and maintainability significantly.

We decided to use the `flutter_test` library[18] to test our application. The BLoC architecture (described in section 5.3.3) makes it easy to test each individual component. We focused on testing the data and UI layers, since the service layer mostly calls methods from the `flutterfire` plugin[19] which connects to Firestore. The Firestore plugin is already heavily tested by its developers and would be complicated to mock. As a result, more than 70% (fig. 5.2), the application code is being tested through over 200 unit and integration tests. We aim to keep this number above the 60% threshold.

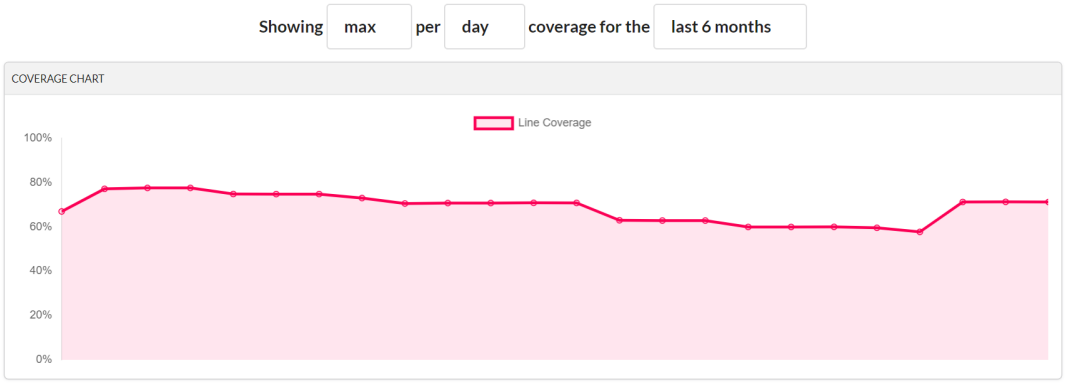


Figure 5.2: Application code coverage according to *codecov*

5.4.2 Deployment

The application is currently being hosted on the web using two free Firebase domains[20]. Deployment is done through the `firebase deploy` command of the Firebase CLI[21].

[18] https://api.flutter.dev/flutter/flutter_test/flutter_test-library.html
[19] https://github.com/FirebaseExtended/flutterfire
[20] https://acs-upb-mobile.web.app/ and https://acs-upb-mobile.firebaseapp.com/
[21] https://firebase.google.com/docs/cli

In the future, the mobile application can be published directly onto the App Store and Google Play (see subsection 6.3.1) for easy access.

5.4.3 GitHub Actions

The code of the application is public in a GitHub repository[22], which makes it easy for students to contribute by creating an issue or submitting a PR. GitHub allows setting up an automated testing and deployment pipeline using GitHub Actions[23], which is what we chose to use for our application.

For Continuous Integration (CI), we created a workflow[24] which runs on every push to the repository. It runs all tests (see subsection 5.4.1) on an Ubuntu virtual machine and sends coverage information to *codecov*[25].

For Continuous Deployment (CD), we defined a deployment workflow[26] which is triggered on the creation of a new tag. It runs all tests and, if they succeed, builds and publishes the web version (see subsection 5.4.2) onto the Firebase servers and creates a new release on GitHub containing the built APK for Android devices.

[22]https://github.com/acs-upb-mobile/acs-upb-mobile/
[23]https://github.com/features/actions/
[24]https://github.com/acs-upb-mobile/acs-upb-mobile/blob/master/.github/workflows/test.yml
[25]https://codecov.io/gh/acs-upb-mobile/acs-upb-mobile
[26]https://github.com/acs-upb-mobile/acs-upb-mobile/blob/master/.github/workflows/deploy.yml

Chapter 6

Conclusion

6.1 Caveats

6.1.1 Scalability

The application should easily be able to support usage from the (roughly 4000) students of ACS. Initially, the free Firebase plan (Spark[1]) should be enough for our relatively low number of users. It offers a daily quota of 50k reads, 20k writes and 20k deletes. Even with the optimistic assumption that every single student would use the application every single day, the limit should not be reached with normal usage. The number of writes per minute could be limited from the application (using a time-out) without causing a significant inconvenience for the users. The invalidation of cached data in the app could be less aggressive in order to restrict the number of daily reads. These measures would ensure that users cannot abuse the database and force it to reach the cap for the free plan too soon.

However, as more students begin to use the application, some "peak" periods (such as the beginning of the year when everyone would set up their application with the new schedule) might lead to reaching the free plan's cap. We should consider upgrading to the pay-as-you-go plan (Blaze[1]) in the future. It offers everything that the free plan offers (meaning if usage stays under the cap, it is still free), as well as the possibility to pay for any transactions that go over the cap (100k additional reads, writes and deletes costing $0.06, $0.18 and $0.02 respectively). It includes many other features such as a daily backup of the database.

[1] https://firebase.google.com/pricing

6.1.2 Collaboration with the university

In its current form, the application is meant to be used and maintained solely by students. However, many more useful features (including some suggestions from students mentioned below, in sections 6.2 and 6.3) could be added to the application if the faculty staff would contribute. On the other hand, an official university application should only contain information published by the staff, which goes against the concept of students contributing with data themselves. Additionally, the university may want to censor some of the resources that the students may want to share, such as written courses that are not available online or old exam subjects. A potential middle-ground solution for this problem would be to find a way to very clearly mark *official information* from *student additions* within the app. This way, the university is only responsible for the official data, and the students can choose to use the information provided by other students or not.

Furthermore, since the application is meant to be free and will not contain ads or another form of monetization, it would be ideal if the faculty could support the running costs. Including the database (section 6.1.1) and store publishing costs (section 6.3.1), these would not surpass $200 a year with the current set-up. Therefore, we believe that it would be a beneficial investment for the sake of the faculty's students.

6.2 Feedback

The feedback we received from the students who tested the application was overwhelmingly positive. They noted the small details like being able to change the language and the theme of the application and showed excitement about the variety of different features offered.

Some students suggested that it would be easier if accounts were created automatically based on their Moodle accounts so that they would not have to sign up.

In terms of UI, the only improvement that was suggested was to use a monospaced font, so that certain pieces of text with the same length have the same size (for instance, the timetable hours - fig. 4.21 and the filter node labels - fig. 4.17).

6.3 Future improvements

6.3.1 Store publishing

In order to publish the app on the **App Store**, an *Apple Developer Account*, which costs $99 per year, is required. Additionally, the app needs to meet all of the requirements specified in the *App Store Guidelines*[2]. The review process usually takes from 24 to 48 hours.

For **Google Play**, a *Google Play Developer Account* with an upfront cost of $25 is required. The Android *Quality Guidelines* specify what the app needs to achieve before being published. The review process generally takes from 3 to 7 days.

One notable requirement that is specified or suggested in both guidelines is providing user agreements such as EULA (End-User License Agreement) and TOS (Terms of Service). Before publishing the application, these documents should be devised accordingly (taking into consideration regulations such ad the GDPR) and shown to the user during the sign up process.

A feature that would be useful once the application is published would be a way to request a review from the user, to ensure a good position in the mobile stores.

Furthermore, a few more improvements should be made to the app before it is publicly available. Thanks to our architecture (see section 5.3.3), the data is almost entirely in the database and can be modified without updating the application. This structure helps avoid the bottleneck of having to publish a new version of the application for any small detail (and require users to update). On the other hand, sometimes an update **is** necessary and users might need to update as soon as possible (for example, a change in the database structure may mean that some parts of an old version of the app might stop working entirely). For that case, we should implement a way to remotely block users from using the app (or certain parts of it) unless they update it.

To help with future debugging, we should introduce a logger feature that logs errors and other information into a file on the device that the user can upload in a bug report.

[2]https://developer.apple.com/app-store/review/guidelines/

6.3.2 Integrations

Since our ultimate goal is to help students, we do not see other published apps meant for students as a competition but rather as an opportunity to learn and improve our application. We believe that building upon the existing technologies is better than trying to re-invent the wheel and create more confusion among students, who would have yet more tools for the same job. Consequently, in the future, we aim to integrate with existing, published applications (e.g. *UPBCampus*, described in section 2.4.2, and *Politehnik*, described in section 2.4.3) in order to extend our app's functionality. Users can select certain options in our app and be redirected to another relevant app. We can implement this feature through URL schemes[3] in iOS or the more well-rounded concept of Intents[4] on Android. Two possible integrations with the tools mentioned above would be, for instance, navigating to a location of an event saved in our application, by using the *UPBCampus* app, or adding the *Politehnik* events users are interested in into the calendar in our app.

Additionally, in the future, data can be extracted from official platforms. For example, news can be obtained from the official university and faculty websites using the Flutter `web_scraper` package[5] and events can be imported directly from the e-learning platform by using the Moodle API.

Finally, users should be able to sync the events in the application with their calendar of choice. This should not be too difficult to achieve, as most calendar applications (including Google Calendar[6]) share a similar format for events which would be possible to export as a *csv* file. An example of the format is pictured in table 6.1.

Table 6.1: Google Calendar importable event fields

Subject	**Start Date**	**Start Time**	**End Date**	**End Time**	**Location**
PC - Exam	28/01/2020	10:00 AM	28/01/2020	12:00 PM	PR001

[3]https://developer.apple.com/documentation/uikit/inter-process_communication
[4]https://developer.android.com/guide/components/intents-filters
[5]https://pub.dev/packages/web_scraper
[6]https://support.google.com/calendar/answer/37118

6.3.3 Additional features

Thanks to the students interviewed for our **Case study** (section 2.1.2) and the survey in our **User study** (chapter 3), we can define additional features that would further improve the usefulness of our application, namely:

- canteen menu information and expected waiting time
- a way to upload files directly into the application for easy access
- feedback from students about classes and professors, for new generations to see and know what to expect
- a special timetable for classroom availability
- an administrative section that allows students to request documents or make payments

6.3.4 Extending to other faculties/universities

Since the application is built to be highly customizable and the entire data is fetched from the database, the codebase can be easily re-used to create an application for a different faculty or university with the same needs. It would just have to be connected to a different Firebase project with the same setup, containing data relevant to the new university.

Some components that would need to be changed would be the application theme, illustration, and logos. Additionally, a tutorial on how to use the application could be added for less digitally-inclined students.

Abbreviations

ACS Faculty of Automatic Control and Computer Science. 6, 7, 10, 14, 20, 21, 25–27, 32, 41, 59

API Application Programming Interface. 62

CD Continuous Deployment. 5, 57, 58

CI Continuous Integration. 5, 57, 58

CLI Command-Line Interface. 57

EULA End-User License Agreement. 61

GDPR General Data Protection Regulation. 61

OS Operating System. 6, 17, 28, 30–32

PR Pull Request. 13, 58

SDK Software Development Kit. 48

TA Teaching Assistant. 12

TOS Terms of Service. 61

UI User Interface. 4, 12, 13, 33–44, 55, 56, 60

UID User Identifier or Unique Identifier. 48

UPB University POLITEHNICA of Bucharest. 6, 10, 14, 20–23

UX User Experience. 4, 13, 31, 33–44

Glossary

BLoC (Business Logic Component) is a design pattern created by Google which uses Reactive Programming to handle the flow of data in an application. 7, 55–57

COVID-19 or coronavirus disease 2019 is a respiratory virus caused by the *Severe acute respiratory syndrome coronavirus 2 (SARS-CoV-2)* strain of coronavirus. 4, 21

Firebase is a mobile and web application development platform developed by Firebase, Inc. in 2011, then acquired by Google in 2014. 6, 24, 47, 48

Flutter is Google's open-source UI software development kit, used to develop cross-platform applications for Android, iOS, Google Fuchsia and the web. 8, 46, 47, 55, 56, 62

JSON (JavaScript Object Notation) is a lightweight data-interchange format based on defining key-value pairs. 22, 47, 56

Moodle (Modular Object-Oriented Dynamic Learning Environment) is an open source, online courseware platform which provides the necessary tools for educators to create a virtual classroom via the Internet[26]. 11, 20, 21, 52, 60, 62

noSQL (non-SQL or non-relational) is a type of database that does not require the data stored within it to be organised tabularly, as is needed for traditional relational databases. 47

SMIS is a web archive of EU-funded projects published by the Romanian Ministry of Public Finance. 21

Bibliography

[1] K. Kelly, *What technology wants*. Penguin, 2010.

[2] T. Lewis, "Studying the impact of technology on work and jobs," *Digital Library and Archives of the Virginia Tech University Libraries*, 1996.

[3] S. M. Kairi. (2019, May) Technology in transportation: Economic and social impacts. Date accessed: 26.05.2020. [Online]. Available: https://blog.mobility.here.com/technology-in-transportation-economic-and-social-impacts

[4] K. D. Lewis and B. M. Burton-Freeman, "The role of innovation and technology in meeting individual nutritional needs," *The Journal of nutrition*, vol. 140, no. 2, pp. 426S–436S, 2010.

[5] J. S. McQuillen, "The influence of technology on the initiation of interpersonal relationships (1)," *Education*, vol. 123, no. 3, pp. 616–624, 2003.

[6] B. Chen and A. Denoyelles, "Exploring students' mobile learning practices in higher education," *Educause Review*, vol. 7, pp. 1054–1064, 2013.

[7] „PLATFORMĂ DE E-LEARNING ȘI CURRICULĂ E-CONTENT PENTRU ÎNVĂȚĂMÂNTUL SUPERIOR TEHNIC", Proiect nr. 154/323 cod SMIS – 4428. Date accessed: 04.06.2020. [Online]. Available: http://curs.pub.ro/index.php/cursuri-upb-despre-proiect

[8] D. Scurtu, "UPB Campus," *Sesiunea de Comunicări Științifice Studențești*, 2020.

[9] S. T. Roberts, *Behind the screen: Content moderation in the shadows of social media*. Yale University Press, 2019.

[10] W. Klemm, "Benefits of collaboration software for on-site classes." 1997.

[11] C. S. Hammer, "The importance of participant demographics," *American Journal of Speech-Language Pathology*, vol. 20, pp. 261–262, 2011.

[12] L. M. Connelly, "Demographic data in research studies," *Medsurg Nursing*, vol. 22, no. 4, pp. 269–271, 2013.

[13] C. Popa. (2019) University Stats: prezența feminină în facultățile cu profil tehnic. Date accessed: 04.06.2020. [Online]. Available: https://codette.ro/blog/university-stats-prezenta-feminina-in-facultatile-cu-profil-tehnic/

[14] StatCounter GlobalStats. Mobile Operating System Market Share Romania. Date accessed: 04.06.2020. [Online]. Available: https://gs.statcounter.com/os-market-share/mobile/romania

[15] C. E. Weeks, "How to design an aesthetically-pleasing, ada-compliant website," in *Proceedings of the 25th annual ACM SIGUCCS conference on User services: are you ready?*, 1997, pp. 319–326.

[16] J. Porter. Testing the three-click rule. Date accessed: 05.06.2020. [Online]. Available: https://articles.uie.com/three_click_rule/

[17] J. Nielsen and H. Loranger, *Prioritizing web usability*. Pearson Education, 2006.

[18] V. Thirumala. Interaction Design patterns : iOS vs Android. Date accessed: 05.06.2020. [Online]. Available: https://medium.com/@vedantha/interaction-design-patterns-ios-vs-android-111055f8a9b7

[19] Apple. Human Interface Guidelines. Date accessed: 05.06.2020. [Online]. Available: https://developer.apple.com/design/human-interface-guidelines/

[20] Google. Material Design. Date accessed: 05.06.2020. [Online]. Available: https://material.io/

[21] Bridged.co. What is content moderation and why companies need it. Date accessed: 09.06.2020. [Online]. Available: https://bridged.co/blog/what-is-content-moderation-why-companies-need-it/

[22] D. Bohn. (2019) Android Q's back gesture controversy, explained. Date accessed: 22.06.2020. [Online]. Available: https://www.theverge.com/2019/8/8/20791457/google-android-q-back-gesture-controversy-app-drawer-navigation

[23] S. Agarwal. 5 Best Mobile App Development Frameworks Businesses Need to Consider Using in 2019. Date accessed: 11.06.2020. [Online]. Available: https://medium.com/quick-code/5-best-mobile-app-development-frameworks-businesses-need-to-consider-using-in-2019-68e8ae9877bf

[24] StackOverflow. (2020) Developer Survey. Date accessed: 11.06.2020. [Online]. Available: https://insights.stackoverflow.com/survey/2020

[25] R. Hilton, "Quantitatively evaluating test-driven development by applying object-oriented quality metrics to open source projects," *PDF]. Available: http://www. nomachetejuggling. com/files/tdd_thesis. pdf, M. Sc thesis, Department of Computer & Information Sciences, Regis University, CO*, 2009.

[26] M. Crosslin, "Course management meets social networking in Moodle," in *Social Computing: Concepts, Methodologies, Tools, and Applications*. IGI Global, 2010, pp. 849–854.

Appendices

Appendix A

Native versus cross-platform appearance

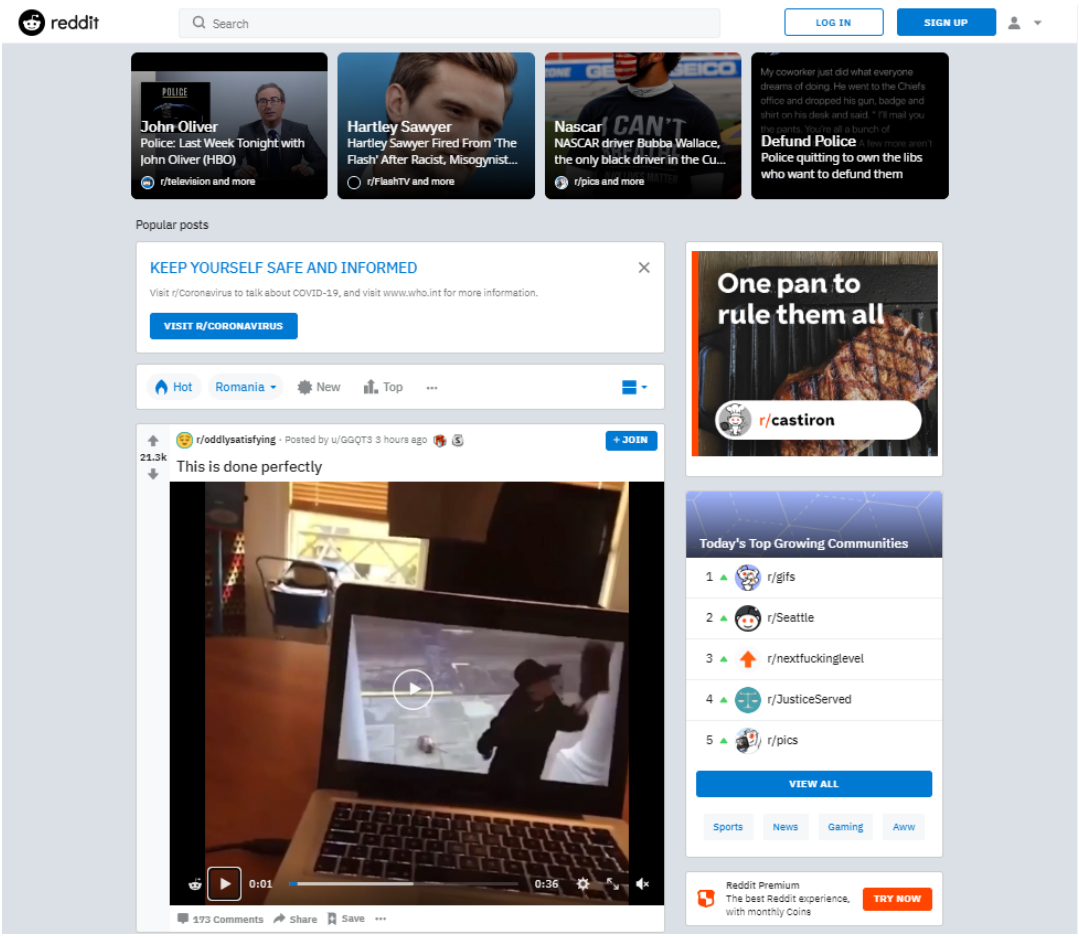


Figure A.1: Official Reddit website

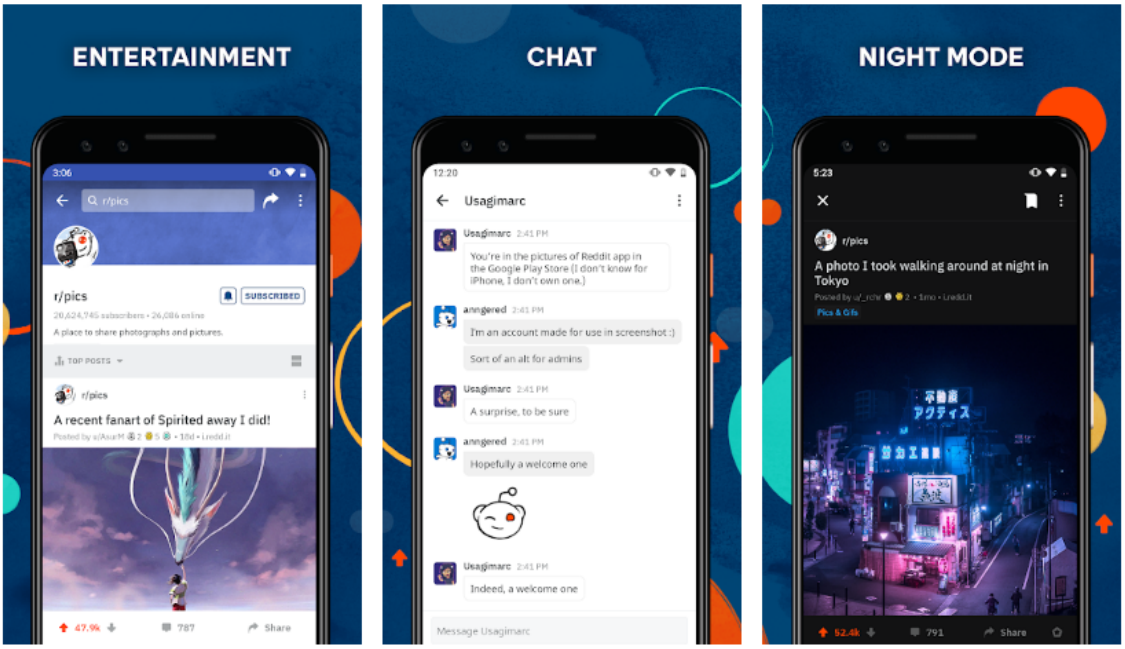


Figure A.2: Official Android Reddit application

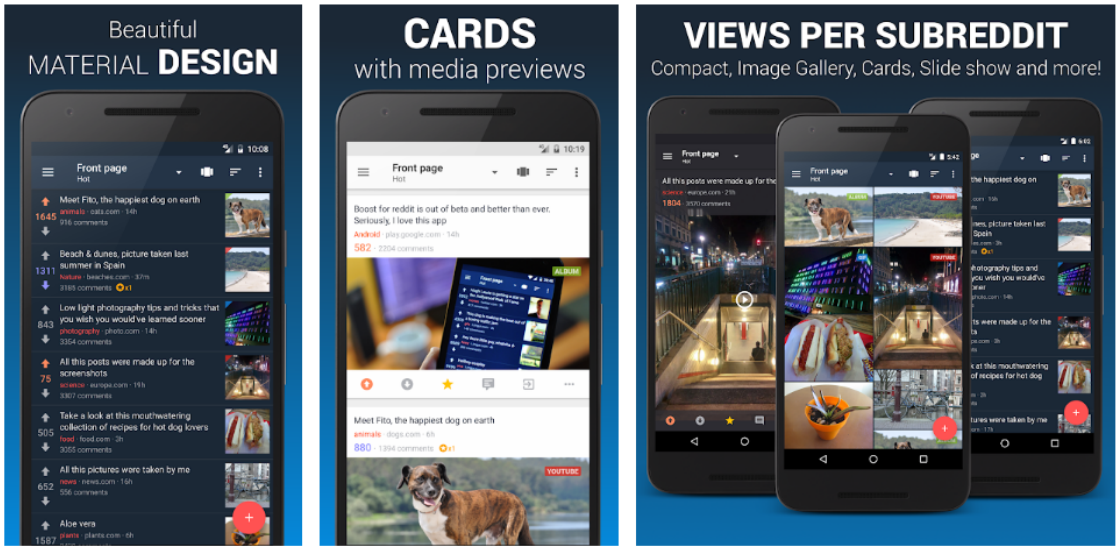


Figure A.3: Boost for Reddit, an unofficial Android application

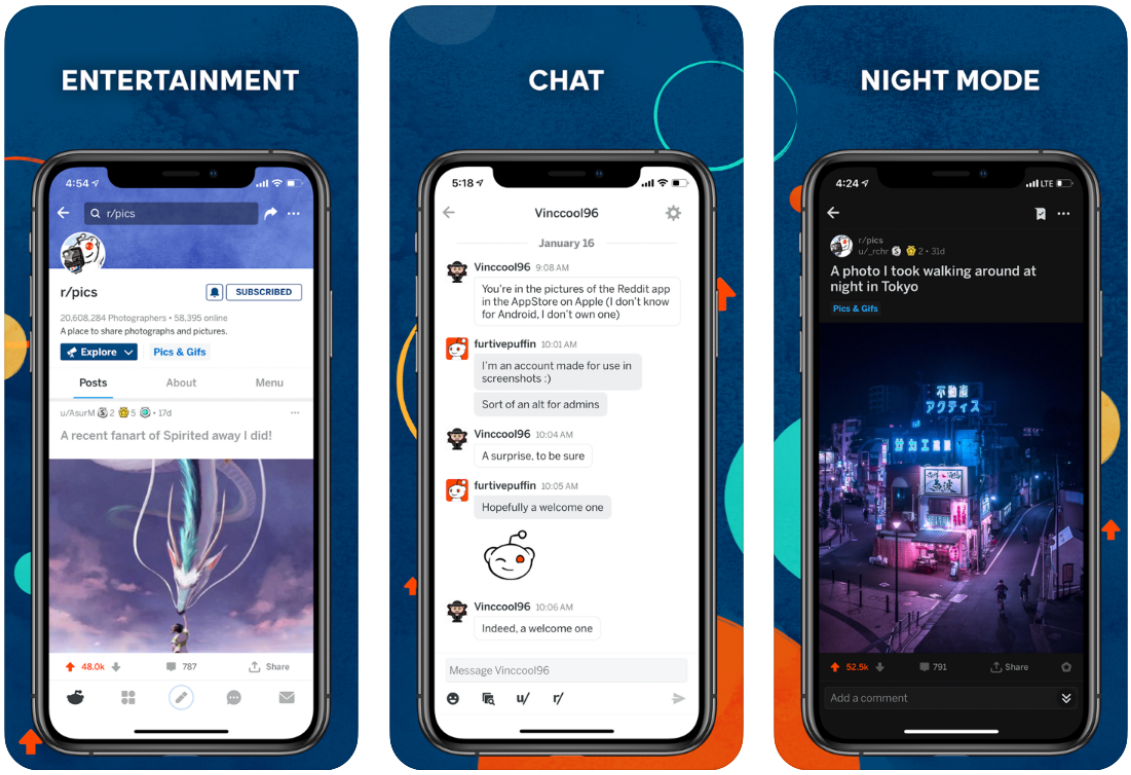


Figure A.4: Official iOS Reddit application

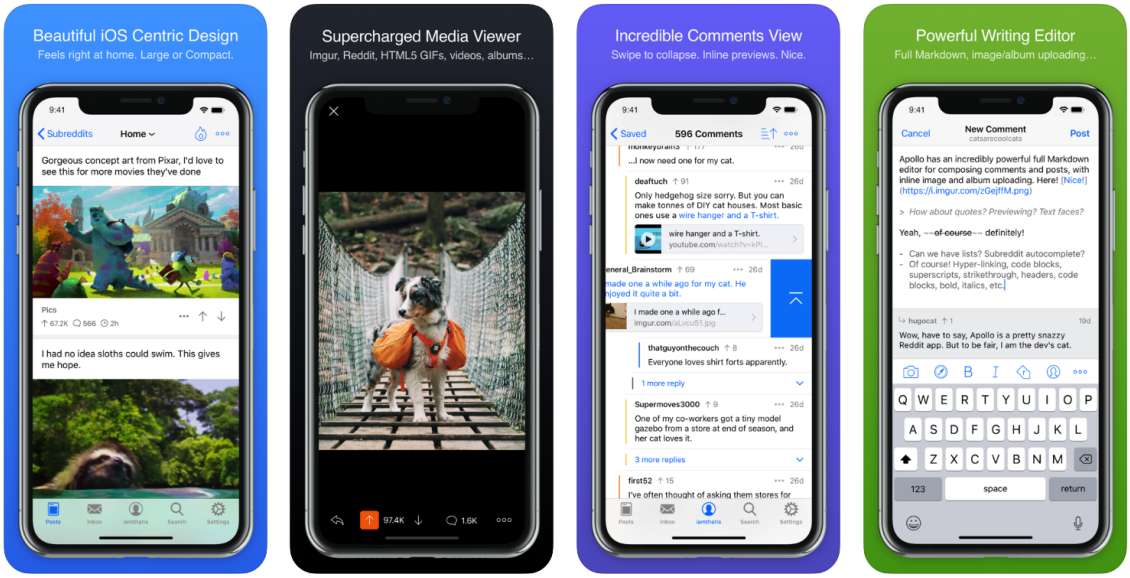


Figure A.5: Apollo for Reddit, an unofficial iOS application

Appendix B

UML diagrams

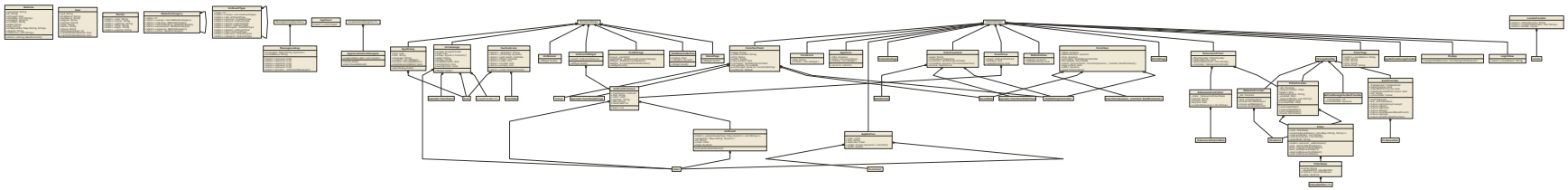


Figure B.1: A simplified UML diagram of the entire system

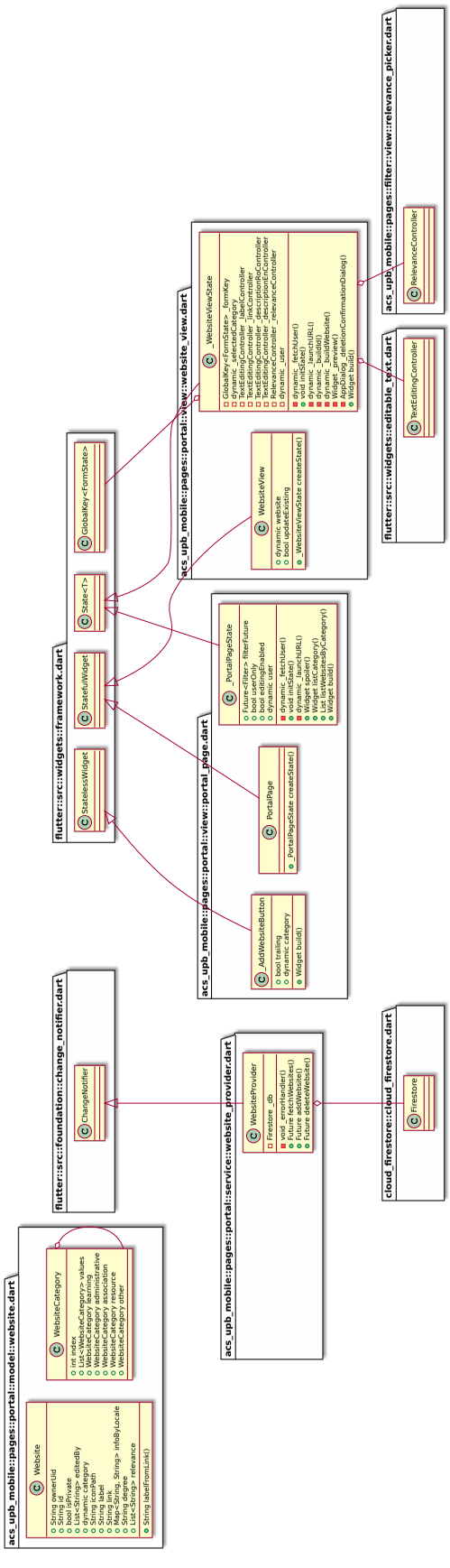


Figure B.2: Diagram of the portal page

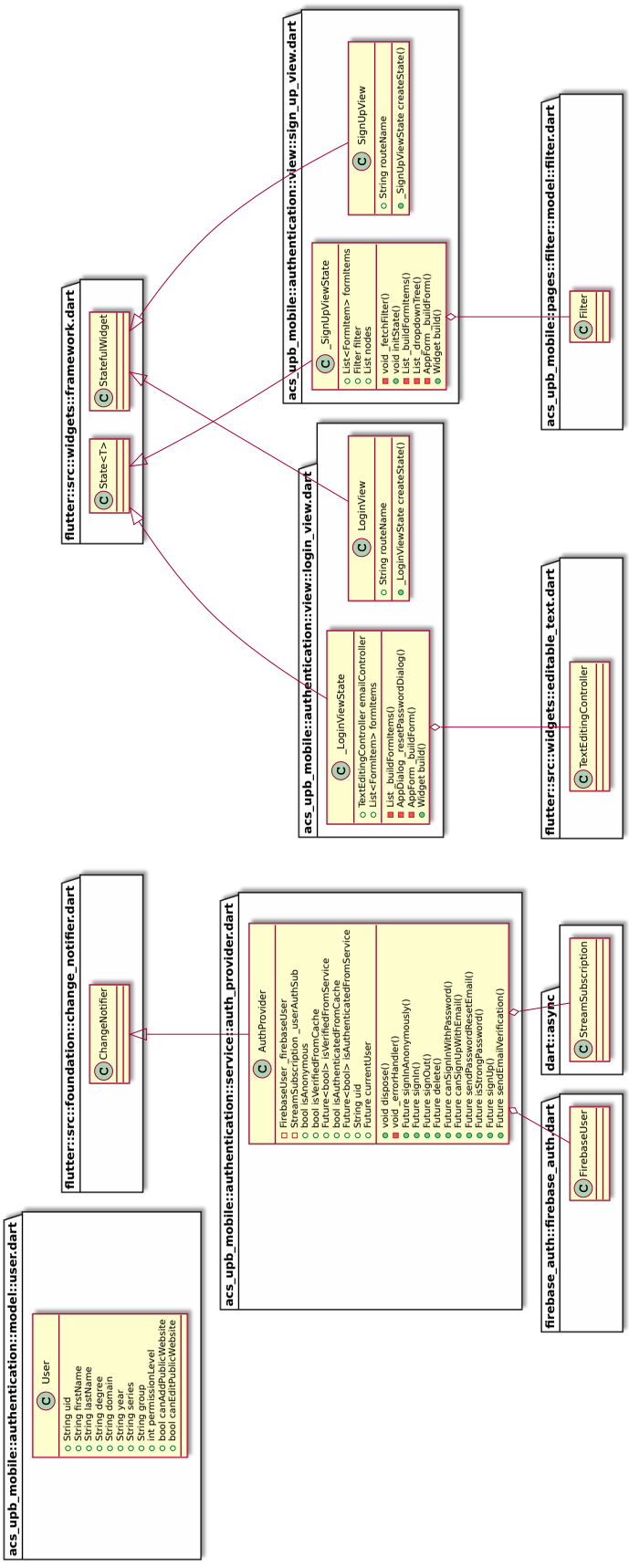


Figure B.3: Diagram of the authentication system