UNIVERSITY POLITEHNICA OF BUCHAREST
FACULTY OF AUTOMATIC CONTROL AND COMPUTERS

Multi-source university news aggregator
Agregator de știri universitare din mai multe surse

Graduate:
Ștefan-Andrei Popa

Thesis supervisors:
Lecturer Dr. Alexandru Grădinaru
Engr. Ioana Alexandru

Bucharest
2022

Această lucrare prezintă implementarea unui modul de agregator de știri, soluție ce își propune să ajute la centralizarea postărilor și noutăților din facultate. Modulul este integrat într-o aplicație deja existentă și are ca scop atât centralizarea informațiilor de pe mai multe platforme, într-un singur feed central, cât și încurajarea studenților de a contribui activ în construirea acestuia.

Din cauza situației pandemice din ultimii ani, viața studenților s-a mutat integral în online, iar buna informarea a acestora a devenit dependentă de o multitudine de platforme online eterogene. Altfel spus, pentru ca un student să rămână constant la curent cu noutățile din facultate, acesta e nevoit să navigheze zilnic pe o serie de platforme și să filtreze o cantitate mare de conținut până să ajungă la informațiile relevante pentru el. Această lucrare tratează problema din perspectiva studenților de la Facultatea de Automatică și Calculatoare, Universitatea POLITEHNICA București.

Încercăm să analizăm problema și impactul ei prin feedback-ul oferit de studenți și să înțelegem cum alte instituții universitare abordează acest subiect. Pe baza informațiilor colectate, dorim să înaintăm o soluție de agregator disponibilă pe o aplicație mobilă open-source contribuită de numeroși studenți în ultimii doi ani.

This paper presents the implementation of a content aggregator module that helps centralize the news related to university life. This module is integrated into an existing mobile application and is aimed at bringing the information from many platforms into one single centralized feed. Simultaneously, this feature should encourage the students to contribute to the feed and populate it with relevant news.

As a direct consequence of the recent pandemic, the students' lives shifted into an online environment, and their information became dependent on several heterogeneous online platforms. In other words, for students to be aware of the official and extracurricular faculty news, they must navigate a series of platforms daily and filter a large quantity of information to reach the news they care about. This paper treats this problem from the students' perspective from the Faculty of Automatic Control and Computer Science, University POLITEHNICA of Bucharest.

Based on the feedback provided by students, we try to analyze this problem and understand how other universities tackle this issue. Our proposed aggregator solution is built over an already available mobile application that many students contributed to in the last two years.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

In the era of hyperconnectivity, information systems, technological devices and people are connected to such a degree that technology has become almost indispensable for the normal flow of our work and personal environments. At the same time, the surge in digitalization has dramatically influenced the amount of daily information load that we are confronted with and has affected our general attention span when it comes to navigating online content [1]. While we may not be short on information, sorting and searching for the content we need has become a challenge, and we are in dire need of tools and solutions that could help curate our large data systems.

Information overload [2] is only one of the modern digital problems that we are confronted with, but it is fair to say that technology is equally responsible for constantly improving our lives. The direction of technology is towards efficiency, optimization and automatization of processes, and even navigating large amounts of content could be a process that can be eventually solved. Therefore, the main focus should be on designing intelligent curating systems that filter and efficiently retrieve our information. According to the American author Clay Shirky [3]: *"It's not information overload. It's filter failure"*.

Nowadays, almost any system can represent a source of information overload, and unsurprisingly, even university ecosystems can fall into this category. For example, each university can employ several online services and platforms to organize its data and activity, thus generating a complex information network.

Consequently, students can quickly become overwhelmed by the tedious task of retrieving their relevant content.

This paper proposes a news aggregator feature built upon an existing faculty mobile application, which aims to help students curate their relevant university content and encourage them to contribute to a centralised news feed that can help their colleagues.

1.2 Background

For the purpose of this paper, we will build a solution aimed at the students of the **Faculty of Automatic Control and Computer Science**[1] from the **University POLITEHNICA of Bucharest**[2]. The name of the existing project that we built upon is **ACS UPB Mobile**[3], and it is a cross-platform mobile application that wants to help students better manage their faculty life from a single platform. As a convention, we will denote the faculty mentioned above as **ACS** and the university as **UPB**.

To better understand how to build our solution, we need to comprehend how a university ecosystem can steadily increase in information complexity. For example, one university can host an online domain for each faculty, and in turn, each faculty can organize its data (timetables, assignments, exams, course materials) on several platforms. Likewise, student organizations can each use a different service for hosting their activity, and, additionally, students can separately organize themselves on a wide range of social networks. Different entities repost each informative post several times, resulting in spam. Moreover, posts from distinct platforms do not generally have the same degree of visibility, meaning that students could be at risk of missing important updates.

Official institutional platforms can make it easier for students to retrieve a specific post, whereas social networks are predominantly convoluted and consume more time to be sorted out. As a result, not only the information load could become an issue for students, but also the daily context switching from one platform to another.

[1] https://acs.pub.ro/
[2] https://upb.ro/
[3] https://github.com/student-hub/acs-upb-mobile

1.3 ACS UPB Mobile

ACS UPB Mobile was initially started by student *Ioana Alexandru* as a Bachelor's thesis [4] in 2020, and ever since, this project attracted more students as direct contributors[4]. The current focus of the app is the students of **ACS**, and it encompasses many core functionalities that all build towards better managing the student life at this faculty. The application is currently published via Google Play Store on Android[5] and has a publicly available web version[6].

ACS UPB Mobile was developed using the Flutter framework[7] from Google. Flutter uses the Dart programming language[8], and it is an open-source framework whose goal is to build beautiful, native-looking Android and iOS applications from the same code base [5]. We will go over Flutter more in detail once we discuss the actual implementation.

Smartphones are an essential and indispensable component for students to go by in their daily activities. According to this study[6], students spend approximately 5 hours a day on their smartphone and check it at least 28 times on a daily basis. Thus, the choice of building a mobile application for students to manage their faculty life becomes logical.

Given recent analytics from the app's admin dashboard, over 650 student accounts are registered on the platform. This project has been thoroughly developed and tested by the passionate community behind it and has had several official releases up to this point. In addition, **ACS UPB Mobile** has a detailed contributing manifest[9], a rigorous deployment pipeline, and a low entry bar for new developers to start contributing features. Therefore, using this existing platform for implementing our solution was a natural decision. Furthermore, we considered it the best choice for reaching a considerable number of students and thus, leaving a significant positive impact.

We have worked in a close feedback loop with the project developers, and we tried to follow their implementation guidelines along the process. Eventually, our news aggregator module should adhere to their quality standards and follow

[4]https://github.com/student-hub/acs-upb-mobile#contributors
[5]https://play.google.com/store/apps/details?id=ro.pub.acs.acs_upb_mobile
[6]https://acs-upb-mobile.web.app/
[7]https://flutter.dev/
[8]https://dart.dev/
[9]https://github.com/student-hub/acs-upb-mobile/blob/master/CONTRIBUTING.md

the design choices present in the app.

1.4 Proposed functionalities

Given the previously described environment at **1.3**, we are going to put forward the functionalities that we need for our module to work as intended. We will separate between user-facing functionalities that impact the overall user experience and system-based characteristics that sustain the whole feature.

User-facing functionalities:

- **news feed page** for students to easily access their content
- **separate details page** for each feed item
- ability to **apply for user roles**
- mechanism for students to **select their information sources**
- **bookmark option** for students to mark their favorite items
- **option to create new post** and **support for Markdown syntax** for custom text formatting
- **external share mechanism** for an individual item

System-based functionalities:

- **central database** with all unique news items
- **web scrapers** that curate online content on different platforms
- **news filtering mechanism** based on user's selected sources and global filters
- **role-based mechanism** that grants different posting rights to students
- **notifications system** that pushes recent posts to users
- support for **deep linking**

1.5 Goals

Information is essential in an environment that shifts increasingly more to the online, and we believe that each student should be empowered with the same access to information. By implementing our proposed functionalities, we want to unify several heterogeneous information sources into one feed and introduce a standard format for all news posts that can be easily browsed and sorted. Besides being a news aggregator from several platforms, we want to give students the possibility to contribute themselves to the main news feed and give them the means to make their posts come across more quickly and effectively.

By having all posts in one place, we want to reduce the spam volume and cut the need to repost the same news to make it more visible. Posts should have an author, a clear target group, and should be timestamped and have a straightforward chronology. By having clear metadata fields attached to each item, students will be able to sort out the content they need and have an easier time browsing the relevant data.

Above all, moderation is essential to keep a quality standard in our news feed. Posts should be informative, have a clear message, and be devoid of harmful or ill-directed intentions. As a result, we will offer on-request access to specific subgroups of users to post content. Later, this strategy could be further improved through an automated gamification system. In other words, students would be able to earn their right to post by being positively involved on the platform and by earning a sufficient amount of points. As this paper states [7]: "Gamification systems are used to motivate individuals to achieve specific goals".

In the end, we want our module to be compatible with the rest of the **ACS UPB Mobile** application and be part of a future release that can arrive in the hands of numerous students.

1.6 Outline

Chapter 1 introduces the motivation and the goals of this paper

Chapter 2 describes the state of the art. We talk to students enrolled at other universities and we try to understand how they manage their information sources. In addition, we try to analyze the existing solutions for **ACS** and how these have tackled our issue so far.

Chapter 3 outlines the results obtained from collected feedback and focuses on what conclusions could be drawn from it. Finally, we try to plot our data using different representations and see what tendencies students at **ACS** follow when they want to retrieve their university-related information.

Chapter 4 explains, based on our user study results, what functionalities we want to have for our news aggregator module and why we consider those most relevant for our use case. Following this, we propose our UI/UX prototype and explain the iterative process from our initial design to our final one.

Chapter 5 describes the implementation process and the general architecture that supports our module.

Chapter 6 presents the testing phase done with users and the results

Chapter 7 describes the final conclusions of our thesis

Chapter 2

State of the Art

Universities could employ several services to organize their data. Depending on the institution, well-informed students may need to navigate one or more distinct platforms to retrieve all the data they need, from course materials, assignments or timetables to information related to student dorms, campus navigation or faculty organizations. This wealth of information could prove incredibly daunting for first-year students in their initial weeks, but it is not excluded that students of all years would benefit from genuinely curated content.

The underlying problem that institutions face nowadays, now more than ever following the COVID-19 pandemic break, is what strategies and technical solutions should these undertake to shift their services and campus environment to the online in a seamless, student-friendly manner.

According to *Statista*[1], an online service for visualizing various statistics, by the first quarter of 2022, the education segment of mobile apps on the App Store[2] places in the third position, with almost **9.7%** market share [8]. In contrast, the education segment on Google Play Store[3] places in the second position, with over **10.4%** market share [9]. The rise of this segment could imply a surge in university-based app adoption, and for high-education institutions, this could prove a competitive advantage in the long term.

Having one or more technical platforms for students to manage their faculty life is one thing. However, we argue that promoting well-tailored and curated content for each individual is another issue that should be considered.

[1] https://www.statista.com/

[2] https://www.apple.com/app-store/

[3] https://play.google.com/

The question in place is how universities could make this content readily available and explicitly costumed for each individual's needs. In other words, the issue is how institutions could help their student manage their information sources efficiently by promoting helpful content, reducing spam or proposing new sources that a student may find useful. Due to the increasing shift to online, this considerable web of information is indeed a challenge, and different university-based apps may handle this one way or another.

While there is no standard format regarding how higher-education institutions manage their wealth of information, it is worth looking at different use cases of universities, locally and abroad. In the following sections, we will investigate what information channels students at **ACS** use and we will receive insights into how other universities tackle this challenge.

2.1 Current university context

As mentioned in **chapter 1**, university ecosystems are equally prone to information overload as any other context that makes use of technology tools and platforms nowadays. When discussing about **ACS**, it is worth pining down first the main information source categories, and further explain how these are all organized in the virtual environment. These main categories are:

- **official sources**, official platforms that directly post institution-related and administrative content
- **student organizations**, associations made up of volunteers that organize activities and events for all students
- **representative students**, specifically-pointed students that represent an intermediary layer between the institution and its students; these students are tasked to communicate relevant decisions from the faculty and give back collective feedback
- **other students**, students with no representative role, but which can still help spread relevant content, mostly active on social networks

Students at **ACS** use *Whatsapp*[4] predominantly to organize themselves in various groups and subgroups and to stay in contact with organizations or repre-

[4]https://web.whatsapp.com/

sentative students. This chatting app receives significant adoption because it is generally user-friendly and constantly pushes notifications on desktop and mobile devices. Additionally, students use this platform to spread news, do surveys or ask all kinds of administrative questions.

Separately, student organizations have each a separate web platform[5] but in practice use *Facebook*[6] and *Whatsapp* to notify events or opportunities. Such organizations work as reliable middleware between companies and students.

Due to the recent pandemic situation, all faculty-related activities were hosted on *Teams*[7] During the online period, courses, labs and several examinations were organized on this platform. While student organizations use social networks to push notifications, official sources use *Teams* heavily for notifying about recent opportunities or conferences. Official sources also have separate web platforms[8], which push weekly faculty-related content, but as we will present in the following chapter **3.3**, these platforms do not benefit from much activity. **ACS** employs *Moodle*[9] to manage assignments, collect student feedback at the end of each semester, and generally host course materials.

Finally, students at **ACS** have an *Outlook*[10] inbox, but the email system pushes mostly spam and less relevant content. Consequently, students rarely check their inboxes and prefer collecting their information from the other platforms. In comparison, as we will later see in the following section **2.2** when discussing other universities, *Outlook* tends to be heavily used as an information source, and in some cases, it is more than sufficient for a student to stay well-informed.

To summarise, there is no single predominant information channel at **ACS**, and students often need to both navigate school platforms and browse social networks to stay well-informed. Chatting apps or social networks tend to push content more visibly due to their notification systems, but their intricate and spammy structure takes a toll on students' time and resources. Furthermore, official platforms tend to be specialized in separate resource categories, and thus, students are at risk of missing out on relevant administrative news.

[5]https://lsacbucuresti.ro/ , https://eestec.ro/
[6]https://www.facebook.com/
[7]https://www.microsoft.com/ro-ro/microsoft-teams/
[8]https://acs.pub.ro/ , https://upb.ro/
[9]https://acs.curs.pub.ro/
[10]https://outlook.live.com/

2.2 Case study

For our case study, we conducted discussions with students from different universities. These discussions were held online, and from the start, we had the following questions in mind:

- How many platforms do they need to navigate to stay well-informed at their faculty?
- Do they have any emailing or feed system that constantly pushes notifications and relevant updates concerning faculty?
- Do they need to filter much spam to get to the relevant information?
- How often do they feel like missing relevant news?

We discussed with a law student at **The University of Cambridge**[11], and they revealed that most of their notifications and information concerning faculty is received over *Outlook*. Besides these, their meetings and conversations with teachers are also scheduled over *Outlook*, and moreover, student societies and organizations have an active newsletter that can be subscribed to via email. Therefore, they claim that they rarely miss any important news or encounter spam. The need to navigate platforms and switch contexts is dramatically reduced since all-important news comes from one information channel.

Similarly to the previous example, another student from **The Department of Computer Science from the University of Oxford**[12] described that *Outlook* is heavily used as an information channel. Students can subscribe to the newsletter of different student organizations or companies that later send them opportunities over email. However, they claim that their email is not devoid of spam and that they need to filter manually much content. Despite mainly using a single information channel, they sometimes miss essential news because of the existing spam.

Another student from **The Department of Physics and Astronomy from the University of Manchester**[13] revealed that they mostly use a single platform called *MyManchester*[14] which is a specialized university-based app for their institution.

[11] https://www.law.cam.ac.uk/
[12] https://www.cs.ox.ac.uk/
[13] https://www.physics.manchester.ac.uk/
[14] https://my.manchester.ac.uk/

From this platform, students can manage their entire faculty life and receive all important news. What is worth mentioning is that there is a separate section called *MyManchester News*[15] which acts as a general news feed from a wide range of sources: official, organizations or individual students. In addition, students can contribute with articles, while readers have the option of subscribing to different authors or sources.

When discussing with a student from the **The Bucharest University of Economic Studies**[16], they described a situation most similar to the one at **ACS**. They claimed they use a number of platforms for different information categories. For assignment management, they use the popular *Moodle*. At the same time, they receive their administrative and relevant news over another internal platform[17]. They revealed that they use mostly *Whatsapp* when communicating with teachers and staff members, and because of this, there is much spam generated, and consequently, they need to be constantly careful not to miss relevant updates.

2.3 Existing solutions

We tried to explore the existing solutions for **ACS** that try to achieve content aggregation and curation. More specifically, we investigated two existing products that have been popular in the latest years: *Politehnik*[18], an Android-exclusive mobile app, and *UPB Campus*[19], a cross-platform university-based app.

Unfortunately, **Politehnik** is an inactive app that has not been updated since October 2019. According to the app description on *Google Play Store*[20], it aimed to aggregate university-related events published on *Facebook*[21], and for this, it relied heavily on user manual input. As far as we know, to centralize these events, students would have communicated them to the app owner, and in turn, these events would have been inserted manually into the database. The app had no automated scraping process and mainly targeted the events published on the *Facebook* platform.

[15]https://studentnews.manchester.ac.uk/
[16]https://www.ase.ro/
[17]https://www.webstudent.ase.ro/
[18]https://play.google.com/store/apps/details?id=ro.politehnik.events
[19]https://unicampusapp.com/
[20]https://play.google.com/
[21]https://www.facebook.com/

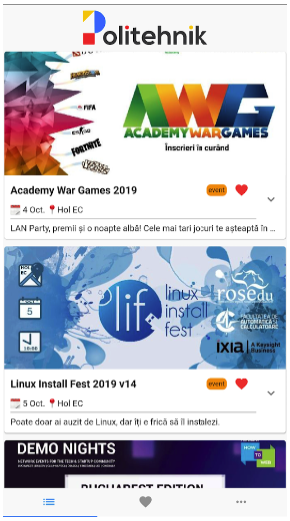

Figure 2.1: Politehnik-1

Figure 2.2: Politehnik-2

Figure 2.3: UPB Campus-1

Figure 2.4: UPB Campus-2

On the other hand, **UPB Campus** is currently maintained by the UPB student **Diana Scurtu** and is developed using the Flutter framework. The app is available on both Android and iOS, and currently, it displays news from the university website[22] and other technical blogs. When discussing with the developer, they revealed that they have the news scraped using HTML scrapers and centralized in a local database. While there is a degree of automation, the information sources are limited, and the news items are more or less in the official category and targeted at the university level.

[22]https://upb.ro/

Chapter 3

User study

Finding the right approach to our implementation requires, first and foremost, collecting feedback and insights from the students of the **ACS**. In the end, we do not intend to have only a functioning product but to deploy a feature that students could use on a daily basis. Therefore, our goals for this user study are:

- to understand how students are impacted by the problem of navigating multiple information sources
- to analyze how students use different platforms to retrieve their data
- to have a clear view of what features would best fit our students' needs

3.1 Methods

We employed two different methods for our research process: a **survey** sent to students of all years from **ACS**, and **discussions** with key student figures who hold representative functions and are well aware of our faculty's present problems. Due to the recent pandemic situation [1], we conducted our research process integrally via online means. While our discussions were held on social networking platforms, our survey was conducted via *Google Forms* [2] and distributed to several students *Whatsapp* [3] groups.

We decided to hold our discussions first to get a general view of our problem and better formulate our survey questions later. When discussing with these

[1] https://en.wikipedia.org/wiki/COVID-19_pandemic
[2] https://docs.google.com/forms
[3] https://web.whatsapp.com/

individuals, we tried to formulate our problem and ask for some general opinions about it. We intended to receive a general feel of how students are affected by the need to navigate several information platforms. Surprisingly, our discussions revealed that people at **ACS** had raised the issue of not having a centralized information channel before, but there had been no particular action on this matter.

Our survey was opened for an entire month in **December 2021** and received **112** total responses. Although Google Forms allows for visualizing data under different representations, we exported our results and used *Python Matplotlib* [4] and *Seaborn* [5] to plot our information.

3.2 Target audience

We tried to target students from both the Bachelor's and Master's stages for our study. **ACS** currently has a four-year Bachelor's cycle and a two-year Master's cycle. Our results follow the distribution illustrated in figure 3.1.

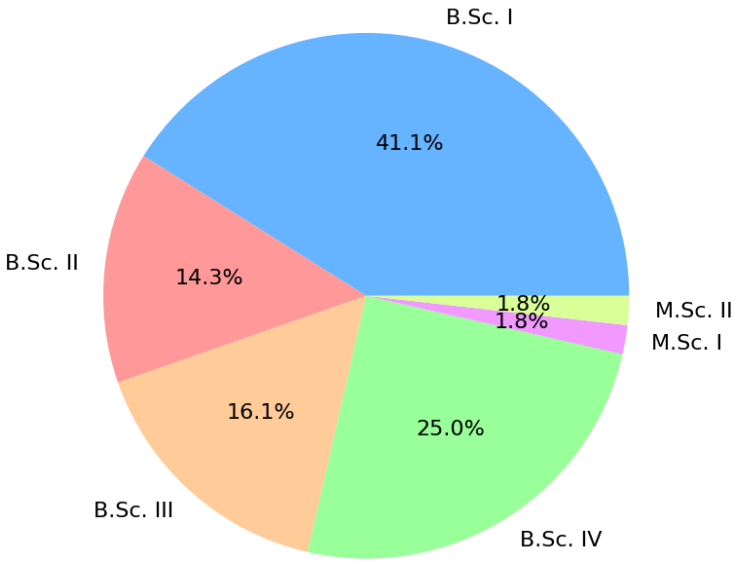

Figure 3.1: Students distribution by year

About **96.4%** of all submissions came from undergraduate students, whereas **3.6%** came from students enrolled at Master's degree. While most of the Bache-

[4]https://matplotlib.org/
[5]https://seaborn.pydata.org/

lor's submissions came from first-year students, the Master's distribution is equal along the two-year cycle.

Concerning the demographics, we tried to analyze the distribution of students that hold any significant or representative role within their collective. Our results are illustrated by the following barchart **3.2**. Students were not limited to choosing only one role, and in practice, some students may have multiple representative roles. Results show that some 14% of all students hold at least one position, 5% may be involved in student organizations, and 1% may be present in the Faculty Council.

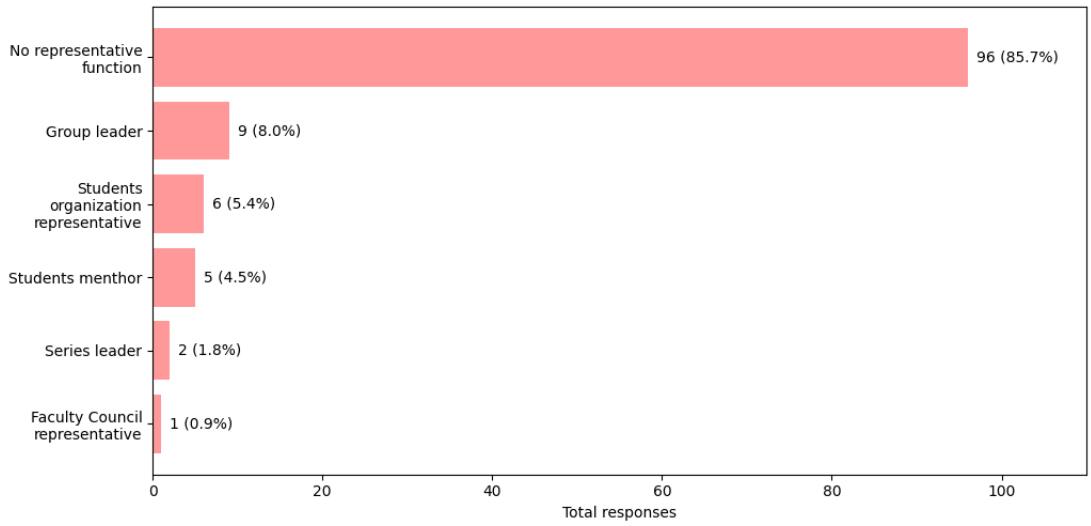

Figure 3.2: Distribution of students in representative roles

Additionally, we tried to assess how involved students are in different student organizations and impressively, close to 50% are present in at least one such association. The following chart **4.1** shows that almost half of all students combine faculty with extracurricular activities.

We concluded that our survey captured a good representation of the real-life demographics. While a small fraction of students (about 1/10 students) are involved in representative responsibilities, an impressive percentage is involved in student organisations or volunteer activities. Because not many Master's students completed the survey, we think our solution best fits the undergraduate group's needs. However, we are reasonably confident that it will positively impact the postgraduate students as well.

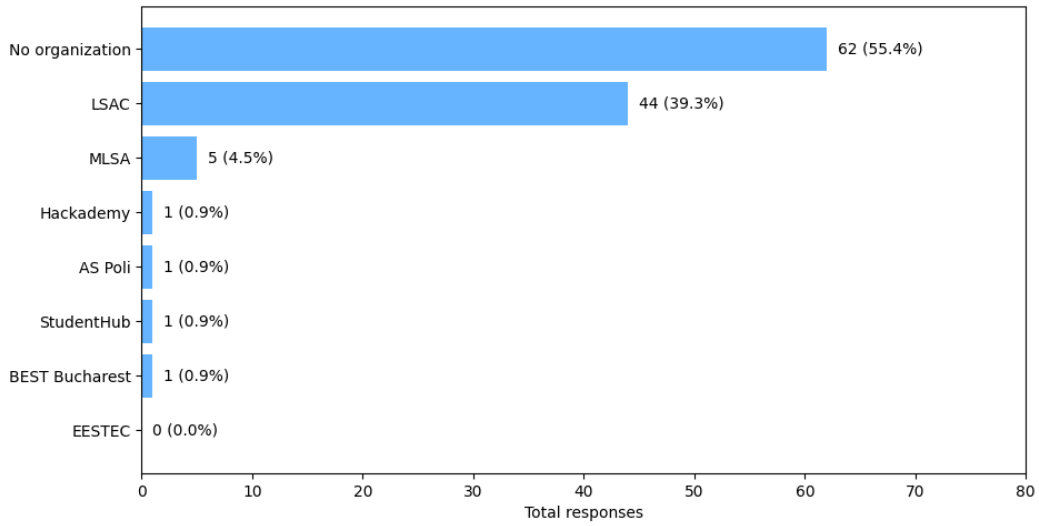

Figure 3.3: Distribution of students involved in organizations

3.3 Results

Following our survey, we tried to reach our goals mentioned in the opening of this chapter. We tried to provide a survey with clear questions and flexible answering methods, so that we capture as many details as possible about the relationship between students and their information sources.

3.3.1 Student behaviour

One of our questions focused on students' behaviour and how they collect their information. We listed the leading platforms for students at **ACS** and several resource categories. Our results consist of a heatmap 3.4 showing what platforms are most used for a specific resource category.

The majority of responses show that the official platforms (*faculty*[6], *university*[7]) are significantly less used in all categories in comparison to all the others. Survey shows that *Whatsapp* is by far, the channel most used when it comes to students

[6]https://acs.pub.ro/
[7]https://upb.ro/

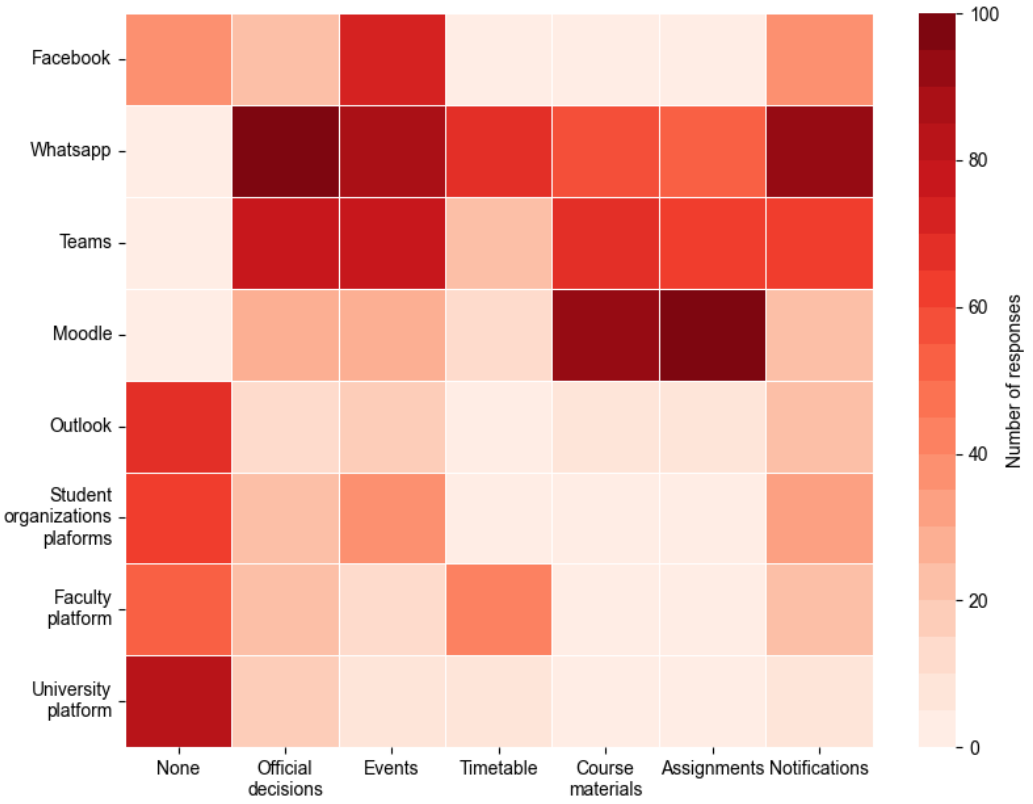

Figure 3.4: Relationship between platform and searched resources

getting their information and resources. This result makes sense since chatting apps are faster and more efficient in terms of real-time communication, but these are not necessarily protected from spam and unregulated content. Since the break of the recent pandemic, all faculty-related activities have moved online on *Teams*[8] and survey shows that is heavily used by students. *Moodle*[9] remains specialized on assignments management, but tends to fail in other categories. Despite its convoluted nature, *Facebook*[10] remains one of students' favorite choice regarding events and faculty announcements.

As a special note, *Outlook* is less relevant to students at **ACS** than the majority of other platforms. While other faculties tend to heavily make use of Outlook or

[8]https://www.microsoft.com/ro-ro/microsoft-teams/log-in
[9]https://curs.upb.ro/2021/
[10]https://www.facebook.com/

any other mailing service (as mentioned in **chapter 2**), students at **ACS** find it less important or visible when navigating their daily faculty content. We could argue that email is generally a powerful tool to push relevant content to people, but in this particular case, it fails to achieve its intended purpose.

Following this result, we could draw a few conclusions. First, students choose to stay informed on platforms that push content and activity constantly. Second, it could be argued that social networks and chatting apps significantly win because they have tailored apps for both desktop and mobile versions and more efficient UI/UX features. Finally, official platforms might lack popularity for many reasons. These may have a weak user interface that does not adhere to most modern standards, rare fresh content and little activity (such websites tend to work more as virtual panels rather than engaging platforms), and these lack a push-notification system.

3.3.2 Problems

Besides students' behaviour, we investigated what problems students deal with when they try to stay informed on a daily basis. We listed several common problems, and we had students give an appreciative score on how much that issue affects their information searching experience.

We had a five-step grading system: **Totally annoyed**, **Annoyed**, **Neutral**, **Relatively indifferent** and **Indifferent**. When putting all data together, we gave a specific score to each step, as shown in the figure 3.5.

The most adverse problems that students confront with are: the spam generated by users, the lack of a precise posting chronology and the lack of search or filtering mechanism. Besides the usual spam, students point out that a clear disadvantage of navigating multiple platforms is the lack of organization and a standard format that would allow them to sort the content uniformly. A precise chronology or a filtering system are issues that should be considered when designing our solution. The need to repost the same content to multiple groups is the most polarizing problem since equally as many students are disturbed as students who are indifferent to this issue. Finally, the lack of notifications visibility is surprisingly one of the least negative issues, but we argue that notifications still impact a lot the students' engagement factor.

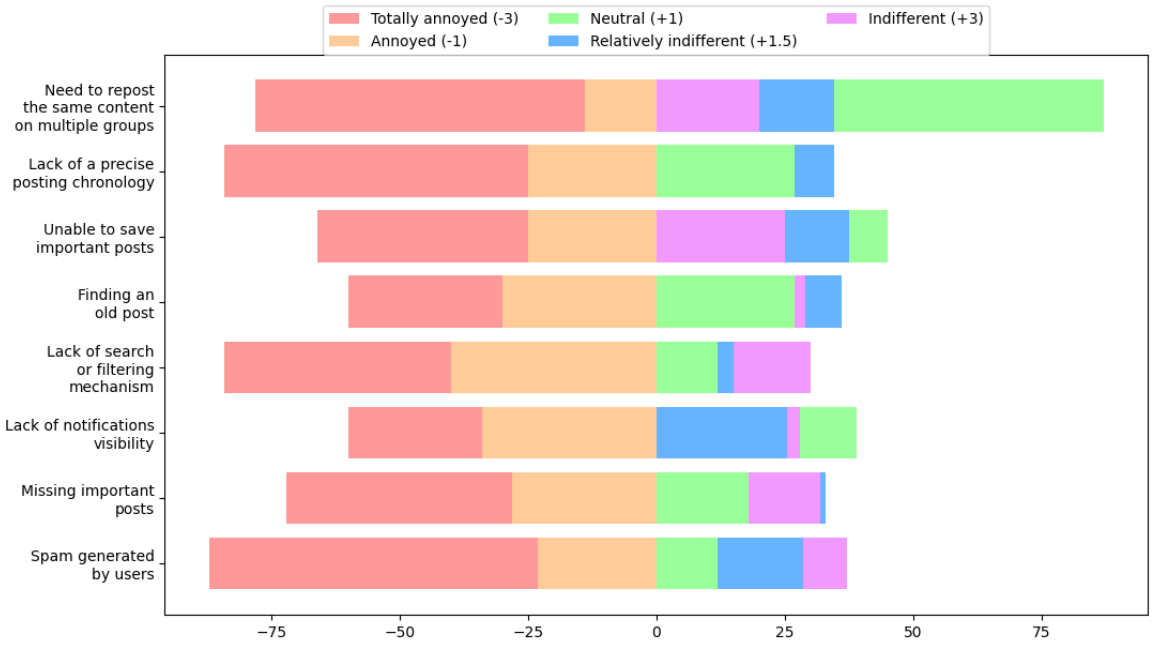

Figure 3.5: Degree of impact for each problem

Having a survey confirmation of what problems are the most priority to deal with, we can follow through and assess which features would best fit the students' needs.

3.3.3 Solutions

For this evaluation, we proceeded similarly to the latest sub-chapter 3.3.2. We listed several features and we had students grade them based on an intuitive five-grade scale: **Useless**, **Not so useful**, **Relatively useful**, **Useful** and **Very Useful**. Each grade was assigned a specific score, and by putting all data together, we obtained the following chart 3.6:

The need to have all news in one centralized feed is undoubtedly the most requested feature with predominantly positive feedback. Second to this, the ability to filter and sort posts received great appreciation from students, and this result ties up naturally with the scores obtained in the previous chart 3.5. Moving forward, the following features received big scores as well and are worth taking into account when designing our news feed aggregator: the ability to share a post externally, the ability to save a post on the device, option to view a post both in

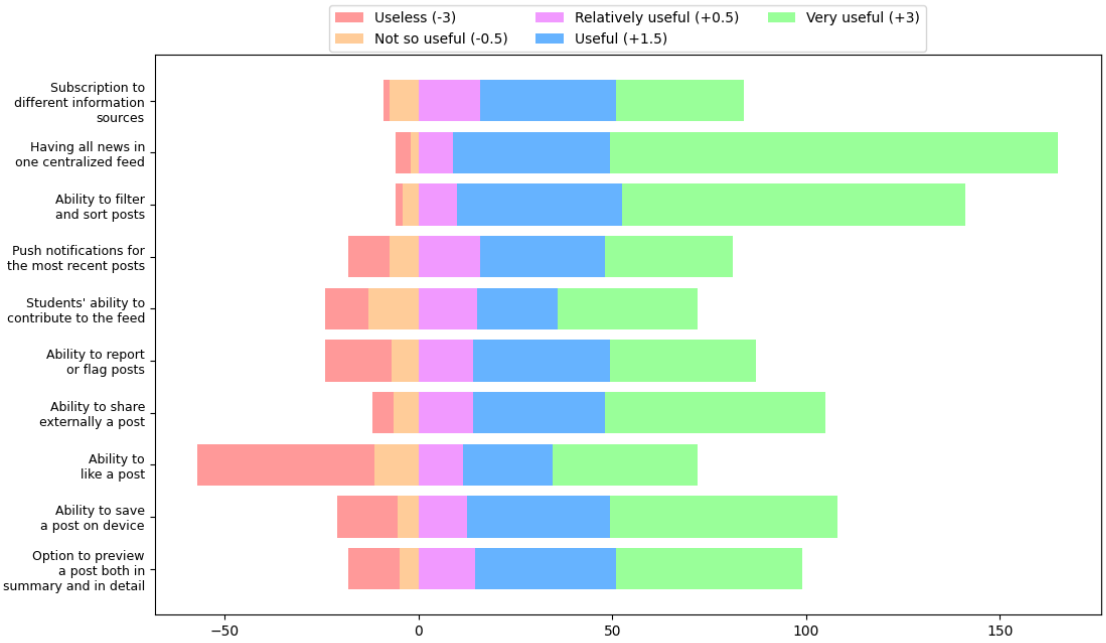

Figure 3.6: Degree of usefulness for each feature

summary and in details, push notifications, subscription to different information sources.

Interestingly, the ability to like a post received the most negative feedback. We could argue that such social actions (like, comments) characteristic of most extensive social networks are not appropriate to our use case. Although the students' ability to contribute to the feed received a similar positive score, we consider this feature to be necessary since, in the end, we want our feed to collect information from several sources, official and non-official, such as students. Nevertheless, having a system that carefully grants posting roles to individuals should strike the right balance between our goal and the students' feedback.

Chapter 4

UI & UX Design

No matter what functionalities we choose to employ in our news aggregator module, we need to build towards an interface that should be usable and easy to pick up regardless of the degree of a user's experience with the rest of the existing app. The new module should blend seamlessly with the current design and aim to complement the other functionalities of the app. We need to follow not only existing design guidelines in the app, but guidelines that are generally followed and predictable in several other modern apps.

Following the results of our previous user study **3.3**, we will discuss in this chapter the core user-facing functionalities of our module and the general framework that influenced our design choices. For prototyping, we used *Figma*[1], a powerful online tool for sketching, and in this chapter, we will describe the iterative process that led from our initial proposals to the final design of our module.

4.1 General design

ACS UPB Mobile is a cross-platform app developed using the Flutter framework from Google[2]. From a single code base, Flutter compiles our app to native code for Android, iOS and web, but being able to run an app on a platform and having it look according to industry standards are two separate issues. Therefore,

[1]https://www.figma.com/
[2]https://flutter.dev/

when discussing the general design framework for our current app, we have to understand what design guidelines and philosophies all these systems follow and what developers should be aware of when adapting an app from one platform to another.

Android apps generally follow the Material Design principles[3], an adaptive set of guidelines and tools introduced by Google at Google I/O 2014 [10]. According to their official documentation[4], this framework *"is guided by print-based design elements...to create hierarchy, meaning, and focus that immerse the user in the experience"*. On the other hand, iOS apps follow the Apple Human Interface Guidelines[5], a set of official principles that apps must adhere to when published to the App Store. Fortunately, to make it easier for developers, Flutter offers them a wide range of out-of-the-box widgets they could use for each platform, more specifically, Material[6] widgets for Android and Cupertino[7] components for iOS. These widgets represent standard components that adhere to each framework's guidelines from a visual and behavioural point of view. Such components are buttons, icons, loading indicators, and alert dialogues; the list is not exhaustive.

However, achieving a perfectly-native design for both platforms with one single code base is a thorough process, and it generally requires platform-aware widgets that adapt their design accordingly or separate implementations for different functionalities. **ACS UPB Mobile** design focuses mainly on the Material guidelines, and for our purpose, we designed our module to follow best this framework guidelines. Nevertheless, our end product will run smoothly on both Android and iOS devices.

4.2 User functionalities

Based on our results obtained from the user study, we selected a number of functionalities worth taking into consideration for our feature. To get a clear view of how we should further develop our UI/UX, we tried to understand how each

[3]https://material.io/design
[4]https://material.io/archive/guidelines/layout/principles.html
[5]https://developer.apple.com/design/human-interface-guidelines/guidelines/overview/
[6]https://docs.flutter.dev/development/ui/widgets/material
[7]https://docs.flutter.dev/development/ui/widgets/cupertino

individual feature would impact our overall module by assessing its importance and usage frequency. We employed a red routes matrix approach to plot the most critical paths and to understand which features would be a high priority for our news aggregator module to work and which features would be more redundant than helpful from a user point of view.

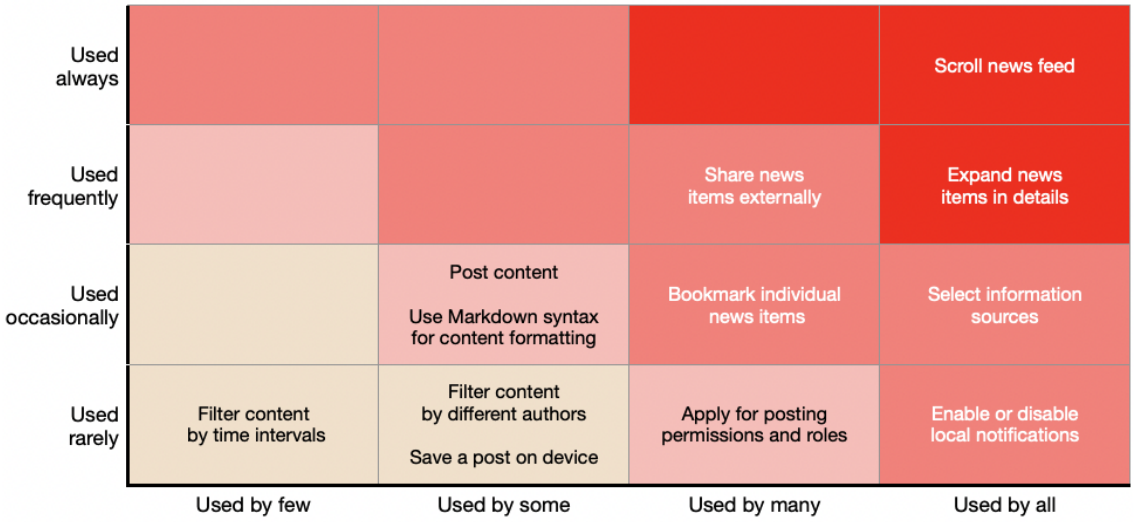

Figure 4.1: News aggregator red routes matrix

Unsurprisingly, the most crucial feature is having a main news feed page where users can scroll for their posts. Needless to say, this is the core functionality of our whole module, and therefore, it will be used by all users whenever they want to check for their news. Students will scroll the feed and see a relevant preview of each news item, and more than often, they will expand a particular post to read its entire content on a separate details page.

Regarding the social actions, following our user study, we concluded that actions such as liking or reporting a particular item do not bear any meaningful impact on our app. For example, liking a post does not influence a student's choice to view or not view an item they are interested in, whereas reporting a post is unnecessary since our information sources would be verified and, more or less, official. However, sharing externally a post is an excellent way to incentivise user traction and motivate other students to use the app. Sharing a dynamic link that sends a user to a specific item within our app would prove a common use case for our module.

Bookmarking is an essential ability for users to build their collection of

favourite items. When considering this feature, we pondered the idea of letting users download their items locally on the device to have these posts available in both connected and offline mode. However, we concluded that while bookmarking an item may be an occasional action, downloading a post may be redundant for our app focus. For example, it is doubtful that a student would need to download a faculty post and access it at some point in offline mode. Moreover, we will employ Firebase services when building our module, which already do some local caching out-of-the-box.

Users should be required to select the information sources or enable their notifications before using the news feed. When logging into the app, users should immediately be prompted to perform these actions. Afterwards, they may occasionally resume to these settings to change something.

Even though students gave positive feedback on having sorting and filtering mechanisms, we concluded that these might not be so relevant for our news feed. It all comes down to our app's content load and users' need to revisit past posts. In practice, students would not resume that often to past news since such items mainly address present issues which are not that relevant in the long term. In other words, there would not be such posts that a user would feel the need to revisit after a long time. Moreover, the reduced frequency of updates would not hinder a user from scrolling directly to a post they are interested in from a few months ago, should that be the case.

Finally, there is a strong need to implement a role-based mechanism and administer which students have the right to post within our app. We wanted to give students the freedom to apply via an in-app form for such rights, and they should be aware of this possibility when using the news feed. We felt that our module should act as a content aggregator from different sources and simultaneously enable students to contribute with valuable posts for their colleagues.

As **Ioana Alexandru** described in her thesis [11], **ACS UPB Mobile** employs different permission levels that control which actions a user can do within the app. The highest level is reserved for just a handful of admin users, who accept or decline permission requests, and below them is the group of users that can add or edit public information. Students must be given these editing permissions beforehand whenever they want to post content. Additionally, since these students might hold one or more official titles in their collective, we want to give them

the possibility of having separate roles within our app. For example, a student that is both president of a student association and a representative in the Faculty Council might hold two roles. Ultimately, these roles should help students better place their posts in the right information source category. Students might post on behalf of the faculty, of a student organization, or themselves; therefore, all posts coming from students should not be placed by default in the same information source category. We will further explain our role-based approach in **chapter 5**.

Although users may not post frequently, this is an essential feature we want to employ for our module. Enabling **Markdown syntax**[8] formatting would help students personalize and design their posts more attractively, so we were confident this would be a feature that they would use.

Summing up, this is how we trimmed down our initial features list to the one described in the proposed functionalities section **1.4**. Again, we tried to take user feedback into account and we tried to understand what our news feed module focus will be.

4.3 General user workflow

4.3.1 User stories

The following user stories reflect how users should interact with different parts of our module. We distinguish between three types of accounts: regular **User** with no publishing rights, **Editor** with both publishing rights and roles, and **Admin** with supreme rights. For clarification, every Editor is a User, but not every User is an Editor. An Admin is both User and Editor.

As a User:

- I want to scroll my news items from a centralized feed page.
- I want to have main categories for my news, such as All or Favorites.
- I want to bookmark a post, so that I can revisit it in my Favorites category.
- I want to share a post externally via URL.

[8] https://www.markdownguide.org/basic-syntax/

- I want to select my information sources, so that I receive filtered content.
- I want to choose to enable or disable notifications for news updates.
- I want to be able to apply for editing permissions, so that I can become an Admin and start publishing content.

As an Editor:

- I want to be able to apply for posting roles, so that I can have different posting titles.
- I want to be able to compose a new post.
- When I compose a post, I want to be able to choose a specific target group and posting role.

As an Admin:

- I want to accept or decline requests from both Users and Editors.

4.3.2 Workflow diagram

To better illustrate how our news feed module and additional admin features will integrate with the rest of the present app workflow, we propose the following state diagram **4.2**.

4.4 Prototyping

With a list of relevant functionalities in mind, we started prototyping our design using **Figma**[9]. During the process, we tried to get a good grip on the design guidelines present in the **ACS UPB Mobile** app, and we worked in a close feedback loop with members of the contributing group. Our approach was more not to reinvent the wheel but try to reuse existing, proven functionalities and practices. Moreover, we try to have our new design not interfere with other critical paths in the app.

[9]https://www.figma.com/

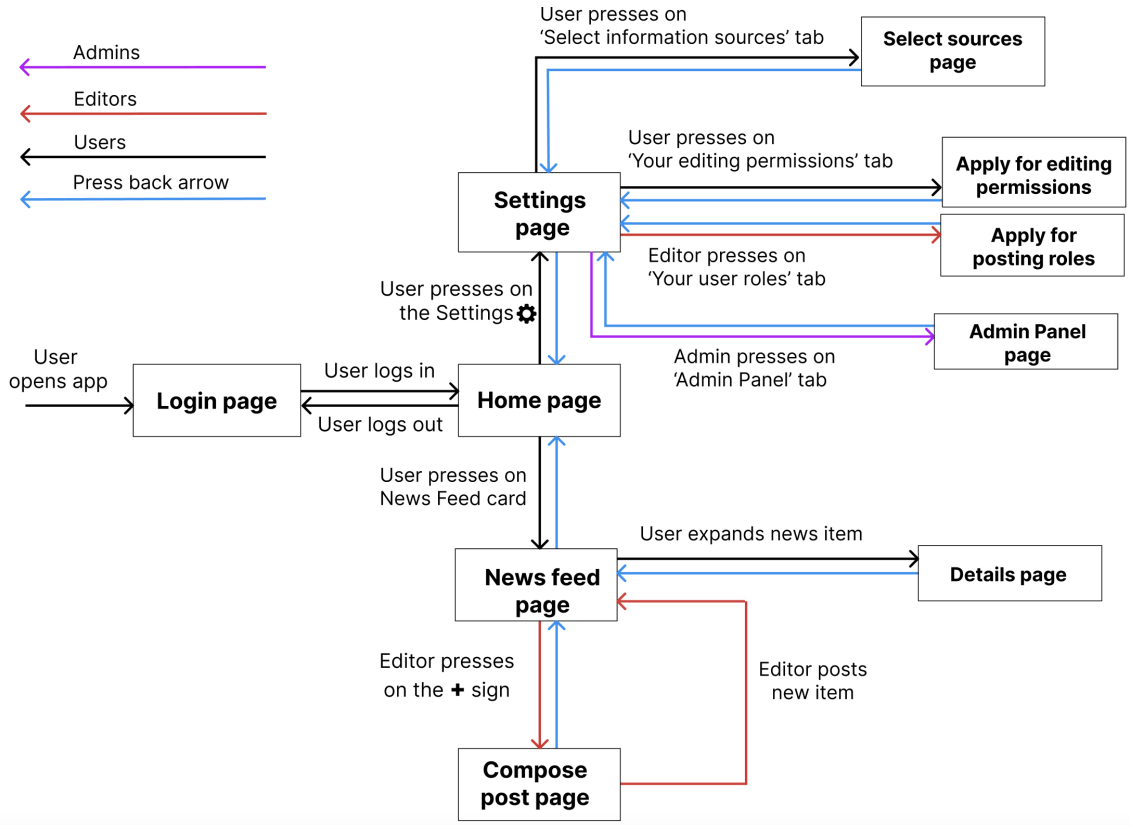

Figure 4.2: General workflow diagram

4.4.1 Initial design

For our initial design, we took inspiration from the current app design and tried sketching the main screen without emphasising the details. In addition, we tried to use icons, several text labels and and follow the app's colour theme.

The main news feed page **4.3** should have two or three big categories that could be accessed via a tab navigation bar: **News**, **Favorites** or **Personal**. The News category should list all available news items, whereas Favorites has the bookmarked ones and Personal has the posts that were composed by that user. Students will be shown the first two or all three categories, depending on their editing permissions. Each news item has a preview body, an author, a target group, a timestamp and a bookmark action, and when the user clicks on a specific item to expand it and is redirected to a separate details page. In addition, the main page has a plus icon in the top-right corner from where a student can compose a post.

The details page **4.4** contains the entire post content and has two actions:

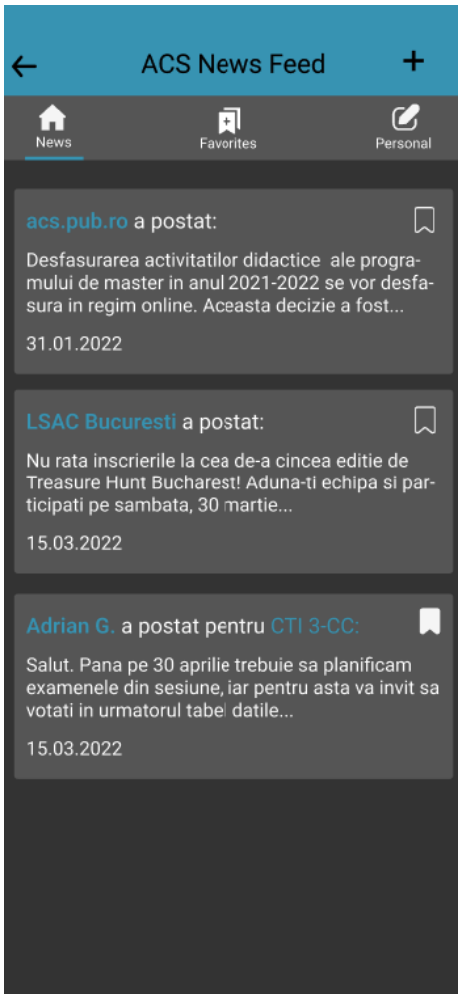

Figure 4.3: Draft news feed main page

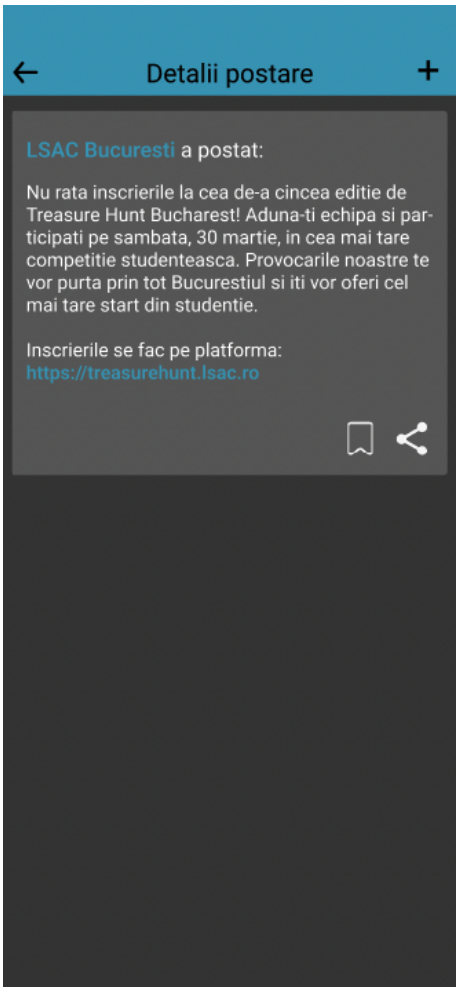

Figure 4.4: Draft news details separate page

bookmark and share. Moreover, whether a post was composed by the current user, we may choose to show a Trash icon so the user can delete the post if they choose.

We pondered on the idea of splitting the compose post action in multiple steps. Initially, the user chooses the title and body **4.5**, then the target group **4.6** or his role. Before publishing the post, they are shown a preview of the post **4.7** to make sure everything looks according to their intention. Later, in the final design, we dropped the idea of having multiple screens and we put everything on a single page to be consistent with the rest of the app.

The user configures their information sources from a separate screen **4.8**. For the moment, we provided three main sources: **Official**, **Student Organizations**

Figure 4.5: Draft compose post - content

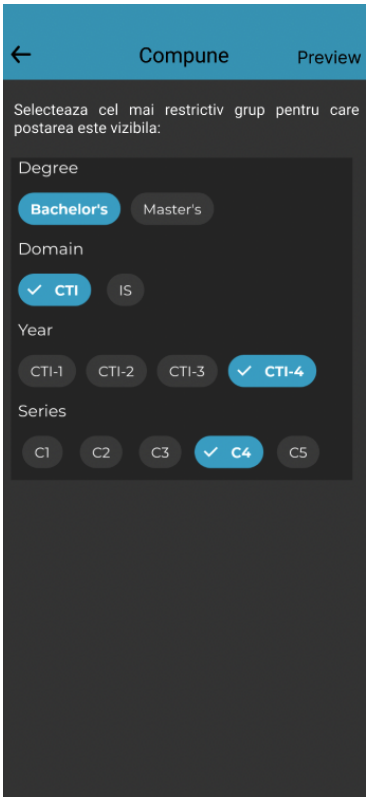

Figure 4.6: Draft compose post - relevance

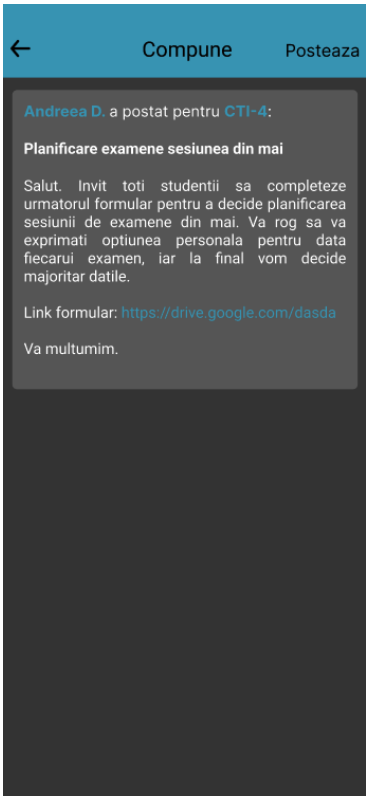

Figure 4.7: Draft compose post - preview

and **Representative Students**. From the same screen, we wanted to give the user the possibility of enabling his notifications as well.

When requesting permissions or roles **4.9**, the user must fill in a form and, most importantly, specify a reason for his request. Requests will be handled separately by admins on a separate screen that is already present in the application.

4.4.2 Final design

Starting from our initial design and our proposed functionalities, we arrived eventually at our final user interface.

As described in our **Workflow diagram 4.3.2**, by the time the user logs in the app, they arrive on the Home page **4.10**. They are presented with the news feed card that transitions them on click to the main news feed page **4.11**. Our feed categories are: **News**, **Favorites** and **Authored** (the last category will be shown depending on whether the user has editing permissions). Users can navigate to

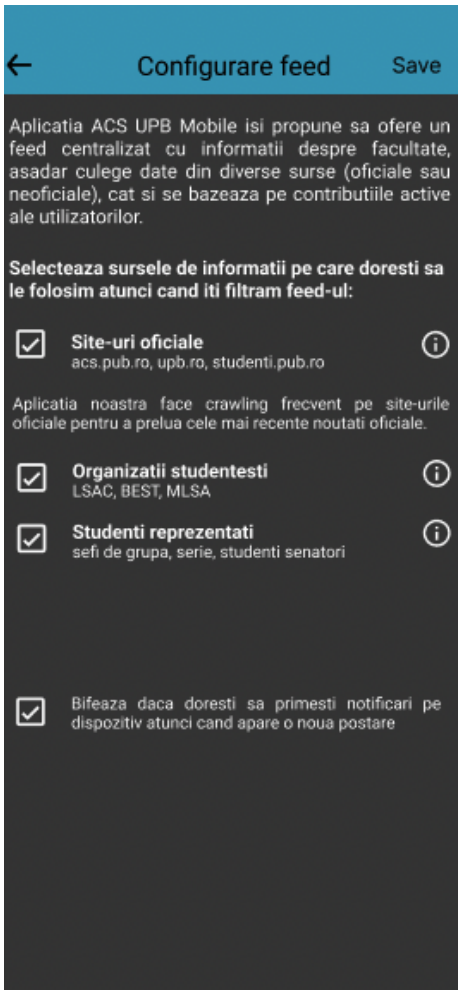

Figure 4.8: Draft select sources

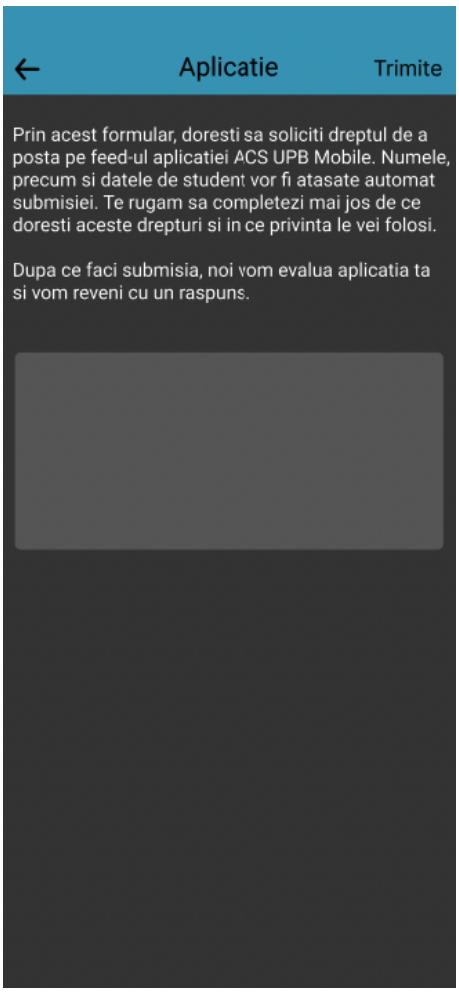

Figure 4.9: Draft request roles

the details page of a news item **4.12** when expanding it on click or opening it from a shared deep-link. Logged-in users can have multiple actions: bookmark, share or delete. The last option is enabled only if the current user is the author of that specific post. Bookmarked posts can be visualized in the favourites category on the main news feed page. Likewise, authored posts can be easily accessed from their according category.

We wanted the plus icon for adding a new post to be visible for all user types. When pressing on it, a normal user would receive a hint **4.13** to fill in a editing permissions request form by navigating to Settings **4.16**. On the other hand, an Editor user is transitioned to a new page, where they can start publishing a new post. When composing a post, users choose a role, target group, title and body **4.14 4.15. ACS UPB Mobile** can filter students based on their current year distribution

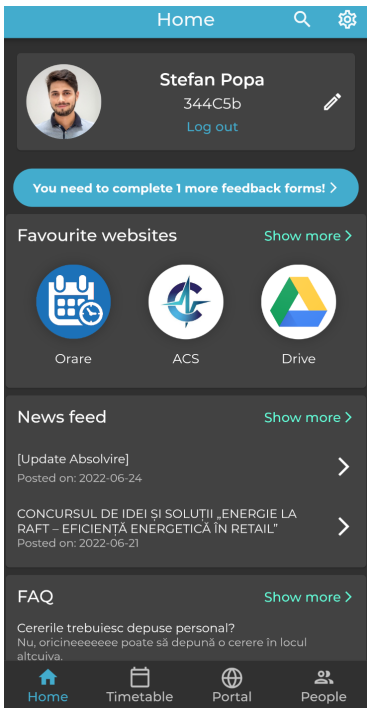

Figure 4.10: News feed card on Home page

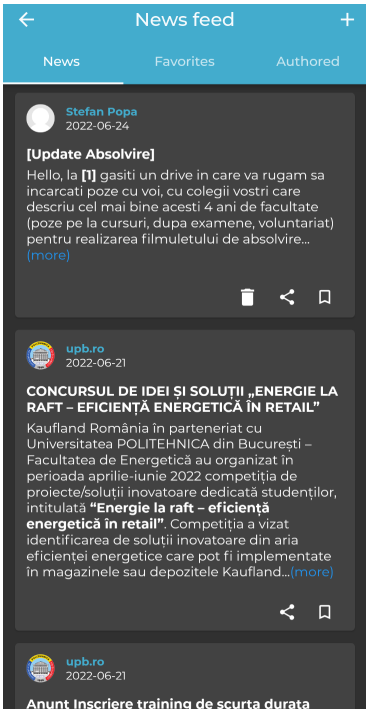

Figure 4.11: News feed main page

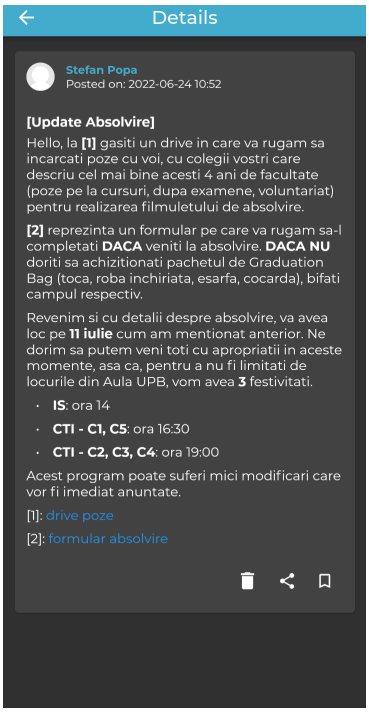

Figure 4.12: News details separate page

(year, specialization, group, subgroup). We use the current architecture to target posts to one or more specific student groups. Editors can use Markdown syntax to give more expressiveness to their posts, and a live preview shows them exactly how the Markdown content is rendered.

Users navigate the settings and toggle the news slider to enable their notifications. When enabled, users receive notifications **4.21** whenever there is a new post available[10]. Pressing on a notification would redirect the user to the news feed page **4.11**. Furthermore, they can configure their information sources from the following settings page **4.17**. When configuring their sources for the first time, users are directly prompted on successful login to this page, thus giving them the opportunity to select their information categories as early as possible. By default, users are subscribed to all categories because we wanted them to have access to all available information from the get-go.

When requesting roles **4.18** users need to select a role category, and a specific role name, and they must acknowledge the responsibilities of their actions. We

[10]notifications are currently available only on Android because iOS devices are more restrictive and require a developer license key

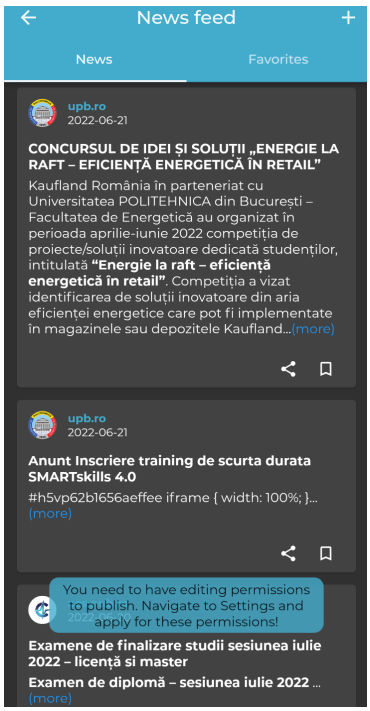

Figure 4.13: User without editing permissions

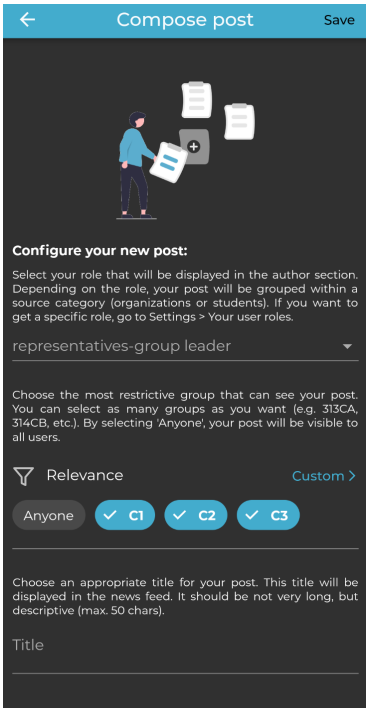

Figure 4.14: Compose post page - 1

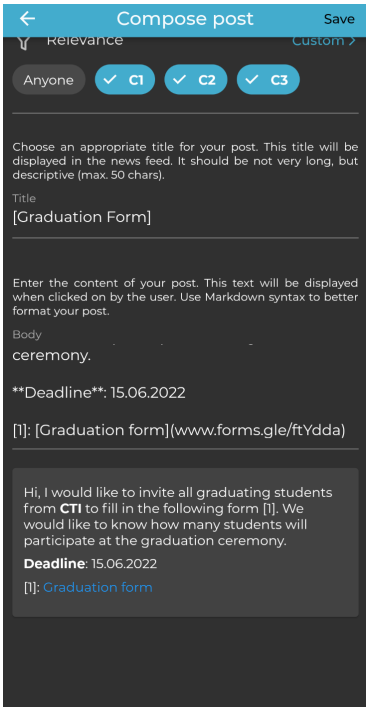

Figure 4.15: Compose post page - 2

took inspiration from the editing permissions request form already present in the app **4.19**. Before applying for a publishing role, a user must acquire their editing permissions since we plan on giving restricted access to who can post content. After filling in a submission, an Admin reviews this application in the Admin Panel **4.20** and decides on accepting or declining it. This panel has two categories, one for editing permissions and the other for publishing roles. Upon accepting an application, the admin is redirected towards an emailing application to send the confirmation email to users.

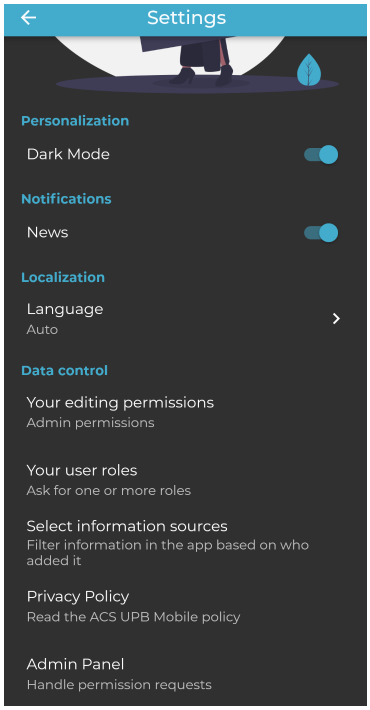

Figure 4.16: Settings main page

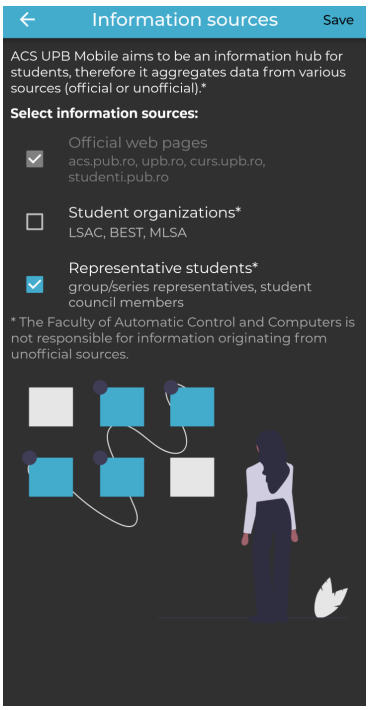

Figure 4.17: Select sources page

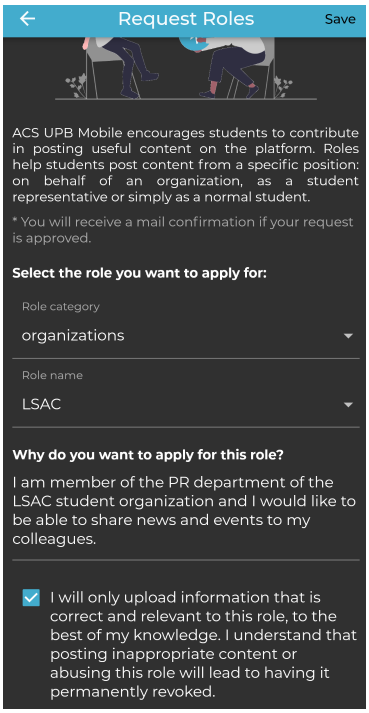

Figure 4.18: Request roles form page

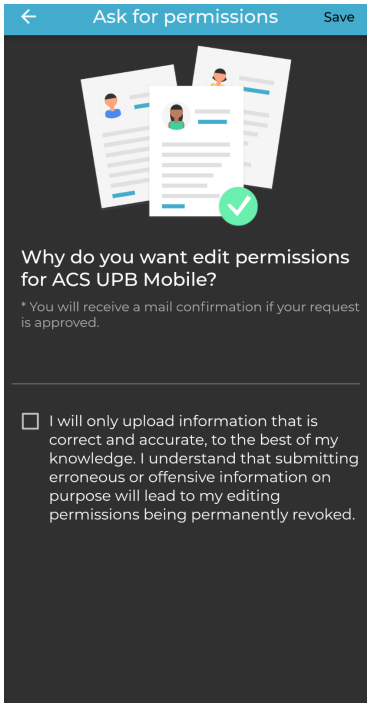

Figure 4.19: Editing permissions request form

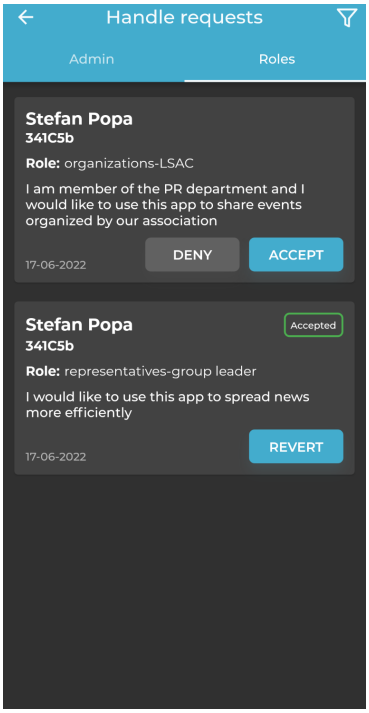

Figure 4.20: Admin panel for handling requests

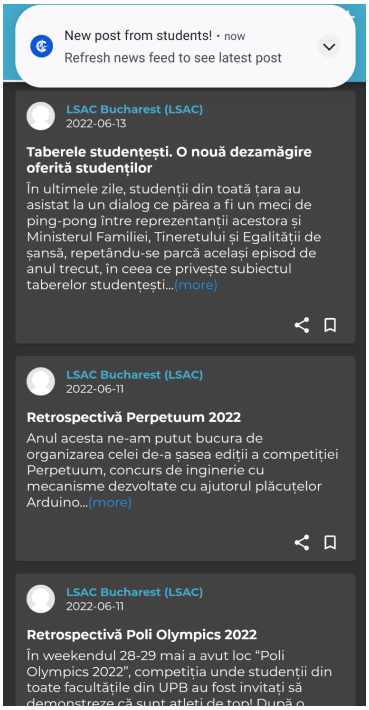

Figure 4.21: Notifications example

Chapter 5

Architecture and implementation

Having clarified the user-facing functionalities and the general design framework for our module, we will explain the system-related features and architecture that support our news aggregator module.

5.1 Flutter and ACS UPB Mobile

ACS UPB Mobile is implemented using the Flutter framework from Google, and the app's most recent release is stable for the **Flutter 1** version. Flutter uses the Dart object-oriented programming language and enables developers to compile their app natively to Android, iOS and the web with one single code base. The ability of targeting cross-platform systems means a larger user audience and solves the need of having separate native developer teams for each platform. **Ioana Alexandru**, the original creator of **ACS UPB Mobile**, has considered Flutter as the choice for this project upon careful and considerable comparison between this framework and other popular solutions [12], such as *Xamarin*[1] or *React Native*[2]. Its code reusability and growing community were strong arguments among others for choosing Flutter.

Our news aggregator module is written using Dart. Dart supports common modern features in the current programming languages spectrum, such as type and null-safety, concurrency, platform independence, flexible compilation, browser support; and in addition, it is open-source and is backed by a large

[1] https://dotnet.microsoft.com/en-us/apps/xamarin

[2] https://reactnative.dev/

developer community.

Flutter developers can publish package repositories on *pub.dev*[3] from where these can be downloaded and used in other projects. Our app uses several such packages to achieve different functionalities, like for example, image picking[4], URL launching[5] or deep-linking[6].

5.1.1 Upgrade to Flutter version 2

Our current target SDK for Android is **version 29** on the release branch, and for several months, there have been increasing efforts to upgrade the code base of our app to **Flutter 2**. The target SDK referes to a property that "tells the system for which Android version the app was designed and tested on" [13]. This upgrade is a necessary change since, according to this official Google Help post[7], new app updates must target SDK **version 30** starting with **November 1, 2021** and existing apps must meet this requirement by **November 1, 2022**. Otherwise, apps that fail this condition will no longer be available on Google Play Store for new users.

Flutter 2 enables target SDK for **version 30** and forces code migration to null-safety standards. For our app to be eligible for migration, all packages used in our app must be null-safety compatible. Consequently, this involves either using packages at their latest null-safety compatible versions or replacing them with other packages altogether. Currently, there are a number of pull requests[8] that await approval or reviewing, without which our migration process could not be finalized. When merged into the main branch, our news aggregator module should already be eligible for Flutter version 2.

5.1.2 Code overview

Flutter follows the declarative programming paradigm, which according to the official documentation[9], "builds its user interface to reflect the current state" of the

[3]https://pub.dev/
[4]https://pub.dev/packages/image_picker
[5]https://pub.dev/packages/url_launcher
[6]https://pub.dev/packages/uni_links
[7]https://support.google.com/googleplay/android-developer/answer/11926878
[8]https://github.com/student-hub/acs-upb-mobile/pulls
[9]https://docs.flutter.dev/development/data-and-backend/state-mgmt/declarative

app. The following **diagram 5.1** from the official documentation[10] illustrates this concept. This programming paradigm is different from the imperative framework because it focuses on the desired output rather than on the processes and actions that lead to that output. The crucial element of the declarative framework is the state, and each state change triggers the rebuilding of the UI.

Figure 5.1: Declarative paradigm equation

Our Flutter UI consists of components called widgets. Everything in Flutter is a widget, and all these components are placed within a data structure called the render tree. Flutter uses *aggressive composability*[11] to compose widgets progressively out of more basic widgets. Each widget component can be stateless (it does not have a state) or stateful (it can change its internal state). When building the UI, the render tree determines which components are affected by the state changes and updates the UI accordingly. For this, Flutter uses a series of optimization techniques[12] to achieve sublinear building times and fast performances.

When writing our code for the news aggregator module, we followed the current code guidelines in the project. **ACS UPB Mobile** uses a variation of the **BLoC (Business Logic Component)**[13] design pattern, which decouples the UI entirely from the business logic. The UI is described inside the **View** layer and regularly emits events to the **Service** layer, which is encapsulated by our business logic component. Our services contain functions that receive events and output values that can be further used in updating the View state. These services wrap the entire widget tree and, as a result, become globally accessible and reusable from any widget component. Finally, the **Data** layer responds to the asynchronous requests coming from services and uses data models for exchanging information.

[10]https://docs.flutter.dev/assets/images/docs/development/data-and-backend/state-mgmt/ui-equals-function-of-state.png

[11]https://docs.flutter.dev/resources/inside-flutter#aggressive-composability

[12]https://docs.flutter.dev/resources/inside-flutter

[13]https://www.mitrais.com/news-updates/getting-started-with-flutter-bloc-pattern/

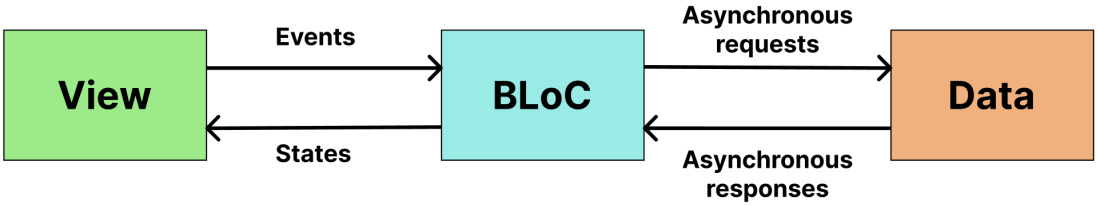

Figure 5.2: BLoC diagram

5.2 Firebase

ACS UPB Mobile employs *Firebase*[14] services from Google for achieving a serverless backend infrastructure. Firebase is part of the same category as *Amazon Web Services*[15] or *Azure*[16] and takes care of many common functionalities such as authentication, database, storage or cloud functions. Some of the most significant advantages of a serverless architecture are less operational complexity, a pay-as-you-go pricing model, and an autoscaling feature. Developers concern themselves less with the cloud maintenance and more with the actual functionalities they want to implement. In addition, the large documentation and available resources make for a straight-forward integration of Firebase with a wide range of projects, including with Flutter.

For our news aggregator module, the most relevant services we used were *Firestore*[17], *Firebase Cloud Functions*[18] and *Firebase Cloud Messaging*[19]. Building upon an existing project meant we did not need to concern ourselves with core functionalities, such as authentication or hosting, but instead, we could focus on directly implementing our features described in **chapter 4**.

5.3 Database

During our implementation process, we worked on a different development environment to not cause any damage or unrelated changes to the production

[14]https://firebase.google.com/
[15]https://aws.amazon.com/
[16]https://azure.microsoft.com/en-us/
[17]https://firebase.google.com/docs/firestore
[18]https://firebase.google.com/docs/functions
[19]https://firebase.google.com/docs/cloud-messaging

one. Moreover, data from production is regularly synced with the development database so that we can replicate issues or start testing features directly on accurate information. For our database layer, we employed Firestore, and we defined or changed several data models to accommodate our needs.

5.3.1 Firestore

According to the official documentation[20], Firestore is a NoSQL document-oriented database service with many advantages concerning performance, scaling and ease of application. Besides this, Firestore provides offline query support, local caching, security rules, and database triggers with defined cloud functions. However, it does not support aggregation queries and has certain limitations, such as 1MB document size limit or document write frequency limit of 1 per second. Nevertheless, for the purpose of **ACS UPB Mobile** and of our news aggregator module, this solution provides more than enough in terms of performance and usage.

Data in Firestore is organized in collections that can hold multiple documents, and each document can have multiple sub-collections with nested sub-documents. Collections are schemaless, and documents organize their data in key-value pairs. In the following subsection **5.3.2**, we will describe our collections' structure.

As a side note, Firestore employs security rules that govern who can read, create or modify existing data in a certain collection. The following code snippet represents an example of the syntax used. The shown example means: anyone can read all documents in the **news** collection, only authenticated users can create new documents, only the admins and the author can modify or delete a specific document.

```
match /news/{news} {
    allow read: if true;
    allow create: if isAuthenticated()
    allow update: if isOwner() || permissionLevel() >= 3;
    allow delete: if isOwner() || permissionLevel() >= 3;
}
```

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

5.3.2 Models

For our news aggregator module, we created new collections or modified existing ones to store all the information we needed. We had to accommodate the need to store additional user information, the data about the available news items, and the information that supports our push-notifications system described at **section 5.4**.

users

This collection stores the information related to users. Since rigorous database rules protect each user's document, we tried to fit as much information related to a user as possible in their own document and avoid using other collections. For simplicity, our model **5.5** describes only the fields that were added for the purpose of our module.

Table 5.1: **users** collection (only additions)

Field	Type	Description
receiveNotifications	`boolean`	checks whether the user has enabled the notifications
roles	`array<string>`	roles that a user can select from when publishing
sources	`array<string>`	information source categories that the user is subscribed to
bookmarkedNews	`array<string>`	list of all references of news documents bookmarked by the user

news

This collection stores all available news items and their information. Each news document contains information about the title, body, author, timestamp and other metadata that helps organize the items according to each user configuration and display them in a user-friendly manner. As a side note, the relevance field is introduced by **Ioana Alexandru** in her thesis [14] and acts as a filter by year, series, group or subgroup for students. Another use case for this relevance filter in the app is showing the proper timetable to a student based on their current year distribution. These filters can be visualized as a tree data structure, starting

from less specific parent nodes that describe year or faculty to more specific leaf nodes that describe groups or subgroups.

Table 5.2: **news** collection

Field	Type	Description
title	`string`	title of the post
body	`string`	content of a post
category	`string`	source information category of the post
categoryRole	`string`	role used by the author when publishing the post
relevance	`array<string>`	target student group(s) for the post (e.g. 342C3, CTI)
userId	`string`	reference to the author's user document
authorAvatarUrl	`string`	link to the author's avatar
authorDisplayName	`string`	author's name used when displaying the news item
externalLink	`string`	external reference to the original post
createdAt	`timestamp`	creation timestamp of the post

`fcmTokens`

This collection stores tokens used in pushing notifications to devices. Each installed instance of the app on a device generates a unique registration token that helps the *Firebase Cloud Messaging* service send notifications to that specific device. These tokens map devices and not authentication instances. Therefore, we need the `receiveNotifications` field from the **users** collection to check whether a user wants to receive notifications.

Table 5.3: **fcmTokens** collection

Field	Type	Description
token	`string`	value used by Firebase when sending push-notifications

`roleRequests`

This collection stores all role requests made by users and, in addition, metadata about the admin user handling these requests. For example, when a user applies for a specific role, they submit a form request mentioning the role and a reason. Afterwards, an admin accepts or declines their application, and upon acceptance, a confirmation email is sent to the user informing them of their new role.

Table 5.4: **roleRequests** collection

Field	Type	Description
id	`string`	request reference
roleName	`string`	name of the role that the user applied for
userId	`string`	reference to the user document
userEmail	`string`	email used for confirmation upon acceptance
processed	`boolean`	checks whether the request was handled by an admin
processedBy	`string`	reference to the admin's user document
requestBody	`string`	content of the request application
dateSubmitted	`timestamp`	submission date of the request

`roles`

This collection stores the user roles organized by source categories. We followed the implementation guideline from the relevance filter and organized our roles in a tree structure, where the parent nodes are the categories (organizations or representatives), and the leaf nodes are role names. We use a map of maps to achieve a tree-like organization in Firebase. In the end, each parent node in this tree structure is a map, and its children nodes are the elements of that map. Another reason for choosing such a structure for organizing our roles is the existing code infrastructure from the relevance filter that can be reused for our use case.

Table 5.5: **roles** collection

Field	Type	Description
root	`map<map<string>>`	tree representation of the user roles

5.4 Web scraping pipeline

For our scraping process, we implemented a pipeline using *Firebase Cloud Functions* and *Firestore* triggers. We chose three faculty-related platforms for start and implemented a different scraper for each one. We could schedule our scrapers to run at specific intervals in different jobs using cloud functions, and once new entries were inserted into the database, we could trigger push-notifications to be sent to devices. The following diagrams **5.3** and **5.4** describe our implemented workflow.

5.4.1 Firebase Cloud Functions

Cloud Functions provide an environment for writing a serverless, event-driven backend that runs on the Google Cloud servers. This service is suitable for writing functions that should not live on the client-side, and developers do not concern themselves with issues such as hosting, autoscaling or security. Cloud Functions run on a Node[21] runtime environment, and therefore, scripts for this service can be written using either JavaScript or TypeScript. For our development process, we used the Firebase Emulators suit[22] to test our JavaScript functions locally on a Firebase copy before deploying them to the cloud.

5.4.2 Web scrapers

Our web scrapers search the available content on the RSS feeds of our targeted faculty platforms. RSS or "Really Simple Syndication" is a standard technology that enables third-party applications to subscribe to one platform's feed and was an essential component in the emergence of content aggregators on the Internet. These RSS feeds consist primarily of text files that contain posts, articles

[21] https://nodejs.org/en/

[22] https://firebase.google.com/docs/emulator-suite

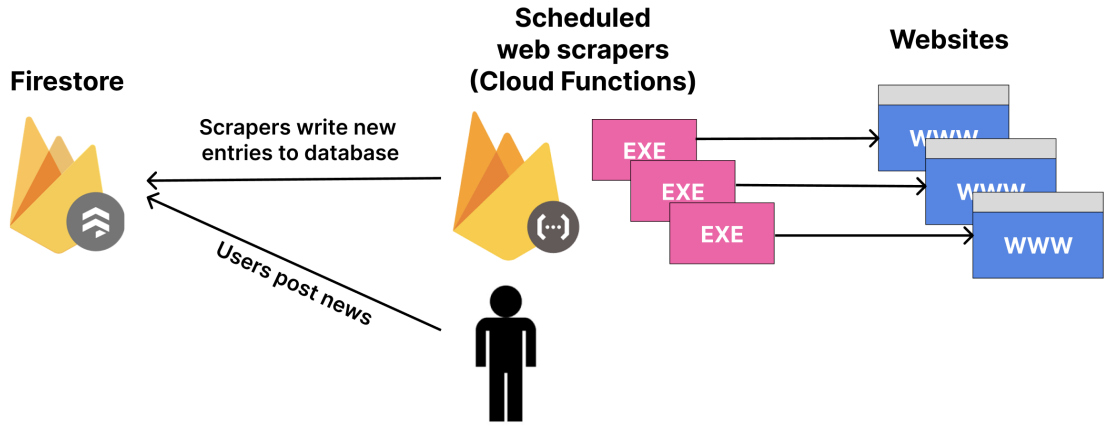

Figure 5.3: Cloud functions pipeline - step 1

Figure 5.4: Cloud functions pipeline - step 2

or updates, and the standard markup language for these files is XML. As a result, we needed to have our scrapers retrieve and parse this format before inserting new entries into the database.

The XML data contains relevant information about a post, such as HTML, author or timestamp. To display the content of a post in our app, we need to parse its HTML from the XML document and then convert the HTML into Markdown syntax. We decided to use Markdown for displaying and publishing posts because it is an increasingly popular markup language that users and programs can easily interpret. According to this Google Developer post[23], Markdown is easier to write by humans, whereas HTML is more expressive in terms of visual elements. Moreover, we have specialized widget components in our Flutter code that can render Markdown on the screen.

Finally, when we started to import posts into our app, we noticed that most of them did not lose any information during the scraping process and preserved their original layout. On the other hand, there were posts that had some original

[23]https://developers.google.com/style/markdown

styling in place and could not be fully translated into Markdown syntax. As a consequence, these posts lost expressiveness or, worse, valuable information that rendered them useless. However, the vast majority of posts did not face any issues, and therefore, we feel pretty confident with the current result. Additionally, since each web scraper runs in a separate function independently from the others, the current architecture can be easily extended with more scrapers in the future.

5.4.3 Firebase Cloud Messaging and Notifications

Due to hardware limitations during the development process, our push-notifications system is tested and currently works only on Android devices. According to this documentation[24], to integrate *Firebase Cloud Messaging* (FCM) with *Apple Push Notification service* (APNs), developers must have an active Apple Developer account and a physical iOS device for receiving messages. For the purpose of this section, we will refer only to Android devices.

Android apps subscribe to notification channels in the background and handle notifications using defined handlers. Our Android app could be in three main states when listening to notifications: **foreground** (the app is active on the screen), **background** (the app is not visible on the screen) and **terminated** (the app process is shut down). For each of these states, we had to define callback functions that treat the incoming messages.

As previously mentioned at **section 5.3.2**, Firebase uses special registration tokens to target messages to devices. Upon creating a new entry in our news collection, the system pushes messages to all available devices, and the logic that decides whether to show a notification resides on the client side.

For the purpose of our news aggregator module, we set up the push notifications system from the ground up. Therefore, other features from the **ACS UPB Mobile** app could benefit from this system in the future since independent notification channels could be easily set up on our implemented code base.

[24]https://firebase.flutter.dev/docs/messaging/apple-integration/

5.5 Deep-linking

Deep links are special URLs that send users directly to the installed app instead of a website. For our purpose, we wanted the users to be able to share external links of news items and have them open directly using our app. Unfortunately, due to hardware limitations, we could not test this functionality on iOS devices.

To create a deep link in Android, we must register a specific URL pattern in our system that can be identified and mapped back to our app. Therefore, whenever the user searches a specific URL, the Android system checks whether that link can be opened by an available app and forwards the action accordingly.

Currently, our functionality is not yet fully complete. While we can open URLs with a specific scheme (e.g. `acs://acs.upb.mobile.dev/news-details?guid=<item-reference>`), our end goal is to define HTTPS-based URLs that can either open the app or send the user straight to Google Play Store to download it. To achieve this functionality, we plan on using the *Firebase Dynamic Links*[25] service in the future once we start releasing new versions of the **ACS UPB Mobile** app in the Google Play Store.

Similarly to the previous push-notification system described at **5.4.3**, there was no existing deep-linking support in the original app. Deep links can trigger various actions within an app and therefore, we believe that our implementation could be reused in the future to support other features.

[25] https://firebase.google.com/docs/dynamic-links

Chapter 6

Evaluation and user feedback

Having implemented our news aggregator module, we wanted to collect feedback from students concerning its user flow and general usability. Thus, we were interested in how students with different levels of familiarity with the app would get around this new module, and we tried to record their feedback and improvement suggestions.

As mentioned in **chapter 5**, some of our features are not fully working on the iOS platform due to hardware limitations during the development process. Consequently, we collected feedback from Android users only so that we could receive fair coverage of our proposed functionalities. We built our app in an APK file for our testing phase and sent it to our test users via online means. Our testers had different levels of familiarity with the app. Some were regular users, some had not opened the app for quite a long time, and some had never used the app before. We gave them an initial description of what features are to be expected when testing, but in the end, we wanted to assess how they get around the news aggregator without much bias.

6.1 Bug discovery

Naturally, our testers found bugs that escaped during the development process. While not critical, some bugs were common among users and could potentially negatively impact the user experience if left unfixed.

For example, when one user had their editing permissions request approved while having the application in the foreground, their app started throwing errors

and shut down. In another instance, when composing a post, one user closed their Wi-Fi connection and started to press the submit button multiple times. These submissions were queued by Firebase locally, and upon network reconnection, all requests were sent to the database. While not necessarily a bug, one tester reported receiving excessively delayed notifications, but we concluded that this issue might have to do more with network connectivity than implementation. However, another tester received irregular notifications that helped us identify a bug in our relevance filtering mechanism. Consequently, that user received notifications about posts they were not supposed to see, as these posts were meant for a different group according to the relevance filter.

Finally, while manual testing helped us identify several bugs and achieve a significant coverage of our features, we plan on employing automated tests for our module in the future. Having automated tests for each new feature is part of the development guidelines promoted by the team behind **ACS UPB Mobile**, and it is a mandatory step when merging into the release branch.

6.2 UI/UX Feedback

Regardless of their previous familiarity with the app, our testers generally got used to the navigation flow fast and had little difficulty understanding how to achieve specific actions. Besides the clear navigation flow, they praised the informative text labels and visual elements that made them aware of what actions could be done at any step during their user journey.

However, there were some observations worth taking into account. For example, one user had trouble with the action buttons because these were too close to one another and caused misactions when pressed. On the other hand, almost all users understood how to apply for editing permissions, but it was not equally clear what the roles meant or how to apply for them. This feedback helped us update the UI and make this part of the user journey more intuitive. While developing, we put great emphasis on designing a workflow that should prevent users from getting lost while navigating the app. Additionally, some users complained about not being familiar with Markdown when composing a post. This issue made us consider allowing the users to choose whether they want to use this markup language or plain text when writing a post in the future.

Chapter 7

Conclusion

Concluding our thesis, we achieved the goal of implementing and testing a news aggregator module for faculty-related news.

7.1 Summary

We started initially by identifying the problem of having multiple heterogeneous information sources in our faculty context. We observed the fact that students at **ACS** need to navigate multiple platforms on a daily basis to stay up-to-date, as their data is not centralized on a single information channel, and we decided to come up with the solution of a news aggregator module that can serve as centralized information channel.

We chose the existing **ACS UPB Mobile** app as the platform basis for building our features, as we concluded this is the best approach to reach a large number of students and leave a significantly positive impact.

Before getting a clear view of what we wanted to achieve, we decided to conduct a study and collect feedback from students to prove that this is an existing issue. Moreover, our goal was to understand what features would be most relevant for us to consider when building our module.

At first, we decided to assess how students from other universities tackle this issue and how they organize their faculty-related information. As a result, these discussions helped us better prepare for collecting direct feedback from students from **ACS**. Finally, we shared an opinion form on multiple student chat groups, and we plotted the collected data to visualize student tendencies.

Upon completing our user study, we started prototyping our functionalities and user interface. We worked in close feedback with the developer team from **ACS UPB Mobile** and started drafting our initial sketches according to their design guidelines. Our final design reflected our proposed functionalities and adhered to the guidelines of the existing app.

We used the Flutter framework to create our required user pages inside the app. Based on the existing platform of the **ACS UPB Mobile** app, we implemented several new user-facing functionalities, and we used the Firebase services to build a serverless backend architecture that supports our whole module. We used scheduled cloud functions to run web scrapers and we built the support for a push-notifications system. Besides aggregating online content from different platforms, our module allows students to publish targeted content for their colleagues using a filtering mechanism. Furthermore, we built the support for different user roles, thus giving different publishing rights in an admin-moderated manner. Finally, we integrated deep links support in the app so that users can share content on other platforms.

Upon completing all of our desired functionalities, we performed a testing session with real users and gathered relevant feedback that helped us better shape our user experience and navigation flow.

In conclusion, we achieved the vertical integration of a news aggregator module over an existing app. As a result, we built support for several new functionalities and created a code base that can be further scaled with new capabilities in the future.

7.2 Future improvements

Based on our collected user feedback and suggestions, we compiled a list of future improvements we plan to bring into our news aggregator module.

7.2.1 Improved scraping

As explained in **section 5.4.2**, we use the available RSS feeds on our target platforms to scrape the needed information. However, although we scrape a satisfactory level of information from our platforms, there are certain visual elements such as images or tables that are mostly skipped during the scraping

process. Thus, we cannot always retrieve the entire content from a post, and the results could weaken the user experience as a consequence.

Currently, we scrape the XML content available in RSS feeds. In the future, we plan on developing more specialized HTML scrapers that could retrieve more elements for our aggregated news items directly from web pages. Each scraper could be specialized on a particular feed, but posts from a single source should follow a standard layout format. Otherwise, it will still prove tedious for a web scraper to reliably get the entire content of a news item on a constant basis.

7.2.2 Lazy-loading lists

Our current news collection is relatively small in terms of item count, but the more sources we scrape and the more time it passes, the bigger it will grow. Bigger lists will take a heavy toll on memory performance on user devices and will deter the user experience without a lazy-loading strategy.

We plan on implementing a paging mechanism that should retrieve posts in chunks rather than all at once, as it is now the case. Besides the clear memory improvements, this strategy should yield the visual effect of an infinite scrollable feed[1], which is a design practice standard on popular social platforms. In the case of our news feed, we do not necessarily expect users to scroll through content for long periods of time, but we believe it will help our feed feel more in tune with other popular platforms that employ this practice.

7.2.3 More granular sources

We currently enable users to select between three types of information sources. For the future, we consider making them more granular to give users more control over their chosen sources. For example, when selecting student organizations, users can subscribe to some organizations and unsubscribe from others. However, we are not totally convinced whether this granular control will be needed for our case. For the moment, we plan on aggregating more sources and encouraging more students and entities to start using our feed. We consider that a necessary condition for having more detailed information sources is having large volumes

[1] https://builtin.com/ux-design/infinite-scroll

of information in the first place. Therefore, we feel that the current level of content curation is enough for the present being.

Bibliography

[1] P. Hemp, "Death by information overload," Sep. 2009, date accessed: 20.05.2022. [Online]. Available: https://hbr.org/2009/09/death-by-information-overload

[2] O. Tunikova, "Are we consuming too much information?" *Medium*, Jun. 2018, date accessed: 20.05.2022. [Online]. Available: https://medium.com/@tunikova_k/are-we-consuming-too-much-information-b68f62500089

[3] O'Reilly, "Web 2.0 expo ny: Clay shirky (shirky.com) it's not information overload. it's filter failure," Sep. 2008, date accessed: 18.05.2022. [Online]. Available: https://www.youtube.com/watch?v=LabqeJEOQyI

[4] I. Alexandru, "Design and implementation of a cross-platform mobile application that facilitates student collaboration," July 2020. [Online]. Available: https://github.com/student-hub/paper/blob/master/Design_and_implementation_of_a_cross_platform_mobile_application_that_facilitates_student_collaboration.pdf

[5] S. Sharma, "What is flutter? its benefits and limitations," *Medium*, Jul. 2020, date accessed: 03.06.2022. [Online]. Available: https://medium.flutterdevs.com/what-is-flutter-its-benefits-and-limitations-c795c94dfb16

[6] A. H. Ataș and B. Çelik, "Smartphone use of university students:patterns, purposes, and situations," *Malaysian Online Journal of Educational Technology*, 2019. [Online]. Available: https://eric.ed.gov/?id=EJ1214011

[7] I. A. Ștefan, A. Ștefan, I. R. Goldbach, and F. G. Hamza-Lup, "Exploring the use of gamified systems in training and work environments," *The 15th International Scientific Conference eLearning and Software for Education Bucharest, April 11-12, 2019*, Apr. 2019. [Online].

Available: https://www.researchgate.net/publication/331197384_Exploring_the_use_of_gamified_systems_in_training_and_work_environments

[8] Most popular apple app store categories as of 1st quarter 2022, by share of available apps. Date accessed: 09.06.2022. [Online]. Available: https://www.statista.com/statistics/270291/popular-categories-in-the-app-store/

[9] Most popular google play app categories as of 1st quarter 2022, by share of available apps. Date accessed: 09.06.2022. [Online]. Available: https://www.statista.com/statistics/279286/google-play-android-app-categories/

[10] G. Developers, "Google i/o 2014 - material design principles," Jun. 2014, date accessed: 09.06.2022. [Online]. Available: https://www.youtube.com/watch?v=isYZXwaP3Q4

[11] I. Alexandru, "Design and implementation of a cross-platform mobile application that facilitates student collaboration," pp. 36–37, July 2020. [Online]. Available: https://github.com/student-hub/paper/blob/master/Design_and_implementation_of_a_cross_platform_mobile_application_that_facilitates_student_collaboration.pdf

[12] ——, "Design and implementation of a cross-platform mobile application that facilitates student collaboration," pp. 46–47, July 2020. [Online]. Available: https://github.com/student-hub/paper/blob/master/Design_and_implementation_of_a_cross_platform_mobile_application_that_facilitates_student_collaboration.pdf

[13] P. Sadowska, "Compilesdkversion and targetsdkversion — what is the difference?" *Medium*, Jun. 2021, date accessed: 20.06.2022. [Online]. Available: https://proandroiddev.com/compilesdkversion-and-targetsdkversion-what-is-the-difference-b4227c663ba8

[14] I. Alexandru, "Design and implementation of a cross-platform mobile application that facilitates student collaboration," p. 51, July 2020. [Online]. Available: https://github.com/student-hub/paper/blob/master/Design_and_implementation_of_a_cross_platform_mobile_application_that_facilitates_student_collaboration.pdf