MyIPB Mobile - REST services aggregator for mobile application centered on student

Eduardo Barbosa de Oliveira - a46679

Thesis presented to the School of Technology and Management in the scope of the Master in Information Systemsunder the double degree with the Federal Technological University of Paraná.

Supervisors:

Professor Doutor Paulo Alexandre Vara Alves

Professor Doutor André Luis Schwerz

Professor Doutor José Eduardo Fernandes

Bragança

2020-2021

MyIPB Mobile - REST services aggregator for mobile application centered on student

Eduardo Barbosa de Oliveira - a46679

Thesis presented to the School of Technology and Management in the scope of the Master in Information Systemsunder the double degree with the Federal Technological University of Paraná.

Supervisors:

Professor Doutor Paulo Alexandre Vara Alves

Professor Doutor André Luis Schwerz

Professor Doutor José Eduardo Fernandes

Bragança

2020-2021

Abstract

Recently COVID-19 emerged and was soon declared as a pandemic due to the high rate of infection. As a result of the public policies arising from COVID-19, millions of people have their daily lives changed. In the case of students, the learning process was impacted. To address this problem, information technology comes into action with solutions such as electronic and mobile learning. In this context, there are several Learning Management System (LMS) that can assist in the learning process, such as Moodle, Blackboard, and Sakai. However, not all LMS have a mobile application to assist students and teachers in accessing their content, thus losing a powerful mechanism to support the learning process. This work proposes three architectures to aggregate and provides a mobile application to LMSs, one that uses the Serverless model and two that uses the Self-hosted model. Each one is composed of a backend and a mobile application. As a result, a performance test is executed on the Serverless architecture to verify the best way to implement the features in terms of performance.
Keywords: Learning management system. Mobile learning. Serverless. Self-hosted.

Contents

1 Introduction **1**

2 State of the art **5**

2.1 Learning management systems 5

2.2 Service-oriented architecture 7

2.3 Mobile learning 8

2.3.1 Mobile development 9

2.3.2 Server-side 12

3 Architecture **15**

3.1 Technologies decisions 15

3.1.1 Mobile 15

3.1.2 Server-side 16

3.2 Architectures proposed 18

3.2.1 Serverless 20

3.2.2 Self hosting 21

3.2.3 Self hosting with local database 23

3.2.4 Discussion 25

4 Implementation **27**

4.1 System description 28

4.1.1 System requirements 29

4.1.2 Mockups 32
4.2 Project management 35
4.3 Code management 36
4.4 Credentials management 36
4.5 Architecture implementation 37
4.5.1 Serverless architecture 38
4.5.2 Self-hosted architecture 42
4.5.3 Self-hosted with local database architecture 43
4.6 Performance test 44
4.6.1 Discussion 47

5 Conclusions **49**

Appendices **51**

A Mockups **52**

B Screenshots **55**

List of Figures

3.1 Serverless architecture diagram. 21
3.2 Self hosting architecture diagram. 22
3.3 Self hosting with local database architecture diagram. 24

4.1 Use case diagram of the mobile application MyIPB on the IPBvirtual part. 31
4.2 Mockup of the login screen. 32
4.3 Mockup of the application sidebar. 33
4.4 Mockup of the announcements screen. 33
4.5 Mockup of the assignments screen. 34
4.6 Mockup of the forums screen. 34
4.7 Mockup of the resources screen. 35
4.8 Implementation of the login screen. 39
4.9 Implementation of the Announcements screen. 39
4.10 Implementation of the Assignments screen. 40
4.11 Implementation of the Forums screen. 41
4.12 Implementation of the Resources screen. 41
4.13 Profiling test of the serverless mobile application. 46

A.1 Mockup of the login screen with another LMS. 52
A.2 Mockup of the announcement details screen. 53
A.3 Mockup of the topic screen. 53
A.4 Mockup of the conversation screen. 54
A.5 Mockup of the resources screens within a folder. 54

B.1 Implementation of the application sidebar. 55
B.2 Implementation of the Announcements screen with a filter applied. 56
B.3 Implementation of the Announcements details screen. 56
B.4 Implementation of the Conversation screen. 57
B.5 Implementation of the Replies screen. 57
B.6 Implementation of the resources within a folder. 58

Chapter 1

Introduction

The COVID-19 emerged in the year 2019 and was soon declared as a pandemic due to the high rate of infection. As a result, governments around the world have declared public policies that include social distancing, isolation, and quarantines. Herewith, millions of people are on lockdown to prevent the spread of the virus and have had their daily lives drastically changed. In the case of students, the major challenge is to adapt their study routine to this new situation. The Information Technology emerges in this context to assist the students in this adaption, proposing solutions such as the LMS [1]. LMS can be defined as web-based technology developed for improving through the application and evaluation of educational institutions [2].

With this need to adapt to this new situation of online learning, the use of LMSs has increased. Herewith, several LMS software become more popular, e.g., Moodle [3], Sakai Project [4], WebCT(Blackboard) [5], etc. However, not every LMS has a mobile application that allows the students to follow the learning in a fast and practical way. Smartphone usage has been growing nowadays [6] and gathering the use of the smartphone and the learning process, could improve the learning rate and make it easier to visualize the LMS content.

Taking these aspects into consideration, the primary objective of this work is to propose the most suitable architecture to support the aggregation of several LMSs. This architecture must provide support for showing the content (e.g., assignments, forums,

etc) of different LMSs. Also, this architecture must be adaptable to two different models, the serverless and self-hosted model. Thus, the secondary objectives are:

- Check if the proposed architecture works on different models, serverless and self-hosted.
- Make a performance analysis comparing the different feature implementations.

To design an architecture that satisfies each requirement, several adversities may be found. It is necessary to determine how the data will be stored in such a way that this data can represent different models and entities. In each LMS, the entities can be named differently, i.e., an assignment can be named as a task or activity, depending on the LMS. Creating a data model that can represent the entity without information losses is essential. Also, define a policy synchronization between the aggregator and aggregated system is a big challenge. It is necessary to choose a policy synchronization that does not overload the aggregated system and keeps the data always synced on the aggregator system. Consume the Web Services of different LMS is another big challenge. The Web Service may not return all necessary information to create the data on the aggregator system, or yet, may not return the data in an efficient way, e.g., return only one register by request or demand more than one request to get the data about a determined entity.

Three architectures are proposed to implement the aggregator system. They all have a mobile application developed in Kotlin Android and a backend. The Serverless architecture has a backend based on the Serverless model using the Firebase tools. The Self-hosted architecture has a backend based on the self-hosted model and uses NodeJS, MongoDB, and GraphQL. The Self-hosted with local database follows the same behavior as the Self-hosted architecture, but with a local database in the mobile application to improve the performance and user experience.

With the conclusion of this work, the implementation of the three architectures was made. Each has a different level of implementation, i.e., one architecture may have more features implemented than the other. The Serverless architecture has all the proposed features implemented. The other two architectures that were made to make a proof of

concept of the main architecture have only one feature (excluding the Login) implemented. Besides, performance results are obtained through a performance test done on the Serverless architecture. On this performance test, results about the different ways to implement the features were obtained and with that, it was possible to conclude which is the best and the worst way to implement the features.

This work is divided into five chapters. Chapter 2 makes a study across the state of the art of the topics related to this work. Chapter 3 describes the technological decisions made and each one of the three proposed architectures. Chapter 4 describes the system, the process for implementing these three architectures, and the results of the implementation and performance testing. Finally, Chapter 5 shows the conclusions made at the end of this work and proposes future works.

Chapter 2

State of the art

This chapter discusses the state of the art regarding the learning management systems, service-oriented architecture, and mobile learning.

2.1 Learning management systems

Learning management systems (LMS) are computer systems used to manage and assist the online learning process. From a technical point of view, an LMS is server-based software that makes communication with a database that contains the information about the users, courses, and contents [7, Chapter 1.3]. Additionally, LMSs enables teachers to manage and distribute this content in an easy and integrated way [8]. An advanced e-learning system must meet the following requirements [9]:

- Compatibility to work with another LMS
- Content management abilities, such as file management
- Creation and management of learning resources as a ”learning object”
- Reusability of the content (content compatibility such as SCORM [10])
- Efficient content creation, distribution, management, integration and authorization tools

- Support to other complementary tools like PowerPoint, Dreamweaver
- Performance and the possibility to extend the environment
- Multi-Language support

Regarding these aspects, a study is made comparing Moodle with other open-source tools [11] . In this study, the Moodle was compared among other LMSs and it was concluded that the Moodle presents three main advantages over the others:

- Its modular design allows more flexibility supporting any teaching style
- Its interface has a rate of usability superior compared to other competitors
- A wider range of authentication, ease of installation, and maintenance increases the frequency of use

The Moodle is often compared to other tools. On [12] a comparison between 36 LMS framework was made to compare the features implementation in each one. This study concluded that all LMS have similar features, all support multimedia, most meet the SCORM standard and less than half have chat support. On [8] a study on predictions is made taking into account data from 17 LMS courses that use Moodle. The objective of the study is to determine the prediction model portability across different courses.

An LMS that is usually compared to Moodle is Blackboard. On [13] a usability comparison between the Moodle and Blackboard is made. The objective of this study is to compare the usability and effectiveness of the two competing LMSs. The study concluded that Moodle is the preferred LMS by the student users and has better usability as well.

Another LMS that is usually present in these comparisons is the Sakai. On [14], a performance comparison between .LRN, Sakai, and Moodle is made. The study aims to compare its performance using several tools for this. The study concluded that the Sakai and .LRN is the best tools from a system perspective. From the point of view of the system administrator, Sakai and Moodle have a similar rating and .LRN has a slightly

lower rating. As a conclusion, if a large number of users are expected, the most suitable LMS is the .LRN or Sakai, if ease of use and large community are expected, the best option would be Sakai or Moodle.

LMS can have different types of architectures. One of them is the Service-Oriented Architecture (SOA). The next section will describe this type of architecture.

2.2 Service-oriented architecture

The software complexity has been increasing with the advance of the history of computing. Several approaches have been proposed to deal with this complexity at different levels, e.g., "structured programming" [15] and Fred Brooks' idea of "conceptual integrity" [16]. The SOA comes into this context as an architecture that modularizes services [17].

SOA can be considered a descendant of the logical evolution of the software modularization techniques that started 50 years ago with structured programming. One of SOA's improvements is that it allows more flexibility in choosing the implementation of technologies and location for the service providers and consumers. The most important aspect of SOA is that it splits the service implementation and the interface, i.e., it separates the "what" from "how". End consumers can visualize just an endpoint that supports a certain request format or contract. They will have no concerns about how the implementation was made, only expected the result [17].

Nowadays, a hot topic regarding SOA is the relation between SOA and Cloud computing. Cloud computing is an emerging topic that enables the rapid delivery of computing resources in such a way that can be scaled dynamically and virtually. Some advantages of cloud computing over the traditional approach include agility, lower entry cost, device independence, location independence, and scalability [18].

SOA and cloud computing are related. SOA is an architectural pattern that guides companies to create solutions that can be reused for their computing resources. Cloud computing is a set of technologies that allows enterprises to build their SOA solution more flexibly. In other words, SOA and Cloud Computing can coexist, support, and

complement each other [19].

On [20], a survey is made to present the latest developments in service-oriented network virtualization to support Cloud computing. In this survey, it was concluded that network virtualization is the main factor to make the convergence of networking and Cloud computing. Also, it was concluded that SOA has been widely used as a mechanism for realizing network virtualization, being a key factor in enabling both.

With the emergence of ubiquitous computing, one of the biggest consumers of SOA is the mobile application. The next section will describe the state of the art of mobile learning.

2.3 Mobile learning

Nowadays, the rapid evolution of information technologies is supplying new ways of learning and creative ways to address the limitations of traditional learning [21]. With the increased availability of mobile Internet and the rapid growth of mobile devices usage, mobile learning is becoming a trend to the future of education and learning [22].

Originated from distance education, mobile learning (m-learning) is a development coming from e-learning [23]. The m-learning can be considered an extension of the e-learning bringing an additional of anywhere and anytime dimensions of learning. One of the advantages of m-learning is to offer learners the possibility to access relevant information with a reduced cognitive load and increased access to other systems and people [24].

Most of the m-learning system is provided to learners via LMS. LMSs such as Blackboard and Moodle provides scaffolding and support to the teachers using the Web 2.0 tools [25]. Nowadays, Moodle already has a mobile app incorporated with the system that can be used on mobile devices[1]. The app is made to assist teachers and students with basic support in education [26].

A wide variety of studies have been made to propose architectures to m-learning.

[1] https://moodle.com/app/

On [26] a study is made to provide a reference to develop and design LMS mobile application for students. Also, a study case of a mobile application developed for Moodle was presented. In this study, it was concluded that there was an increase in the number of students shifting from using the mobile device browser to use the mobile application.

On [27] a study is made to propose a software architecture that can support the adaptation of LMS features and mechanisms to the mobile scenario. The study takes into account the SCORM and shows a study case of a mobile application developed to the Moodle. The study concluded that the improvement into the interoperability related to e-learning systems is necessary and will help the teachers and students to access the LMS contents through mobile devices.

Regarding the m-learning, mobile development is necessary to make the mobile application that will support the teachers and students. Taking it into consideration, it is necessary to understand what options are available to make this development most appropriately. The next subsection will describe what technologies regarding mobile development and server-side technologies can be used to achieve this. The Subsection 2.3.1 will discuss mobile development and its technologies and Section 2.3.2 will discuss the server-side development and its technologies.

2.3.1 Mobile development

With the technological advances, ubiquitous computing has become increasingly present in everyday life. In conjunction with this, services are becoming portable and the use of mobile devices to access these services has been widely adopted. Consequently, there is an increase in demand for portable services.

From this demand, several frameworks for mobile development have emerged (e.g., *React-Native*, native development on *Android* with *Kotlin* and on *iOS* with *Swift*). Each of them has its specificities, advantages, disadvantages and will be better suited to a specific situation. For example, in a situation where all of your users have an *Android* device, developing a hybrid *framework* (for example, *Flutter*, *Ionic*) may not be the best

option, as this type of framework is focused at developing applications for two platforms.

From this, a comparison of the advantages of hybrid and native development is made. The purpose of this comparison is to understand the advantages of each of the development model, and then observing this, define a model which best fits the objective of this work. Furthermore, a brief description of the web application development model is given as well.

Mobile Web app development

Mobile Web applications (Web apps) is an approach to developing mobile applications. It consists of using a browser within its runtime environment. Using this approach, websites optimized for mobile devices will be interpreted and rendered within the application. This optimization needs to take into consideration several screen sizes of mobile devices [28].

However, mobile Web apps cannot access natives functionalities of the device [28]. Thus, it is not able to implement push notifications.

Native mobile development

The native application development model consists of developing applications that are specifically developed for a specific operating system. To create truly native applications, you need to use the *Kotlin* or *Java* programming languages for developing applications for the Google *Android* operational system and for developing applications for the operational system *iOS* it is necessary to use the languages *Objective-C* or *Swift* [29].

There are several advantages to using the native application development model, some of them are [30]:

- By running at the system level, native applications have more advantages for functionalities on the device (e.g., camera, GPS, fingerprint reader). In addition to these advantages, it is possible to access system notifications in this way, implementing notifications more functionally;
- Applications offer a better experience and interaction for the user;

- Applications have a better performance.

In a nutshell, the native application development model will produce applications with higher performance, and improved User Experience (UX) and access to device functionality more effectively. In contrast, the model has problems with the speed of development, the learning curve, and the distribution to multiple platforms [31].

Hybrid mobile development

In recent years, the hybrid application development model has been growing and is showing a promising alternative to the native application development model. With hybrid development, it is possible to create a single mobile application using web development standards and distribute it across multiples mobile platforms with small or no code changes [32].

There are several advantages to using the hybrid application development model [33]:

- Developers can use standard web technologies like *HTML5*, *CSS3* and *JavaScript* to develop;
- It has active wrappers that allow the application to be packaged, deployed and distributed between platforms;
- It has a smaller learning curve to start building a multi-platform mobile application using common web languages (e.g., *HTML5*, *CSS* and *JavaScript)*;
- Maximizes the concept of *"Write once, run anywhere"* (WORA) [34], which means that the developer can develop the application as a common browser application with responsive features provided by the hybrid development frameworks.

In summary, the hybrid application development model will produce multi-platform applications faster due to the low learning curve. In contrast, the model suffers from problems such as performance, functionalities that involve the device resources and does not provide the best UX for the user [31].

Discussion

From the analysis and study done on the models of development of native and hybrid applications, it was possible to conclude that the hybrid model is easier to be produced, maintained, and distributed but it is necessary to pay a price in performance and UX, besides having access the features of the device less effective than the native. On the other hand, native applications provide UX, performance and access to the device's functionalities effectively and functionally, but suffers in issues to distribute to multiple platforms and the language learning curve.

2.3.2 Server-side

For the development of the mobile application server-side, there is two main architectures model that can be followed and these two options are described below.

Self-hosting

The Self-hosting model, also known as the on-premise model, is an architecture model where the hardware and the software are maintained by the owner, i.e., the organization is responsible for all the infrastructure process, such as buying the hardware, install, maintain, and running the hardware and software.

These aspects imply a series of factors. In this model, the cost is a key aspect. The owner must purchase a license and is also responsible for installing, maintaining, and updating the system. Furthermore, the owner must sign maintenance and service contracts [35].

Moreover, this model has problems to scale. When a scale is required (when the demand is too high, for example), the owner must be responsible to make this scale, i.e., the owner must be responsible to maintain each server instance, requiring an IT know-how and staff to keep the hardware and software up to date and make the required change for the scale [35].

Another problem of this model is the data reliability. In IT systems, backups need to

be made. In this model, the owner is responsible to make the entire backup process, i.e. the IT team is who defines how reliable and how many times this process occurs [35].

The flexibility of this model is a key aspect. Nowadays, IT systems must be flexible to go along with the business changes. Companies with volatile business such as seasonal or project-oriented can have high peaks of demands according to the month of the year. In this model, even this peak occurs only a few times a year, the infrastructure has to operate at peak-load capacity at all time, even if this capacity is not used most of the time [35].

Security is a hot topic on this model. In this model, all data is stored on local servers and is not exposed to the Internet. As a result, the company is less susceptible to attack over the Internet. However, most data thefts occur in-house by employees [35].

This model is widely used nowadays by several organizations, however, it presents several problems and expensive costs. In contrast with this model, the following section describes the Serverless model.

Serverless

Cloud computing has been proliferating since its emergence in the context of application deployments. A wide variety of providers is offering services like that in different levels, such as Amazon Web Services (AWS), Google Cloud Platform (GCP), Microsoft Azure, IBM Cloud, and others [36].

In this context, the term "serverless computing" is gaining momentum to represent the peak of the cloud-native model: a "serverless" cloud application is deployed to infrastructure parts that are completely transparent to the application developer. The fundamental principle of the serverless concept is not to require operations (also referred to as "NoOps") to maintain a serverless application [36].

The most well-known implementation of the serverless model is Function-as-a-Service (FaaS) offerings, such as AWS Lambda, Azure Functions, and Google Cloud Functions. With the FaaS, application developers provide the source code to be executed by a trigger. The source code is usually short-running and atomic functions and the triggers are usually

defined by an HTTP request. For example, a common use case of these functions is to define a unitary function that is triggered by an HTTP request. So, the FaaS provider executes and bills these functions on isolated instances and can scale this horizontally as needed on demand. Ideally, FaaS will: provide simplified deployments; reduce the operation efforts; provide a pricing plan pay-as-you-go. Furthermore, the providers can support a huge functions call with few resources [36].

One of the open issues on the FaaS offerings is to build larger applications. Currently, the FaaS tends to be used on microservices architecture model with not so many lines of codes [36].

Discussion

As seen above, the two models have their advantages and disadvantages. The serverless model can reduce the DevOps efforts, scale horizontally and make the process of deployment simpler. On the other hand, the model has problems when the application logic grows, demanding more lines of code. The self-hosting model can grow in lines of code without problems, but the process of deployments is more complex to be done. Also, this model demands the system maintainer to design an architecture that can scale horizontally; manage the load balance; make backups of the data, etc.

Chapter 3

Architecture

This chapter describes the decisions made to acquire the system architectures and how they were designed, i.e., perform a discussion around the possible technologies and which to use, describes which frameworks are used, how these frameworks are integrated, and how all these functionalities and tools work together to make the system work properly. As a result, three architectures are proposed.

3.1 Technologies decisions

In this Section, a discussion is made around the possible technologies and which one is the most suitable to use on this project. The Subsection 3.1.1 will guide the architecture decision through the mobile context. The Subsection 3.1.2 will guide the architecture decision regarding the server-side model to be followed.

3.1.1 Mobile

Taking into account the information provided on the subsection 2.3.1, the following conclusions can be made across the mobile development implementation of this project.

One of the main features of this system is the possibility of send push notifications. Therefore, looking at this perspective, the native approach is the best option due to

the best implementation of push notifications. Moreover, the learning curve, one of the disadvantages of the native approach, is not a problem for those who will develop the app due the previous development experience in this approach. With the native approach, we can develop a mobile app with an optimum UX and performance. However, it will not be possible to distribute to multiple platforms. This is the price to be paid. Even with this disadvantage, the trade-off to choose the native approach is better than the trade-off of the hybrid approach.

Looking into native development, it is necessary to define on which platform the development will be made, on *iOS* or *Android*. According to the statistical website Stat Counter[1], the operational system *Android* is used by more than 70% of the mobile devices in the world. Looking at this data, the native approach to *Android* is the option that reaches most users.

In the context of *Android* native mobile development, it is possible to use two programming languages: *Java* or *Kotlin*. Kotlin is the official language for *Android* supported by *Google*. It is a strong indicator that is the best language option to develop for *Android* nowadays. In addition, *Kotlin* has compatibility with several *Java* features, making it more complete and dynamic.

However, the mobile application alone is not enough to create a LMS mobile application. A server-side implementation is also necessary. The next Subsection will describe the decision process to the server-side with details.

3.1.2 Server-side

Taking into account the information provided on the subsection 2.3.2 made on Chapter 2, the following conclusions can be made around the server-side implementation of this project.

Creating a server-side for the mobile application is a key factor to achieve the best architecture for LMS mobile applications. Without a server-side structure, all the code

[1]https://gs.statcounter.com/os-market-share/mobile/worldwide

and business logic will be inside the mobile application. That is not good, considering that the mobile application is supposed to be only an interface, without huge business rules.

With these rules within the mobile application, if it is desired to make some changes to these rules, it is needed to launch a new version. Not all the users will update the mobile application recurrently, i.e., the application can not be expected to always be in the latest version and this is a problem. For example, in a case that the mobile application consumes a service API, and this API launch a new version that changes an endpoint response (e.g., changing a field name). With that change, the mobile application may not work properly. So, a new version is needed to be launched and all the users will have to update the application or it will not work properly.

On the other hand, if these rules and service API consumption are kept on a server-side structure, when the above example occurs, just deploy on the server side is needed, and all users will have the mobile application working properly, without the need to update it. If that change on response API is scheduled, is possible to update the server-side before the change occurs and this change will be transparent for the user, i.e., the user will not notice that something changed, unlike the other solution that requires the user to update the application.

Taking into account the above observations, everything indicates that have a server-side structure is the best solution. For this structure, two models are proposed, the self-hosting and the serverless.

Taking the aspects shown in subsection 2.3.2, the serverless model is the most suitable for this application. The application is not huge and it is not worth increasing the complexity of the system with DevOps matters. Choosing the serverless model is only necessary to develop the code responsible to implement the application logic and the deployment process can be done with few commands.

Nevertheless, the place where the system will be hosted is not defined yet and the self-hosted model is adopted by several companies. Herewith, the system must be suitable for both, serverless model and the self-hosted model.

The following section will show three architectures that were obtained taking into account these conclusions made.

3.2 Architectures proposed

The first architecture consists in use the serverless model on the server-side. In this architecture, the GCP and Firebase tools are used to build the entire server-side structure. This is described in more detail on 3.2.1.

The Self-hosted architecture is using the self-hosting model on the server-side. In this architecture, an API is built to provide the data to the mobile application using NodeJS as the Web API framework and MongoDB as the database storage. This is described in more detail on 3.2.2.

The Self-hosted architecture with local database architecture follows the same behavior as the Self-hosted architecture, but in this architecture, a database will be used on the mobile application. Therefore, this architecture is an uplift in the Self-hosted architecture and makes improvements to the user experience in the mobile application. This is described in more detail on 3.2.3.

All three architectures will have the following entities in common:

- **Database**: it is the entity responsible to store all the data.
- **Firebase Cloud Messaging (FCM)**[2]: it is cross-platform messaging solution who lets the developer to send messages at no costs. With this solution, it is possible to send push notifications whenever the developer wants.
- **Login**: it is a single function responsible to make the user's login. For this, the function will make an HTTP request to the provider to make the authentication. The provider that will be requested is defined by the parameters sent by the mobile application. If the provider's response is successful, the function will make another request to get the user's subjects. After this, the function will create a register in

[2]https://firebase.google.com/docs/cloud-messaging

the Database to store the user's subject, authentication token, and the FCM token sent by the mobile application.

- **Database Triggers**: it is a group of functions responsible for implementing the database triggers. The create and update triggers of collections of interest (e.g., announcements, assignments) are implemented to send the push notifications. Therefore, at every creation or update on the collections of interest, the trigger function will be called and send the push notifications. For this, the function must check which subject the created/updated record is related to and get all users who are related to the same subject. After getting all users, the function will send the push notification via the FCM token with the appropriate content.

- **Synchronizers**: it is a group of functions responsible for implementing the synchronization between the provider and Firestore. With an admin user (i.e., a user who has access to the entire system), this function will make an HTTP request to the providers and get all records of all types, i.e., all assignments of all subjects, all announcements of all subjects, etc. After getting the records, the function will create the record if it does not exist, update if exists and has a difference between the records, or do nothing if it already exists and nothing has changed between the records. Furthermore, each collection of interest has the own synchronizer.

- **Google Cloud Scheduler**[3]: this is the entity responsible for triggering the synchronizers. For this, a job is scheduled using the cron syntax for each synchronizer. This job will make a simple request HTTP and trigger the synchronizer. Each job has the own cron time. So, for collections that have constant updates, the interval between the execution of the jobs will be shorter than for collections that have sporadic updates. For instance, the announcements synchronizer will be triggered more than the resources synchronizer.

Therefore, the entire application flow can be summed up by:

[3]https://cloud.google.com/scheduler

1. User makes login;

2. System make an HTTP request on the provider to make the login and store in the Database the user's tokens and subjects;

3. At some moment, the jobs of Google Cloud Scheduler will be executed and will create/update records in the Database;

4. With the create/update in Database, the Database Triggers will be executed and will send the push notifications to the users via FCM.

The following describes how the three architectures implement all entities and using which tools and concepts.

3.2.1 Serverless

This Serverless architecture is built using the serverless model described in 2.3.2. For this, the Firebase tools are used.

All functionalities from Firebase tools used (except the FCM that was described previously) are:

- **Google Functions**[4]: it is a serverless framework that permits the developer to write backend code to be executed by triggers in HTTP requests and Firebase Features. The JavaScript or TypeScript code is stored on Google's cloud and the developer does not need to manage and scale the servers.

- **Firestore**[5]: it is a scalable NoSQL cloud database used in client-side and server-side. It keeps data between the server-side and client-side synchronized through realtime listeners and offers offline support as well.

These are all the functionalities that are used in this architecture. The Figure 3.1 describes how these functionalities are grouped and how they are related.

[4]https://firebase.google.com/products/functions
[5]https://firebase.google.com/docs/firestore

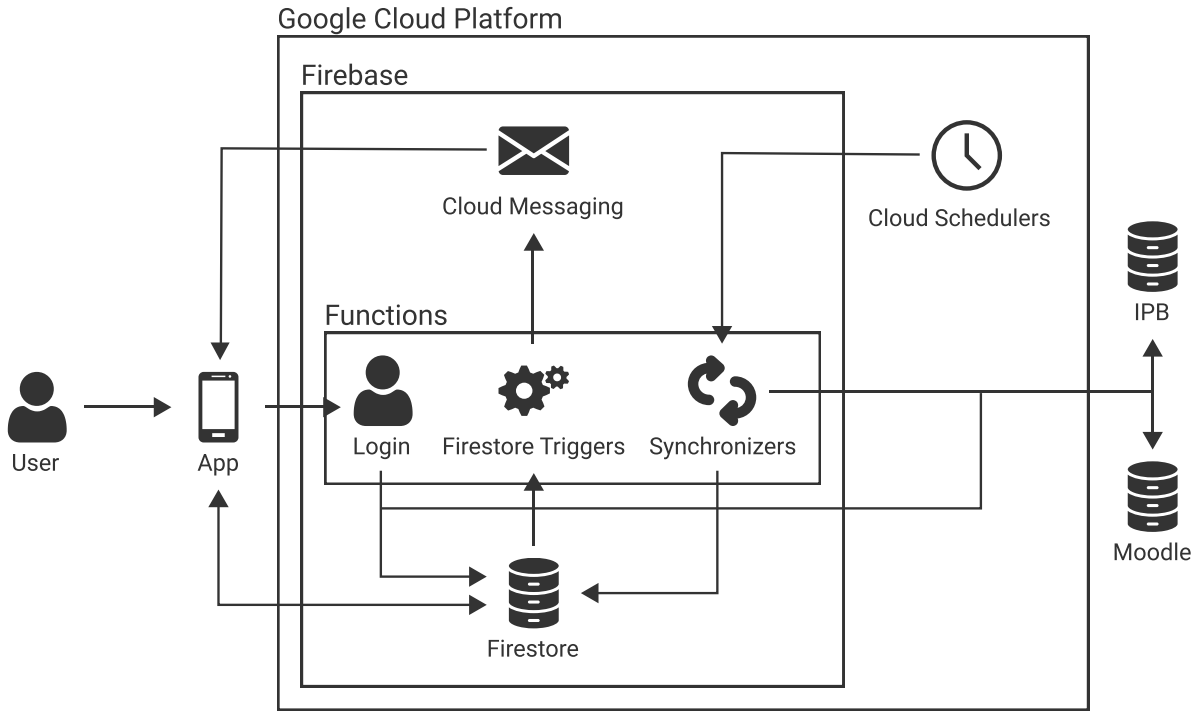

Figure 3.1: Serverless architecture diagram.

Looking at the image is possible to see that the Firebase is responsible for implementing the Database entity and the Google Functions is responsible for implementing the entities Login, Database Triggers, and Synchronizers.

Also, it is important to mention that the mobile application access the Firestore directly and not via API. With this, in the development, the calls for access to the storage is transparent, the developer only needs to call the method, and all logic to get the data is kept by the Firestore.

This is the Serverless architecture proposed using the serverless model. The following section describes the Self-hosted architecture that uses another model.

3.2.2 Self hosting

This architecture is built using the self-hosted model described in Subsection 2.3.2. This architecture was designed to aim at the several companies that use the self-hosted model and do not use cloud services.

The diagram illustrated in Figure 3.2 shows how this architecture was designed.

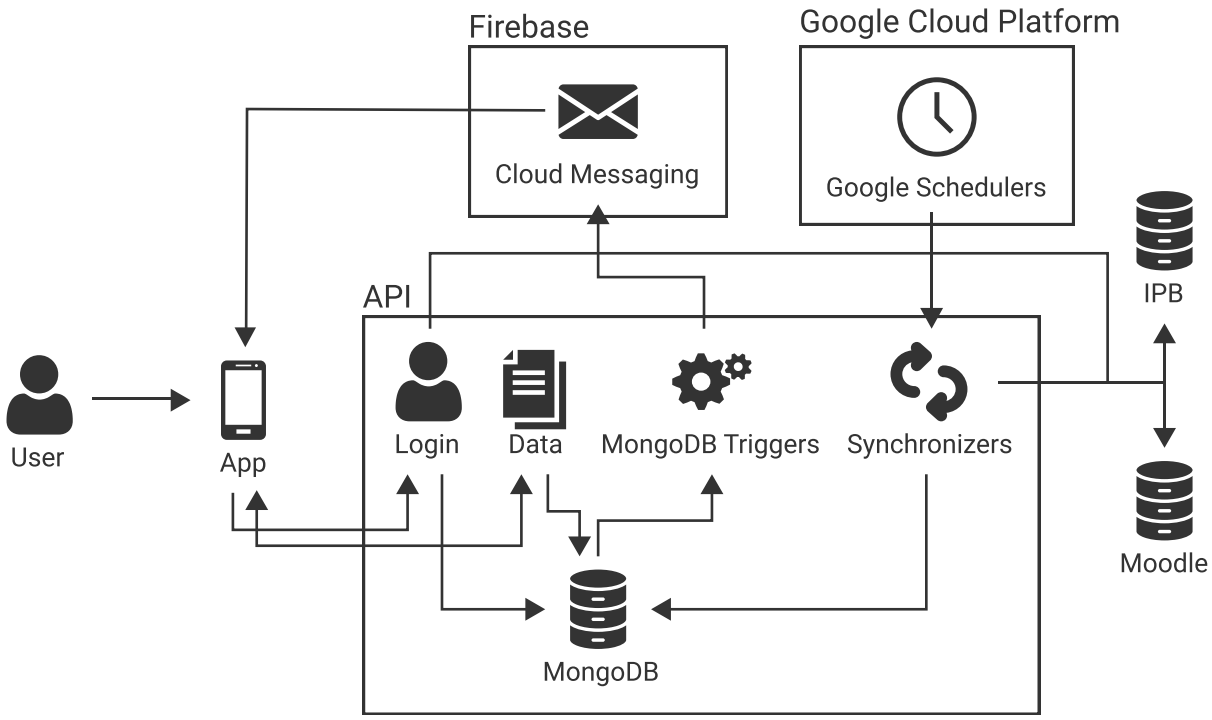

Figure 3.2: Self hosting architecture diagram.

Looking at the image it is possible to notice that on this architecture the MongoDB is responsible to implement the entity Database and the API is responsible to implement the entities Login, Synchronizers, and Database Triggers. As a very well know and NoSQL database, MongoDB was chosen, maintaining the same data structure of the Serverless Model. Also, the deployment and initialization of the MongoDB are fast and simple to be done.

Furthermore, an important point to note is that this architecture uses Cloud solutions only at two points, on Firebase Cloud Messaging and Google Schedulers. These two features were implemented using the Cloud to reduce the implementation time. These features could be implemented without using Cloud solutions, but it would not be feasible. All other architecture entities do not use Cloud solutions.

The entity Data is an addition to this architecture. This entity is responsible to provide all data for the mobile application. This entity did not exist in the serverless architecture

because the Firestore transparently provides the data. Besides, in this architecture, it is necessary to create the endpoint that will provide the data for the mobile application.

As seen in the image, the mobile application does not have a local database, this implies that the app always has to access the API to get the data. With this, every time the app will open a new screen, it will be necessary to make an HTTP request on the API to provide the data.

For the implementation of the API, the combination JavaScript + NodeJS[6] was chosen. JavaScript was chosen to maintain the standard and not increase the complexity with another language, once the serverless model must be implemented with JavaScript. Also, JavaScript is a widely used language, with several resources for documentation and help. The NodeJS was chosen because it is a consistent and widely well-known engine.

Nevertheless, the API is not fully implemented with NodeJS. Only the synchronizers are implemented with NodeJS, the rest (the Data and Login entities) are implemented with GraphQL[7]. GraphQL is a query language for the API. With this engine, the application has a single endpoint to provide all the features. It has many advantages in using GraphQL, but the most suitable advantage for this application is: with a single query, it is possible to return all the data to the mobile application. In this query, it is possible to specify which attributes are necessary (such as the username, the user's access token, the user's assignments, etc). Herewith, it is not necessary to implement several endpoints that return only one entity (for instance, an endpoint to return the user's assignments or user's access token). All information can be returned on a unique endpoint. Yet, if the mobile application accesses this unique endpoint but does not request some information (such as assignments), that information will not be processed and loaded.

3.2.3 Self hosting with local database

This architecture is also built using the self-hosted model described in Subsection 2.3.2. Therefore, this architecture is an uplift from the Self-hosted architecture. This uplift

[6]https://nodejs.org/en/
[7]https://graphql.org

comes from an implementation of a local database within the mobile application.

The Figure 3.3 show how this architecture was designed.

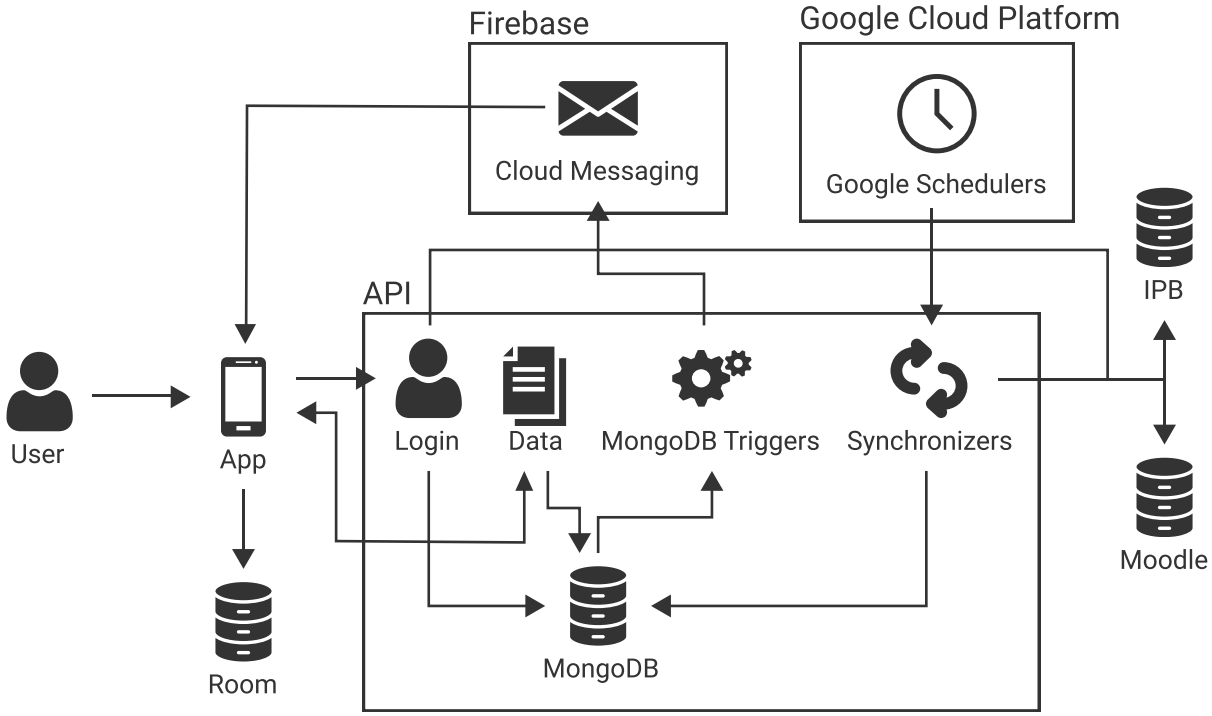

Figure 3.3: Self hosting with local database architecture diagram.

The above image shows the architectural arrangement. This arrangement is very similar to the previous Figure 3.2, with one more item, the local database Room[8]. Room is a local database for Android applications and is recommended by Android. There are other options for the local database, such as SQLite[9] and Realm[10]. SQLite is very similar to the Room, but it has more problems and is less powerful. The Android recommends using Room instead of SQLite for this reason. In turn, the Realm is a cloud solution. The data will be stored on a cloud server and the synchronization is made by the engine, very similar to the Firestore, which is used on architecture 3.2.1. However, the main idea of this architecture is to build a self-hosted solution, so using the Realm database means using the cloud. Taking these aspects into consideration, the local database Room was

[8]https://developer.android.com/training/data-storage/room
[9]https://developer.android.com/training/data-storage/sqlite
[10]https://realm.io/docs/java/latest/

chosen.

Using a local database brings a better user experience, saves network data, and has the possibility of working offline, but also, it brings problems with the data synchronization between the app and the server.

The user experience is increased because as the data is stored locally, the time to access this data is very fast compared to accessing the data in the API (which has to access the Internet). As a result, the time loading the screens and the data is significantly reduced. It brings the user a better experience, without the need to wait for the screen load, which can be slow due to internet access.

The network data can be saved because as the data is stored locally, it is not necessary to request the API every time that one screen will be opened (that is what happens on the architecture 3.2.2). Once the data is stored locally, they will be accessed locally. As a result, the app can work offline and show the data from the local database.

The biggest problem is that this data may be out of date. Depending on the synchronization policy, the information (which is based on local data) that is shown to the user may have an older version of the data. For example, an assignment that has a due date for one date is rescheduled to have a different due date. As long the app does not perform the synchronization, the local database will be inconsistent with reality and the information presented to the user will be incorrect.

3.2.4 Discussion

This chapter described three architectures for the system. The first one uses the serverless model on the backend and mobile use the cloud services. The second uses the self-hosted backend and the data is requested on open the screens. The Self-hosted with local database architecture also uses the self-hosted backend, but the data is stored within a local database.

The Serverless architecture has higher productivity. Using cloud solutions, problems such as data synchronization and infrastructure efforts are significantly mitigated. As a

result, the application can be done quickly and time can be saved. Thus, it is possible to implement more features considering the time saved. However, if the application grows in proportions of logic (e.g., implementing different services and features), the serverless model can become a problem.

The Self-hosted architecture has greater flexibility. Once the server is fully implemented by the developer, it can increase the logic without any problem. However, efforts with infrastructure (e.g., ensuring the data availability, data replication, etc) are necessary. This means that time must be spent on these efforts. As a result, features may not be implemented due to the time spent on infrastructure efforts. Also, as this architecture does not have a local database, the user experience is not better due to the loading time when opening the screens. Furthermore, this architecture consumes a greater amount of network data than the first, since the data is requested each time a screen is opened.

The Self-hosted with local database architecture is an uplift of the Self-hosted architecture. It mitigates the problems found on the Self-hosted architecture, such as the consumption of network data, and improves the user experience reducing the loading time on opening the screens. However, this architecture needs to define a good policy synchronization to avoid data inconsistencies. This means that time must be spent implementing the local database and synchronization policy. As a result, features may not be implemented due to the time spent on them. The Serverless and Self-hosted architecture do not have these problems. On the first, the local database is implemented using Firestore, which takes care of the data synchronization. The Self-hosted architecture does not have a local database, so there is no need to implement data synchronization. The most suitable solution for solve the data synchronization problem is to use a web socket. With the web socket, each time the data is modified in the backend, the communication is made with the mobile application and the data is changed on the local database. However, this solution requires more time and it is not feasible to implement it in the requested time.

Chapter 4

Implementation

This chapter will describe the system specifications and its implementation process. This includes what implementation decisions were made, which and where the features were implemented, and how the process to achieve them was for each architecture implemented. Also, this chapter will show the results of a performance test made on the Serverless architecture application.

The architectures have different levels of implementation, i.e., some features may be implemented on a specific architecture and may not be implemented on another. All entities that are supposed to implement were described in Section 4.1.

This chapter is divided into five sections. Section 4.1 will describe the system and its features in different ways. Sections 4.2, 4.3 and, 4.4 describe the implementation of the three architectures proposed. Section 4.2 will describe which kind of methodology was used to make the project management. Section 4.3 will describe which tools and concepts were used to make the code management. Section 4.4 will show which tool is used to make the credentials management within each project. Section 4.5 will describe which features were implemented in each architecture and how were the decisions made to implement them. Finally, the Section 4.6 will show the results obtained by a performance test on the Serverless architecture application.

4.1 System description

This system has the purpose to enable LMS users to visualize their information and follow the learning process through the mobile application. This proposed system is made to validate if the architectures proposed in Chapter 3 will work properly. Also, it is important to note that this system will not have data input, just data output, i.e., the user will only be able to visualize the data and will not be able to make data inputs such as sending messages, submitting assignments, etc. This decision, of not allowing users to make data inputs, is made due to the Web Services available.

The features chosen to be implemented were: announcements, assignments, forums, and resources. These features were obtained by two main factors. How much the feature is used was the first criterion to define which feature will be implemented. The second criterion was the feasibility of implementing that feature. Not every feature was available on IPB Web Services. Also, the feature may have an implementation available on IPB Web Services, but not support the use of that Web Service efficiently and completely.

The entities definition for each feature are:

- Announcements: it represents an announcement made within a subject. Usually, this is used to inform the students about something, such as reminders, information about the subject, etc.

- Assignments: it represents an assignment to a student within a subject. This is used to create a delivery for some task.

- Forums: it represents a forum within a subject. Within the forum, there are the topics. Within the topics, there are the conversations. Yet, within the conversation, there are the replies. This feature is usually used to make a conversation around some specific topic. The student and teacher can create a topic within a forum, after creating a conversation within the topic, and finally, create a reply for a specific conversation.

- Resources: it represents the resources within a subject. This is usually used by the

teachers to provide files and links for the student. So, it has folders, files, and links that the student can access and download, in the case of the files.

This section is divided into two subsections. Subsection 4.1.1 will describe the system requirements in two different ways. Subsection 4.1.2 will show the mockups of the main screens of the proposed system.

4.1.1 System requirements

This section will describe the system in two different ways. On 4.1.1 a description of the system with User stories is made and on 4.1.1 the system is described with a use case diagram.

User stories

- As a student, I want to make login on the system, so that I can view my LMS content;
- As a student, I want to change the institution I am going to log in, so that I can log in on the correct institution;
- As a student, I want to visualize my announcements for a given subject, so that I can stay informed about that subject;
- As a student, I want to visualize all announcements of all subjects, so that I can stay informed about all subjects;
- As a student, I want to visualize the details of the announcement, so that I can check all information about that announcement;
- As a student, I want to visualize the assignments deliveries deadlines for a given subject, so that I can manage myself to not delay any of them;
- As a student, I want to visualize all the assignments deliveries deadlines from all subjects, so that I can have a general overview of the deliveries;

- As a student, I want to visualize the forums for a given subject, so that I can stay informed about what is being sent on that forum;

- As a student, I want to visualize the forums for a given subject, so that I can follow what is happening in that forum;

- As a student, I want to visualize the topics for a given forum, so that I can stay informed about what is being sent on that topic;

- As a student, I want to visualize the replies for a given conversation, so that I can stay informed about what is being sent on that conversation;

- As a student, I want to visualize the resources for a given subject, so that I can visualize which resources are available for me on that subject;

- As a student, I want to browse through the resources folders for a given subject, so that I can visualize which resources are available for me on those folders;

- As a student, I want to download files from resources, so that I can visualize documents that the teacher has made available to me;

- As a student, I want to open the links from the resources, so that I can open important links that the teacher has made available to me;

- As a student, I want to receive push notifications when an assignment is created, so that I can stay informed about the assignments without having to open the mobile application constantly.

To provide a better understanding of the system, a Use Case Diagram for it is described following.

Use Case Diagram

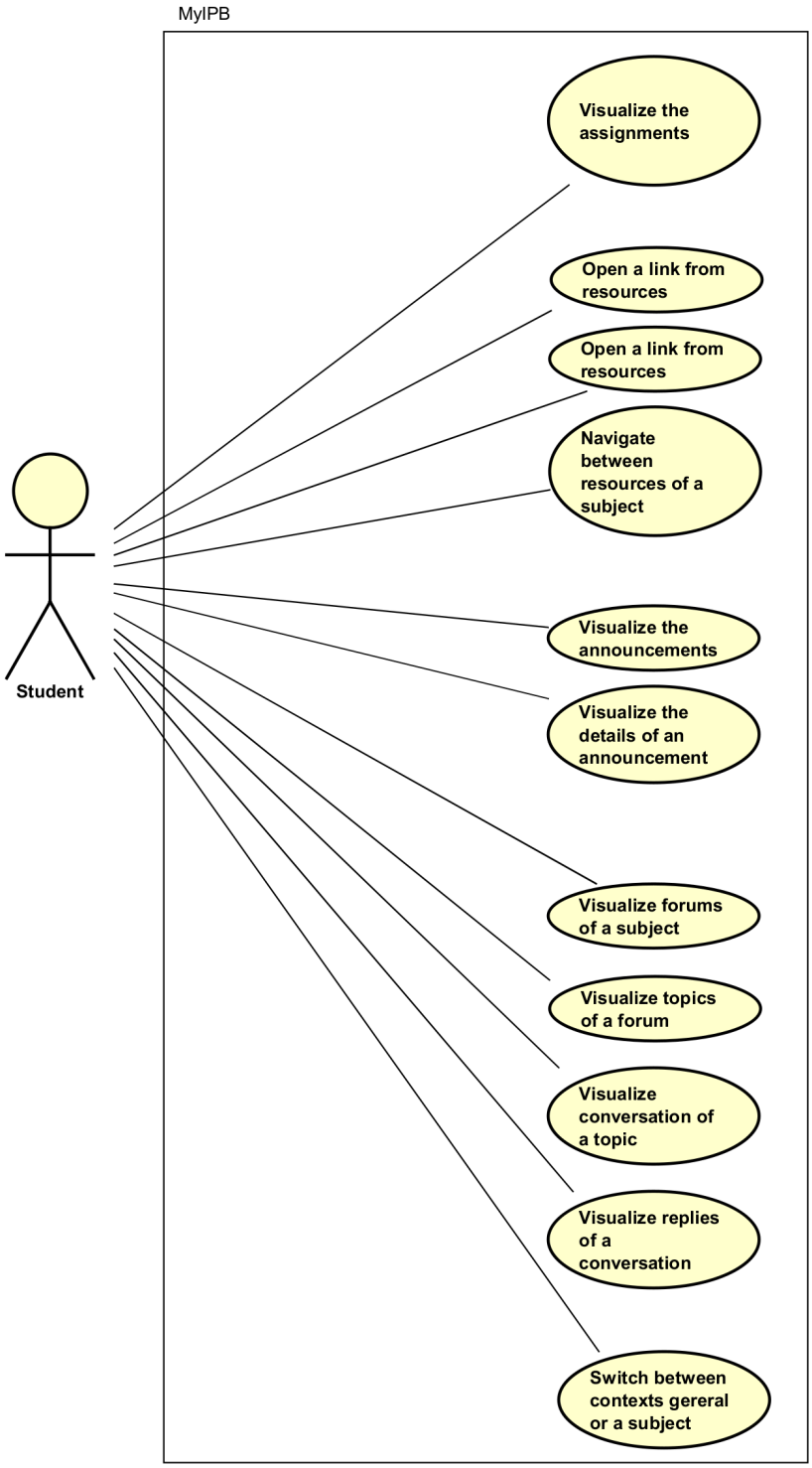

Figure 4.1: Use case diagram of the mobile application MyIPB on the IPBvirtual part.

To have a visual representation of this system, the next Subsection will show the mockups of the main screens of this system.

4.1.2 Mockups

In this Subsection, six Figures showing the mockups of the main screens of this system will be presented. These mockups were made with the collaborative interface design tool Figma[1].

The Figure 4.2 shows the mockup of the screen responsible for enabling the login to the student. This screen is the home screen, when the user is not logged in.

Figure 4.2: Mockup of the login screen.

The Figure 4.3 shows the mockup of the screen responsible for enabling the student to browse on the application through a sidebar.

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

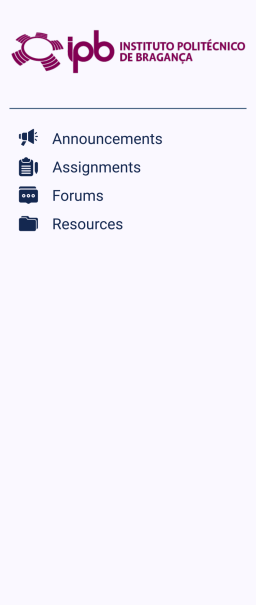

Figure 4.3: Mockup of the application sidebar.

The Figure 4.4 shows the mockup of the screen responsible for showing the announcements to the student. This screen is the home screen, when the user is logged in.

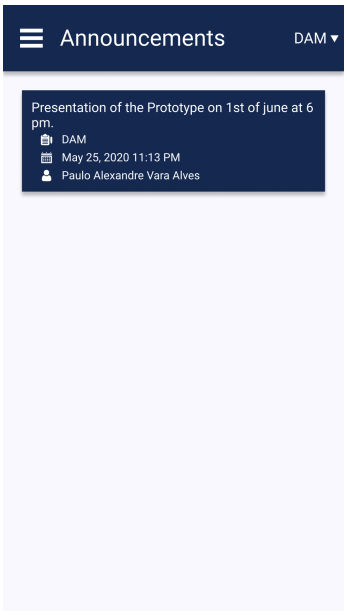

Figure 4.4: Mockup of the announcements screen.

The Figure 4.5 shows the mockup of the screen responsible for showing the assignments to the student.

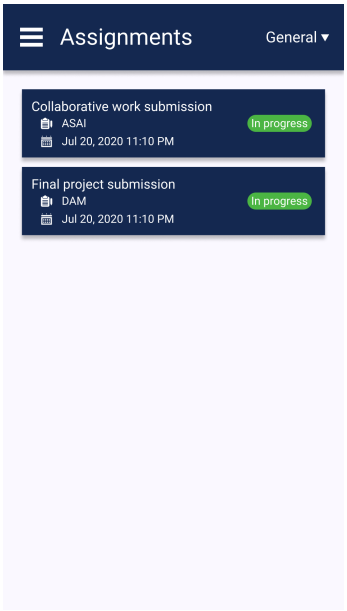

Figure 4.5: Mockup of the assignments screen.

The Figure 4.6 shows the mockup of the screen responsible for showing the forums to the student.

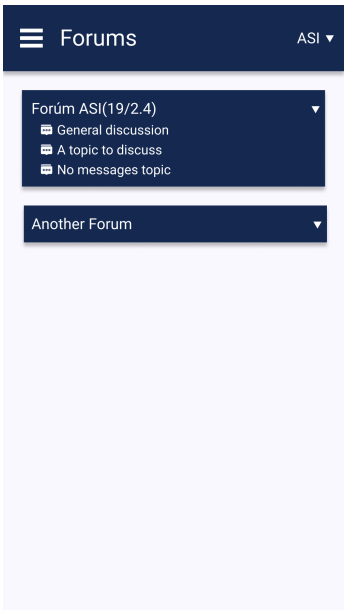

Figure 4.6: Mockup of the forums screen.

The Figure 4.7 shows the mockup of the screen responsible for showing the resources to the student.

Figure 4.7: Mockup of the resources screen.

On the login screen, there is a dropdown selector that enables the student to change the institution. Also, on each logged in screen, there is a dropdown selector on the app bar. This dropdown selector is where the student selects the context of the visualization, i.e., whether the components will show the content of a given subject or the general context, which includes all subjects.

The Appendix A shows the remaining mockups for the other screens.

The following section will describe which methodology for the development process was used.

4.2 Project management

For the development process, the agile methodology was chosen. The agile method is a widely used methodology that consists of a set of practices to make the development process productive. In this methodology, the development process is made in sprints, which can be defined as a short and repeatable phase to achieve some deliverables. The sprints usually have weeks of length.

In our project, the duration of the sprints was two weeks. Then, once every two weeks, a meeting is held with a discussion about what was delivered and what is the next deliverable for the next two weeks. Besides, in this meeting, problems, solutions, and project decisions such as defining the new features that will be implemented are discussed. The task priority evaluation was carried out at the meetings to deliver the most important features first.

4.3 Code management

For the code, the version control was made with the GitHub[2]. GitHub is one of the most popular version control tools and is widely used across the world. Within GitHub, the branching model was used. The idea of the branching model is to use branches that represent an isolated version of the code that is modified locally and after can be merged on the main branch. It allows parallel development on projects.

In our project, the branching model was applied to divide the features within each branch. Herewith, the implementation of each feature is divided within a branch and the development history can be easily understood. So, occasionally, if there is a need to review what has been done before, the implementation of the feature will be on a specific branch and can be done quickly.

4.4 Credentials management

For the credentials management, the *sops*[3] was used. This tool allows to encrypt and decrypt data using the Google credentials. So, using this tool, a file containing the credentials (i.e., admin user and admin password for the LMSs) is encrypted and can be pushed to GitHub, and after, a decrypt can be made.

[2]https://github.com
[3]https://github.com/mozilla/sops

4.5 Architecture implementation

This project has three architectures that were described before in Chapter 3. Each architecture has a different implementation level, i.e., depending on the architecture, it can have more or less features implemented. It occurs because the main focus of this project is to find an architecture that works in different contexts (serverless and self-hosted). To check if the main architecture design will work in a determined context (e.g., in self-hosted context) it is not necessary to implement all features. Taking these aspects into account, the implemented features divided by context were:

- Serverless architecture: in this architecture the entities announcements, assignments, forums, and resources were implemented for the IPB implementation of the Sakai. Also, the push notification system was implemented to work when a new assignment is created.

- Self-hosted and Self-hosted with local database architectures: in these architectures, only the entity assignment was implemented for the IPB implementation of Sakai and the Moodle. The push notification system for assignment creations was implemented as well.

Therefore, looking at the implementation level described above, it is possible to conclude that the Serverless architecture was the chosen architecture to implement all features. The Self-hosted and Self-hosted with local database architecture was chosen in such a way to make a proof of concept to check if the proposed architecture will work in the context of self-hosted and using a different LMS. This different LMS was the Moodle and it was chosen due to the popularity of the LMS.

The following Subsections will describe each architecture implementation individually, bringing details about the implementation.

4.5.1 Serverless architecture

The Serverless architecture uses the Firebase Functions and Firestore within the Firebase tools. The mobile application code was made on the master branch and can be found on GitHub[4].

Using the Firebase Functions, five functions were implemented:

- *login*: responsible to implement the entity Login described on Chapter 3. Must receive the username, password, FCM Token (token used to send push notifications), and the institution required. Will return the access token.

- *assignmentsSync*, *subjectsSync*, *announcementsSync*: responsible to implement the entity Synchronizers described on Chapter 3. These 3 functions are responsible to make the synchronization between backend data and the LMS data. Receive no parameters and returns just a status code.

- *onCreateAssignment*: responsible to implement the Database Triggers described on Chapter 3. This function is not possible to receive a external call.

These five functions were implemented on the backend part and can be seen on the GitHub[5]. It is possible to notice there is no synchronizer for the forums and resources. It occurs due the IPB Web Service does not have an endpoint that returns all forums or resources for a user. So, to populate the Firestore with the forums and resources data, a request for each subject must be made. Taking into account that the number of subjects may be very high, it is not an efficient way.

On the mobile application, five main screens were implemented.

The Login screen is responsible for enabling the user's login. The logo and information about the institutions are loaded from the Firestore. This information about the institutions must be inserted on Firestore manually. The Figure 4.8 shows the resultant screen.

[4]https://github.com/eboliveira/LMS-App
[5]https://github.com/eboliveira/LMS-App-functions/

Figure 4.8: Implementation of the login screen.

The Announcements screen is responsible to show all announcements. The announcements is loaded from the Firestore. The Figure 4.9 shows the resultant screen.

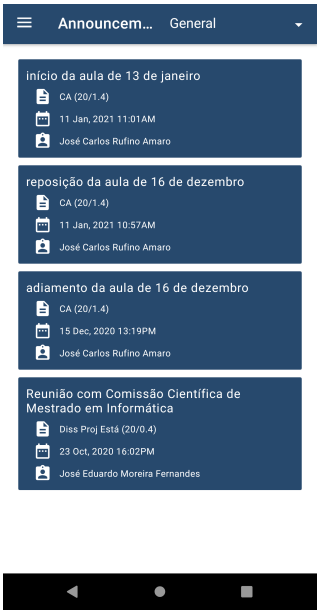

Figure 4.9: Implementation of the Announcements screen.

The Assignments screen is responsible to show all assignments. The assignments is loaded from the Firestore. The Figure 4.10 shows the resultant screen.

Figure 4.10: Implementation of the Assignments screen.

The Forums screen is responsible to show all forums for a subject. By this screen, is possible to open the topic screen, and after, open the conversation screen. Finally, by the conversation screen is possible to open the replies screen. The data loaded on these screens are loaded in real-time requesting the IPB Web Services. The Figure 4.11 shows the resultant screen.

The Resources screen is responsible to show all resources for a subject. This screen is implemented using WebView[6]. This method was chosen due the IPB Web Service does not have an endpoint that returns the content within each folder. Then, the entire screen is a WebView for resources. Thus, this screen will be quite different from the mockup shown in Figure 4.7. When a file is picked, if it is a link, the link will be opened within the same WebView, if it is a download file, the device web browser will open to make the download. The Figure 4.12 shows the resultant screen.

[6]https://developer.android.com/reference/android/webkit/WebView

Figure 4.11: Implementation of the Forums screen.

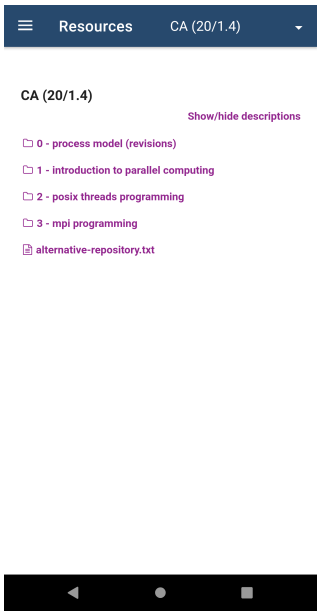

Figure 4.12: Implementation of the Resources screen.

Beyond these 5 main screens, there is the conversation, topic, replies screens that are part of the context of the forums. Also, there is a screen that shows the details for a specific announcement.

For the entire application, on the app bar, is possible to change the context of the

visualization. For example, on the announcements screen is possible to change the context to visualize only announcements of a specific subject. Is possible to select all user's subjects on this app bar. Also, for the announcements and assignments screen is possible to select the *General* context, that will return the assignments/announcements of all subjects. As the forums and resources screens are not loaded from the Firestore, it is not possible to add the *General* context for these screens.

The Figures that show the remaining screens implemented for this application can be found on the Appendix B.

4.5.2 Self-hosted architecture

The Self-hosted architecture uses NodeJS for the Web API and MongoDB as Database. The GraphQL is used as well to provide more efficiency and productivity. The mobile application code was made on a branch from the main repository and can be found on GitHub[7]. The backend code was made into a separated repository on GitHub[8] Using the Node JS, 3 endpoints are provided:

- *subjectsSync* and *assignmentsSync*: responsible to implement the entity Synchronizers described on Chapter 3. Note that on this architecture, just the synchronizers for subjects and assignments are implemented. This endpoints receive no parameters and returns just a status code.

- *graphql*: this endpoint (the only one) is responsible to implement the GraphQL. Within this endpoint, is possible to make the mutation *Login* that receives the username, password, institution, and FCM Token and is responsible to implement the entity Login and the query *Me* that receives only the access token and is responsible to implement the entity Data described on Section 3.2.2.

As described before in Section 3.2.2, the GraphQL enables the requester to require specific fields. Then, the query *Me* returns a User that contains all the required fields on the query.

[7]https://github.com/eboliveira/LMS-App/pull/6
[8]https://github.com/eboliveira/LMS-App-sh

It is possible to notice that on this architecture, the trigger was not implemented as shown in Subsection 4.5.1 that describes the implementation of the Serverless architecture. On the MongoDB, the implementation of triggers works differently. To implement the entity Database Triggers, it is necessary to define a callback function for change events on a specific collection. This implementation was done, but not on an endpoint, as seen previously.

On the mobile application, two main screens were implemented:

- Login: responsible to implement the entity Login that was described in Chapter 3. This implementation was made using the GraphQL mutation *Login*. It is possible to make login into IPB or UTFPR (which is an institution that uses the Moodle LMS).

- Assignments: responsible to show all assignments for the user. The data is loaded by using the GraphQL query *Me* when opens the screen.

These screens are equal to the screens shown in Section 4.5.1, there are no changes made on the interface, just on the functional side.

Is possible to notice that on this architecture, the data from assignments is loaded when open the screen by the GraphQL query. So, each time that the screens open, the mobile application will make a GraphQL request to the backend to get the assignments data. If other features (e.g., announcements) were implemented on this architecture, they will follow the same behavior of making the *Me* query to get the data.

4.5.3 Self-hosted with local database architecture

This architecture follows the same behavior of the Self-hosted architecture that was described above. The difference in this architecture is to have a local database. The mobile application code was made on a branch from the main repository and can be found on GitHub[9].

[9]https://github.com/eboliveira/LMS-App/pull/7

This local database is responsible for providing all data for the mobile application, i.e., on open the assignments screens, instead of request the API for the data, the screen will load the screen information by the data stored on the local database.

One important detail on this architecture is the choice of the synchronization policy. A synchronization policy chosen is to synchronize the data between the backend and the mobile application each time that the home activity is opened. With this choice, a request is made every time that the app opens. It is not a problem considering that the request made does not consume so much network data. Implementing in this way prevents the data inconsistency for almost the entire cases. There is a case that the user opens the app, keeps opened for a long time, and the backend changes, but the app does not synchronize. In this specific case, the data will be inconsistent.

Regarding performance, the next Section will make a discussion about the performance of the Serverless architecture.

4.6 Performance test

In this Section, results regarding performance will be shown. To acquire these results, the Android Studio Profiler[10]. With this tool, it is possible to measure in real-time the app performance. Four main aspects are shown: CPU Usage in %, memory usage in MB, network usage in KB/s, and energy consumption in four different levels.

For the tests, we used the mobile application of the Serverless architecture. The device used was an Android emulator with the following specifications:

- Device name: Pixel 2 XL
- RAM Memory: 512MB
- Android version: 9
- Core numbers: 4

[10]https://developer.android.com/studio/profile/android-profiler

Given this device, the flow of the test was:

1. Open the app (already open on announcements screen)
2. Open the assignments screen
3. Open the forums
4. Click to show the topics of a specific forum
5. Open a topic
6. Open a conversation
7. Back to conversation screen
8. Back to the topic screen
9. Back to forums screen
10. Open the resources screen
11. Change the context for another subject
12. Open a folder of the resources

The results obtained were:

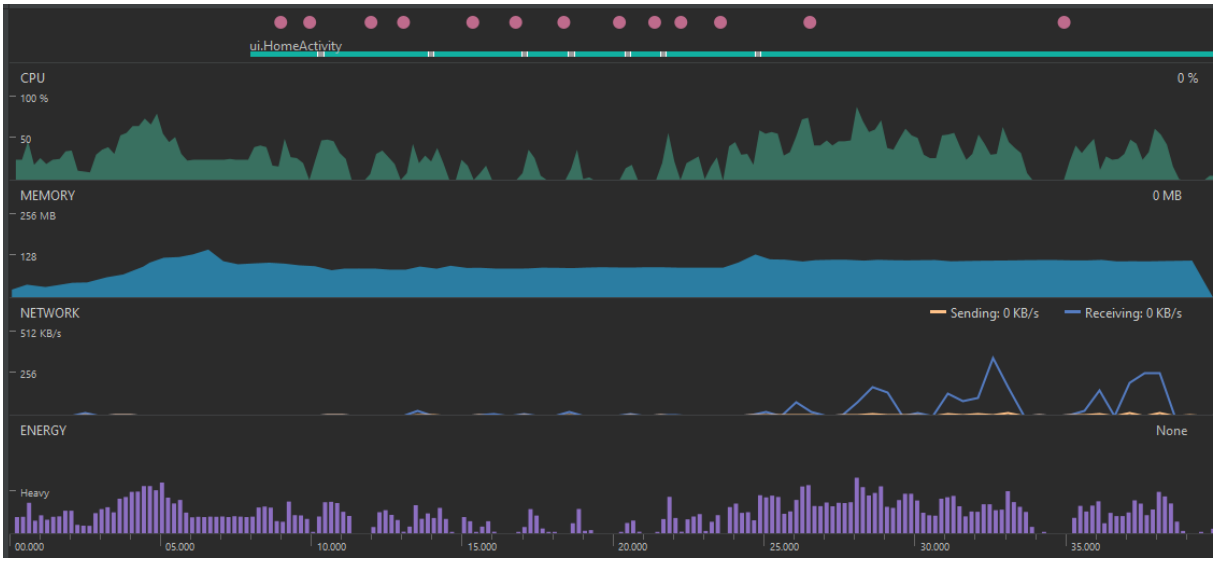

Figure 4.13: Profiling test of the serverless mobile application.

At the image, each dot on the top of the image represents a click made by the user. Below the dots, has a bar. This bar represents the activities that are running on the app. Below that bar, has four lines. The first line shows the CPU usage in %, the second line shows the memory usage in MB, the third line shows the network usage in KB/s, and the fourth image shows the energy consumption.

The entire test took 39 seconds. Looking at the bar of running activities, it is possible to notice that in the first 8 seconds, no screens are visible to the user. Also, it is possible to notice that has a big consumption of resources (except network data). These first 8 seconds was the app startup.

After the app starts, the user made two clicks, one to open the sidebar menu and another to open the assignments activity. It is possible to notice that these actions, clicking to open the side menu and opening the assignments, do not consume so many resources. The time between the end of the click to open the assignments screen and the screen is completely loaded is about 300ms.

After these two clicks, there are more two clicks, to open the side menu and open the forums. Similar to the previous one, these actions do not consume so many resources. When the user clicked to open the forums: the CPU usage went to 44%; RAM usage of 90

MB; a network consumption with a peak of about 25 KB/s of receiving data and 2 KB/s of sending data; and an energy consumption considerate medium. The forum screen took about 835ms until the click end and the screen is completed opened.

After the forum screen was opened, one click to open the topics list of a specific forum was made and one more to open the topic. So, the conversation screen of the topic was opened and another click to open the replies conversation. After this, two more clicks to back to the forum screens and one more click to open the sidebar menu. This entire process did not consume so many resources. No peaks and no problems of performance were found in this process.

After the side menu was open, a click to open the resource screen was made. The time between the end of the click and the screen loads by complete took about 1155ms. But this screen is a WebView, so, the screen ends the load does not mean that the WebView content was visible to the user. Before the WebView content was loaded, the user clicked to change the context to another subject. This click to change the context is represented with the last but one dot. Is possible to notice when the click to change the context, there was a higher network consumption. After the click to change the context was made, there is a peak of network consumption and CPU usage. The time between the click to change the context and the content of the WebView was completely loaded took 7620ms. At this time, there is a peak of network consumption of 347 KB/s of receiving data and 13.2 KB/s of sending data. Also, there is a peak of 88% of CPU usage.

After this, a click to open a folder of the resources was made and the results also consume a higher value of resources with a peak of CPU usage of 62% and a peak of network consumption with 262 KB/s of receiving data and 14 KB/s of sending data.

4.6.1 Discussion

Looking at the results, it is possible to notice that the biggest issue on the application regarding performance is the resources screen. The assignments and announcements screens are the more efficient screens regarding performance. Also, the forums screen did

not prove to have performance issues.

These results can be explained by the implementation of the screens. The announcements and assignments screens have the data loaded through the Firestore. Using the Firestore, that have a behavior similar to a local database, there are no performance problems. The forum data is loaded by HTTP requests, so, there is no frontend data and does not impact on performance. The major difference between from load the data from Firestore and load the data by HTTP requests can be noticed by the time until the screen is completely loaded. Using the Firestore, the time was about 300ms, using the HTTP requests was 835ms. And the worst results regarding performance was the resources screen. It can be explained due to the implementation made with the WebView. Using the WebView, it is necessary to load the data and the frontend part. As a result, a bad performance is expected. Also, there is a big network data consumption compared with other screens.

Looking at these aspects, it is pretty clear why the mobile application was not entirely implemented using the WebView concept. The resources screens, which use the WebView concept, got the worst results regarding every aspect analyzed on the performance test. Also, the interface that will be present is not controlled by the mobile application, thus, if the screen provider does not pay attention to the responsiveness, the screen rendered may not render correctly. As a result of not having control of what is shown, the user experience and usability may be impaired.

Chapter 5

Conclusions

During the execution of this work, three architectures were purposed to create a system that can aggregate LMSs. At the Serverless architecture, the serverless model on the back-end was used. At the Self-hosted architecture, the self-hosted model was used. Finally, the Self-hosted with local database architecture uses the self-hosted model as well but has an uplift on the mobile application including a local database. Each one of these architectures has advantages and disadvantages. Serverless architecture is the most productive architecture in terms of time and ease of implementation. The Self-hosted architecture is the median architecture in terms of productivity, however, is the worst regarding performance. The Self-hosted architecture with a local database is the worst in terms of productivity, however, has a better performance than the Self-hosted architecture.

These three architectures have different levels of implementation due to the Self-hosted architecture and the Self-hosted with local database architecture are a proof of concept. The Serverless architecture contains 4 features: assignments, announcements, forums, and resources. The other two architectures just have the assignments. Taking into account, the Serverless architecture was tested to check mobile application performance. This test was made with the Android Studio Profiler and from them was possible to conclude that the resources screen is the biggest drawback regarding performance. Also, this performance problem is due to the screen implementation is made with WebView concepts, which have a bad performance compared to the others. However, the mobile navigation was

prejudiced only on this screen, there is no issues regarding performance in the other three screens (mainly the announcements and assignments, that uses the Firestore).

Several improvements can be made to this system. Regarding improvements that can be made on all architectures, it is possible to implement the remaining features (e.g., online tests, send the push notification for all backend events) to be more complete. Also, it is possible to implement new features such as a home screen with a calendar showing the next events for the week or month or a screen that shows every update/creation of data on the backend. Another big improvement would be the implementation of data inputs. Currently, the app just shows the data for the user, but do not receive any data to send for the backend. Sending data such as replies on a conversation, create a conversation or a topic in forums can be an improvement. Some of these improvements can be made just with time, others depend on external factors (e.g., improvements on Web Service that the backend consumes).

On the Serverless architecture, it is possible to make the forums and resources be stored on Firestore, to achieve a better user experience and performance.

Regarding the Self-hosted with local database architecture (the Self-hosted architecture will be not taken into account because the Self-hosted with local database architecture is an uplift of the Self-hosted architecture), make an improvement on the data synchronization policy or just make a Web Socket to communicate with the backend would be a big improvement. With this web socket, the problems with data synchronization will be reduced drastically, fewer network data will be consumed and the performance will be improved.

Also, for future work, a performance test in the Self-hosted and Self-hosted with local database architectures could be made. With the results of these tests, it will be possible to conclude which architecture of the three is the better regarding performance. Also, other possible future work is to make usability tests for all three architectures to conclude which architecture has the better usability taking into account the user's opinion.

Appendices

Appendix A

Mockups

Figure A.1: Mockup of the login screen with another LMS.

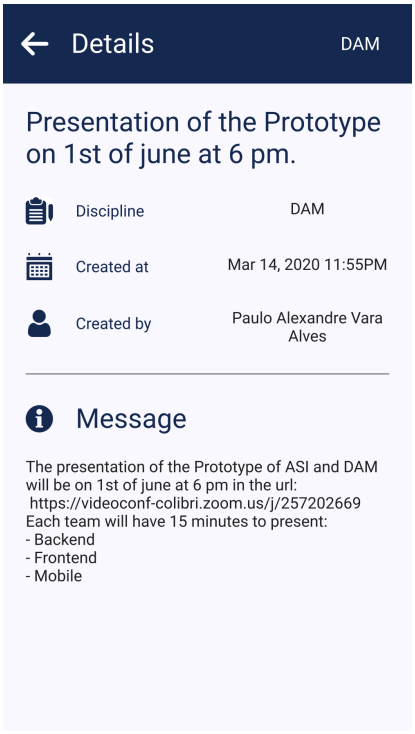

Figure A.2: Mockup of the announcement details screen.

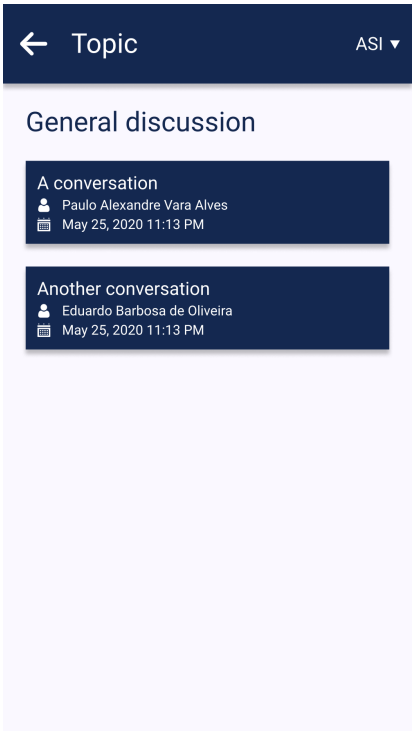

Figure A.3: Mockup of the topic screen.

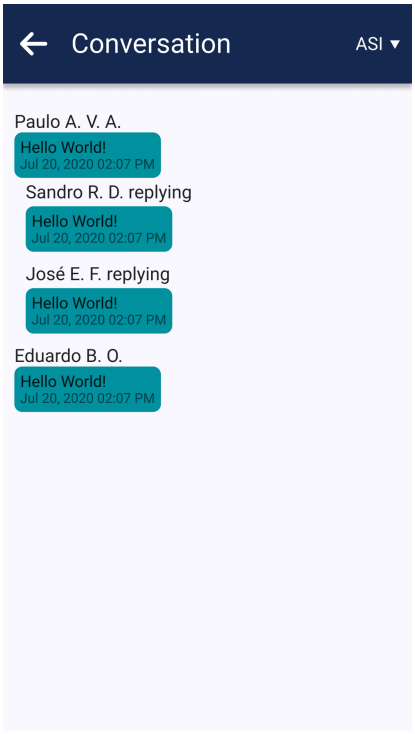

Figure A.4: Mockup of the conversation screen.

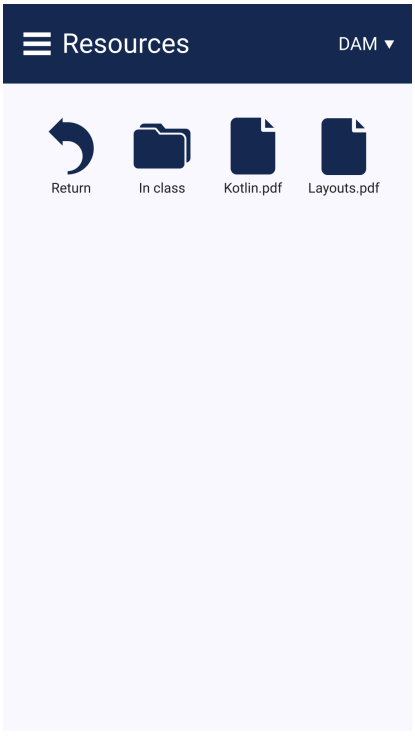

Figure A.5: Mockup of the resources screens within a folder.

Appendix B

Screenshots

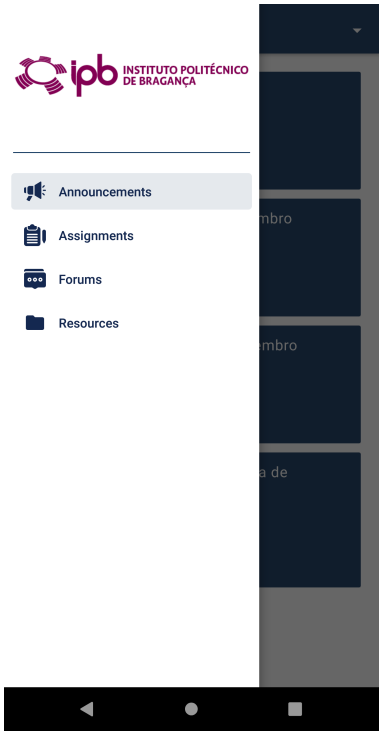

Figure B.1: Implementation of the application sidebar.

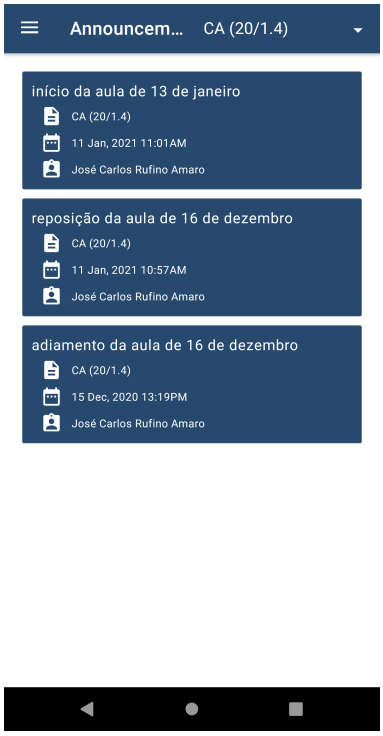

Figure B.2: Implementation of the Announcements screen with a filter applied.

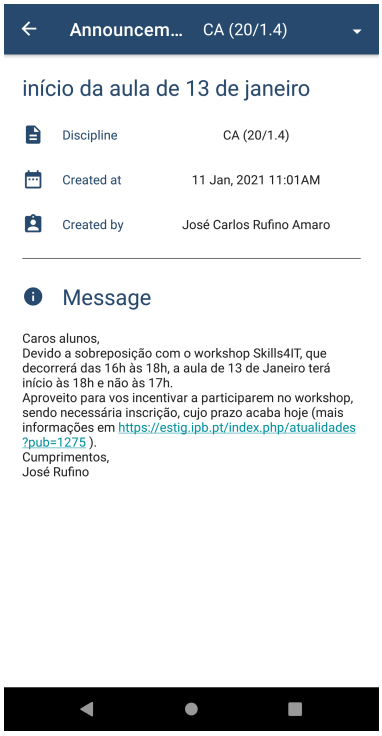

Figure B.3: Implementation of the Announcements details screen.

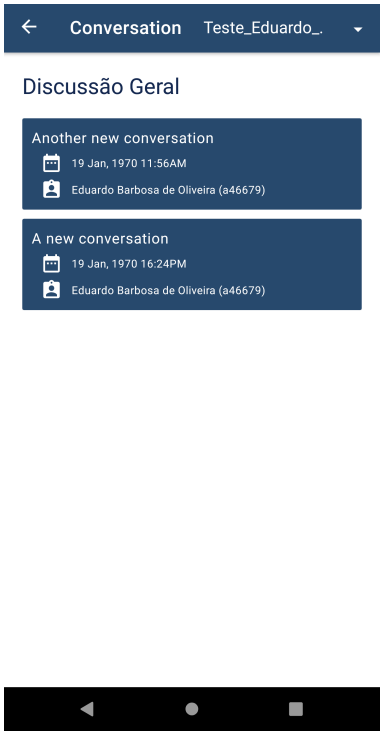

Figure B.4: Implementation of the Conversation screen.

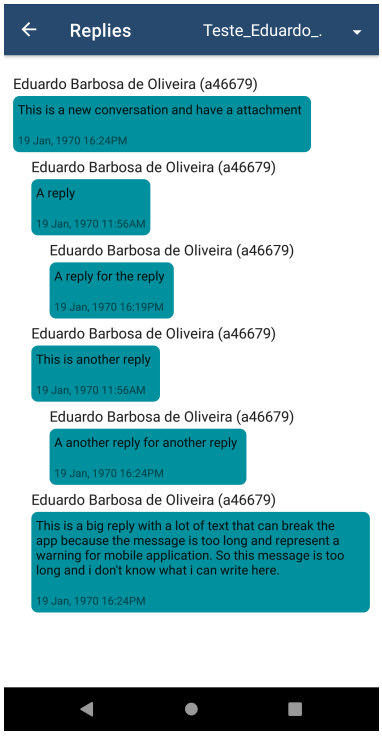

Figure B.5: Implementation of the Replies screen.

Figure B.6: Implementation of the resources within a folder.

Bibliography

[1] S. A. Raza, W. Qazi, K. A. Khan, and J. Salam, "Social isolation and acceptance of the learning management system (lms) in the time of covid-19 pandemic: An expansion of the utaut model", *Journal of Educational Computing Research*, 2020. DOI: 10.1177/0735633120960421.

[2] N. O. Oluwaniyi, B. O. Afeni, and O. O. Lawal, "Development of an asynchronous web based e-learning system", *Journal of Computer and Communications*, vol. 03, no. 12, pp. 84–99, 2015. DOI: 10.4236/jcc.2015.312008.

[3] S. Kumar, A. K. Gankotiya, and K. Dutta, "A comparative study of moodle with other e-learning systems", in *2011 3rd International Conference on Electronics Computer Technology*, vol. 5, 2011, pp. 414–418. DOI: 10.1109/ICECTECH.2011.5942032.

[4] J. Li and X. Wang, "Study the use of sakai in distance education", in *2010 International Conference on Educational and Information Technology*, vol. 3, 2010, pp. V3-167-V3–170. DOI: 10.1109/ICEIT.2010.5608399.

[5] S. Kamel, "Webct - evaluating the use of technology, lessons learnt from the american university in cairo", in *The 3rd ACS/IEEE International Conference on Computer Systems and Applications, 2005.*, 2005, pp. 893–896. DOI: 10.1109/AICCSA.2005.1387150.

[6] W. Gadzama, B. Joseph, and N. Aduwamai, "Global smartphone ownership, internet usage and their impacts on humans", Sep. 2019.

[7] *Virtual learning environments: concepts, methodologies, tools and applications*. Information Science Reference, 2012.

[8] R. Conijn, C. Snijders, A. Kleingeld, and U. Matzat, "Predicting student performance from lms data: A comparison of 17 blended courses using moodle lms", *IEEE Transactions on Learning Technologies*, vol. 10, no. 1, pp. 17–29, 2017. DOI: 10.1109/TLT.2016.2616312.

[9] Y. Kritikou, P. Demestichas, E. Adamopoulou, K. Demestichas, M. Theologou, and M. Paradia, "User profile modeling in the context of web-based learning management systems.", *J. Network and Computer Applications*, vol. 31, pp. 603–627, Jan. 2008.

[10] N. K. Singh and Y. N. Singh, "Scorm in brihaspati-3 learning management system", in *2011 3rd International Conference on Electronics Computer Technology*, vol. 4, 2011, pp. 97–101. DOI: 10.1109/ICECTECH.2011.5941865.

[11] C. C. Aydin and G. Tirkes, "Open source learning management systems in e-learning and moodle", in *IEEE EDUCON 2010 Conference*, 2010, pp. 593–600. DOI: 10.1109/EDUCON.2010.5492522.

[12] R. Kraleva, M. Sabani, and V. Kralev, "An analysis of some learning management systems", *International Journal on Advanced Science, Engineering and Information Technology*, vol. 9, no. 4, pp. 1190–1198, 2019, ISSN: 2088-5334. DOI: 10.18517/ijaseit.9.4.9437. [Online]. Available: http://ijaseit.insightsociety.org/index.php?option=com_content&view=article&id=9&Itemid=1&article_id=9437.

[13] M. Machado and E. Tao, "Blackboard vs. moodle: Comparing user experience of learning management systems", in *2007 37th Annual Frontiers In Education Conference - Global Engineering: Knowledge Without Borders, Opportunities Without Passports*, 2007, S4J-7-S4J–12. DOI: 10.1109/FIE.2007.4417910.

[14] A. C. Caminero, R. Hernandez, S. Ros, A. Robles-Gómez, and L. Tobarra, "Choosing the right lms: A performance evaluation of three open-source lms", in *2013 IEEE Global Engineering Education Conference (EDUCON)*, 2013, pp. 287–294. DOI: 10.1109/EduCon.2013.6530119.

[15] O. J. Dahl, E. W. Dijkstra, and C. A. R. Hoare, Eds., *Structured Programming*. GBR: Academic Press Ltd., 1972, ISBN: 0122005503.

[16] F. Brooks Jr, *The Mythical Man-Month: Essays on Software Engineering*. Jan. 1995, ISBN: 978-0-201-83595-3.

[17] M. H. Valipour, B. Amirzafari, K. N. Maleki, and N. Daneshpour, "A brief survey of software architecture concepts and service oriented architecture", in *2009 2nd IEEE International Conference on Computer Science and Information Technology*, 2009, pp. 34–38. DOI: 10.1109/ICCSIT.2009.5235004.

[18] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud computing and grid computing 360-degree compared", *Cloud Computing and Grid Computing 360-Degree Compared*, vol. 5, Jan. 2009. DOI: 10.1109/GCE.2008.4738445.

[19] W. Tsai, X. Sun, and J. Balasooriya, "Service-oriented cloud computing architecture", in *2010 Seventh International Conference on Information Technology: New Generations*, 2010, pp. 684–689. DOI: 10.1109/ITNG.2010.214.

[20] Y. Wei and M. B. Blake, "Service-oriented computing and cloud computing: Challenges and opportunities", *IEEE Internet Computing*, vol. 14, no. 6, pp. 72–75, 2010. DOI: 10.1109/MIC.2010.147.

[21] X. Pan, L. Ding, X.-y. Zhu, and Z.-X. Yang, "A social approach to high-level context generation for supporting context-aware m-learning.", *Eurasia journal of mathematics, science and technology education*, vol. 13, pp. 3675–3686, 2017.

[22] J. C.-Y. Sun and K.-Y. Chang, "Design and development of a location-based mobile learning system to facilitate english learning", *Universal Access in the Information Society*, vol. 15, no. 3, pp. 345–357, 2014. DOI: 10.1007/s10209-014-0392-x.

[23] A. Litchfield, L. Dyson, E. Lawrence, and A. Bachfischer, "Directions for m-learning research to enhance active learning", *ASCILITE 2007 - The Australasian Society for Computers in Learning in Tertiary Education*, Jan. 2007.

[24] S. Al-khamayseh, A. Bachfischer, E. Lawrence, and G. Culjak, "Mobile learning systems for digital natives", pp. 252–257, Mar. 2007.

[25] M. Kearney, S. Schuck, K. Burden, and P. Aubusson, "Viewing mobile learning from a pedagogical perspective", *Research in Learning Technology*, vol. 20, Feb. 2012. DOI: 10.3402/rlt.v20i0.14406. [Online]. Available: https://journal.alt.ac.uk/index.php/rlt/article/view/1225.

[26] P. Hung, J. Lam, C. Wong, and T. Chan, "A study on using learning management system with mobile app", in *2015 International Symposium on Educational Technology (ISET)*, 2015, pp. 168–172. DOI: 10.1109/ISET.2015.41.

[27] A. Bakhouyi, R. Dehbi, and M. Talea, "Toward an adaptive architecture for integrating mobile apps with lms using next generation of scorm", in *2019 2nd International Conference on Computer Applications Information Security (ICCAIS)*, 2019, pp. 1–7. DOI: 10.1109/CAIS.2019.8769575.

[28] H. Heitkötter, S. Heierhoff, and T. A. Majchrzak, "Evaluating cross-platform development approaches for mobile applications", vol. 140, Jan. 2013, pp. 120–138. DOI: 10.1007/978-3-642-36608-6_8.

[29] W. Jobe, "Native apps vs. mobile web apps", *International Journal of Interactive Mobile Technologies (iJIM)*, vol. 7, pp. 27–32, Oct. 2013. DOI: 10.3991/ijim.v7i4.3226.

[30] 2015. DOI: 10.4135/9781473933279. [Online]. Available: http://dx.doi.org/10.4135/9781473933279.

[31] P. Que, X. Guo, and M. Zhu, "A comprehensive comparison between hybrid and native app paradigms", Dec. 2016, pp. 611–614. DOI: 10.1109/CICN.2016.125.

[32] I. Malavolta, S. Ruberto, T. Soru, and V. Terragni, "End users' perception of hybrid mobile apps in the google play store", Aug. 2015. DOI: 10.1109/MobServ.2015.14.

[33] M. Q. Huynh, P. Ghimire, and D. Truong, "Hybrid app approach: Could it mark the end of native app domination?", *Issues in Informing Science and Information Technology*, vol. 14, pp. 049–065, 2017.

[34] M. Q. Huynh and P. Ghimire, "Browser app approach: Can it be an answer to the challenges in cross-platform app development?.", *Journal of Information Technology Education : Innovations in Practice*, vol. 16, pp. 047–068, 2017.

[35] B. Link and A. Back, "Classifying systemic differences between software as a service- and on-premise-enterprise resource planning", *Journal of Enterprise Information Management*, vol. 28, pp. 808–837, Oct. 2015. DOI: 10.1108/JEIM-07-2014-0069.

[36] P. Leitner, E. Wittern, J. Spillner, and W. Hummer, "A mixed-method empirical study of function-as-a-service software development in industrial practice", Jun. 2018. DOI: 10.7287/peerj.preprints.27005.