Mobile Education Platform

Smart Caching Learning Materials

Project report

P4 - GR452

AALBORG UNIVERSITY

STUDENT REPORT

Aalborg University
Internet Technologies and Computer Systems

Internet Technologies and Computer Systems
Aalborg University
http://www.aau.dk/

AALBORG UNIVERSITY
STUDENT REPORT

Title:
Mobile Education Platform

Theme:
Embedded Real Time Systems

Project Period:
Spring semester 2019

Project Group:
GR452

Participant(s):
Daniel Brtize
Robert Nedergaard Nielsen

Supervisor(s):
Per Printz Madsen

Copies: 4

Page Numbers: 108

Date of Completion:
May 28, 2019

Abstract:

This report presents a perspective on how to work with the UN Sustainable Development Goals within IT. The project is a collaboration between students from Brazil and Denmark, with the goal to create a system cable of providing course content to Brazilian waste pickers even with erratic mobile coverage. Starting by examining the terms for the project which will provide a base for the initial concept. The possibilities for developing this concept is then examined and end up as an implementation. This implementation is then compared to the established requirements and a plan for further development is presented.

Abstract

Denne rapport præsenterer et perspektiv på hvordan man kan arbejde med FN's verdens mål inden for IT. Projektet er et samarbejde mellem studerende fra Brasilien og Danmark, med formål at lave et IT-system, der kan levere kursusmateriale til brasilianske skraldere på trods af den sporadiske adgang til mobildata. Først er vilkårene for projektet undersøgt hvilket munder ud i en problemformulering som er grundlaget for det initierende koncept. Mulighederne for udviklingen af dette koncept bliver undersøgt hvilket munder ud i en implementering som afslutningsvis bliver sammenlignet med de opstillede krav. Denne implementering opfylder ca. 70% af kravne opstillet. På baggrund af de manglende opstillede krav opstilles videruviklings punkter.

Contents

Todo list

1 Introduction

The way Brazil handles their waste is shifting, this results in the closure of numerous dump sites[2]. This has lead to a challenge, wherein a lot of people who used to make a living at the dump sites by scavenging the trash for recyclable materials, commonly referred to as waste pickers[26], are left without a stable source of income.

This introduction serves the purpose of creating an overview of the challenges of these waste pickers, and if an IT project can help solve it. To do so a problem identification stage is commenced, highlighting the problems the Brazilian waste pickers face. From this problem identification, a initiating problem statement is generated, allowing for an in depth problem examination. In order to examine this problem, both field and desk research is conducted. Since this case if very specific to the waste pickers special focus is given towards different aspects of the waste pickers problems related to this project. This includes examining both their conditions trough an interview during field research and trough desk research, looking into how the United Nations(UN)' Sustainable Development Goals(SDGs)[31] are part of this project. Furthermore an examination of smartphones, their adoption rates, coverage and the waste pickers ability to use them is examined. Before concluding the international context of the project is taken into account, and a broader examination of similar problems exists around the world and if a similar implementation could help in these other contexts as well. Lastly an overview of the international project members is given, to create an overview of the different parties involved and their perspectives. With this initial research stage completed, a problem statement is made, determining the topics and scope of the research needed to be done.

Figure 1.1: Waste pickers working at the new facilities

To help these waste pickers the Brazilian government created some central recycling centers, where they hire the waste pickers to sort the waste under better conditions. Effectively, this means that they are contracted by the municipalities and get paid on a weekly or monthly basis as opposed to daily as they are accustomed to. Most of these people do not have a basic education[8] and as such they have limited knowledge on mathematics and subsequently financial management, see the interview in Appendix .1.

In order for them to better understand handling monthly finance, they need knowledge of basic mathematics such as addition and division. This lead to the initiating problem statements, see Section 1.2. However before arriving at the initial problem statement, the pre-project problem identification process is presented.

1.1 Problem identification

Originally, the idea of creating a framework for learning was developed during a summer school held in Brasília, Brazil during August 2018.

The goal of this summer school was to initiate the collaboration between Universidade de Brasilia(UnB) and Aalborg University(AAU). The goal of this collaboration is to work with and help solve part of the United UN' SDGs. Another part of the goal is to create project proposals that the students from AAU could bring back home and use as a semester project.

The proposals that were developed ended up not fitting into the curriculum of the danish participants and as a consequence of this, the projects where forced to be worked on as extracurricular activities. The issue with spare time projects, is that it is hard for students to prioritize compared to their own studies. Beyond this, there is not any obligation to do any work. With these two factors it was quickly found that projects will suffer and ultimately end up in a total stand-still. That was what happened to the original education project, working with creating a system that could deliver mathematics tuition to students in primary schools, much like the Green Shoots project, described later on in Section 1.5.

After the initial project died due to a lack of communication which lead to a lack of motivation as well, it was decided that it should be attempted, to see whether it was possible to get it going again by meeting up to talk about the progress already done and how to scope the project so that it fit into the curriculum of the AAU students.

During the second expedition, Global Students SDG Summit, in January 2019, the Brazilian professor João Mello, see Section 1.5 suggested that the target group could be the waste pickers mentioned in Chapter 1, because it would narrow down the target group and give the ability to make the project more concrete.

With the understanding of the origin of the project, as well as having uncovered a perspective for this project, an initial problem statement is made

1.2 Initiating problem statement

How can a system helping the Brazilian waste pickers gain knowledge on mathematics and financial management be build?

1.3 Problem examination

After having done research to identify issues within Brazil that this group might be able to help alleviate, examination of the problem defined in the initial problem statement in Section 1.2 can begin. Firstly. the field research conducted in collaboration with Grupo Gestão [11] and students from techno anthropology, see Section 1.5, during the Global Students SDG Summit is examined. In this field research an interview of the Brazilian waste pickers is conducted to better understand their situation.

With the field research completed, the broader perspective of the international context is explored. As this project aims to help in solving part of the SDGs, the goals themselves and how this project attempts to help it is highlighted. To specify the issue to smartphones in other locations, an examination of smartphone

adoption rates in emerging economies is given, to examine whether or not this project could potentially be modified to different parts of the world. Lastly the international collaboration highlighting the different project members is given, in order to showcase the different perspectives both professional and cultural.

1.3.1 Field research

To get an insight into the lives and needs of our target group some field research were done during the Students SDG Summit. This information will be used as an offset for the final problem statement, but also as input to the requirement specification.

As this is an interdisciplinary project the knowledge of the techno anthropology students can be used to generate the questions that the waste pickers are asked. These questions will focus on how the life as a waste picker changed after the closure of the dump site, their education and the education they are offered as a part of the job that Serviço de Limpeza Urbana do Distrito Federal (SLU)[9] hired them for.

To establish an understanding of the actuality of the problem some of the answers are highlighted below, a transcription of the complete interview can be found in Appendix .2, the interview is done by our local collaboration partner Matheus Halbe from Grupo Gestão, since the waste pickers does not speak English.

Firstly, knowledge about how the income of the waste pickers changed after the closure is needed, in order to see how the closure affected them.

Mateus Halbe: *So, let's start. We see things are better than before.*

Waste picker: *No, things are not better than before.*

Mateus Halbe: *Why?*

Waste picker: *Because before we had money.*

Mateus Halbe: *Are you making less money today?*

Manager of administration: *What is improved today in infrastructure, or space to work. The point of before they gained more is that before they worked individually then it was a war for the garbage, because the one that reached first would have property of garbage. Now at the Triage Center everyone works and the whole amount of revenue is divided. And who works more compared to other people ends up feeling hurt. So they think here is not better. He works a lot here and works a little and receives less here.*

This quotation from the interview shows that the waste pickers earn less after the closure of the dump site. Which means that the economic situation have not

been improved due to the closure, this can also be seen in quote bellow, however the work environment has improved.

Waste picker: *Most people here have already been through this extremely difficult financial situation, and some are still through. There are months here that we need to help each other, otherwise it is very difficult.*

To understand how bad the salary is, the waste pickers are asked how much the earn each month, which is R$ 1000 if it is a good month and they participate in the courses from SLU. If you compare this to the minimum wage in Brazil, which is R$954[6] it is only a bit more than that, however if you compare it with the average wage which is R$2270[40], then it is easily seen that the waste pickers does not have much money to work with.

Mateus Halbe: *How much do you get here?*

Waste picker: *With the benefit payed for the participation in the classes, R$ 1000. R$360 from the participation in the course.*

Since the idea is to use technology to improve the life of the waste pickers it is needed to know to which grade they have access to internet, so they are asked to show their phones. Based on the phones that was shown the conclusions is that they are pretty good. During the research trip, it was noticed that most people used WhatsApp. To see how they used it, Maria was asked about how she use the application.

Mateus Halbe: *Maria, how do you talk in the WhatsApp? you just listen to audio, is this?*

Maria: *Yes. I listen to audio and record the audio.*

This highlights that content such as videos or sound is important, as the waste pickers might not be able to read the content, this leads to another possible problem, data use as these file types are larger. With this understanding of how the waste pickers use their phones and their salary an examination of the Brazilian phone plans is done to understand how much data the waste pickers have available.

1.3.2 Brazilian phone plans

According to the Brazilian telecommunications association, the largest phone plan provider is Vivo[35]. Looking at their "regular" plans, they start at around R$229,99[42]. On a salary of R$700 this would be a high percentage of their wage to spend on

a phone subscription. However as it turns out, Vivo provides relatively tiny packages for a lot smaller fee[41]. The cheapest of these plans, which only provides 500MB of data is only R$36 per month[41]. The thing of note with these plans is that they all come with unlimited data for WhatsApp. Without having it confirmed by the waste pickers. It is assumed that since many of them where found to use WhatsApp, that they use plans similar to these. These plans however have very limited data outside of WhatsApp, and in order to be as non intrusive as possible, the application made in this project must therefore use as little cellular data as possible.

Having completed the interview process, the international context of this project as well as desk research is done.

1.4 International context

With the answers from the interview concluded upon, it is important to recognize the international context wherein this project exists. Firstly the SDGs concerning education are examined, and whether or not the project has potential to help is evaluated. To further broaden the project perspective to a wider global context, subjects such as smartphone adoption rates in emerging economies are examined. This will help see if a more generalized version of this project could potentially be implemented on a larger scale.

Since a vital part of this project is the internet connectivity on the Brazilian waste pickers smartphones, a look into the telecommunication infrastructure in emerging economies is done. To examine if this part of the system has potential to be is useful in a broader global context.

Lastly an overview of the international partners for this project is given, clarifying the active participants and cooperation partners. With this international context given, the problem statement based on both the field and desk research can be provided.

1.4.1 Desk research

This desk research phase seeks to cover any holes in the field research conducted in Brazil. Beyond that it also seeks to broaden the perspective of the project into a larger international context. Firstly all problems found during the field research are backed up using reliable sources. Any arguments wherein local desk research in Denmark provides insufficient evidence, the Brazilian cooperators are used as sources.

1.4.1.1 Sustainable Development Goals

The UN have compiled several SDGs in order to help improve our world or as they describe it:

The Sustainable Development Goals are a call for action by all countries – poor, rich and middle-income – to promote prosperity while protecting the planet. They recognize that ending poverty must go hand-in-hand with strategies that build economic growth and address a range of social needs including education, health, social protection, and job opportunities, while tackling climate change and environmental protection[31].

The focus of this project is directly set to help with goal number four, Education. While the scope of this project is purely to help the Brazilian waste pickers, a more generalized system could in theory be used on a larger scale. Take for example target one in goal number four:

By 2030, ensure that all girls and boys complete free, equitable and quality primary and secondary education leading to relevant and Goal-4 effective learning outcomes[30].

By providing the smartphone application for free, this could help to achieve the this goal. Beyond that, further development can lead to more versions specialized for different parts of the world. A requirement for this to be viable however, is that the people in need of this education have access to smartphones. Therefore an examination of smartphone adoption rates in developing countries is done.

1.4.1.2 Smartphone adoption rates

To gain insight into smartphone adoption rates in emerging economies, several statistics done by the Pew Research Center are used[34][28]. On Figure 1.2 the smartphone adoption rates of advanced and emerging economies are shown.

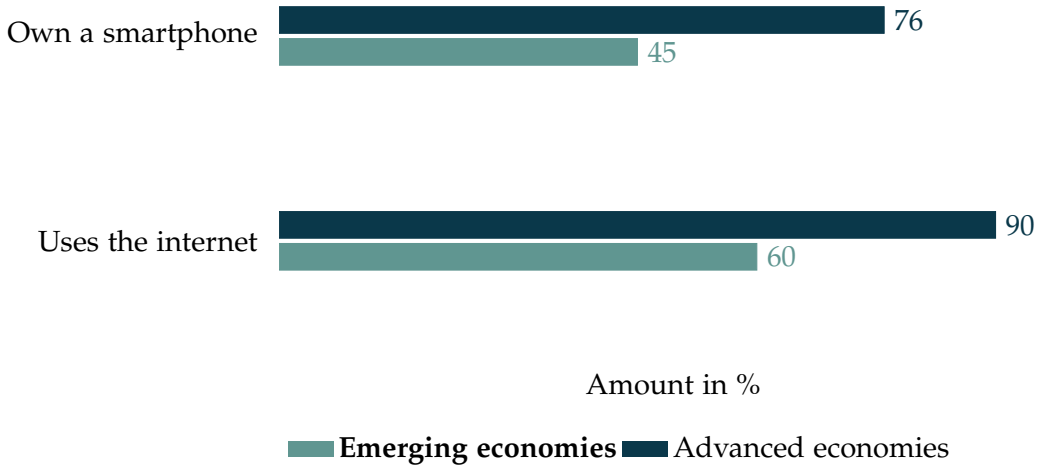


Figure 1.2: Mobile and internet use in advanced and emerging economies[28]

An important thing to note is that Brazil is part of the emerging economies. However it can be seen that as of 2018 even in emerging economies, almost half own a smartphone. While this means that many people can be reached with a smartphone application, considerations must be made to if the people who own smartphones also are the target demographic for this project.

It could be argued that the people who are in the most need of the base education system are those to poor to afford smartphones. While it is impossible to determine on a global scale, based on both observations and the interview conducted during field research, see Section 1.3.1, the waste pickers almost all had smartphones.

This is further supported by a research showing that 85% of Brazilians aged 18-34 own a smartphone as of 2018 see Figure 1.2. While current statistics are good, they cannot provide an overview of the growth or decline of smartphone adoption rates in all emerging economies. To gain an overview of this, a second Pew research paper is used[28]. On figure Figure 1.3 the growth of smartphone adoption percentage between 2015 and 2018 are compared, in three emerging economies one from each continent.

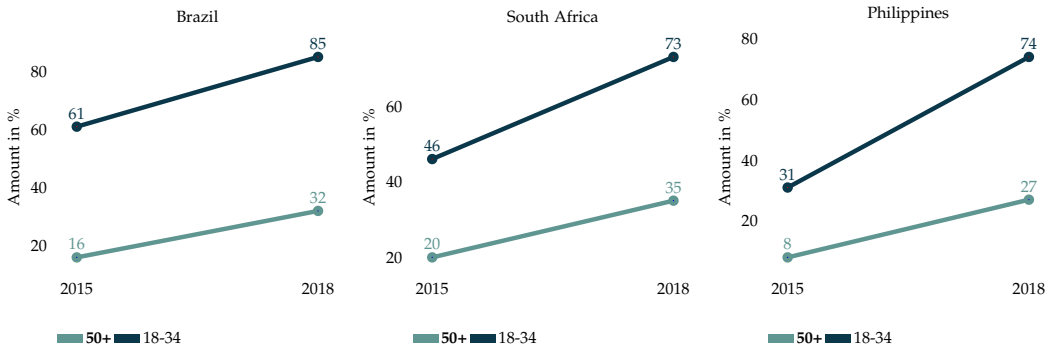


Figure 1.3: Smartphone adoption rate growth in emerging economies[28]

As can be seen in Figure 1.3, there is global trend of significant increase in smartphone ownership. If this trend continues, then a smartphone application to help educate people and improve their financial skills has potential to reach many people. The article also concludes the following:

Smartphone ownership rates have skyrocketed in many countries since 2013. This includes increases of over 25 percentage points among the total population in large emerging economies such as Turkey (+42 points), Malaysia (+34), Chile (+26) and Brazil (+26)[28].

Again showcasing the fact that smartphones are beginning to become more and more widespread even among traditionally poorer communities. However the question also stands if the issue with erratic internet connectivity is as big a concern in other developing countries. To find out about this an examination of telecommunication infrastructure in developing countries is done.

1.4.1.3 Telecommunication infrastructure

Similar to the research on smartphone adoption rates, the research on telecommunication is based of studies done by the pew research institute[4]. When it comes to telecommunication infrastructure, it must be considered that even in countries with stable cellular network infrastructure, the data rate might be to high for the target subjects of this application. This is the issue observed from the Brazilian waste pickers.

The graph shown on Figure 1.2 shows 60% of adults in emerging economies uses the internet. This is 15% higher than the amount of smartphone users. The people with internet access is also increasing [22]. More so the amount of internet users is growing from 1 billion in 2005 to almost 4 billion in 2018[37].

Overall this means that the amount of people with internet access is very much increasing alongside the amount of smartphone users. However making a system that works alongside caching is still a sensible choice as it allows for everyone who

can get to free internet access to use the system, further decreasing the barrier of entry.

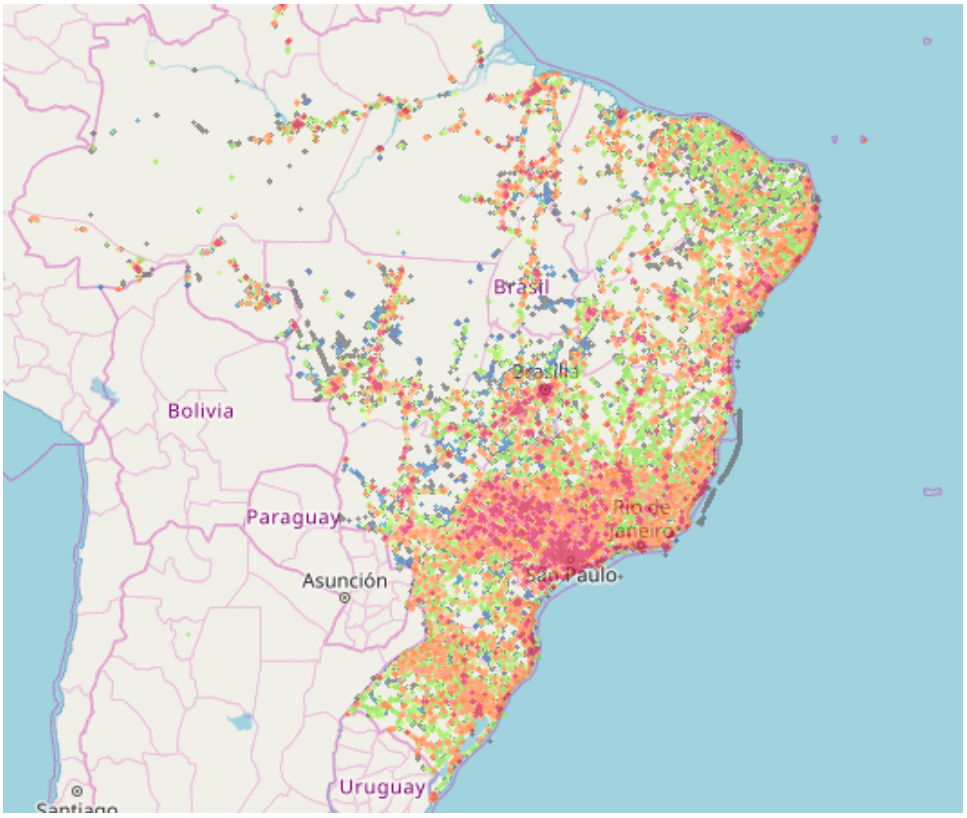


Figure 1.4: Vivo coverage map[33], darker color means better connection

It can be seen on Figure 1.4 that the major cities in Brazil all seem to have 4G connectivity, but moving further inland coverage becomes much more sparse. While this means the telecommunications infrastructure in Brazil in densely populated areas should be usable, the issue of low amount data in data plans as described in Section 1.3.2 means, that in this project scope, the Brazilian waste pickers must be seen as not having a functional data connection.

Having examined the telecommunication infrastructure, the cooperation partners for this project is listed.

1.5 International collaboration

Having completed the initial desk research phase, it becomes necessary to recognize the context this project is in. More specifically the fact that this project is part of a major cooperation between different groups. Accordingly, this section

will list all the collaboration partners and what their role will be in this project. As mentioned earlier this is a interdisciplinary project which means that not only people studying computer engineering will be participating. This will be a joint venture together with people who studies techno anthropology and Production engineering, an overview of the participating students can be seen in Table 1.1.

Denmark

Name	Field of study	Semester
Daniel Britze	Computer engineering	4th
Peter Lundgaard	Techno anthropology	8th
Robert Nielsen	Computer engineering	4th

Brazil

Name	Field of study	Semester
João Mello's subjects	Production engineering	TBD
Matheus Halbe	Production engineering	9th

Table 1.1: Table containing students participating in the project

This project will also be done in collaboration with SLU, Christian Hundborg Liboriussen(Christian) from the Green Shoots-project and Grupo Gestão as mentioned before.

SLU is the recycling waste management organization that oversees the closure of dump sites, and the creation of waste management facilities where waste workers sort recyclable materials from non-recyclable materials. The goal is to develop this system for SLU to use, so that they can offer education to more waste pickers and track more efficient whether they actually learn something. The collaboration with this organization will also facilitate the contact and dialogue with the waste pickers, which is needed to create a system that fit their needs.

Christian from the Green Shoots-project has extensive experience in creating IT solutions in a somewhat similar educational setting in South Africa, wherein internet connectivity is erratic[5]. In his project, his group created a system capable of synchronizing courses and data from a local server to schools in rural areas of Africa. This made the non-governmental organization (NGO)[32], Green Shoots[21] able to provide their education to even more schools. The system that will be developed, will have some similarities with what he did and his experience within this field, will aid the project.

The focus on this work is that of creating the back-end system, making it possible to deliver data in form of teaching materials to the Brazilian waste pickers. João Mello's subjects will focus on creating said teaching materials, meaning it is not part of this demarcations scope.

This means that the project scope for this demarcation is defined purely to a back-end system, that allows data to be transferred to the Brazilian waste pickers in a sensible way, allowing data to be cached, due to the waste pickers erratic internet connection. This is due to the project members of making this demarcation and project are studying IT system building. Meaning that there is very little focus within the education on things such as user experience.

Having gained this overview of the different cooperation partners and with the initial research phase complete, the problem statement for this project can be found to be:

1.6 Problem statement

How can a system with the ability to work with erratic internet connectivity deliver teaching materials to smartphones through smart caching be created?

2 Initial concept

Having completed initial research in order to find a problem statement, the problem must then be broken down into smaller parts. This is done as to make the problem statement simpler to overview, in accordance with the idea that more smaller problems are more easy to solve as supposed to one giant intertwined problem. Breaking down the problem statement into smaller problems is done via a brainstorming project, wherein an active conversation with all members is done. This input is taken and noted on one big paper, where the potential problems are categorized. This categorization provides different focal points, which all together gives an initial ideas as to which sub systems could lead to a full solution. This chapter will in turn showcase this initial concept model. A note is that the model is purely surface level and does not go into greater technical detail. The goal is to create an overview of the overall system the project is creating and set the framework for which research can be conducted. To do so efficiently, the initial concept model also splits the system into several subsystems that have to work together. This allows for each subsystem to be consequently examined, but before this technical analysis, an examination of potential implementation methods, inclduing an examination of existing solutions is done.

2.1 Concept explanation

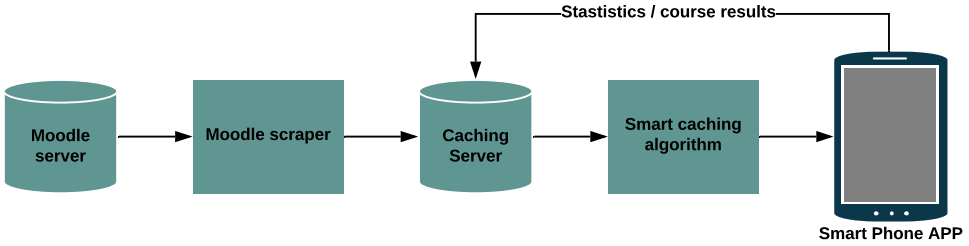


Figure 2.1: The initial concept for the overall system

The model on Figure 2.1 shows the overall concept of the project in simple terms, and outlines how the system is expected to function. Due to this being in the early stages of development, it was seen fit to do a more simple overview of the main concept. If at this stage a full requirement specification is made, it would not be based on research. Therefore each sub system is described in broad terms

so that it can be researched in detail later. On Figure 2.1 firstly a Moodle server is used for creating teaching materials for the waste pickers. Moodle is a open-source learning management system created to support blended learning, distance education, flipped classroom and other forms e-learning[43]. Using Moodle will give the Brazilian students whose role is to create said teaching materials a standardized system to work on, making both their and our job simpler. Moodle is also chosen because we already know how it works and the choice of the learning management system is not in scope of this project.

The teaching materials on the Moodle, will then be scraped in order the put the Moodle courses on the caching server, in a format that makes it easy for the caching server to send it to the users. In turn this creates an interface between the Moodle server and the caching servers database. By clearly defining this interface, it will allow future improvements. This will also allow for another learning management system to be used depending on the use case. The caching server consists of several systems and will be explored in much greater detail later on. However, in this overall concept it can bee seen as having four functions:

- Storing scraped Moodle courses.
- Storing identifying data on users, such as usernames and passwords.
- Calculating the optimal amount of course material needed for each user.
- Transferring data to the application.

All the calculations are meant to be done at this caching server in real time, so that each user always receives the correct amount of course data, when they are connected to the internet. Lastly a smartphone application is meant to facilitate the courses to the waste pickers. This application must also store statistics about the user to be sent to the caching server, but this concept will be explained in more detail later. With this initial concept model explored, an analysis of existing solutions is done, to ensure that this problem statement has not already been solved, and to see if any existing solutions could give valuable insight towards the development of the project.

2.2 Existing solutions

This section will focus on existing or similar solutions to this problem. The goal is to examine whether or not any existing solutions solve part of if not all of the initial problem statement. Examination of the existing solutions will therefore be mainly observations made from research. Since the entire project is based around a smartphone application, an examination of existing educational applications is done. Of note is that the contents of the applications and their presentation is the focus of the techno anthropologists and not this project. Effectively meaning that this analysis will not be as extensive as theirs. To further specify the applications to

be examined, they must have a sort of offline capability. Beyond these applications a previous project, mentioned in Section 1.5, has attempted to overcome the issue of erratic internet connectivity in South Africa. Because of this, the implementation used on that project, will be examined. Finally a summary of existing solutions is done, leading to the conclusion of the analysis.

2.2.1 Smartphone application

Firstly a list of relevant applications are found and a comparison between them is done. This will allow some important requirements to be extracted, if for example every version has something in common. Examination of the android applications will be split into two categories: Education and financial management. Importantly, this examination is not as detailed as the actual course is designed by other projects, namely the techno anthropologist as well as the students in Brazil.

2.2.1.1 Education

Whilst searching the Google play store for educational applications, a number of indicative observations where made: Quite a large selection is available, allowing for generalized learning. However considering the educational level of the waste pickers both from our observations during the field research in Section 1.2 and the statistics provided by WIEGO [25], most applications on that technical level are aimed at children. Examples include Khan Academy for kids [17]. Using these applications would therefore be quite demeaning towards adults and are therefore not a great fit. This is due to the way in which they attempt to appeal to children, which can come of as condescending to adults.

2.2.1.2 Financial management

When it comes to financial management applications, there are tons of applications to help the user manage their money. Many of these come highly rated by their users. While researching these apps a focus was put on what kind of educational content they provide to the user. Meaning a system with complex financial management systems is not nearly as relevant as an app that provides relatively simple financial help, but explains itself in detail. This would allow an app to effectively both provide assistance in the financial management as well as an educational platform for this very subject.

Examples include 1Money, Monefy and Easy Home Finance [15]. While these applications all aim to make financial simpler for the user, very few of them provide teaching materials. This means that while the user can possibly gain an overview of their finance, there is very limited help in how to do so.

When seeking applications for education on financial management, the selection was a lot more limited. One application found to be of particular interest

is American India Foundation(AIF) Financial literacy. This application consists of four different sections:

1. Modules
2. Schemes
3. Calculator
4. Quiz

Modules The modules page of the AIF application contains PowerPoint style presentations on different topics[12]. The application splits these presentations up into two categories: Important topics and How to. Important topics contains presentations on basic financial management such as how different types of income works and why saving money is important. It even goes into somewhat more complex topics such as financial planning and how inflation works. In the how to section, large amounts of practical information on various parts of finance is given. This is everything from what is needed in India to open a bank account to how to book a train ticket online. Essentially these modules visually present different vital parts of financial management, which is also something this project aims to do as described in the initial problem statement, see Section 1.6

Schemes The schemes page show different banking / pension schemes and what is needed to be eligible to join them. A way in which to spread information to the target group of this application as to what kind of special offers are potentially available to them. This is not within the scope of this project, so it does not make sense to attempt to make a similar tab in the projects smartphone application. However, perhaps a future iteration would gain from having a similar, but localized system.

Calculator The Calculator has two options. A saving per day calculation, where the user can set how much something costs and how many days the user want to spend saving op for the item. The calculator then provides the amount the user needs to save each day. The other calculation is loan repayment, where the user can set the loan amount, interest rate and how long until it has to be repaid in months. The calculator will then provide an estimate of the monthly installment, the total to be repaid and the interest paid. This would allow a user who is not mathematically adept to compare different loans plans from how much they would have to pay in interest in total. Notably in the interview with the Brazilian waste pickers in Appendix .2, a waste picker informed us that, they used the calculator application on their smartphone. Therefore providing a calculation system for different financial applications could prove a valuable tool for the waste pickers.

Quiz The quiz page takes the slideshows within the modules page and quiz's the user on the information within the modules page. Effectively allowing the user of the application to ensure they have a good understanding of the teaching materials provided within the modules. Allowing the user to test their knowledge of the teaching materials could help the courses become better based on the statistics to be gathered from the users. Beyond that it ensures the Brazilian waste pickers gain the knowledge to a usable degree within the application. Therefore a quiz / testing system should be implemented in the smartphone application. With the AIF finance application examined, a subsidiary conclusion is made.

2.2.2 Subsidiary conclusion

Haven examined different android applications and gone into detail on the AIF finance application, some conclusions are drawn. Firstly there is not a system covering substantial amounts of the initial problem statement. However some similarities can be drawn to several applications. More specifically the AIF finance application is somewhat similar, but aimed at a different target group. From the AIF application, different important features such as quizzing and a calculator to help Brazilian waste pickers are identified. However none of the applications examined used a caching system, but rather downloads the whole application at once. Effectively this means any updates are done trough the Google play store, and this is not a suitable solution for this project. To make a smart caching system, that only works when the correct type of data is available whilst working with erratic internet connectivity, another solution to a similar problem. Therefore the Green shoots project, which focused on erratic internet connectivity is examined.

2.2.3 Green shoots project

The section will describe how the Green Shoots system is build and how Christians group helped the NGO overcome the erratic internet connection in South Africa[5].

Green Shoots provides learning content through a browser using Moodle as well. Each user have their own unique profile, which allows Green Shoots to analyze the user data, which is crucial for them and the teachers at the schools to optimize both the learning material and environment. Since the connection at each school is different Green Shoots is forced to split the schools into two categories, online schools and offline schools. The online schools are the ones with a connection good enough to use the system directly which means the user-data goes directly to the system analyzing it.

The offline school a remote server is installed at the school, meaning that Green Shoots will be able to provide materials in a reliable way, the problem with this solution is the synchronization between the local server and Green Shoots servers. This forced Green Shoots to spend a lot more resources on either driving to the

school or have a staff member of the school to establish a connection through a 3G/LTE dongle, in order to update the local database, using a TeamViewer client. One would think that the solution was to just leave 3G/LTE dongle on at all time, but the problem with this was that there are no way of automatically refilling the cards subscription with data in the SIM-cards used in those.

What Christians group created was a system utilizing a heartbeat package combined with a transmission error handler to determine when the offline school had a connection good enough to send the data or how much data can be transferred at a time. The system also automated the process of synchronizing the databases minimizing the amount of human resources needed to complete the update.

As the wish is to create a system utilizing the available time slots where the application is connected to WiFi, it could be relevant to look into the heartbeats to examine if it could provide a base for the smart caching algorithm.

Having looked at both applications and back-end system implementations, a summary of conclusions is made.

2.3 Conclusion

This section will summarize the most important conclusions made in the analysis of existing solutions.

From Section 2.2.2 it can be concluded that it might be relevant to include a version of a loan and savings calculator in the application. It is also concluded that most applications within the topic uses the Google play, meaning that updates to specific courses will result in an update for every one. In Section 2.2.3 it is concluded that it might be viable to use heartbeat packages for the initial communication between the smart caching server and the smartphone application.

With this analysis complete, the more technical aspects of implementing a smart caching system can be examined.

3 Technical analysis

This chapter will examine how to build the different parts of the system as well as the packages to use when building the system. Firstly the theory behind a Moodle scraper is explored including methodology and interfaces. With the theoretical aspect of a Moodle scraper covered, the way in which to approach developing a smartphone application is examined. As the smartphone application must communicate with a database, the way in which to build a database system will also be covered. Lastly, due to the erratic internet connectivity, the theory behind creating a way in which to cache data onto the application in a smart way is examined.

3.1 Moodle scraper

This section will examine different ways scraping is and what to consider while building a scraping system. For example how the scraping will be done, as there is several way of doing this. As the wish is to deliver courses that fits the needs off the waste pickers the ability to update the courses is crucial. As these courses will be updated continually, a system to keep track whether the newest version is scraped is needed, this will be done using the HTTP ETag which is explored in Section 3.1.1.

One way of scraping a website is using the websites build in Application Programming Interface(API). An API is not the same as a remote server but rather a part of the software that allow other systems to interact with it. When sending a request to an API you will get requested data as a file format rather that a rendered website. As the response you will be getting is in a predefined format, often JSON or XML, it is much easier to parse out the data needed, as many APIs also allow the user to specific what data and the amount of data needed.

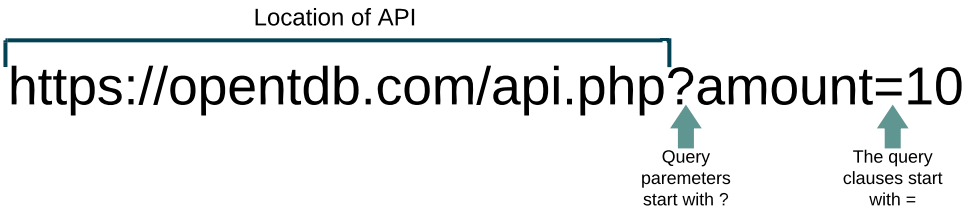


Figure 3.1: An example on how to use Open Trivia DBs API

An example of an API could be Open Trivia DB[7], their API allow users to

request a specific amount of trivia questions, on Figure 3.1 it is shown how to send a request asking for 10 questions. In this example the API only requires one query parameter which is the amount of questions, but more can be added by simply adding a & followed by the next query, in this case it could be for example a category. The query parameters can be found in the documentation for the specific API. The example on Figure 3.1 yield the result seen in Code-block 3.1.

```
{"response_code":0,
"results":[
    {"category":"Geography",
    "type":"multiple",
    "difficulty":"medium",
    "question":"What is the region conjoining Pakistan, India, and China with ↩
        unknown leadership called?",
    "correct_answer":"Kashmir",
    "incorrect_answers":["Andorra","Gibraltar","Quin"]}]}
```

Code-block 3.1: Example of a response from Open Trivia DB in JSON format

Moodle does also have a API, but after taking a look into how to utilize this API it was found to be very sophisticated, and without any clear description of how to use it. And the examples given in the file_API documentation[44], is minded on people wishing to build a module for Moodle. The API also requires that the location of each file is known before hand in order to be used, this could be done by accessing the file database on the Moodle server, however before starting to build a system able of doing this other ways of scraping is examined.

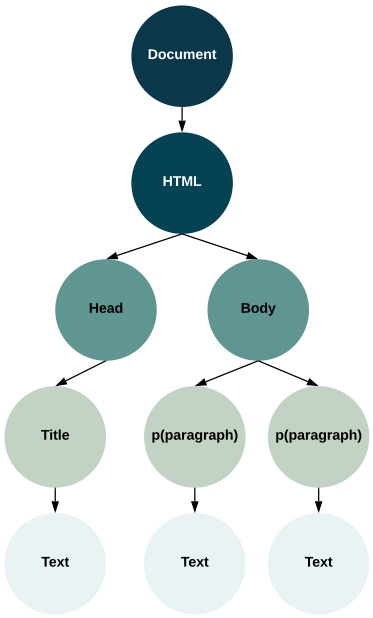
Another way of scraping is by sending HTTP requests to a server and then parsing the response either by text pattern matching or by parsing it into a Document Object Model(DOM). DOM is a language-independent API treating HTML, XHTML and XML as a tree structure, where each node represents a part of the document, an example of this can be seen on Figure 3.2.

```
<!DOCTYPE html>
<html>
<head>
  <title>My First HTML</title>
  <meta charset="UTF-8">
</head>
<body>

<p>The HTML head element contains meta data.</p>
<p>Meta data is data about the HTML document.</p>

</body>
</html>
```

(a) Example of HTML code



(b) Tree structure of HTML code seen on Figure 3.2a

Figure 3.2: Example on how HTML translates into a DOM tree

The DOM API have predefined methods for accessing and manipulating the tree and its elements. Each node can have multiple event handlers attached to them, making the content dynamic, this could for example an on-click event which allows the user to submit login data or changing the color of an element when you mouse over it. This works more efficient than parsing the response by text pattern matching, as you will be getting complete elements and not just text strings matching the input. However, text pattern matching is often used in combination with machine learning to do data mining. As the scope of this project is smart caching and not data mining, the responses will be parsed into a DOM tree which also will allow us to access the download file event attached to the content on the Moodle server that will be used. Essentially this means that if the system is able to see the content in the response from the Moodle server then it should be able to download the files by finding them in the DOM tree.

With the different methods of scraping examined both using the Moodle API and using DOM parsing seem like a suitable solutions for scraping, however one must be picked. Looking at using the two solutions the API would require us to split the system up into two smaller subsystems one handling download files based on information served by the other subsystem that should keep track of updates in the Moodle file database. Using a DOM parser would require that the system is able to send a HTTP request to the Moodle and parse the response into a DOM

tree, in which it should be able to identify content and activate the download event which is connected to the nodes.

With the two examples of scraping examined the way in which HTTP keeps track of a files concurrency is examined, to give an understanding of how to do it in the system and whether the same system can be used.

3.1.1 ETags

As explained in Section 3.1 the courses that need to be scraped will be updated continuously, therefore a system cable of tracking the concurrency of a file must be found. The HTTP have a system like this already called entity tags (ETags). This tag is not a mandatory tag, which means that how it is generated is also not specified, but is up to the individual server owner to specify the composition of it. The most common way of generating this value is by using a collision-resistant hash function and hashing either the content or the time stamp of last modification. By requesting a file from Moodle it can be seen that they implement ETags.

This will allow us to use this tag for comparing whether the caching server has the newest file download already by saving the ETags of already downloaded files in a format explained in Section 3.1.2, and comparing the ETag of the header to this database. Section 3.1.2 will also explain the format in which the files are stored.

3.1.2 File database

As described in Section 5.1 one of the priorities in this project is to define the interfaces between the different subsystems. This will allow people to exchange subsystems for a system that the prefer, like exchanging Moodle for Blackboard or another LMS. As mentioned in Section 3.1.1, this section will explain in which way the files and ETags will be stored.

To make it easy to keep track of each course the files will be stored accordingly to that and with a folder for course and a folder for each topic within the course, this structure is shown on Figure 3.3.

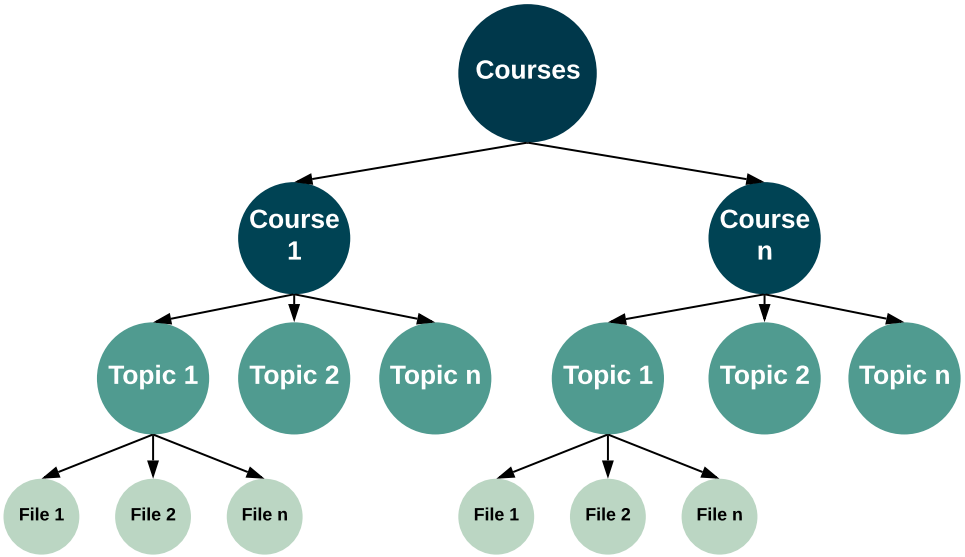


Figure 3.3: Visualization of the file system

Storing the ETags is a bit more complicated as they must be stored together with the URL they where downloaded from, as the download URL for an updated file would be the same. It is found that JSON is a suitable format for this, as it is a lightweight format saving and sending data objects. It will allow us to save each pair of information as one element in the file, making it easy the check whether it is a complete match which would mean that the file is in its newest format or just a partial which would mean that the file was updated. The files will be saved in a array of the type File when downloaded, this type contains the URL and the ETag as seen on Code-block 3.2. The two entries is of the type string to allow a combination of both letters, numbers and special characters, as these types will appear in URLs and ETags.

```
1 type File struct {
2     Href string `json:"href"`
3     Etag string `json:"etag"`
4 }
```

Code-block 3.2: File object

```
1 type Files struct {
2     Files []File `json:"files"`
3 }
```

Code-block 3.3: Object that contains all downloaded files

This element will the be saved in another struct, Files, with the the entry files

which is an array of the type File explained before, this struct can be seen at Code-block 3.3. After each entry in both Code-block 3.2 and Code-block 3.3 there is a 'json:' this tag tells the the JSON package how to organize the elements in the database file. This can be seen on Code-block 3.4 where an example with two files downloaded from AAUs Moodle is shown. Each file is within curly brackets separated by a comma.

```
{"files": [
    {"href": "https://www.moodle.aau.dk/pluginfile.php/1032158/mod_folder/content↩
        /0/coursenotes/AAUgraphics/aau_logo_new_circle.eps?forcedownload=1",
     "etag": "\"01aea7c56b349e2d215bc9250b081c763055c6fa\""},
    {"href": "https://www.moodle.aau.dk/pluginfile.php/1032158/mod_folder/content↩
        /0/coursenotes/AAUgraphics/aau_logo_new_circle.pdf?forcedownload=1",
     "etag": "\"6b9fe325c9e9fe2611795112feb2872967d60fa2\""}
     ]}
```

Code-block 3.4: Example of how the database would look if the files where downloaded from the AAU Moodle

With the formats explained an conclusion of how the system will be build can made in Section 3.1.3.

3.1.3 Subsidiary conclusion

Based on the findings in Section 3.1 it is found that it is suitable to utilize DOM parsing to scrape Moodle for files, as using the Moodle file API would require a system for interacting with the Moodle database in order to download the files, which would lead another problem as this database contains both new and out-dated files which would have us making more than two comparisons which is the amount need if DOM parsing is used. It was concluded in Section 3.1.2 that using JSON as file format for the database of downloaded files would provide an easy way of keeping track of those. For storing the scraped files on the smart caching server the tree system shown on Figure 3.3 will be used.

With the Moodle scraping system explained the next step is to examine the smartphone application.

3.2 Smartphone Application

This section will break down the smartphone application part of the initial concept model, described in Chapter 2. Examining eash part of building a smartphone application. Firstly considerations towards which operating system to develop must be used is done. Then an examination of user identification is commenced. As the intention with the smartphone application is to facilitate course content stored on a Moodle server, an analysis of what is needed to be facilitated is done. Since these Moodle courses are stored on the caching server as described in Section 3.3, an

analysis of how the two are to communicate is done. To enable the smart caching described in Section 3.3.2.3 some statistics must be collected, but to understand which statistics an analysis is done. Importantly, the smartphone application must be able to determine if it is connected to the internet. To establish how a smartphone application can do that, an analysis of how connectivity is handled on such a device is done. Lastly a summary of conclusions for the smartphone application is done, allowing for requirements to be establish. Before analyzing the smartphone application, it must be specified which operating system the application must be developed for.

3.2.1 Operating system selection

Since the goal of the project is to get this educational system to as many of the Brazilian waste pickers as possible, the operating which the most of them uses myst be found. Optimally the smartphone application would be on all operating systems used by the waste pickers phones. However in order to limit the scope of the project, only one operating system is selected. While no conclusive observations where made doing the field research as to which operating system was most widespread, a conclusion will be made from more generalized statistics. The global market share for each major smartphone operating system is therefore showcased on Figure 3.4.

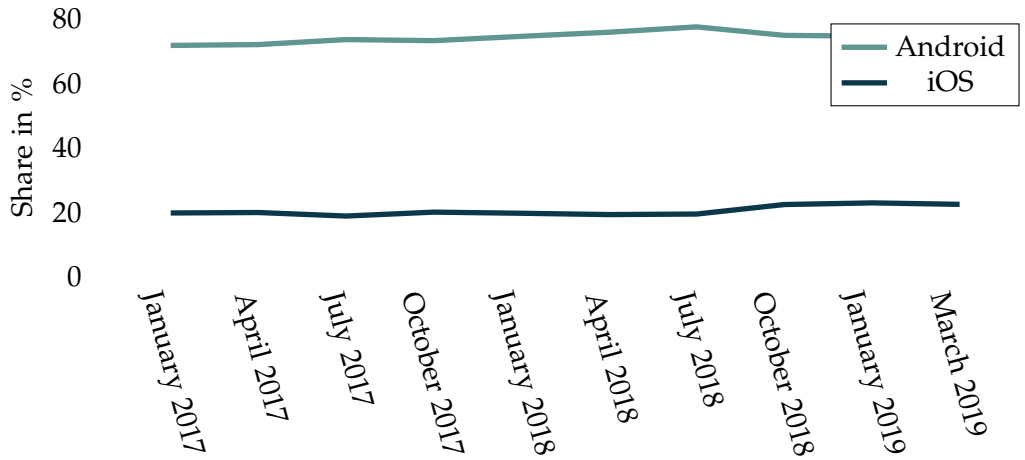


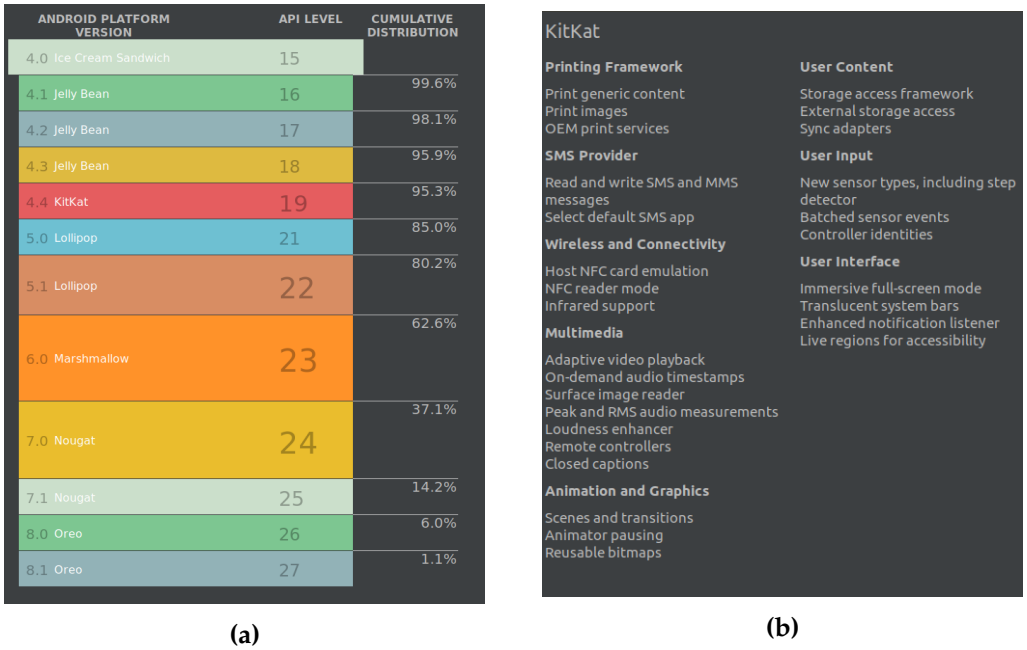
Figure 3.4: Smartphone operating systems market share in Brazil[36]

As can be seen in figure Figure 3.4, Android is the most prominent smartphone operating system by far, having over 80 % market share in Brazil. Due to this higher usage of Android smartphones and the goal of reaching as many as possible, an Android application is the most sensible choice. Of course developing for iOS also would almost ensure that almost every waste picker with a smartphone could

be reached. With Android selected as the operating system to develop for, the different APIs of the Android operating system are examined.

3.2.2 Android API level

Android continuously receive major updates in form of new releases. This could be from version 4.4 to 4.5 or 7 to 8 etc. With each major release Android updates their "API". These APIs are linearly enumerated from API 1 to the current API 29 (pie). The users phones API will depend on if the phone manufacturer releases an update for the phone. This means that older phones that are no longer supported by their manufacturer are not going to receive an update. Luckily APIs are backwards compatible, so a phone using API 28 can run applications designed for API 27 and down. Because of this, selecting an API is a balance between the features needed and how old phones the application needs to run on. To help with this selection process Android studio, the development tool for Android applications, provide both a feature overview of each major Android release and their respective features as seen on Figure 3.5a.



(a) **(b)**

Figure 3.5: Pictures from Android studio showing the support for each API level on Figure 3.5a and features of KitKat on Figure 3.5b

When choosing the API to be used in this project, the smartphones the waste pickers have must be taken into account. As observed during the field research in Brazil the waste pickers phones where decent. However, as the observations

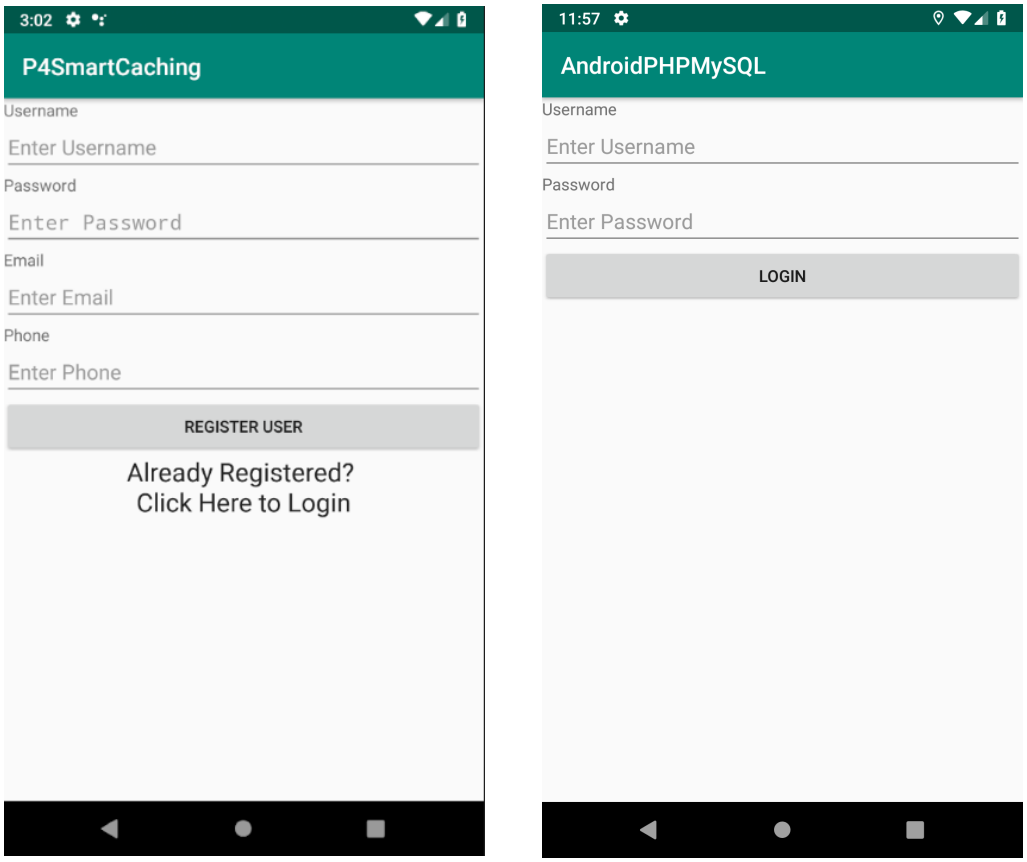
where very limited, a focus on a lower API version is done, to ensure a higher compatibility. But in order to help facilitate these Moodle courses as simply as possible a feature introduced in Android version 4.4 API 19 KitKat called "External storage access". This will allow the courses to be saved to a phones SD card. Effectively allowing it to use more potential space.

Since this API also works with 95,3% of Android phones, see Figure 3.5b, almost every phone should work with it. Having selected the API for the project, the user identification system on the Android application is examined.

3.2.3 User identification

Since the application has to have several users each with their own course progress, a user identification system is needed. While the back-end is done trough the MySQL system described in Section 3.3.1, the Android application must therefore be able to communicate with this MySQL server. As the Brazilian waste pickers are meant to create their own account, the Android application must therefore be able to send create account requests. Likewise the waste pickers must be able to login to their own account. This would allow the waste pickers to keep their progress even when switching phones.

As the focus of this project is the smart caching aspect of this system, the Android system must be built as simply as possible. For simplicity sake, the Android application could be made to just send text from text input fields. This works by giving the user some text fields to input their information and a button to "send" this information. For the sake of visualizing just how simple this implementation could be see Figure 3.6.



(a) **(b)**

Figure 3.6: Basic implementation of a login and register screen.

Logically, the user must also be able to logout in case another person wants to continue their courses on another persons phone. So a logout feature should also be implemented. Once it is possible to identify unique users, the an examination of how to implement the Moodle courses can be done.

3.2.4 Moodle courses

Since this project uses Moodle courses as the learning material, the application must be able to display said courses. However more specifically the different types of files to be displayed must be examined. While Moodle supports a plethora of file types[29], the focus will be on having the following types of media supported:

- Text
- Audio
- Video
- quiz

The reason for these specific four is that it will allow for varied content. Especially audio and video are important to allow the waste pickers who cannot read or write to also gain from the courses. Furthermore a quiz system will allow the progress and proficiency of the waste pickers to be monitored.

To simplify the project only one file type of each category will be implemented. This means that multiple text file types are out of scope for this project, but would be a good idea for future improvements. The file types to facilitate are chosen to be:

- Pdf
- Mp3
- Mp4

An examination of how to facilitate these inside the Android application is now done.

Pdf In order to simplify the Pdf facilitation a library will be used. An examination of GitHub for available libraries, found that the "Android PdfViewer"[1] is the most commonly used implementation. Using this library allows for a super basic PDF reader with the two lines of code shown in Code-block 3.5.

```
1 pdfView=(PDFView)findViewById(R.id.filename);
2 pdfView.fromAsset("documentname.pdf").load();
```

Code-block 3.5: PDF reading using the AndroidPdfViewer library

This allows opening a pdf file from an asset folder inside an Android application.

Mp3 Playing mp3 files in Android can be done by using a built in function known as the MediaPlayer API[18]. Effectively this means that mp3 support is part of the Android operating system and implementation of same can be done with relatively few lines of code. To showcase this a sample with basic functions are showcased on Code-block 3.6.

```
1  public void play(View v) {
2      if (player == null){
3          player = MediaPlayer.create(this, R.raw.filename);
4          player.setOnCompletionListener(new MediaPlayer.OnCompletionListener() {
5              @Override
6              public void onCompletion(MediaPlayer mp) {
7                  stopPlayer();
8              }
9          });
10     }
```

```
11      player.start();
12  }
13
14  public void pause(View v) {
15      if (player != null) {
16          player.pause();
17      }
18  }item
19
20  public void stop(View v) {
21          stopPlayer();
22
23  }
24
25  private void stopPlayer(){
26          if (player != null) {
27              player.release();
28              player = null;
29          }
30  }
31
32  @Override
33  protected void onStop() {
34      super.onStop();
35      stopPlayer();
```

Code-block 3.6: Basic media playback using the MediaPlayer API [10]

Just as with the pdf reading, the file is selected from a folder, in the case on Code-block 3.6 the file *filename* is chosen from the folder *raw* and a new player is created on line 3. The actual media playback is done trough the build in MediaPlayer API. In order to improve the experience three different functions are implemented play, pause and stop. Each of these functions can be called via a button and allow the user to control the playback. Of note is that the media playback take up considerable amount of resources. If for example the code just created new instance of a MediaPlayer every time a new file is played without closing it after, it would take quite a bit of resources and impact the phones on screen time. To ensure that the Android application does not fall into this issue, the stopPlayer function is automatically called at the end of every media playback. This means that if the user listens to the audio until the end, it will automatically end that instance of the MediaPlayer.

Mp4 Android has a build in VideoView class that can be implemented to display video. Using this VideoView class, all the natively supported video file types are supported, including mp4 files. The implementation of VideoView can be seen on Code-block 3.7.

```
1  VideoView videoView = findViewById(R.id.video_view);
2  String videoPath = "android.resource://" + getPackageName() + "/" + R.raw.↩
       filename;
```

```
Uri uri = Uri.parse(videoPath);
videoView.setVideoURI(uri);

MediaController mediaController = new MediaController(this);
videoView.setMediaController(mediaController);
mediaController.setAnchorView(videoView);
```

Code-block 3.7: Video playback using the VideoView package [19]

In this code example a VideoView is creating in the layout page and given the id video_view. This id is then found in the main Java. From there a string is made to the video path which in this case is a video with the name filename within the folder raw. This file is then parsed. Finally it is set to be played via the setAnchorView command.

3.2.5 Data logging

As each end users smartphone application is supposed to cache only the courses and topics relevant to them, some data must be gathered. The different data points needed will therefore be examined. Firstly the course results data points and their potential usage within the project is explored. However not only the results of the course are relevant as the surrounding data, such as time between course results must also be tracked. Therefore the more general usage statistics data points and their respective usage is also explored.

3.2.5.1 Course results

As the Brazilian waste pickers complete parts of the course, this completion must be logged. This will allow the smart caching system to gain an overview of each users progress in the courses. Since the courses consist of mostly text video and audio completion could be set to:

- text: When scrolling down to the button of a text document
- audio: When reaching the end of the audio
- video: When reaching the end of the video

While this doesn't mean that a user necessarily understood or even paid attention to the different files, it still shows a sort of completion. Of course to ensure that the user has actually gained knowledge from the courses, as well as just paying attention instead of scrolling trough it, a test system should be implemented. However, this test system is out of the scope of this project. So while understanding this limitation of this data, it will still provide an estimate of material the users gone trough. By gather this data on the progress of the user, the smart caching algorithm can understand the current progress of the user. By attaching timestamps to this data, it also becomes possible to do calculations on the average amount of time between progress. This average can be used in the smart caching algorithm

as described in Section 3.3.2. Using this data point for the course results will therefore provide an indication of the users progress as well as the speed at which they progress. However more data must be gathered for the smart caching system to be effective.

3.2.5.2 Usage statistics

One of these data points is when and for how long the phone is connected to Wi-Fi. By collecting data on the Wi-Fi connection times and duration, the algorithm can begin factoring this in. If a user is rarely connected to Wi-Fi, then logically bigger amounts of data is needed to be cached to the smartphone, compared to a user who is frequently connected to Wi-Fi. As to how specifically this will affect the smart caching algorithm, is out of scope for this project, however a very important step for further development. Consequently, since both the data needs to be stored on when the phone is connected to Wi-Fi as well as course downloads being exclusively trough Wi-Fi, Internet connectivity inside of Android must be examined.

3.2.6 Internet connectivity

As described in Section 1.3.2, the Brazilian waste pickers do not have much data on their cell phone plans. Therefore if a course with several videos needs to be downloaded, it must be done over Wi-Fi. The smartphone application must consequently be able to determine if it is connected via Wi-Fi or cellular data. In order to do so the ConnectivityManager build into Android can be used. An example showcasing how the ConnectivityManager is shown on Code-block 3.8.

```
1  ConnectivityManager connMgr =
2          (ConnectivityManager) getSystemService(Context.CONNECTIVITY_SERVICE);
3  boolean isWifiConn = false;
4  boolean isMobileConn = false;
5  for (Network network : connMgr.getAllNetworks()) {
6      NetworkInfo networkInfo = connMgr.getNetworkInfo(network);
7      if (networkInfo.getType() == ConnectivityManager.TYPE_WIFI) {
8          isWifiConn |= networkInfo.isConnected();
9      }
10     if (networkInfo.getType() == ConnectivityManager.TYPE_MOBILE) {
11         isMobileConn |= networkInfo.isConnected();
12     }
13 }
14 Log.d(DEBUG_TAG, "Wifi connected: " + isWifiConn);
15 Log.d(DEBUG_TAG, "Mobile connected: " + isMobileConn);
```

Code-block 3.8: Broadcast Receiver used to see if Wi-Fi is enabled [13]

This code creates two booleans defining whether or not Wi-Fi or mobile data is connected. This means that code needed to run exclusively when Wi-Fi or mobile

data is on can look at these booleans. Effectively this could be used to ensure that the downloads run only when Wi-Fi is connected. Furthermore it can be setup to periodically check for Wi-Fi and functions can be setup to run when Wi-Fi is found to be connected. This could be starting the whole smart caching function for example. When the Android phone has established that it is on Wi-Fi it can then begin communicating with the server.

3.2.7 Communication

For the Android to commence the smart caching sequence, it must first ensure not to use the users data-plan but only Wi-Fi as described in Section 1.3.2. Once this is done, the Android application can then begin communicating with the server. This gives a one to many relationship between the server and the Android users. Furthermore since the communication process is started by the Android application and not the caching server, this is a client-server relationship. While the algorithmic calculations are to be done inside the caching server in order to minimize the impact on the smartphones performance, the entire communication loop starts at the smartphone.

The communication itself can be done trough the HTTP protocol. This allows for both communication and user identification, as well as file transfers between the server and the smartphone application.

In Android HTTP requests can be done via a library called "volley" [20]. A simple string request within volley can be seen on Code-block 3.9.

```
1 StringRequest stringRequest = new StringRequest(Request.Method.GET, url,
```

Code-block 3.9: Volley string request

This sends a simple string HTTP GET request to a defined "url". For this to be usable, the response from the server must be read. Therefore a "Response.Listener" is setup which records the string received when the server responds to the Android application. Then it is up to the user to define what to do with this response. Beyond that Android also requires that an "Response.ErrorListener" is setup. This will then catch if an error occurs doing to the server response, such as no response, a responds using user defined illegal characters etc. An example of a "Response.ErrorListener" can be seen on Code-block 3.10

```
1 new Response.Listener<String>() {
2                     @Override
3                     public void onResponse(String response) {
4                       //Do something with the response.
5                     },
6 new Response.ErrorListener() {
7           @Override
```

```
8             public void onErrorResponse(VolleyError error) {
9                 //Do something with the error.
10            }
```

Code-block 3.10: Volley Response and error listener

How this StringRequest system is implemented will be explained in greater detail in Section 5.5. However this forms the basis for communication between the smartphone and the caching server.

3.2.8 Subsidiary conclusion

As part of the larger initial concept model, the smartphone application forms the end-user system for the Brazilian waste pickers. Firstly it was found that Android should be used due to the much larger market share in Brazil. While creating both an iOS and Android application would be optimal, due to time constraints, only one is chosen. After selecting Android as the operating system, the API level is chosen. Due to external storage access being added in API 19 (Android 4,4 kitkat) and it covering almost 96% of users, it is chosen. As each user must be able to have their own individual progress tracked, a user management system is needed. This system is chosen to be MySQL and the features needed are that a user must be able to both create an account as well as login directly from the application. The main feature of the Android application for the Brazilian waste picker, is the Moodle course facilitation.

To further specify which types of data is to be supported within the application, a selection process is done leading to pdf, mp3 and mp4 being chosen to be supported. Logically, more supported file types would be optimal, but these are selected to limit the scope of the project. It is also found that quizzes / tests has the potential the greatly help the end-user and ensure the people overseeing the courses that the waste pickers are gaining an understanding of the teaching materials. However this is chosen, not to be within the scope of the project, as it seems a larger task, and time is limited. A way of facilitating each type of supported file types is then presented.

With the file types facilitated an examination of data logging is given. This examination concludes that the users progress within the courses must be monitored in order to allow for the smart caching. It is also found that the times in which the user is connected to Wi-Fi as well as the duration for same should be logged. Then a way in which to check this is presented. Lastly the communication between the Android application and the caching server is examined and a Android specific code example is given.

With this technical examination complete, the basis on which the Android application is built upon is created. As this application has to communicate with the caching server, the way in which that side of the communication is done, must be

examined. Therefore an analysis of the caching server is commenced.

3.3 Caching server

This section will analyze the different aspects of the caching server used in this project. Firstly a system able to handle the databases needed throughout the system is found and justified in Section 3.3.1. Secondly a look into the necessities for a smart caching system to function is taken in Section 3.3.2. Including which data is necessary to calculate how much material should be sent to the user and when it makes sense to transfer this material.

3.3.1 Databases

This section will examine how to create a system cable of storing the data needed in multiple cases. These cases being user information used for authentication and smart caching, but also data about available courses and files within these. As multiple parts of the system might need access to the data simultaneously, it would not make sense to store it in JSON like it is done in the database of downloaded files, described in Section 3.1.2. Instead MySQL and PostgreSQL is considered as both system supports multiple users at once and allows for creation of multiple tables within the same database.

However, the final result ended up being MySQL as it is easier and less time consuming to setup[3]. With the platform for saving data defined the data which need to be stored on the is examined.

In Section 3.2.3 it is concluded that a table of users is need in order to both identify an keep track of the their progress. The most important thing is to be able to identify each user, as the information about their progress first becomes relevant as more courses is added. The identification can be done by creating a table in the database called *users* this table will contain five columns, containing following values id, username, password, email and phone so that i looks like Table 3.1.

id	username	password	email	phone
1	user1	f249ff8dcd6675e9a	some@mail.br	54854548
2	user2	f249ff8dcd6675e9a	some@mail.bk	54854538
3	user3	f249ff8dcd6675e9a	some@mail.eu	54854528
4	user4	f249ff8dcd6675e9a	some@fail.br	54854518

Table 3.1: Users table on MySQL server

These values will allow the smartphone application to send request to the server asking whether the information entered is connected to a valid user. The table

containing the available files will be described later on in Section 5.2.2.

As the wish is to be able to track a users progress in the courses another table might be created later on containing relevant information about this. The same goes for the information relevant for the smart caching examined in Section 3.3.2. These tables can then be joined so that identifiers from the users table can be used in other tables. For example a table used in smart caching could contain the rows: username, last connect, last sync, average sync amount, users row can then be join with the username row in the users table. With MySQL found to be a suiting solution for handling our databases the smart caching system is examined.

3.3.2 Smart caching

This section will be examining the aspects of building a caching system like which statistics is needed and algorithms for calculating how much data should be sent. The first thing that will be examined is statistics needed to be able to build the algorithms. The statistics will be split up into two categories, network statistics and user statistics.

3.3.2.1 Network statistics

This section will explore the statistics that can be logged within network traffic. The first thing that might be relevant is timestamps for when the application tries to connect to the server, one case could be that the application is set to send a heartbeat with a specific interval while connected to a Wi-Fi signaling that is connected and ready to receive data. This will give the server the ability to calculate an average time for an users connection, which can be used to calculate how much data can be send.

Another thing that can be useful to calculate how much data that should be possible to transfer is the package loss. This information will be available is the data is transferred through a TCP connection counting the amount of retransmissions.

How these things will affect the caching is evaluated in Section 3.3.2.3, but before a model for the smart caching can be created the user statistics is considered.

3.3.2.2 User statistics

This section will examine the user statistics that could be useful to log in order to calculate the amount of course material needed to be transferred to make sure that the user always have enough content to not run out before the next synchronization.

The first thing that should be tracked it how much time the user spends within the application, this could be used to give an idea of which parts of the application that is used the most. The next thing is the time spend within each course but also

within the materials provided. This data can be compared to an estimate given by the creator of the course giving an idea whether they have seen the whole video or listened to the podcast. Information on how the user performed in the tests in the courses should also be taken into account as this will give an indication on whether the materials matches the level of the waste pickers and therefore also give an idea of how fast they can go through it.

With an idea of the time spend within the application and the time spend on the materials this can be sent to the caching server and used to calculate how much material that should be downloaded.

3.3.2.3 Smart caching model

This section will look at how the data from the previous sections, network statistics and user statistics, can be used to calculate how much data that will have to be send but also how much it is possible to send for each connect.

The idea is to create an algorithm the uses some kind of a weighted average, exactly how the algorithm will work and be implemented will be described in Section 5.4. The reason for using a weighted average instead of just the average is that a users habits might change over time, this means that synchronizations will have a higher weight if they are newer, as they will give a clearer picture of when the next synchronization will be. So that can be taken into account when deciding how much data needs to be transferred. However, the connection must also be taken into account as it is important that all the data send is actually received, so if the package loss it to high then it is better to slow the transmission down. Another thing that must be taken into account is file sizes, meaning that the algorithm should be able to calculate when it is a good idea to send larger files based on stats from recent synchronizations.

As mentioned in Section 3.3.2.2 the amount must also be calculated based on how active the user are, meaning that when the application contacts the server, the information described is transferred. This gives us a smart caching model like the one on Figure 3.7.

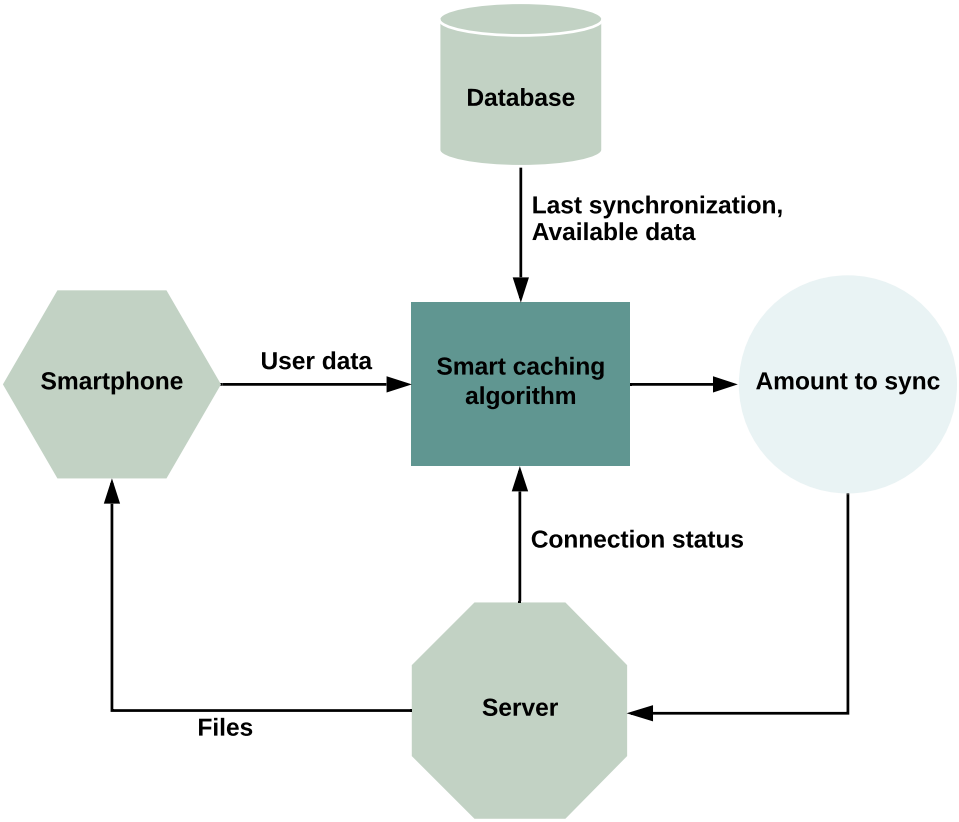


Figure 3.7: Smart caching model

The smartphone, database and server feeds the algorithm with data, which then calculate the amount that should be send. The server then starts to send this amount to the smartphone, while doing this, the connection status is send to the algorithm for it to calculate whether it is possible to reach the initial target.

With the initial smart caching concept explained an conclusion on the caching server can be made.

3.3.3 Subsidiary conclusion

This section will summarize the features found to be relevant for our caching server to work. In Section 3.3.1 it is found that system cable of handling multiple users at the same time is needed. Both in handling multiple connections at the same time but also having multiple users. This system should also be cable of saving more data regarding each user in the same database so that the information is easily accessed by the smart caching algorithm. The system should also be able to run the Moodle scraper with set interval to make sure that all the files exist in their newest version on the server and to know how much data each course contains.

In order to communicate with the smartphone the server should be able to handle incoming requests and write responses.

How all this is done will be examined in Section 5.4. But before moving on to the implementation chapter a requirements specification is done, summarizing and testing all requirements found throughout this chapter.

4 Requirement specification

This chapter will summarize the conclusions gathered trough field research, initial concept models and a technical analysis. With the conclusions summarized the requirements specification methodology is explained so that the format of the requirement specification is clear and understandable. An example of testing a requirement in the requirement specification is then given. With the methodology explained, the requirements is presented in a list from highest to lowest importance. With the final requirements completed the conclusion of the requirements is given. This conclusion sets the scope of the project and its development while drawing clear and concise lines on what is within scope.

As this project has potential to quickly become to large scaled for two fourth semester students, requirements that are out of scope of this demarcation are briefly discussed, in order to help future development of the system. However before an overall conclusion on the requirements specification is completed, it is necessary to recall the conclusions made in the problem and technical analyses.

4.1 Summary of conclusions

When building a requirements specification, summarizing the conclusions made in previous chapters will ensure that the requirements are generated on the basis of what have been concluded through out problem and technical analyses. To ensure that every requirement have a reason to exist.

In Section 1.3.1 the waste picker Maria tells that the way she uses her smartphone is by sending audio clips instead of writing messages, which means that the learning materials offered should include either video or audio, to allow illiterate to also gain from the courses.

It is also found out that the waste pickers does not have a lot of money to work with. In the context of money the Brazilian phone plans were examined in Section 1.3.2, which showed that subscriptions with "a lot" of data where quite expensive considering their salary. The subscriptions that the waste pickers might only have around 500MB of data which they might not want to spend on downloading courses. Considering the last conclusion that the learning materials probably should be video or audio the 500MB data would also be used up quite fast.

Section 1.5 describes that the application is developed for SLU to use in providing courses to the waste pickers. In order for them to check whether the waste pickers actually participate in the courses or just joined to get the extra R$ 360,

progress tracking should be implemented.

To understand the market for the proposed solution, some existing solutions are examined and concluded upon in Section 2.2.2. The conclusion is that it might be relevant to include a calculator similar to the one in the AIF finance application. Section 2.2.2 also concludes that most of the applications uses static courses, which mean that every time a course is updated this need to be done through Google play store. This way also means that the application contains all available courses at all times taking up a lot of space, which the waste pickers phones might now have as they are older generations. This means that a system with the ability to transfer courses when needed must be developed.

With all conclusions summarized the requirements can be generated after a short description of the methodology.

4.2 Requirement specification

The requirements will be generated and tested using the method described in [24, p. 98]. By the methodology being transparent, a better insight can be gained into how the demarcation came to its conclusion in terms of requirements. To showcase this methodology an example of a requirement generation is given.

One of the conclusion found in Section 4.1, was that the Brazilian waste pickers did not have much data on their smartphone plans. Therefore creating a system that detects when their smartphone is connected to Wi-Fi would help ensure that this system does not waste their data. Therefore the requirement listed is:

The smartphone application must be able to determine if it is connected to Wi-Fi

The requirements specification is presented prioritized from most to least important. However before the requirements can be listed, some clarification is needed. In order to avoid having an excessive amount of requirements the following terms are used the in the final requirements specification:

Stats: The different statistics gathered are described in greater detail in Section 3.2.5.

Content: The content facilitated in Moodle courses are described in greater detail in Section 3.1.

As the demarcation splits this system into several subsystems, logically the requirements are listed for each subsystem. The subsystems are listed in the order shown on the initial concept model in Figure 2.1.

4.2.1 Requirements moodle scraper

1. Find content on a Moodle server.
2. Download files from Moodle.
3. Save files in the format described in Section 3.1.
4. Find ETags in a HTTP-response.
5. Compare ETags to local database.

4.2.2 Requirements caching server

1. Receive requests.
2. Write responses.
3. Access local files.
4. Save stats from application.
5. Run the Moodle scraper at a set interval.
6. Perform CRUD on users, see Section 3.2.3.
7. Determine the amount of data to be sent.
8. Handle several users simultaneously.
9. User password recovery.
10. Display stats graphically.

4.2.3 Requirements smartphone application

1. Facilitate course content.
2. Determine if it is connected to Wi-Fi.
3. Write requests as described in Section 3.2.
4. Read responses as described in Section 3.2.
5. Facilitate a create user and login system.
6. Download files from caching server.
7. Gather stats as described in Section 3.2.5.
8. Must handle large files such as video.
9. Be able to playback mp4 and mp3 files.
10. Have a savings and loan calculator similar to the AIF finance application described in Section 2.2.1.2.

4.3 Partial conclusion

With the requirements specified, a important observation is that these systems rely on each other for the entirety of the system to function. This means that the different subsystems must all be made for this system to function. While the requirements for each subsystem is listed from most to least important, the subsystems

are not. These requirements will however give the project the ability to move into the implementation of these subsystems.

5 Implementation

This chapter documents the implementation of the system based on the requirements generated in the requirements specification. Firstly the programming philosophy used during programming is discussed. Having established ground rules for programming, the implementation will be listed in the order shown in the initial concept model. Effectively the implementation will be presented in the following order:

1. Moodle scraping
2. Database system
3. Caching algorithm
4. Android application

Firstly the programming philosophy used throughout development is discussed.

5.1 Programming philosophy

Programming philosophy in this context is the intention to make this system modifiable and simplify future development. Including making clear interfaces between the different subsystems effectively making it modular. Additionally the philosophy becomes important when it comes to which extend the code documentation is done. Therefore both of these points will be explained and the reasons behind the choices made in this project. Firstly the interfaces will be explained.

5.1.1 Interfaces

The interfaces linked between different parts of the project have already been outlined in the initial concept model as shown on Figure 5.1. Essentially by making the data / communication formats within the "arrows" clear and understandable changing it for future development in order to simplify it. This modularity reflects the way in which both the initial concept model as well as the implementation is split into different subsystems. With the Moodle scraper communicating with the caching server and the caching server communicating with the Android application.

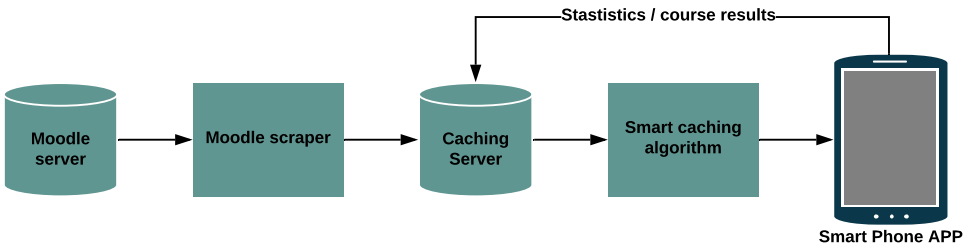


Figure 5.1: The initial concept for the overall system

For the Moodle scraper the interface would then be how it stores both the files within the caching server as well as the file-data used for the Android application, which is stored in a MySQL database. The Android application communicates with caching server using HTTP requests as described in Section 5.3. Effectively making the interface the PHP-scripts. The smart caching is theoretical at this point, therefore the theoretical interfaces are described in Section 5.4. All of these interfaces are described in more detail in their respective implementation section. With the interfaces philosophy explained how the code will documented is examined.

5.1.2 Code documentation

While the interfaces are to be clearly defined as showcased in Figure 5.1, the extend in which the code is documented must also be clearly defined. In order to document the code two things are done.

Firstly this demarcation, gives and overview and background theory for the modules being created. Furthermore it also explains in detail the most central functions, and the decisions leading to their implementation. This means that all important functions are going to be documented in great detail, to help the reader or future development gain a good understanding of what is made. In extension, future development suggestions are also given in this section, with all relevant summarized in the conclusion. To assist with this, the code is also commented within the code itself. With these suggestions alongside the requirement specification, and the comments within the code, future development is greatly simplified.

5.2 Moodle scraper

This section will document how the Moodle scraper is implemented based on the considerations discussed in Section 3.1, and requirements established based on these. The requirements include finding content on a Moodle server and downloading it, see Section 4.2.1 for a complete list of requirements. This section will start out by examining how the part finding courses and content is build in Section 5.2.1, followed by Section 5.2.2, which will explained how the files is down-

loaded and stored on the caching server, including how it is verified that a file is concurrent.

5.2.1 Scraping

This section will document the functions used for scraping the Moodle website. The site is scraped using DOM parsing, which is explained in Section 3.1, this is done by using the package for Golang called Soup[27]. Soup contains functions for parsing a HTTP response into a DOM tree and searching it for given arguments. The easiest way to find these arguments is by inspecting the site in a web-browser by right clicking and clicking inspect, this will show you exactly where the element can be found in the DOM tree.

Firstly, it is needed find the available courses on the site, these is found by using the function *FindCourses*, which is seen on Code-block 5.1.

```
1  func FindCourses(baseURL string, client *http.Client) {
2      resp, err := soup.GetWithClient(baseURL, client)
3      if err != nil {
4          log.Fatalln(err)
5      }
6      parsed := soup.HTMLParse(resp)
7      courselinks := parsed.FindAll("h4")
8      for i := range courselinks {
9          links := courselinks[i].Find("a")
10         link := links.Attrs()["href"]
11         name := sanitizeCourseName(links.Text())
12         if findCourse(name) == true {
13         } else {
14             courses.Courses = append(courses.Courses, Course{
15                 Href: link,
16                 Name: name,
17             })
18         }
19     }
20     findTopics(client)
21 }
```

Code-block 5.1: FindCourses function

FindCourses takes two inputs a string and pointer to a HTTP client, the string is the web address for the Moodle site and the HTTP client is a client that is defined in the main package, see Code-block 1 in Appendix .3.1, this allows logging into Moodle.

Firstly, *FindCourses* sends a GET request to the URL. This is then parsed into a DOM tree on line 6 returning a pointer to the start node. Then the function *FindAll*, from the Soup-package, is used to search for the HTML-tag **h4**, this will then return an array of pointers to nodes containing this HTML-tag.

This array is the looped over and the link to and name of each course is then found, this information is compared to the database of courses. If it is a new course

it is added to the list otherwise it just continues. The last thing *FindCourses* does is to call the function *findTopics*, but before the is examined the course database is examined.

The database consists of four structs, Courses, Course, Topic and Resource, see Figure 5.2a. Courses contains an array of the type Course, the type Course is the on holding relevant information about each course on Moodle this information is the name and link for each course, but also an array of the type Topic which contains all the topics within the course.

The type Topic contains the name of each topic and an array contents which is of the type Resource each instance of this type contains the type, name and link to each resource in the topic. This data structure will be described in more detail in Section 5.2.2.

With this structure explained the function *findTopics* is documented. This function loops over each course, sends a request to the link and scrapes for topic names and files, the function is seen on Code-block 5.2.

```
1  func findTopics(client *http.Client) {
2      for j := range courses.Courses {
3          resp, err := soup.GetWithClient(courses.Courses[j].Href, client)
4          if err != nil {
5              log.Fatalln(err)
6          }
7          parsed := soup.HTMLParse(resp)
8          list := parsed.FindAll("li", "class", "main")
9          for i := range list {
10             topic := list[i].Find("span", "class", "sectionname").Text()
11             clean := sanitizeTopicName(topic)
12             courses.Courses[j].Topics = append(courses.Courses[j].Topics, Topic{↩
                   Name: clean})
13             Content := list[i].FindAll("div", "class", "content")
14             for k := range Content {
15                 file := Content[k].FindAll("li", "class", "resource")
16                 for f := range file {
17                     resource := file[f].FindAll("a")
18                     for l := range resource {
19                         link := resource[l].Attrs()["href"]
20                         r, _ := client.Get(link)
21                         d := r.Header["Content-Disposition"]
22                         re := regexp.MustCompile(`filename=\"(?P<Name>.+)\"`)
23                         filename := re.FindStringSubmatch(strings.Join(d, ""))[1]
24                         courses.Courses[j].Topics[i].Content = append(courses.↩
                               Courses[j].Topics[i].Content, Resource{Href: link, ↩
                               Modtype: "resource", Name: filename})
25                     }
26                 }
27                 page := Content[k].FindAll("li", "class", "page")
28                 for f := range page {
29                     resource := page[f].FindAll("a")
30                     for l := range resource {
31                         link := resource[l].Attrs()["href"]
32                         name := resource[l].Find("span", "class", "instancename")↩
```

```
                                .Text()
33                          filename := sanitizeTopicName(name) + ".txt"
34                          courses.Courses[j].Topics[i].Content = append(courses.↩
                                Courses[j].Topics[i].Content, Resource{Href: link, ↩
                                Modtype: "page", Name: filename})
35                      }
36                  }
37              }
38          }
39      }
40  }
```

Code-block 5.2: findTopics function

Firstly the parsed response is searched for, the tags identified, describing the topic element. This nodes children is then searched to find the one containing the topic name on line 10, returning the text within the node by using the *Text* function. This is then sanitized so that all names follow the same naming conventions, which means no spaces and capital letters, the sanitized name is then appended to the database. After the topic name is found, the nodes children is searching for any nodes containing content. This array is lopped over to find nodes with the class **resource** and then nodes with the class **page**. Subsequently, each of these nodes is added to the database.

With all the wished materials found and added to a data structure how the files is downloaded and stored on the caching server is examined.

5.2.2 Downloading and storing files

This section will examine how downloading and storing of files is handled. Documenting how the file structure described in Section 3.1.2 is achieved and how a list of files available within each course is created.

As explained in Section 5.2.1 all the courses, topics and files that is found will be added to the data structure seen on Figure 5.2.

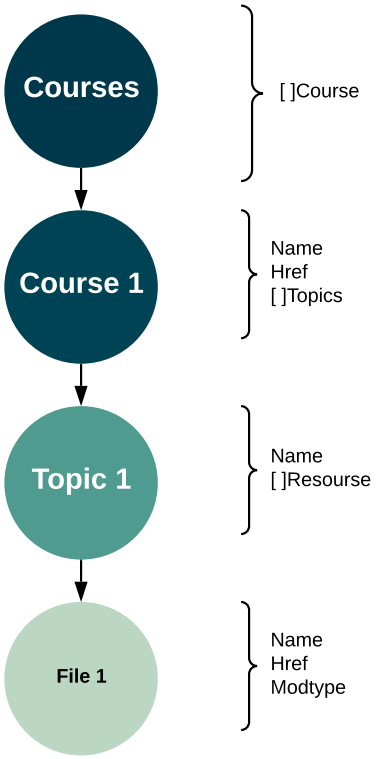
```go
//Courses contains an array of courses
type Courses struct {
    Courses []Course `json:"courses"`
}

//Course contains the name, link and topics of each course on moodle
type Course struct {
    Name   string  `json:"name"`
    Href   string  `json:"href"`
    Topics []Topic `json:"topics"`
}

//Topic contains the name and the Contents of a topic within a course
type Topic struct {
    Name    string     `json:"topic"`
    Content []Resource `json:"resource"`
}

//Resource is a description of each file
type Resource struct {
    Modtype string `json:"type"`
    Href    string `json:"href"`
    Name    string `json:"name"`
}
```

(a) Data structure



(b) Visualization of the Go code on Figure 5.2a

Figure 5.2: Data structure used for files found on Moodle

Every time a new course is found it us added to an instance of the type **Courses** called courses as an element in the array Courses. This gives the ability to create a folder for each course by looping over the array and creating a folder for each element. Since the name of each course is a part of the type Course of which the array is, the name for the folder is easily found, the same principle goes for the topics within the courses and the content of these.

This data structure is used for both downloading the found files and also for adding the files to the MySQL database. The MySQL database being used by the smartphone application to identify available files. Before examining this part closer the function that downloads the content is documented. The function is called *DownloadContent* and takes three inputs.

Firstly, it takes a string called outputDir which is the directory that all the files will be downloaded to. The next input is a string called database, this is not the database explained in this section, but the database containing a link to each file downloaded and the Etag from the HTTP header. The last input is a

pointer to a HTTP client for the same reason as the *FindCourses* function described in Section 5.2.1.

```
1  func DownloadContent(outputDir, Database string, client *http.Client) {
2      for i, c := range courses.Courses {
3          for j, t := range courses.Courses[i].Topics {
4              for _, k := range courses.Courses[i].Topics[j].Content {
5                  if k.Modtype == "resource" {
6                      wg.Add(1)
7                      go files.GetFile(k.Href, outputDir+"/"+c.Name+"/"+t.Name, ↩
                           Database, client, &wg)
8                  } else if k.Modtype == "page" {
9                      wg.Add(1)
10                     go files.GetPage(k.Href, outputDir+"/"+c.Name+"/"+t.Name, ↩
                           Database, client, &wg)
11                 }
12             }
13             wg.Wait()
14         }
15     }
16 }
```

Code-block 5.3: DownloadContent function

On line 2 on Code-block 5.3, the function *DownloadContent* loops over each element within Courses accessing both the index of the element and the content, the index is written to the variable **i** and the content is saved in **c**.

The loop on line 3 then loops over the topics within each course and saves the index of each to the variable **j** and contents to **t**. The last for loop then looks at the contents of each topic only saving the content to the variable **k**. The modtype of the content is then checked, if it is of the type "resource" or "page" and is downloaded using the fitting function. This download process is done using multi threading, allowing multiple files to be downloaded simultaneously.

To ensure all threads finishes before continuing the package *sync* is used. Facilitating the creation of wait groups, which waits for a collection of go routines to finish. The wait group uses three functions *Add*, *Done* and *Wait*. The function *Add* adds an int to a counter, *Done* decrements the counter by one, and *Wait* block the main tread until the counter is zero. This ensures that all the files within each topic is downloaded before continuing. The *Add* function is used on both line 6 and 9 just before starting a new go routine for either grabbing a file or a page, each of these functions take a pointer to the wait group allowing them to call *Done* when they are done.

Both functions utilize that the contents of the different elements is saved in variables. They use the variables to access the course name and topic name, which is used to place the file in the right folder. This process saves the download file in the structure described in Section 3.1.2.

With the way in which the file download is handled examined, the next step is to examine the way the list of files is communicated to the smartphone application. As the smartphone is able to communicate with a MySQL database this is used. A table called Files is created on the MySQL server, containing three columns so that it looks like Table 5.1.

Course name	Topic name	File name
test-course-1	test-topic	data.csv
test-course-1	test-topic	tasks.docx
test-course-2	test-topic	summary.txt
test-course-4	test-topic	something.pdf

Table 5.1: Files table on MySQL server

This table is then filled with the data from the data structure shown on Figure 5.2 using the function *DoTheDBThing*, shown on Figure 5.2. *DoTheDBThing* loops over the data just like *DownloadContent* does and prepares a SQL statement for each file in the database. This statement is then send to the database instance created on line 4 in Code-block 5.4.

```
1  func DoTheDBThing() {
2      data := fmt.Sprintf("%s:%s@tcp(%s:%d)/%s",
3          user, password, host, port, dbname)
4      db, err := sql.Open("mysql", data)
5      if err != nil {
6          log.Fatal(err)
7      }
8      defer db.Close()
9      for i, c := range courses.Courses {
10         for j, t := range courses.Courses[i].Topics {
11             for _, k := range courses.Courses[i].Topics[j].Content {
12                 sqlStatement := fmt.Sprintf("INSERT INTO `Files` (course, topic, ↩
                       filename) VALUES ('%s','%s', '%s');", c.Name, t.Name, k.Name)
13                 fmt.Println(sqlStatement)
14                 fmt.Println(k)
15                 err = db.QueryRow(sqlStatement).Scan()
16                 if err != nil {
17                     log.Println(err)
18                 }
19             }
20         }
21     }
22 }
```

Code-block 5.4: DoTheDBThing function

With these functions examined a conclusion is made, concluding whether the Moodle scraper fit the requirements established in Section 4.2.1.

5.2.3 Subsidiary conclusion

This section will conclude on implementation of the Moodle scraper by taking each of the requirements from Section 4.2.1 and evaluate whether it is met by the implemented code. The section will end with a list of features that are to develop towards for future development.

Starting with the first requirement, which is that the Moodle scraper should be able to find content on a Moodle server. This requirement is meet with the functions *FindCourses* and *findTopics*, however these functions does not support all the file formats that Moodle does so this can be expanded to also support these. The second requirement is that the Moodle scraper should be able to download the found content, this is also met for the file formats supported by *FindCourses* and *findTopics*. Adding more file formats will require that more functions is added handling the specific formats. In Section 5.2.2 it is explained how the scraper saves the files according to the structure described in Section 3.1.2 which is the third requirement.

The last two requirement addresses the use of ETags as concurrency verification. When a file is downloaded the HTTP header is checked whether it contains an ETag, if it does this ETag is compared to the local JSON database using the *FindFile* function. This function can be seen on line 72-86 in Appendix .3.2, this means that the last two requirements is met. With all the requirements met some points of development is established.

As previously mentioned all types off content on Moodle is not supported yet, some of the missing types is quizzes and forums, which both could improve the system greatly. Especially quizzes as it will help provide data on the course progress and understanding by the Brazilian waste pickers. Another point that could be improved is function for submitting answers from specific users in order to scrape grades. The scrapers multi threading could also be improved to make the scraper more efficient, this could starting a thread for each course as well and improving protection against race conditions. The function for adding files to the MySQL database could also be improved so that is checks if the file exists in the database already. With the Moodle scraper examined the different databases is examined.

5.3 Database implementation

This section will examine how the databases discussed in Section 3.3.1 is implemented and how the caching server and smartphone application can communicate with it in order to authenticate users and receive a list of files. The section will also give an example of all the tables within the database. Firstly the database

for user authentication is examined, hereunderwhich tables might be added if the functionality is expanded, like implementing quizzes and grades in the application. After examining all the tables handling user information the tables handling files is examined.

Before all of that a short examination into which database system and how it is implemented is taken. In Section 3.3.1 it is concluded that MySQL fits our needs, being able to handle multiple requests at the same time. MySQL also supports connected tables meaning that login information can be in one table and statistics can be in another.

Our MySQL server is installed on cloud computer rented from Amazon Web Services(AWS) together with Apache2, PHP and PHPMyAdmin. Apache2 is a HTTP server, which means that it will be able to send HTTP requests to the server these will then be processed by our PHP-scripts. These will in turn be able to communicate with the database and create an API Facilitating authentication: create and delete users in the database, by sending a HTTP request. PHPMyAdmin will allow the management of the databases from a graphical interface instead of doing it using the command line. With the basics in place an examination of the user tables is done.

5.3.1 User tables

This section documents how information is stored in the table and how it is accessed through the PHP-scripts. Before this can be done an overview of the user table i given.

The main table is the users table also explained in Section 3.3.1, the table contains the the columns id, username, password, email and phone, structured as seen on Table 5.2. Id is an integer which increments by one each time a new user is created, the username is a string picked by the user, the password is the sha1 hash of the password picked by the user.

id	username	password	email	phone
1	user1	f249ff8dcd6675e9a	some@mail.br	54854548
2	user2	f249ff8dcd6675e9a	some@mail.bk	54854538
3	user3	f249ff8dcd6675e9a	some@mail.eu	54854528
4	user4	f249ff8dcd6675e9a	some@fail.br	54854518

Table 5.2: Users table on MySQL server

This table can be accessed sending a POST request to the server. The two functionalities build at this stage is to register a new user and login an existing

user. These functionalities are handled by the PHP-scripts *userLogin.php* and *registerUser.php*, which will be examined hereunder.

Starting with *userLogin.php*, this PHP-script starts by including the script *DbOperations*, see Appendix .4.3, this script contains all the functions for preparing SQL statements based on the inputs given. On line 4, in Code-block 5.5, the script ensures that the HTTP method used is POST, after that it check if a username and password is set in the request, if it is then the then is is passed to the login function within *DbOperations*.

```
1  <?php
2  require_once '../includes/DbOperations.php';
3  $response = array();
4  if ($_SERVER['REQUEST_METHOD'] == 'POST'){
5      if (isset($_POST['username']) and isset($_POST['password'])){
6          $db = new DbOperations();
7          if ($db->userLogin($_POST['username'], $_POST['password'])){
8              $user = $db->getUserByUsername($_POST['username']);
9              $response['error'] = false;
10             $response['id']       = $user['id'];
11             $response['username'] = $user['username'];
12             $response['email']    = $user['email'];
13             $response['phone']    = $user['phone'];
14             $response['message'] = "Logged in successfully";
15         }else{
16             $response['error'] = true;
17             $response['message'] = "Invalid username or password";
18         }
19     } else {
20         $response['error'] = true;
21         $response['message'] = "Required fields are missing";
22     }
23 }
24 echo json_encode($response);
25 ?>
```

Code-block 5.5: PHP-script for logging in

After logging in it calls the function *getUserByUsername* and writes the results to the HTTP response that will be send to the smartphone application. This concludes the *userLogin.php* and a look into the *registerUser.php* can be taken.

```
1  <?php
2  require_once '../includes/DbOperations.php';
3  $response = array();
4  if ($_SERVER['REQUEST_METHOD'] == 'POST') {
5      if (isset($_POST['username']) and isset($_POST['email']) and isset($_POST['↩
       password']) and isset($_POST['phone'])) {
6          $db = new DbOperations();
7          $result = $db->createUser(
8              $_POST['username'],
9              $_POST['password'],
10             $_POST['email'],
```

```
11              $_POST['phone']);
12          if ($result == 1){
13              $response['error'] = false;
14              $response['message'] = "User registered successfully";
15          }elseif($result == 0){
16              $response['error'] = true;
17              $response['message'] = "User already exists, please choose a ↩
                    different email and username";
18          }elseif($result == 2) {
19              $response['error'] = true;
20              $response['message'] = "An error occurred, Please try again";
21          }
22      }else{
23          $response['error'] = true;
24          $response['message'] = "Required fields are missing";
25      }
26  } else {
27      $response['error'] = true;
28      $response['message'] = "Invalid Request";
29  }
30  echo json_encode($response);
31  ?>
```

Code-block 5.6: PHP-script for registering a new user

Like *userLogin.php*, *registerUser.php* starts by including *DbOperations* and checking the HTTP method of the request. Following that, it checks whether all the information that is needed is within the request. This is then passed to *createUser* from *DbOperations* and the return values is checked to see if any errors occurred.

These two functionalities are the ones that are created, but as explained in Section 3.3.1 some thoughts to how to build the rest have been made. The concept is to create another table containing synchronization data, like last synchronization and average time between synchronizations both measured in seconds, so that that the table looks like Table 5.3.

userid	last sync	average
1	1558426608	86400
2	1558424648	432000
3	1558426336	43200
4	1558433333	1209600

Table 5.3: Table containing synchronizations data

The userid column can then be linked to the id in the users table using one of the SQL *JOIN* commands depending on the set up the same goes if a table of all synchronizations is created which can be used to calculate the average in Table 5.3, then each synchronization can be linked to the user. The relation between the users table and a table containing synchronizations is illustrated on Figure 5.3.

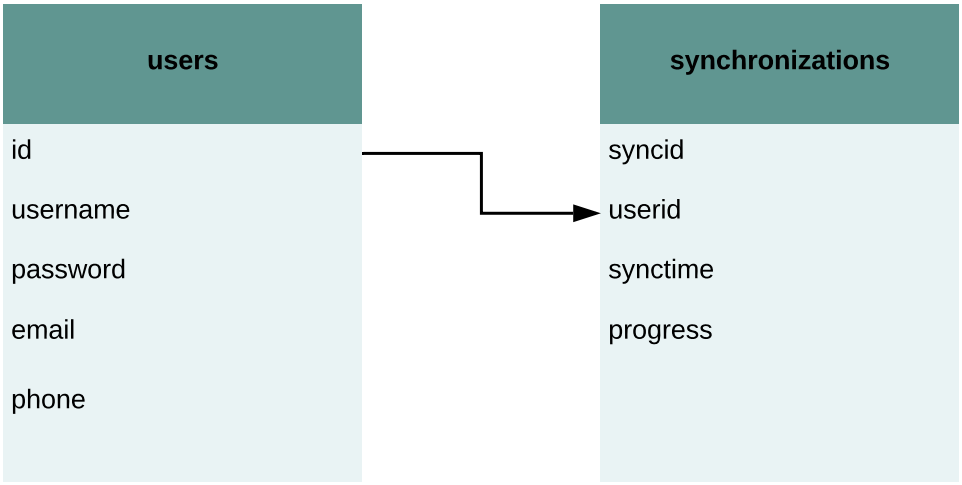


Figure 5.3: Illustration of how the databases is connected

But as previously stated, these tables are not created yet as the functionality is not implemented in the caching system. This leads into the next section, that will be documenting the tables for storing the course files.

5.3.2 Course tables

This section will examine the way that information about courses stored on the MySQL server. Specifically, how it can be extracted and thoughts about this could be improved. At the current implementation stage, only table containing information about the content on the server is the Files table, which is also described in Section 5.2.2, the table contains all the files on the server. The table is structured like Table 5.4, containing course name, topic name and a file name.

Course name	Topic name	File name
test-course-1	test-topic	data.csv
test-course-1	test-topic	tasks.docx
test-course-2	test-topic	summary.txt
test-course-4	test-topic	something.pdf

Table 5.4: Files table on MySQL server

This table can be accessed by navigating to *fetchCourses.php* this will provide output the content of the table.

```
1 <?php
2 require_once '../includes/Constans.php';
```

```
3  // Create connection
4  $con = mysqli(DB_HOST, DB_USER, DB_PASSWORD, DB_NAME);
5  // Check connection
6  if (!$con) {
7      die("Connection failed: " . mysqli_connect_error());
8  }
9
10 $sql = "SELECT course, topic, filename FROM Files";
11 $result = mysqli_query($con, $sql);
12 if (mysqli_num_rows($result) > 0) {
13     // output data of each row
14     while($row = mysqli_fetch_assoc($result)) {
15         echo "course: " . $row["course"]. " - Name: " . $row["topic"] . " - Name:↩
               " . $row["filename"]. "<br>";
16     }
17 } else {
18     echo "0 results";
19 }
20 mysqli_close($con);
21 ?>
```

Code-block 5.7: PHP-script for listing all files on the server

It starts by including *Constans.php* which is the file containing the login information for our MySQL server and the use this to open a new connection to the database. Then the SQL statement created on line 10 on Code-block 5.7 is send to the server which will return all row as nothing else is specified. Then the while loop on line 14 takes each of these and echoes them out as HTML, before the connection is closed back down. If this script should be improved it should do an user authentication before writing out the content of the table. This could be done by building it together with *DbOperations* using the login function in this file. Another thing that could be improved is they way the files is stored in the database, it might be an idea to create more table like one containing only courses. This could probably make it easier to keep track of which user is working on which course. With some improvements for the the course tables listed, a final conclusion on the database implementation can be made.

5.3.3 Subsidiary conclusion

This section will conclude on the database by looking at the database relevant requirements in Section 4.2.2 and evaluating if they are met by the implemented system. As the database is only a part of the caching server not all requirements have to be met in this section, they will also be examined in Section 5.4.1 before a final conclusion on the system and all subsystems is made in Section 5.6. This section will also summarize the improvements found to help with future development. The requirements relevant for the database system is the following.

- Receive Requests.
- Write Responses.

- Save stats from application.
- Perform CRUD on users.
- Handle several users simultaneously.
- User password recovery.

Starting from the first requirement is that the caching server should be able to receive HTTP requests, which goes hand in hand with the second requirement. Both of those are met by the the PHP-scripts *userLogin.php*, *registerUser.php* and *fetchCourses.php* as they receive a HTTP and responds as JSON or plain HTML depending on the function. These functions also take care of most of the fourth requirement that the system should be able to perform CRUD on the users. As *userLogin.php* and *registerUser.php* allow for the creation of users and reading their data from the database. However, the U and D part is not implemented yet. This also means that the requirement stating that the system should have password recovery is also not met in this version of the system.

The next requirement is that the system should be able to save stats from the application, this part is not yet implemented, as the logging system in smartphone application in not implemented yet anyways.

The last requirement is that the system should be able to handle multiple users. This is met as the database allows us for creating a table containing a lot of users, the system is also cable of handling multiple requests as the time making it a double check.

To summarize the things that should be either developed or improved, starting with the PHP-script *fetchCourses.php*. This should use the already implemented authentication system to verify that it is an actual user that access the site. Another thing that should be done is creating tables for storing the stats coming from the application. While in same take restructuring the tables containing the content information optimizing them to store information about participants as well.

With the database implementation examined the next step is to examine the smart caching implementation.

5.4 Caching

This section will be examining how much of the smart caching explained in Section 3.3 is implemented. As explained in Section 3.3 the caching server is split up into two parts, there are the databases, which implementation explained in Section 5.3, and then there are the caching algorithm and with that a server cable of sending data also comes.

This section will be based on an examination of the requirements established in Section 4.2.2. Some of the requirements are already met by the database system as explained in Section 5.3.3, like the possibility to handle HTTP traffic. However, in order to make the smart caching work probably some of this need to be modified

drastically. For example the way files is transferred should be changed so that it is the server that is in control instead of the smartphone application just grabbing a file path from a database. The server should just send data to the smartphone based on the information received as explained in Section 3.3.2.3. The initial plan was to create some kind heartbeat package, which could be send with a given interval while connected to a Wi-Fi, containing all the necessary user statistics. However, progress did not get to this stage, as there is time left after getting a basic file transfer up and running between the application and the server.

Which mean that the implementation is done with the PHP-scripts explained in Section 5.3 running on an Apache2 server. Which allow us to place the files in a specified folder on the server and generate a URL to each file which can then be downloaded by the application by sending a simple HTTP GET request to this URL. With the explanation of the small bit of the smart caching that was completed, the next step is to look at the requirements.

5.4.1 Subsidiary conclusion

This section will section will conclude on the caching implementation involving the database conclusion, see Section 5.3.3 as they should function as one in a final product.

As described both in Section 5.3.3 and Section 5.4 the two first requirements are met to some extent just not in the way original specified. The next requirement is that the caching should have access to the local files, which it does, the files however must be places in the right folder in order to be accessible from the internet. The requirement stating that the Moodle scraper should run at a set interval is not met yet. The reason for that is that it would make more sense to build it into the caching systems so that is runs whenever the server have spare resources instead of forcing a scrape while busy. As the database for storing stats from the application is not implemented yet this removes the possibility to meet the requirements:

- Save stats from application.
- Determine the amount of data to be sent.
- Display stats graphically.

As these requirements rely on the ability to access a database containing the information that it needs in order to do these things. With all the requirements to the caching server assessed in this section and Section 5.3.3 the next step is taking a look at the implementation of the smartphone application.

5.5 App

This section will document the implementation of the Android application. This includes but is not limited to:

1. Constants
2. Basic navigation
3. User identification
4. Course facilitation
5. Data collection

With the implementation of these parts of the Android application described a summary of the implementation is done, followed by showcasing the caching mechanism implemented in the project.

5.5.1 Constants

The constants used within the Android application is the link to each of PHP-scrips allowing the application to communicate with the caching server hereunder the MySQL server.

```
1 public class Constants {
2     //Defines the root URL. This is a selectable root in this projects case it is↩
          in the AWS server on /var/www/html/
3     private static final String ROOT_URL ="https://save-brazil.lanestolen.dk/↩
          android/v1/";
4
5     //Defines the different scripts that the phone calls
6     public static final String URL_REGISTER = ROOT_URL+"registerUser.php";
7     public static final String URL_LOGIN = ROOT_URL+"userLogin.php";
8     public static final String URL_GETDATA = ROOT_URL+"fetchCourses.php";
9 }
```

Code-block 5.8: The constants within the java button

These PHP-scripts are the ones explained in Section 5.3, and the usage of these constants is to help simplify code later on, when different the PHP-scripts are used. With this is mind, the way in which basic navigation is implemented within the application can be documented.

5.5.2 Basic navigation

The users need to navigate throughout the Android application as it is the end user product. The basic way in which to navigate to different screens is by moving between "activities". An activity is essentially a screen with its own functions, both being visually different for the user as well as having its own background functions.

In the current implementation of the Android application the following activities exists:

- MainActivity: The user registeration screen
- LoginActivity: The user login screen

- ProfileActivity: The main user screen
- SettingsActivity: The settings screen

These activities can be navigated by different means. The simplest of which is to simply start a new activity via a trigger such as a button click. In the *ProfileActivity* the *SettingsActivity* can for example be navigated as shown here:

```
buttonSettings.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        startActivity(new Intent(ProfileActivity.this, SettingsActivity.class));
    }
});
```

Code-block 5.9: Simple on button press navigation

In this way of navigation, the new activity is opened, however the old one remains open in the background. This means that pressing the "back" key built into android (implementation vary from phone to phone), will simply return the user to the previous activity. If a user logs out, it would be pretty bad if the user could go back into his / her account by pressing the back key. For this reason it is tested if the user is logged in. If it is found that they are not the *finish()* function is called. This simply closes the current activity.

```
if (!SharedPrefManager.getInstance(this).isLoggedIn()) {
    finish();
    startActivity(new Intent(this, LoginActivity.class));
}
```

Code-block 5.10: Simple on button press navigation where the old activity is closed.

In this example the *SharedPrefManager* is called and the script detecting if the user is logged in is called. If the user is not logged in then they are sent back to the *LoginActivity* (the login screen) and the old *ProfileActivity* is closed. This is essentially how navigation between screens and activities work within this implementation of the Android application. Of course this would not be possible without the users actually being identifiable, therefore the implementation of the user identification system is examined.

5.5.3 User identification

As shown in the requirements specification, the Android application must be able to differentiate users. This section will go into how this is implemented within the android application. Firstly the way in which the users register within the application is explained and then the way in which they login.

5.5.3.1 Registering user

The first screen a new user is presented with is the user registration screen. Within the screen the user has four text fields to input and a button to register, and a way in which to get to the login screen in case they already have a user as seen on Figure 5.4.

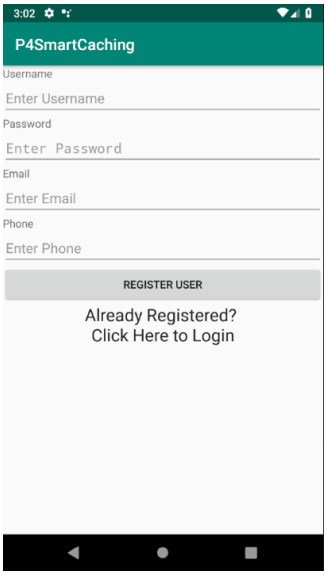


Figure 5.4: The android registeration screen

Whenever the user presses the register user button an *OnClickListener* shown on Code-block 5.11, is used.

```
1  public void onClick(View view){
2      if (view == buttonRegister)
3          registerUser();
4      }
```

Code-block 5.11: On click listener for user register

This in turn calls *registerUser()* function which is shown on Code-block 5.12.

```
1  private void registerUser(){
2      final String email = editTextEmail.getText().toString().trim();
3      final String username = editTextUsername.getText().toString().trim();
4      final String password = editTextPassword.getText().toString().trim();
5      final String phone = editTextPhone.getText().toString().trim();
6
7      progressDialog.setMessage("Registering user...");
8      progressDialog.show();
9      StringRequest stringRequest = new StringRequest(Request.Method.POST,
10             Constants.URL_REGISTER,
```

```
11              new Response.Listener<String>() {
12                  @Override
13                  public void onResponse(String response) {
14                      progressDialog.dismiss();
15
16                      try {
17                          JSONObject jsonObject = new JSONObject(response);
18
19                          Toast.makeText(getApplicationContext(), jsonObject.↩
                                getString("message"), Toast.LENGTH_LONG).show();
20
21                      } catch (JSONException e){
22                          e.printStackTrace();
23                      }
24                  }
25              },
26              new Response.ErrorListener() {
27                  @Override
28                  public void onErrorResponse(VolleyError error) {
29                      progressDialog.hide();
30                      Toast.makeText(getApplicationContext(), error.getMessage(), ↩
                            Toast.LENGTH_LONG).show();
31                  }
32              }){
33          @Override
34          protected Map<String, String> getParams() throws AuthFailureError {
35              Map<String,String> params = new HashMap<>();
36              params.put("username", username);
37              params.put("email", email);
38              params.put("password", password);
39              params.put("phone", phone);
40              return params;
41          }
42      };
43
44      RequestHandler.getInstance(this).addToRequestQueue(stringRequest);
45
46  }
```

Code-block 5.12: The user registration

Firstly, a string for all four input fields are made. Since the input fields are in text and not string formats, they must be converted. Then a *progressDialog* is used to keep the user updated on what the function is doing. In this implementation a simple print or Toast in Android would have sufficed, but the *progressDialog* can be more easily expanded.

Then a StringRequest as shown in Section 3.2.7 is used to send a POST request to the *URL_REGISTER* as defined in a constants Java file. This *URL_REGISTER* is a PHP-script and how it functions is shown in Section 5.3. For the applications purpose this PHP-script sends a response to the application. In case of an error in the script it will send a response in JSON and this response is then printed.

The Volley library also requires an error listener, which will give an error in case of lost connection. In case an error occurs, the error message is printed. If no errors occurs, then the application will proceed to insert parameters into the parameters.

Finally it is all sent to the request handler. With this the user has finally created a user and a *Toast* (a popup) will display the message received from the server in this case *"User successfully created"*. Optimally the user would then automatically be logged in, however this is not the case in the current implementation. Therefore the login screen and hereby code is used next.

RequestHandler On line 44 on Code-block 5.12, a request handler is called, which is used in order to make it simpler to send requests compared to programming a *RequestQueue* every time communication with the server is needed. The code behind that is very simple and will not be examined. What the *RequestHandler* does is startup and insert the *stringRequest* into the *RequestQueue*. It is only after it is added to the *RequestQueue* that the *stringRequest* is actually sent.

5.5.3.2 User login

The user login feature requires that the data being inputted in the Android application is compared with the MySQL database containing the users. While the user has given both a username and a phone, they will not be used for this login. This is simply to save time for the user, and requiring four fields perfectly filled out, could prove a tall order. Especially for the waste pickers with trouble reading and writing. Just as in the MainActivity register, the login sequence is started by first filling out text fields and then pressing a button with an assigned *OnClickListener*. When the user presses the login then the code show on Code-block 5.13 is executed.

```
1  private void userLogin(){
2          final String username = editTextUsername.getText().toString().trim();
3          final String password = editTextPassword.getText().toString().trim();
4
5          progressDialog.show();
6          StringRequest stringRequest = new StringRequest(
7                  Request.Method.POST,
8                  Constants.URL_LOGIN,
9                  new Response.Listener<String>() {
10                     @Override
11                     public void onResponse(String response) {
12                         progressDialog.dismiss();
13
14                         try {
15                             JSONObject obj = new JSONObject(response);
16                             if(!obj.getBoolean("error")){
17                                 SharedPrefManager.getInstance(↩
                                       getApplicationContext())
18                                         .userLogin(
19                                                 obj.getInt("id"),
20                                                 obj.getString("username"),
21                                                 obj.getString("email"),
22                                                 obj.getInt("phone"));
23
```

```
24                                  startActivity(new Intent(getApplicationContext(),↩
                                        ProfileActivity.class));
25                                  finish();
26
27                                  //After the error check code, it is then added to↩
                                        the request queue
```

Code-block 5.13: The login script in the LoginActivity

As in the user register code, the first thing is to take the input for username and password and turn them into strings. Then a POST request with the two strings are setup to the *URL_LOGIN* which once again is a PHP-script on the server explained in Section 5.3. Depending on the response from the PHP-script several things can happen. In case of a successful login, the user data (id username email and phone) are returned to the application and the user is moved to the *ProfileActivity*. As described in Section 5.5.2, the previous activity *LoginActivity* is closed due the *finish()* command. Just as with the register user code, both JSON has an error reader as well as Volley. With the user being able to both register and login, the course facilitation is the next step for the application. It should be noted this is not the complete code as all the error listener are the same. In order to actually send the string to the server the the request handler is called.

5.5.4 Shared Preferences

In order to simplify the code used in different activities a *SharedPrefManager* manager is used. This deals with a lot of the formatting and checking on the user. For example it is used in the user login code in Code-block 5.13. In that use case it to read the different variables inserted by the MySQL database as a response in case of a successful login. Furthermore the *SharedPrefManager* also does the checks if the user is logged in and the user logout.

```
1  public boolean userLogin(int id, String username, String email, int phone){
2
3      SharedPreferences sharedPreferences = ctx.getSharedPreferences(↩
           SHARED_PREF_NAME, Context.MODE_PRIVATE);
4      SharedPreferences.Editor editor = sharedPreferences.edit();
5      editor.putInt(KEY_USER_ID, id);
6      editor.putString(KEY_USERNAME, username);
7      editor.putString(KEY_USER_EMAIL, email);
8      editor.putInt(KEY_USER_PHONE, phone);;
9
10     editor.apply();
11
12     return true;
13     }
14
15     //Checks if the user is logged in
16     public  boolean isLoggedIn(){
```

```
17          SharedPreferences sharedPreferences = ctx.getSharedPreferences(↩
                SHARED_PREF_NAME, Context.MODE_PRIVATE);
18          if(sharedPreferences.getString(KEY_USERNAME, null) != null){
19              return true;
20          }
21          return false;
22      }
23
24      //Logs the user out
25      public boolean logout(){
26          SharedPreferences sharedPreferences = ctx.getSharedPreferences(↩
                SHARED_PREF_NAME, Context.MODE_PRIVATE);
27          SharedPreferences.Editor editor = sharedPreferences.edit();
28          editor.clear();
29          editor.apply();
30          return true;
31      }
```

Code-block 5.14: Simple on button press navigation where the old activity is closed.

The *SharedPrefManager* checks if the user is logged in, by checking if the username key is not empty. Of course the actual login check is done when the login click as shown in Code-block 5.13 is called, and only then is the username inserted into the *sharedPreferences* editor. This means that the editor should only have a non-empty key when the user has successfully logged in.

Lastly the user must also be able to log out. In the *SharedPrefManager* it works by simply clearing all fields in the editor, making it look like no user is there. The logout function simply calls *SharedPrefManager.getInstance(context).logout()* and that way it is cleared. Finally as shown in basic navigation in Section 5.5.2 the user is then taken back to the *LoginActivity* and the old activity is closed via the *finish()* command.

5.5.5 Retrieving the file data

In order to facilitate the courses, they must first be downloaded to the application. However the file names and location on the server isn't known. If only certain constant files have to be downloaded the simplest way of downloading them would be by just inserting them as an array into the DownloadManager. How this *DownloadManager* works can be found on in the documentation[14]. Since the files are dynamically changed by the Moodle scraper as described in Section 5.2.1, the file names and folders will update as described in Section 5.3.2. In order to get the full file names as well as its location within the caching server the code shown on Code-block 5.15 is used.

```
1   private void sendGetRequest() {
2       if (isWifiConn = true) {
3           StringRequest stringRequest = new StringRequest(
4                   Request.Method.GET,
```

```
                Constants.URL_GETDATA,
                new Response.Listener<String>() {
                    @Override
                    public void onResponse(String response) {
                        try {
                            outputStream = openFileOutput(filename, Context.↩
                                MODE_PRIVATE);
                            outputStream.write(response.getBytes());
                            outputStream.close();
                        } catch (Exception e) {
                            e.printStackTrace();
                        }
                    }
                }, new Response.ErrorListener() {
            @Override
            public void onErrorResponse(VolleyError error) {
                debugtext.setText("A network error has occured");
            }
        });
        RequestHandler.getInstance(this).addToRequestQueue(stringRequest);
    }
}
```

Code-block 5.15: File information retrieval

As the goal is to minimize the amount of cell phone data used, the first thing done is to check if there is an active internet connection. This is done as described as in android technical analysis as shown in Section 3.2.6. With internet connection over Wi-Fi established, the Android application can begin sending data. Just as with the user registration and login, this is done via HTTP whilst calling a PHP-script on the server as described in Section 5.3. In this it is *URL_GETDATA* as defined in the constants Section 5.5.1. The string response that the PHP-script provides a response that the Android application must store. Since this data could be stored for a longer period of time, it makes sense to save it in a file. This is implemented under *filename* where the string response is saved. The filename chosen is *Stuff.txt* meaning the string response from the caching server will be saved in the Android applications internal storage under *Stuff.txt*

```
String filename = "Stuff.txt";
```

Code-block 5.16: The filename used

Exactly as with the other requests a Volley *ErrorListener* is implemented, in case of network errors. As described in Section 5.3 the data the PHP-script responds with is: "*course/topic/filname.filetype;* The course and topic being folders and the filename being the actual file and type being its format. Having this data collected now means the Android application can commence the downloading of the files.

5.5.6 Downloading and saving files

The current version of the downloading script is non-dynamic. Meaning that it only downloads a test file. How a dynamic implementation could be made is described throughout this section. Downloading a file in Android can be done relatively simply trough a built in library called DownloadManager[14]. This download manager is able to get files via HTTP requests. The implementation in the application is seen on Code-block 5.17

```
1  public void Download_Click(View view) {
2      downloadManager = (DownloadManager) getSystemService(DOWNLOAD_SERVICE);
3      DownloadManager.Request request = new DownloadManager.Request(Uri.parse("↩
           https://save-brazil.lanestolen.dk/test.mp4"));
4
5      request.setAllowedNetworkTypes(
6              NETWORK_WIFI)
7              .setDestinationInExternalPublicDir("/sb", "test.mp4");
8
9      File sdCard = Environment.getExternalStorageDirectory();
10     String folder = sdCard.getAbsolutePath() + "/sb";
11     File dir = new File(folder);
12     if (!dir.exists()) {
13         if (dir.mkdirs()) {
14         }
15     }
16
17     queueid = downloadManager.enqueue(request);
18 }
```

Code-block 5.17: The filename used

Firstly the *DownloadManager* is called and the context is set to *getSystemService(DOWNLOADSERVICE* which defines that the *DownloadManager* is about to be used to download files via a HTTP request. Then the destination of the request needs to be set. In this case it is set to *https://save-brazil.lanestolen.dk/test.mp4* on line 3. To make this a dynamic system this must be changed to a variable defined from the information gathered from the file data retrieving system inside *Stuff.txt*. Critically this is the part of the communication which takes by far the heaviest data footprint. If for example as 20 minute full HD video file needs to be downloaded, then this would be rather sizable. Therefore the *request.setAllowedNetworkTypes* is set to only allow download over Wi-Fi.

With this defined, the way in which the file is to be saved must be done. As described inside the Android technical analysis, API 19 is chosen and in here the permissions needed is utilized. The external storage access which became available in API 19 is now used to mount the files to the phones storage. Firstly the path is defined and then the filename. These should then be dynamically defined by variables as well and run in a loop.

Lastly, it is needed to ensure that the folder the Android application is trying

to save in exists or it is needed to create a new folder. This is done in the last part of the code.

Lastly the request is sent via the: *downloadManager.enqueue(request)* It is notable, that this uses its own request queue and not the *RequestHandler*, since the *DownloadManager* uses its own request system. In order for this system to function the application must have permissions and the implementation of a permission system is documented next.

5.5.7 Permissions

In Android, different permissions are required to accomplish different thing. Firstly the permissions defined needed is done in the *AndroidManifest.xml* file. Since the application uses the internet, a permission is needed for that. It also turns out that Android requires the user to have given permission for an application to read and write to the external storage. This means that the following lines are needed in the *AndroidManifest.xml*

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

Code-block 5.18: The permissions in the Android manifest

While this tells the Android application that these permissions are needed, the user must consent to the external storage. In some versions of Android they also need to consent to the application using the internet. While this can be done simply by doing a one time popup, it can have issues if the user at some points removes these permissions in their phones settings. Therefore a check is made every time the user either boots up the application for the first time, or whenever the user reaches the main screen. How this is done can be seen on Code-block 5.19.

```
private void veryPermissions(){
    String[] permissions = {Manifest.permission.READ_EXTERNAL_STORAGE,
            Manifest.permission.WRITE_EXTERNAL_STORAGE,
            Manifest.permission.INTERNET};

    if(ContextCompat.checkSelfPermission(this.getApplicationContext(),
            permissions[0]) == PackageManager.PERMISSION_GRANTED &&
            ContextCompat.checkSelfPermission(this.getApplicationContext(),
                    permissions[0]) == PackageManager.PERMISSION_GRANTED &&
            ContextCompat.checkSelfPermission(this.getApplicationContext(),
                    permissions[0]) == PackageManager.PERMISSION_GRANTED){
        //We are confirmed
    } else {
        ActivityCompat.requestPermissions(ProfileActivity.this,
                permissions,
                MY_PERMISSIONS_REQUEST);
    }
```

```
18 }
```

Code-block 5.19: The permission request system

Firstly a String array is used to save the different requests. Then a test is run to see if the permissions are granted for all three requirements. If they are not granted a permissions request is sent to the user. In case the user does not grant access to the data storage system, making course facilitation impossible. Furthermore it means that the file download will not work, due to a lack of permissions. This completes the permission implementation. With permissions now complete, the course facilitation implementation can be explored.

5.5.8 Course facilitation

In order to facilitate the course a simple file explorer is used. This means that in case the user presses the button "pick", then a file browser is opened. Optimally this is opened in the main folder of the saved courses. Allowing the user to firstly select a course, then a topic and finally which file to open. There are many ways of the different ways of using a file explorer / picker. In this implementation the material file picker is used [39]. This file picker opens the external storage upon a button press.

```
1 button.setOnClickListener(new View.OnClickListener() {
2     @Override
3     public void onClick(View v) {
4         new MaterialFilePicker()
5                 .withActivity(ProfileActivity.this)
6                 .withRequestCode(1000)
7                 .start();
8     }
9 });
```

Code-block 5.20: The filename used

```
1  @Override
2  protected void onActivityResult(int requestCode, int resultCode, Intent data) {
3      super.onActivityResult(requestCode, resultCode, data);
4
5      if (requestCode == 1000 && resultCode == RESULT_OK) {
6          String filePath = data.getStringExtra(FilePickerActivity.RESULT_FILE_PATH↩
               );
7          //open file. TBD
8          debugtext.setText(filePath);
9      }
10 }
```

Code-block 5.21: The filename used

Firstly an *OnClickListener* sees if the button is pressed. In case it starts up the file picker and sends a *requestCode* of 1000 (default for the library) and starts the *MaterialFilePicker*. Once the filer picker has selected a file, the *debugtext* is set to the exact location of the file including the filename. Of course this should be opened, but this is not yet supported inside the application, as the default way of opening files, wherein the phone simply asks what the file should be opened with, does not work with the *MaterialFilePicker*. Moreover a bug within the *MaterialFilePicker*, means that a top menu cannot be used to a rather complicated conflict, that has not been solved. Because of these two issues, a point of further development is to change the *MaterialFilePicker* to something else. This concludes the course facilitation of the application, and the data collection will be examined next.

5.5.9 Data collection

Since the smart caching system needs the data described in Section 3.2.5, the Android application must be able to record this data. While Android has a built in library called *Log*, the data collection part of the Android application has not been implemented. This means that Log is not used in the current implementation. This must therefore be done at a later date in order to properly begin the smart caching implementation as described in Section 5.4. With this quickly examined, a final conclusion on the Android application is done.

5.5.10 Subsidiary conclusion

The Android application in its current implementation, completes parts of the requirement specification. Firstly the requirements met will be showcased, followed by those partially met, and finally those not met. More specifically the following requirements are met:

- Determine if it is connected to Wi-Fi.
- Write Requests as described in Section 3.2.
- Read responses as described in Section 3.2.
- Facilitate a create user and login system.
- Must handle large files such as video.

The application is able to determine if it is connected to Wi-Fi, and all data-intensive communication is done exclusively over Wi-Fi. Registering a user and logging in is still possible via mobile data, but it is a few small HTTP requests. Both the file data and download is done exclusively via Wi-Fi in order to conserve data. The requests and responses are done using HTTP and work by these requests being send to PHP-scripts on the caching server. These work, although improvements such as requiring a user to be logged in before fetching files and file data. The user is able to both create a user and login with said user. It is also checked if the user

is logged in, and if not they are automatically redirected to the login page. They are also able to logout with the press of a button. Furthermore the login system is successfully connected to the caching based server. Due to external access, the application is able to save as much data as the phone has space. This means that large files such as video can be saved.

However, a system to automatically clean up files and delete them, after the user has passed the courses is needed. If this is not implemented in future development, the Brazilian waster pickers phones will quickly fill up, with old courses. With the met requirements concluded, the partially met will be examined:

These requirements are partially met:

- Facilitate course content.
- Download files from caching server.

The course facilitation is to an extend complete, meaning that the user is able to find the courses within the *MaterialFilePicker*. However, it does not actually open the file and only displays a debug message on the file the user selected. While Android has a default syntax for opening files, attempts at making it work with the *MaterialFilePicker* were unsuccessful. The user can open the files trough their own Android file explorer, but this is a poor option. While the application is able to download files from the caching server using the download manager, it is not dynamic.

This means that in its current state, only one file is downloaded. What should be done is that the file data taken from the caching server should determine the files downloaded trough variables. This concludes the partially met requirements and lastly the non-met requirements are examined.

These requirements are not met:

- Gather stats as described in Section 3.2.5.
- Be able to playback .mp4 and .mp3 files.
- Have a savings and loan calculator similar to the AIF finance application described in Section 2.2.1.2.

As described in [16], Android does have a built in API for logging the data needed, however development never reached that stage, and therefore it is not implemented. Due to the issues with the *MaterialFilePicker*, video and audio within the application is not working. Lastly the calculator is the lowest priority requirement, and since more important requirements have not been met, development of the calculator has not been done.

Overall the Android application needs some back-end work in order to fit with the requirements as layout in the requirements specification. In extension the front-end was never considered as it is not within the scope of the project, but for a formal implementation to the Brazilian waste pickers, it will need an update.

5.6 Partial conclusion

This section will summarize the conclusions of the different parts of the system, and lead into the final conclusion of the project. In Section 5.2.3 it is concluded that the Moodle scraper meets the established requirement. However, there are some of the functions that can be improved so that the system supports all the formats that Moodle does which allows course creators to have more freedom to create content that fits the waste pickers.

Section 5.4.1 and Section 5.3.3 concludes that only some of the requirement are met. The caching server have most of the crucial parts implemented, like the a database for storing data and a way of communicating through HTTP. What needs to be created new is the algorithm for calculating the amount of data which should be send to the smartphone. A part of this is also to create as system that can analyze the network in order to provide this information to the algorithm. However, some of the working parts will also need an update, like the database, this will need to be restructured as a part of the further development so that it can contain the data in an organized way.

In Section 5.5.10 it is concluded that the application meets some of the requirement but not all of them. The application is able to authenticate users by sending requests to the caching server and reading the responses. The application is also able to determine if it is connected to a Wi-Fi and handle large files downloaded on the this connection. This download function is not yet able to download files dynamically so this is a feature that needs to be developed fully. The application also needs to be able to gather stats which is an important part of the caching algorithm so this should be one of the first things to be developed moving on. A thing that is done but is not a part of the requirements is to make the HTTP traffic encrypted. All the data transferred between the smartphone and the server is encrypted using SSL, so that all HTTP traffic is actually transferred using HTTPS and plain HTTP traffic is redirected to HTTPS enforcing encryption.

As seen through this section most of the requirement are met with place for improvements. In order to show specifically which requirements are met, they are put inside a table.

5.6.1 Requirements met

This section will present a checkbox containing all the requirements established. Each of the requirements is then assessed with a symbol, the requirements which are completely implemented is assessed with a ✓, however this does not mean that the implementations cannot be improved. The requirements which are only partially implemented like the CRUD requirement will be presented with a •, and the requirements which are not implemented at all is presented with a %. This table can be seen on Table 5.5.

Moodle Scraper	state
Find content on a Moodle server.	✓
Download files from Moodle.	✓
Save files in the format described in Section 3.1.	✓
Find Etags in a HTTP-response	✓
Compare Etags to local database.	✓
Caching server	
Receive Requests.	✓
Write Responses.	✓
Access local files.	✓
Save stats from app.	•
Perform CRUD on users, see Section 3.2.3	•
Determine the amount of data to be sent.	%
Handle several users simultaneously.	✓
User password recovery.	%
Display stats graphically.	%
Smartphone application	
Facilitate course content.	•
Determine if it is connected to Wi-Fi.	✓
Write Requests as described in Section 3.2.	✓
Read responses as described in Section 3.2.	✓
Facilitate a create user and login system.	✓
Download files from caching server.	•
Gather stats as described in Section 3.2.5.	%
Must handle large files such as video.	✓
Be able to playback .mp4 and .mp3 files.	%
Have a savings and loan calculator similar to the AIF finance application described in Section 2.2.1.2.	%

Table 5.5: List of requirement with a symbol presenting the state

Conclusion based on requirements From the table above in Section 5.6.1 it can be concluded in text both how much was completed and which parts are missing. Out of the 24 requirements 14 are completed which is is around 58% of the require-

ments completely fulfilled. Counting the partially concluded then 20 out of 24 are at least partially completed being 83%.

6 Conclusion

In Chapter 1 the problem following the shift in the way Brazil handles their trash was examined. It was found that by closing the dump sites a large group of people, the waste pickers, were left with out a stable income. By conducting both field research and desk research it was found that most these waste pickers had a inadequate understanding of finances. Alongside students and a professor from UnB, it was found that a potential way of helping these waste pickers was to create an educational system, to help them get a basic education and help with financial management. The desk research also confirmed that an educational system for smartphones could be a valuable tool not just for the waste pickers, but also other groups of people around, as the adoption rates is rising. This research all is rooted in the wish to work with the SDGs, and through them working with a problem that can change the lives of people.

However, it was found that a problem with creating a platform like this is that the files that would be needed to provide adequate teaching materials would be quite large. This lead to an examination of data plans in Brazil. It was found that the waste pickers most likely do not have much of data on their cell plans, and thereby usage of Wi-Fi is critical. From this research an initial concept model describing a potential implementation of this system in broad terms is developed. By exploring the different technical aspects of this concept, a requirement specification and a prove of concept implementation is made.

This implementation manages to be the first building block in a larger system. Having successfully built and connected each subsystem within the system with clear guidelines for future development for each subsystem. Consequently a first implementation is not very far away from the project in the state as of writing, as only a few obstacles remain. Overall this project can from an implementation aspect can be considered a success, especially due to the clear future development path given. Furthermore, going by the amount learned by the project members and the evolving of their respective technical abilities, this project is considered a complete success.

6.1 Future development

A big part of a successful project like this is the ability to scope to what is possible within the timeline available. As the development of this project progressed it became clear that not everything was going to be ready. Simply some develop-

ment is still needed on the project. While this proof of concept is not ready for deployment, it has potential to help countless people get a basic education as it is. Due to this larger perspective and the potential seen, laying out a concise plan for future development is important. In order to begin an implementation phase the Android application must become able to read and download selected lines from the MySQL filedata database. Furthermore the course facilitation needs to get to a working state. Combine this with a rudimentary caching algorithm, an the system would be able to be deployed in Brazil with the courses created by our cooperation partners. Whenever this system is to be deployed within new applications, its important that the courses are changed to work best with the locals. A basic mathematics course might not require much change, as it is pretty universal, however a finance course might need some changes. This could be in a sense of informing different types of loans and savings accounts available to the target group. The best way of having these implementations done is by working together with local non profits of universities to have the create the courses. That way it is ensured that it fits with local standards. With just these final implementation improvements, this project thereby has the potential to help a large amount of people all around the world.

7 Reflection

The reflection is split into two sections: international and internal. The goal of this reflection is to gain insight into how the collaboration functioned and lessons to be taken away from the same. Firstly the international section will showcase the standards for communication used and challenges faced. This will in turn allow for reflection on the methodology effectiveness. Since both project members have been part of a previous international project, a comparative section between reflection is given. With this comparison a summary of the reflection for the international collaboration is given. After the international collaboration is done a reflection of the internal collaboration is commenced.

As before the focus will be on the methodology and a discussion of how well it worked and what important lessons can be taken from it. A point wished to be highlighted as it has been a focus throughout the project: time management will be given a more in depth discussion. As a part of this, a discussion of the collaboration with the project supervisors are given followed by a final summary of the reflection.

7.1 International collaboration

Here the international collaboration will be reflected upon. Firstly, the different international collaborators are originally listed in Section 1.5, but are re displayed here for convenience sake:

Denmark

Name	Field of study	Semester
Daniel Britze	Computer engineering	4th
Peter Lundgaard	Techno anthropology	8th
Robert Nielsen	Computer engineering	4th

Brazil

Name	Field of study	Semester
João Mello's subjects	Production engineering	TBD
Matheus Halbe	Production engineering	9th

Table 7.1: Table containing students participating in the project

Around two months into the project, we where informed that João Mello's subjects had been set to do other work, as a result they did not directly contribute to this demarcation. With the roles and parties surmised. A reflection upon the initial field research is commenced.

7.1.0.1 Methodology

The collaboration between the different partners was not based on deliverables, but rather maintaining contact, keeping each other informed on progress on their respective parts of the project. This communication was done over the Discord platform. The reasons for using Discord are described in [38]. In order to keep in contact, a weekly meeting set on Sunday at 8pm Danish time was set. As a consequence of being set on a weekend, it was sometimes forgotten by some danish project members.

However, as this was only a main project for the writers of this dissertation (Daniel and Robert), a missed meeting was not a big issue. Furthermore a more relaxed attitude was taken by all project members. This let to a good loose collaboration, however this might be due to the lower importance of the collaboration in these early stages of the project. As both members of the group have previous experience in an international collaboration, a quick comparison to the reflection is given.

7.1.1 Compared with previous experience

We (Daniel and Robert), have experience doing international cooperation in a somewhat similar project: P2 - HoneyJar [38]. In that project much of the time was spent communicating with the other project members. Beyond that, a lot of time was spent attempting to handling a project member, who was not doing her part of the project. A valuable lesson learned back then was to quickly take actions in order to not to spent times thinking on it.

In this project, the more relaxed attitude helped alleviate this issue, and ended up resulting in more time being spent on the project as supposed to thinking about the international collaboration. This also comes as a result of the collaboration not being nearly as integral as compared with the HoneyJar project. Overall we are quite pleased with how the international collaboration went, and think for similar projects in the future, a similar approach could be useful.

However, if the project is more tightly integrated between the different parties within the project, a focus on deliverables should probably be done. This concludes the summary of the international collaboration, and the Internal collaboration will be explored next.

7.2 Internal collaboration

While working with international partners helps the project, it does not make a difference unless the internal collaboration works well. Therefore a reflection of the internal collaboration will be given. Firstly discussing and reflecting upon the methodology and time management used in the project. Then a reflection upon the Supervisor collaboration is given.

7.2.0.1 Collaboration methodology

Internally the methodology used can gain insight into the process used while making the system and this methodology will be reflected upon. This project has been very different in a number of key ways as compared to previous projects. Firstly the group size of two, instead of five plus. This has meant quite a few things for the project and how it is done. Firstly since the project is limited to two people, it is important to trust and believe in the other person, as they are responsible for half the project. As a result the way the which work is done is quickly what could be considered more "professional", in a sense that each project member is expected to do something, and left to their own devices in doing so.

A vital ability for this project to get as far as it has, is the way in which we take a big problem and try to find a fitting solution and break this solution down to its components. By making the initial concept model, it become possible to take these different components and do in depth research for each. This methodology means that each component and its respective requirements are based on tangible writing. It also allows each component to be considered as to if it sensible within the project, if it is decided not to, then it becomes necessary for the group to discuss alternatives. This means, that since the different parts worked on don't communicate with each other, then the chances of the different members code coming in conflict with each other was very low.

Of course this also meant that collaboration might not have been as close as it could have been, but this methodology used, did allow each project member large degrees of freedom. It can best be summarized as "freedom under responsibility", which was agreed by the members in the beginning. While some people might find this intimidating, depending on personality types, the project members found it very useful. Having discussed the collaboration methodology, the time management will be examined next.

7.2.0.2 Time management

As time management is a critical aspect of the project, it is sensible to reflect upon it. Time management in this project was somewhat different from previous projects. Since we where only two people, and had work stations right next

to each other, knowing what the other person was doing and their progress was pretty simple. While a more strict plan was laid sometimes in the project, it was quickly realized that some things take the time they do for the person doing it, and remaining more flexible in the time management approach became important. This also meant that the lack of an actual time management system, such as Trello. Partially due to this needed flexibility and the project members being able to quickly learn their co-workers progress, but also partially due to the time it takes to maintain such a system. In hindsight it has quickly become apparent that this solution is lacking, and that more detailed time management should be done. Within the group an idea is made to always have the things people are working on, in writing on a blackboard, where each person periodically updates the percentage done. The period would have to be decided within the group.

Working as two in a group as meant time for a group member is more limited compared to groups with five or more, as groups size usually are in this semester. Of course, this does not mean no structure was used for time management. A focus on the results of what the project member had done is the approach taken. Usually, this will result into a supervisor meeting being set as the next "milestone" and the things to do beforehand would be defined as the meeting was planned. This gave each group member a tangible thing to work on within about a two week basis.

An additional challenge this semester to the time management, has been the workshops from the courses, as they proved more difficult than previous semesters, and therefore took quite a bit more time. Again this means that flexibility had to be the keyword for all time management, and overall was successful this semester. In addition as with all project, a supervisor is part of the project and the collaboration with them is therefore reflected on.

7.2.0.3 Supervisor

Having now spent two years working with supervisors in the context of AAU, the lessons learned from previous projects are definitely beginning to show. Firstly, the supervisor is used to give periodic feedback on the project demarcation, which helps create a better end result. In previous projects, we almost always concluded that the supervisor needed to be used more for this. However in this project a much better balance was struck. Resulting in supervisor meeting with the goal of providing feedback to the demarcation, might be somewhat far apart, but only "finished work" is shown to the supervisor allowing a model developed during Daniels 1st semester to be used [23]:

The A-B-V-C model: Person A writes a section. Person B corrects a section and looks for errors. The supervisor V gives feedback on the section during a supervisor meeting. Lastly Person C (Person A or B in this case) corrects the errors found

by the supervisor or any new ones he finds himself. Then this part of the demarcation can be crossed off as finished unless new information leads to the section needing changes.

Using this model has allowed for a very productive semester, where a demarcation of considerable size for two people has been created. Overall the supervisor collaboration has been really successful, as feedback has been provided when needed. Unlike some previous projects, few issues presented themselves during the time spent on this project. Effectively meaning, the supervisor was not needed to help with any issues be it internal collaboration or the international collaboration.

7.3 Summary

In summary a more relaxed attitude in the international collaboration was taken. Due to the different parts of the project not being very tightly bound, this proved a good thing, as little time was spent on issues, allowing for more focused time on working. The internal collaboration was quite successful based on the ability to break down a solution into smaller bites and splitting them up between the project members. Working with a "freedom under responsibility" mindset, helped the productivity of both project members. The supervisor has played a good role within the project providing feedback to the demarcation when asked. Finally the reflection conclusion table is presented.

7.4 Reflection conclusion table

The reflection conclusion table presents lessons learned from this project within a three structure system: Start doing, keep doing, stop doing. For each point in the table, the lesson will be presented.

Start doing
More structured time planning.
Concrete and measurable deadlines.
Unified code structure.
Better internal communication.

Keep doing
Freedom under responsibility.
Weekly meetings.
Group contract.
Use the help from people around us.
Active information sharing.
Take notes during every supervisor meetings.
Choosing real world applications as a project
Shared calendar.

Stop doing
Waiting for collaboration partners.
Spending time on irrelevant activities.
Write Requests as described in Section 3.2.
Read responses as described in Section 3.2.
Facilitate a create user and login system.

Table 7.2: List of lessons learned during the project

Bibliography

[1] barteksc. *Android PdfViewer*. https://github.com/barteksc/AndroidPdfViewer. [Visited 08-04-2019]. 2019.

[2] bbc.com. *Huge Brazil rubbish dump closes after six decades*. https://www.bbc.com/news/world-latin-america-42757085. [Visited 26-03-2019]. 2018.

[3] Sagar Bhatia. *PostgreSQL vs. MySQL: [2019] Everything You Need to Know*. https://hackr.io/blog/postgresql-vs-mysql. [Visited 15-05-2019]. 2019.

[4] Pew Research Center. *Pew Research Center*. https://www.pewresearch.org/. [Visited 27-05-2019]. 2019.

[5] Rasmi Vlad Mahmoud Christian Hundborg Liboriussen Ivan Kjær Gammeljord. "Connecting South African Schools With Unreliable Internet". In: *Unpublished* (2018), p. 121.

[6] countryeconomy.com. *Brazil National Minimum Wage - NMW*. https://countryeconomy.com/national-minimum-wage/brazil. [Visited 14-03-2019]. 2019.

[7] Open Trivia DB. *Open Trivia DB*. https://opentdb.com/. [Visited 05-04-2019]. 2019.

[8] ASonia M. Dias. *Statistics on Waste Pickers in Brazil*. http://www.wiego.org/publications/statistics-waste-pickers-brazil. [Visited 25-03-2019]. 2011.

[9] Serviço de Limpeza Urbana do Distrito Federal. *Serviço de Limpeza Urbana do Distrito Federal*. http://www.slu.df.gov.br/. [Visited 13-02-2019]. 2019.

[10] Coding in Flow. *How to Play a Sound File Using the MediaPlayer Class - Android Studio Tutorial*. https://www.youtube.com/watch?v=C_Ka7cKwXW0. [Visited 20-05-2019]. 2018.

[11] Grupo Gestão. *Grupo Gestão*. https://www.grupogestaoconsultoria.com/. [Visited 27-05-2019]. 2019.

[12] Google. *AIF Financial Literacy - googe play store*. http://tinyurl.com/y3djxloy. [Visited 27-03-2019]. 2019.

[13] Google. *ConnectivityManager*. https://developer.android.com/training/basics/network-ops/managing. [Visited 20-05-2019]. 2019.

[14] Google. *DownloadManager*. https://developer.android.com/reference/android/app/DownloadManager. [Visited 23-05-2019]. 2019.

[15] Google. *finance education - googe play store*. http://tinyurl.com/y3djxloy. [Visited 27-03-2019]. 2019.

[16] Google. *Log*. https://developer.android.com/reference/android/util/Log. [Visited 23-05-2019]. 2019.

[17] Google. *math education - googe play store*. http://tinyurl.com/y68rjfhl. [Visited 27-03-2019]. 2019.

[18] google. *MediaPlayer overview*. https://developer.android.com/guide/topics/media/mediaplayer.html. [Visited 08-04-2019]. 2019.

[19] Google. *VideoView*. https://developer.android.com/reference/android/widget/VideoView. [Visited 20-05-2019]. 2019.

[20] Google. *Volley Overview*. https://developer.android.com/training/volley. [Visited 20-05-2019]. 2019.

[21] GreenShoots. *GreenShoots*. https://www.greenshootsfoundation.org/. [Visited 13-02-2019]. 2019.

[22] miniwatts marketing group. *Internet Users in the World*. https://www.internetworldstats.com/stats.htm. [Visited 14-03-2019]. 2019.

[23] Christopher Damsgaard. Daniel Britze. Kristian Noesgaard Nielsen. Jacob Vejlin Jensen. Magnus Stensli. Mikkel Steen Hansen. *AAU HoneyPot*. https://projekter.aau.dk/projekter/da/studentthesis/aau-honeypot(4b016229-e96e-46d2-b421-c2757946bcd0).html. [Visited 23-05-2019]. 2018.

[24] Daniel Britze. Johan Hempel Bengtson. Jacob Vejlin Jensen. Robert Nedergaard Nielsen. Mikkel Steen Hansen. *Internet of Things - Smart Home: Smart Home protocol design*. https://projekter.aau.dk/projekter/da/studentthesis/internet-of-things--smart-home(4bb4cdfc-02eb-44b7-9804-724327401dbb).html. [Visited 23-05-2019]. 2018.

[25] Women in Informal Employment: Globalizing and Oraganizing. *Empowering Informal Workers, Securing Informal Livelihoods*. http://www.wiego.org/. [Visited 25-03-2019]. 2019.

[26] Women in Informal Employment: Globalizing and Oraganizing. *Waste Pickers*. http://www.wiego.org/blogs/term/25. [Visited 25-03-2019]. 2019.

[27] Anas Khan. *package soup*. https://godoc.org/github.com/anaskhan96/soup. [Visited 28-05-2019]. 2019.

[28] Laura Silver Kyle Taylor. *Smartphone Ownership Is Growing Rapidly Around the World, but Not Always Equally*. http://www.pewglobal.org/2019/02/05/smartphone-ownership-is-growing-rapidly-around-the-world-but-not-always-equally/. [Visited 13-03-2019]. 2019.

[29] Moodle. *Working with files*. https://docs.moodle.org/36/en/Working_with_files. [Visited 12-04-2019]. 2019.

[30] United Nations. *4 Quality Education*. https://www.un.org/sustainabledevelopment/education/. [Visited 13-03-2019]. 2019.

[31] United Nations. *About the Sustainable Development Goals*. https://www.un.org/sustainabledevelopment/sustainable-development-goals/. [Visited 13-03-2019]. 2019.

[32] ngo.org. *DEFINITION OF NGOs*. http://www.ngo.org/ngoinfo/define.html. [Visited 13-02-2019]. 2019.

[33] nperf. *2G / 3G / 4G coverage map, Brazil*. https://www.nperf.com/en/map/BR/-/161704.Vivo-Mobile/signal/?ll=-12.168225677390119&lg=-47.06542968750001&zoom=5. [Visited 25-03-2019]. 2019.

[34] Jacob Poushter. *Smartphone Ownership and Internet Usage Continues to Climb in Emerging Economies*. http://www.pewglobal.org/2016/02/22/smartphone-ownership-and-internet-usage-continues-to-climb-in-emerging-economies/. [Visited 13-03-2019]. 2019.

[35] Stastistica. *Mobile telephony market share in Brazil from 2000 to 2016, by telecommunications provider*. https://www.statista.com/statistics/523798/mobile-market-share-in-brazil-by-operator/. [Visited 14-03-2019]. 2019.

[36] statcounter.com. *Mobile Operating System Market Share Brazil*. https://tinyurl.com/y5gnatmx. [Visited 22-05-2019]. 2019.

[37] statista. *Number of internet users worldwide from 2005 to 2018 (in millions)*. https://www.statista.com/statistics/273018/number-of-internet-users-worldwide/. [Visited 14-03-2019]. 2019.

[38] Peter Bolstad Møller. Daniel Britze. Jacob Vejlin Jensen. Robert Nedergaard Nielsen. Magnus Stensli. *EPIC HoneyJar*. https://projekter.aau.dk/projekter/da/studentthesis/epic-honeyjar(ab23958c-8a53-4847-9538-623cb821616a).html. [Visited 23-05-2019]. 2018.

[39] nbsp team. *aterialFilePicker*. https://github.com/nbsp-team/MaterialFilePicker. [Visited 23-05-2019]. 2019.

[40] tradingeconomics.com. *Brazil Real Average Monthly Income*. https://tradingeconomics.com/brazil/wages. [Visited 14-03-2019]. 2019.

[41] VIVO. *Conheça as Ofertas*. https://celular.vivo.com.br/planos/pre/. [Visited 14-03-2019]. 2019.

[42] VIVO. *Vivo Pós*. https://www.vivo.com.br/portalweb/appmanager/env/web?_nfls=false&_nfpb=true&_pageLabel=P10340028869144831327915 9&WT.ac=portal.movel.planosepacotes.planospos.smartvivopos_&. [Visited 14-03-2019]. 2019.

[43] Wikipedia. *Moodle*. https://en.wikipedia.org/wiki/Moodle. [Visited 04-04-2019]. 2019.

[44] Petr Škoda. *File API internals*. `https://docs.moodle.org/dev/File_API_internals`. [Visited 09-04-2019]. 2017.

Appendices

.1 Field research questions

Research question	Interview question	Translation Portuguese
How are the workers financial situation?	- Do you feel like you have enough money to last the rest of the week/Month?	- Voce sente que voce tem dinheiro o suficiente para o resto da semana?
	- Do you know how much money you are making every week/month?	- Voce sabe quanto dinheiro voce faz por semana?
	- Do you know how much you pay in tax?	- Voce sabe o quanto voce paga em impostos?
	- Do you have a retirement plan?	- Voce tem algum plano para sua aposentadoria?
How well do the workers fare with mathematics?	- Have you ever been to school?	- Voce frequentou a escola atë qual serie?
	- How well would you say you understand mathematics?	- O quao bem voce entende matematica?
	- Do you feel like your level of understanding of mathematics creates problems in your everyday life?	- Voce acha que voce perde alguma coisa por nao saber matematica?
Does the workers experience problems in their daily in regards to finance?	- Have ever experienced not having enough money for basic necessities?	- Alguma vez faltou dinheiro para suas necessidades basicas?
	- What do you feel about your financial situation? Are you happy or do you feel like it is a problem?	- Como esta sua situacao financeira? Voce esta feliz ou acha que ë um problema?
How do the workers feel about education and learning?	- Are you interested in learning?	- Voce se interessa por aprender?
	- How do feel about the education you have already had?	- O que voce acha da educacao que voce ja teve?
	- Do you feel like you have gained new knowledge from those classes?	- Voce considera que voce ganhou novos conhecimentos dessas aulas?
	- How have the classes impacted your life?	- Como as aulas impactaram suas vidas?
	- Do you want to participate in an educational course about basic mathematics and financial management?	- Voce teria interesse em participar de um curso de matematica e educacao financeira?
	- What are your thoughts about education in general?	- O que voce acha da escola e da educacao em geral?
What problems do the workers see as the biggest in their lives?	- What do you feel is the biggest problem in your life right now?	- Qual e o principal problema da sua vida agora?

.2 Field research interview

Translation of the "interview" with the waste pickers during the tour through the dump site area (01/30/19).

Transcribed and translated by Mateus Halbe, see Section 1.5.

Mateus Halbe:So, let's start. We see things are better than before.

Waste picker:No, things are not better than before.

Mateus Halbe:Why?

Waste picker:Because before we had money.

Mateus Halbe:Are you making less money today?

Manager of administration:What is improved today in infrastructure, or space to work. The point of before they gained more is that before they worked individually then it was a war for the garbage, because the one that reached first would have property of garbage. Now at the Triage Center everyone works and the whole amount of revenue is divided. And who works more compared to other people ends up feeling hurt. So they think here is not better. He works a lot here and works a little and receives less here.

Mateus Halbe:How much do you get here?

Waste picker:With the benefit payed for the participation in the classes, R$ 1000. R$360 from the participation in the course.

Mateus Halbe:What do they teach in the course for you?

Waste picker:Now Work safety practices, management, how to operate the machines. A more technical learning. It's once a week. We have learned about the cooperative's financial management, where we apply our money.

Mateus Halbe:Your financial situation could be better right? but the thing is evolving right?

Waste picker:Yes it is evolving.

Mateus Halbe:How do you consider your math knowledge? and you suffer some kind of limitation in your everyday life because of some lack of math knowledge?

Waste picker:I do not know how to make head or paper calculations. But I do everything in the calculator just fine.

Mateus Halbe:What is your schooling?

Waste picker:I finished high school and now I wanted to do social service college. But due to the conditions of money, schedules and everything ends up not being possible. But here the minority has second degree (high school).

Mateus Halbe:And here there are a lot of people who can not read or write, right?

Waste picker:Yes. We have a program in progress that made two people who did not write, and would not be able to write their names, to now be able to write at least their names.

Mateus Halbe:Excellent

Waste picker:They separate people into groups depending on the level of education they have given these reading and writing classes to who needs it there in IFB. The rest do the other courses, from Senai, for example. We have here five people who do not write in any way. Not even their name.

Mateus Halbe:And you, Maria? as...

Maria:No! I have nothing to say! You don't need to talk to me.

Mateus Halbe:No. Just a little question here, Maria. Come on! I just want to ask you this: did you study until which school level?

Maria:The only thing I know is to sign my name. Nothing else.

Mateus Halbe:Interesting. And do you read something?

Maria:No.

Mateus Halbe:Not yet, right, Maria?

Maria:Yes. Not yet.

Mateus Halbe:Because you'll be able to read soon, right, Maria?

Maria:Ah! study is not with me! I don't think so.

Mateus Halbe:But then Maria, as you write your name, you are not in the group of five that are considered as unfit to read and write, right?

Maria:Yes. I am not on that group. I sign my name.

Mateus Halbe:(Min11) and here how does division of functions work?

Maria: We decide who are in the directors team of the cooperative, but even the administrators and directors do the screening. Everyone here does sorting. everybody here selects garbage.

Mateus Halbe: Maria, Do you have a cell phone and use WhatsApp?

Maria: Yes

Mateus Halbe: Is your cell phone with you right now?

Maria: Yes

Mateus Halbe: Can I take a look at your cell phone?

Maria:(Laughs)

Mateus Halbe: Calm down, there's a camera here and everything. I will not steal your cell phone.

Maria:(Maria shows her cell phone)

Mateus Halbe: Cool. It's a Samsung that holds two chips. It's a good cell phone. You can solve your things, right? and even if you do not have the ability to read you can use the cell phone and solve your things?

Maria: Yes

Mateus Halbe: Maria, how do you talk in the WhatsApp? you just listen to audio, is this?

Maria: Yes. I listen to audio and record the audio.

Mateus Halbe: Another question, Maria. Have you experienced financial difficulties before or now that made you unable to meet your basic needs?

Maria: Now it's difficult but it's possible to survive well. Another thing is my past. if I talk about my past it will be very crying here. I can not tell my story here. I certainly had this situation. I was raised without a father..., anyway. But now the children have grown and everything; grown well; and now I have some support from them.

Waste picker: Most people here have already been through this extremely difficult financial situation, and some are still through. There are months here that we need to help each other, otherwise it is very difficult.

Maria:I'm old enough to retire and I can not retire. Things are too difficult these days.

Mateus Halbe: It's one day after another, right? And nowadays if you were to select Which is the biggest problem you face today? in general, that problem that hinders his life.

Waste picker: It is so much. So many things. I think family is the most important, if family is well Everything is fine. Now at this point I feel like I have a big problem with that. The love of many today It's gone cold. For me today my biggest problem is my family.

Maria: It has a whole range of problems, today the growing child already wants to get involved with drugs. So this is a very big problem, but I thank God I created my five children and my children are a blessing. Everyone works, everyone has their service ... for me, what hurts me most today is that I could not retire. I've worked too much! in the field doing heavy duty since the 8 years old. I grew up like this and married, I can not even calculate how much time I worked.

Mateus Halbe: So, to ask another question. What is the verdict, do you prefer to work here or in the dump?

Waste picker: Dumping ground.

Waste picker: Dumping ground.

Waste picker: Dump site with certainty.

Maria: Here everyone is for the dump site

Mateus Halbe: Is this serious or are you all joking?

Waste picker: Seriously! There, I worked three days and I already supported the entire month

Maria:I particularly prefer here

Mateus Halbe: So if the Dumpsite went back to operation, would you go back there?

Waste picker:I have five children to support

Manager of administration: We have to think about the benefits highlighted today. All Social Security contributions from you today are paid. I do not know if this will work for all of you, But you are all up to date on these payments in Welfare. no one had it there. Here he earns little, because it is not like there that we were just going to get the garbage and leave. Here you have a whole structure to keep. Here we have a thousand of things to pay.

Waste picker: But if we worked there right now and still had to pay all that, we would still earn more money. There we had more money. The material there was totally different from what has come to here and I used my salary to pay the social security fees and even then I had left over. Here everyone only takes $ 150.

Mateus Halbe:$ 150 per week?

Waste picker: No. R $ 150 per fortnight. R $300 per month; and R $350 when it gives too much.

Waste picker: And there is the benefit of compulsory classes that is R $360. It is not enough. R$700 for the whole month when we have a very good and productive month.

Mateus Halbe: Yes, indeed! It makes sense, I got the perception from everyone, but what the management waste picker said, it's also relevant; when you pay the Pension Plan correctly you are investing in the future, and this was not done before when you were in the dump site. It's complicated. It's hard to choose, I know.

Waste picker: Yes. But I can not keep thinking about my future. Because it is already difficult to eat today.

Manager of administration: Here the advantage is that we work in the shade, with a dining room, bathroom to shower and a whole structure. It's dignity. We think about how many people died working in the open dump pressed by trucks or with sickness.... Here we have dignity.

Waste picker: For my dignity it's money

Mateus Halbe:(Asks everybody to show their phones and check their capacities. The phones are good. And says goodbye to the waste pickers and go away.)

.3 Moodle scraper

.3.1 Moodle scraper - main

```
1 package main
2
3 import (
4     "io/ioutil"
5     "log"
6     "net/http"
7     "net/http/cookiejar"
```

```
    "net/url"
    courses "test2/courses"
    files "test2/files"
)

const (
    baseUrl = "http://frudio.lanestolen.dk:8080/"
)

var (
    username  = "admin"
    password  = "Smart-caching-2019"
    Database  = "database.json"
    outputDir = "output"
)

type App struct {
    Client *http.Client
}

func (app *App) login() {
    client := app.Client
    loginURL := "http://frudio.lanestolen.dk:8080/login/index.php"

    data := url.Values{
        "username": {username},
        "password": {password},
    }
    resp, err := client.PostForm(loginURL, data)
    if err != nil {
        log.Fatalln(err)
    }
    defer resp.Body.Close()
    _, err = ioutil.ReadAll(resp.Body)
    if err != nil {
        log.Fatalln(err)
    }
}
func main() {
    jar, _ := cookiejar.New(nil)
    app := App{
        Client: &http.Client{Jar: jar},
    }
    files.InitDatabase(Database)
    app.login()
    files.MakeDir("output")
    courses.FindCourses(baseUrl, app.Client)
    courses.MakeCourseDirs("output")
    courses.DownloadContent(outputDir, Database, app.Client)
    courses.SaveDatabase()
    courses.DoTheDBThing()

    files.SaveDatabase(Database)
}
```

Code-block 1: Main go

.3.2 Moodle scraper - files

```go
package files

import (
    "bytes"
    "encoding/json"
    "io"
    "io/ioutil"
    "log"
    "net/http"
    "os"
    "regexp"
    "strings"
    "sync"

    "github.com/anaskhan96/soup"
)

var (
    DB = Files{}
)

type Files struct {
    Files []File `json:"files"`
}

type File struct {
    Href string `json:"href"`
    Etag string `json:"etag"`
}

func MakeDir(path string) {
    os.Mkdir(path, 0777)
}

func fileWrite(filePath string, text []byte) {
    file, err := os.Create(filePath)
    if err != nil {
        log.Fatal(err)
    }
    file.Write(text)
    if err != nil {
        log.Fatal(err)
    }
}

func InitDatabase(Database string) {
    startValue := []byte("{\"files\": []}")
    if _, err := os.Stat(Database); err != nil {
        if os.IsNotExist(err) {
            fileWrite(Database, startValue)
        } else {
            // other error
        }
    }
}
```

```
56 func SaveDatabase(Database string) {
57     db, err := json.Marshal(DB)
58     if err != nil {
59         log.Fatal(err)
60     }
61     ioutil.WriteFile(Database, db, 0777)
62 }
63 func FindFile(href, etag string, Database string) string {
64     data, err := os.Open(Database)
65     if err != nil {
66         log.Fatal(err)
67     }
68     defer data.Close()
69     byteValue, _ := ioutil.ReadAll(data)
70     _ = json.Unmarshal([]byte(byteValue), &DB)
71     for i := 0; i < len(DB.Files); i++ {
72         if href == DB.Files[i].Href && etag == DB.Files[i].Etag {
73             return "File is concurent"
74         }
75     }
76     return "File is outdated or doesn't exist"
77 }
78
79 func getFile(url string, client *http.Client) *http.Response {
80     tempfile := "file.tmp"
81     log.Println("Getting files with url - ", url)
82     resp, err := client.Get(url)
83     if err != nil {
84         log.Fatal(err)
85     }
86     defer resp.Body.Close()
87     file, err := os.Create(tempfile)
88     if err != nil {
89         log.Fatal(err)
90     }
91     defer file.Close()
92     io.Copy(file, resp.Body)
93     return resp
94 }
95 func GetFile(href, folder, Database string, client *http.Client, wg *sync.↩
       WaitGroup) string {
96     r := getFile(href, client)
97     var etag string
98     for k, v := range r.Header {
99         if k == "Etag" {
100            etag = strings.Join(v, "")
101            log.Println(etag)
102        }
103    }
104    if FindFile(href, etag, Database) == "File is concurent" {
105        log.Println("File is concurent")
106        wg.Done()
107        return "File is concurent"
108
109    } else {
110        DB.Files = append(DB.Files, File{
111            Href: href,
112            Etag: etag,
113        })
```

```
114     }
115     d := r.Header["Content-Disposition"]
116     re := regexp.MustCompile(`filename=\"(?P<Name>.+)\"`)
117     filename := re.FindStringSubmatch(strings.Join(d, ""))[1]
118     os.Rename("file.tmp", folder+"/"+filename)
119     SaveDatabase(Database)
120     wg.Done()
121     return filename
122 }
123 func GetPage(href, folder, Database string, client *http.Client, wg *sync.↩
        WaitGroup) string {
124     r, err := soup.GetWithClient(href, client)
125     if err != nil {
126         log.Fatal(err)
127     }
128     parsed := soup.HTMLParse(r)
129     name := parsed.Find("h2").FullText()
130     re := regexp.MustCompile(`.+`)
131     small := re.ReplaceAllFunc([]byte(name), bytes.ToLower)
132     re2 := regexp.MustCompile(`([A-Za-z]+)\s(\d+)`)
133     nopunc := re2.ReplaceAll(small, []byte("$1-$2"))
134     clean := string(nopunc) + ".txt"
135     text := parsed.Find("div", "class", "text_to_html").FullText()
136     fileWrite(folder+"/"+clean, []byte(text))
137     wg.Done()
138     return clean
139 }
```

Code-block 2: Files main

.3.3 Moodle scraper - courses

```
1  package courses
2
3  import (
4      "database/sql"
5      "encoding/json"
6      "fmt"
7      "io/ioutil"
8      "log"
9      "net/http"
10     "regexp"
11     "strings"
12     "sync"
13     "test2/files"
14
15     "github.com/anaskhan96/soup"
16     //driver for mysql
17     _ "github.com/go-sql-driver/mysql"
18 )
19
20 const (
21     host     = "localhost"          //Address for the mysql database
22     port     = 3306                 //Port for communicating with mysql
23     user     = "admin"              //Mysql user
```

```
24      password = "Smartcaching-2019" //Password for mysql
25      dbname   = "android"           //Name of the database
26  )
27
28  var (
29      courses = Courses{}    //instance of courses
30      wg      sync.WaitGroup //WaitGroup for syncronizing multithreading
31  )
32
33  //Courses contains an arry of courses
34  type Courses struct {
35      Courses []Course `json:"courses"`
36  }
37
38  //Course contains the name, link and topics of each course on moodle
39  type Course struct {
40      Name   string  `json:"name"`
41      Href   string  `json:"href"`
42      Topics []Topic `json:"topics"`
43  }
44
45  //Topic contains the name and the Contents of a topic within a course
46  type Topic struct {
47      Name    string     `json:"topic"`
48      Content []Resource `json:"resource"`
49  }
50
51  //Resource is a description of each file
52  type Resource struct {
53      Modtype string `json:"type"`
54      Href    string `json:"href"`
55      Name    string `json:"name"`
56  }
57
58  //findCourse checks whether a course already exists in Courses
59  func findCourse(name string) bool {
60      for i := 0; i < len(courses.Courses); i++ {
61          if name == courses.Courses[i].Name {
62              return true
63          }
64      }
65      return false
66  }
67
68  //FindCourses find the available courses and adds then to courses and calls ↩
        findTopics
69  func FindCourses(baseURL string, client *http.Client) {
70      resp, err := soup.GetWithClient(baseURL, client)
71      if err != nil {
72          log.Fatalln(err)
73      }
74      parsed := soup.HTMLParse(resp)
75      courselinks := parsed.FindAll("h4")
76      for i := range courselinks {
77          links := courselinks[i].Find("a")
78          link := links.Attrs()["href"]
79          name := sanitizeCourseName(links.Text())
80          if findCourse(name) == true {
81
```

```
82          } else {
83              courses.Courses = append(courses.Courses, Course{
84                  Href: link,
85                  Name: name,
86              })
87          }
88
89      }
90      findTopics(client)
91  }
92
93  //MakeCourseDirs creates a folder for each course and subfolders for each topic
94  func MakeCourseDirs(outputDir string) {
95      for i, c := range courses.Courses {
96          courseDir := outputDir + "/" + c.Name
97          files.MakeDir(courseDir)
98          for _, t := range courses.Courses[i].Topics {
99              topicDir := courseDir + "/" + t.Name
100             files.MakeDir(topicDir)
101         }
102     }
103 }
104
105 //findTopics finds topics within each course and the Contents and adds it to the ↩
        courses instance of Courses
106 func findTopics(client *http.Client) {
107     for j := range courses.Courses {
108         resp, err := soup.GetWithClient(courses.Courses[j].Href, client)
109         if err != nil {
110             log.Fatalln(err)
111         }
112         parsed := soup.HTMLParse(resp)
113         list := parsed.FindAll("li", "class", "main")
114         for i := range list {
115             topic := list[i].Find("span", "class", "sectionname").Text()
116             clean := sanitizeTopicName(topic)
117             courses.Courses[j].Topics = append(courses.Courses[j].Topics, Topic{↩
                    Name: clean})
118             Content := list[i].FindAll("div", "class", "content")
119             for k := range Content {
120                 file := Content[k].FindAll("li", "class", "resource")
121                 for f := range file {
122                     resource := file[f].FindAll("a")
123                     for l := range resource {
124                         link := resource[l].Attrs()["href"]
125                         r, _ := client.Get(link)
126                         d := r.Header["Content-Disposition"]
127                         re := regexp.MustCompile(`filename=\"(?P<Name>.+)\"`)
128                         filename := re.FindStringSubmatch(strings.Join(d, ""))[1]
129                         courses.Courses[j].Topics[i].Content = append(courses.↩
                                Courses[j].Topics[i].Content, Resource{Href: link, ↩
                                Modtype: "resource", Name: filename})
130                     }
131                 }
132                 page := Content[k].FindAll("li", "class", "page")
133                 for f := range page {
134                     resource := page[f].FindAll("a")
135                     for l := range resource {
136                         link := resource[l].Attrs()["href"]
```

```
137                         name := resource[1].Find("span", "class", "instancename")↩
                                .Text()
138                         filename := sanitizeTopicName(name) + ".txt"
139                         courses.Courses[j].Topics[i].Content = append(courses.↩
                                Courses[j].Topics[i].Content, Resource{Href: link, ↩
                                Modtype: "page", Name: filename})
140                     }
141                 }
142
143             }
144         }
145     }
146 }
147
148 //DownloadContent downloads each file within the courses starting a new thread ↩
    for each file
149 func DownloadContent(outputDir, Database string, client *http.Client) {
150     for i, c := range courses.Courses {
151         for j, t := range courses.Courses[i].Topics {
152             for _, k := range courses.Courses[i].Topics[j].Content {
153                 if k.Modtype == "resource" {
154                     wg.Add(1)
155                     go files.GetFile(k.Href, outputDir+"/"+c.Name+"/"+t.Name, ↩
                            Database, client, &wg)
156                 } else if k.Modtype == "page" {
157                     wg.Add(1)
158                     go files.GetPage(k.Href, outputDir+"/"+c.Name+"/"+t.Name, ↩
                            Database, client, &wg)
159
160                 }
161             }
162             wg.Wait()
163         }
164     }
165
166 }
167
168 // DoTheDBThing is the function that adds all the files to a mysql database for ↩
    the Smartphone app
169 func DoTheDBThing() {
170     data := fmt.Sprintf("%s:%s@tcp(%s:%d)/%s",
171         user, password, host, port, dbname)
172     db, err := sql.Open("mysql", data)
173     if err != nil {
174         log.Fatal(err)
175     }
176     defer db.Close()
177     for i, c := range courses.Courses {
178         for j, t := range courses.Courses[i].Topics {
179             for _, k := range courses.Courses[i].Topics[j].Content {
180                 sqlStatement := fmt.Sprintf("INSERT INTO `Files` (course, topic, ↩
                        filename) VALUES ('%s','%s', '%s');", c.Name, t.Name, k.Name)
181                 fmt.Println(sqlStatement)
182                 fmt.Println(k)
183                 err = db.QueryRow(sqlStatement).Scan()
184                 if err != nil {
185                     log.Println(err)
186                 }
187             }
```

```
        }
    }
}
func SaveDatabase() {
    db, err := json.Marshal(courses)
    if err != nil {
        log.Fatal(err)
    }
    ioutil.WriteFile("test.json", db, 0777)
}
```

Code-block 3: Course main

```
package courses

import (
    "bytes"
    "regexp"
)

func sanitizeCourseName(input string) string {
    re := regexp.MustCompile(`([A-Za-z]+)\s([A-Za-z]+).\s([A-Za-z]+)`)
    small := re.ReplaceAllFunc([]byte(input), bytes.ToLower)
    nopunc := re.ReplaceAll(small, []byte("$1-$2-$3"))
    clean := string(nopunc)
    return clean
}
func sanitizeTopicName(input string) string {
    re := regexp.MustCompile(`.+`)
    small := re.ReplaceAllFunc([]byte(input), bytes.ToLower)
    re2 := regexp.MustCompile(`([A-Za-z]+)\s(\d+)`)
    nopunc := re2.ReplaceAll(small, []byte("$1-$2"))
    clean := string(nopunc)
    return clean
}
```

Code-block 4: Course sanitizer

.4 PHP-scripts

.4.1 Register user

```
<?php

require_once '../includes/DbOperations.php';
$response = array();

if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    if (
        isset($_POST['username']) and
        isset($_POST['email']) and
        isset($_POST['password']) and
```

```
        isset($_POST['phone'])) {
        //Data operation becomes possible

        $db = new DbOperations();

        $result = $db->createUser(      $_POST['username'],
                                        $_POST['password'],
                                        $_POST['email'],
                                        $_POST['phone']
                                );
        if ($result == 1){
            $response['error'] = false;
            $response['message'] = "User registered successfully";

        }elseif($result == 0){
            $response['error'] = true;
            $response['message'] = "User already exists, please choose a ↩
                different email and username";

        }elseif($result == 2) {
            $response['error'] = true;
            $response['message'] = "An error occurred, Please try again";
        }

    }else{
        $response['error'] = true;
        $response['message'] = "Required fields are missing";
    }

} else {
    $response['error'] = true;
    $response['message'] = "Invalid Request";
}

echo json_encode($response);
```

Code-block 5: Register user

.4.2 User login

```
<?php
require_once '../includes/DbOperations.php';

$response = array();

if ($_SERVER['REQUEST_METHOD'] == 'POST'){

    if (isset($_POST['username']) and isset($_POST['password'])){
        $db = new DbOperations();

        if ($db->userLogin($_POST['username'], $_POST['password'])){
            $user = $db->getUserByUsername($_POST['username']);
            $response['error'] = false;
            $response['id']         = $user['id'];
            $response['username']   = $user['username'];
```

```
            $response['email']      = $user['email'];
            $response['phone']      = $user['phone'];
            $response['message'] = "Logged in successfully";
        }else{
            $response['error'] = true;
            $response['message'] = "Invalid username or password";
        }

    } else {
        $response['error'] = true;
        $response['message'] = "Required fields are missing";
    }
}

echo json_encode($response);
?>
```

Code-block 6: User login

.4.3 DbOperations

```
<?php

class DbOperations
{

    private $con;

    function __construct()
    {

        require_once dirname(__FILE__) . '/DbConnect.php';

        $db = new DbConnect();

        $this->con = $db->connect();
    }

    /*CRUD -> C -> Create*/

    public function createUser($username, $pass, $email, $phone)
    {
        if ($this->isUserExist($username, $email)) {
            return 0;
        } else {
            $password = sha1($pass);
            $stmt = $this->con->prepare("INSERT INTO `users` (`id`, `username`, `password`, `email`, `phone`) VALUES (NULL, ?, ?, ?, ?);");
            $stmt->bind_param('sssi', $username, $password, $email, $phone);

            if ($stmt->execute()) {
                return 1;
            } else {
                return 2;
            }
```

```
34          }
35      }
36
37      public function userLogin($username, $pass){
38          $password = sha1($pass);
39          $stmt = $this->con->prepare("SELECT id FROM users WHERE username = ? AND ↩
                password = ?");
40          $stmt->bind_param("ss",$username,$password);
41          $stmt->execute();
42          $stmt->store_result();
43          return $stmt->num_rows >0;
44      }
45
46      public function getUserByUsername($username){
47          $stmt = $this->con->prepare("SELECT * FROM users WHERE username = ?");
48          $stmt->bind_param("s",$username);
49          $stmt->execute();
50          return $stmt->get_result()->fetch_assoc();
51      }
52
53
54
55      private function isUserExist($username, $email){
56          $stmt = $this->con->prepare("SELECT id FROM users WHERE username = ? OR ↩
                email = ?");
57          $stmt->bind_param("ss", $username, $email);
58          $stmt->execute();
59          $stmt->store_result();
60          return $stmt->num_rows > 0;
61      }
62      //Test for authentication before getting courses
63      private function getFiles(){
64          $stmt = $this->con->prepare("SELECT course, topic, filename FROM Files");
65          $stmt->execute();
66          $stmt->store_result();
67          return $stmt->get_result();
68      }
69  }
```

Code-block 7: DbOperations

.4.4 DbConnect

```
1   <?php
2
3       class DbConnect{
4
5           public $con;
6
7           function __construct()
8           {
9
10          }
11
12          function connect(){
```

```
13              include_once dirname(__FILE__).'/Constans.php';
14                  $this->con = new mysqli(DB_HOST, DB_USER, DB_PASSWORD, DB_NAME);
15
16                  if (mysqli_connect_errno()){
17                      echo "failed to connect with database".mysqli_connect_error()↩
                            ;
18                  }
19
20                  return $this->con;
21          }
22      }
```

Code-block 8: DbConnect

.4.5 Constants

```
1  <?php
2      define('DB_NAME','android');
3      define('DB_USER','admin');
4      define('DB_PASSWORD','Smartcaching-2019');
5      define('DB_HOST','127.0.0.1');
```

Code-block 9: Constants used in PHP