Educado

An Educational Mobile Application

Anton Olsen, Charlotte Thaarup, Johan Nissen Riedel, Kevin Thor Hansen,

Laurits Lippert, Mathias Gigas, and Pouya Goran

April 11, 2023

AALBORG UNIVERSITY
STUDENT REPORT

Department of Computer Science
Aalborg University
http://www.aau.dk

Title:
Educado: An Educational Mobile Application

Theme:
Complex Front- and Back-end

Project Period:
5. Semester

Project Group:
cs-22-sw-kbh-5-gr1

Participant(s):
Charlotte Thaarup
Johan Riedel
Kevin Thor Hansen
Laurits Lippert
Mathias Gigas
Pouya Goran

Supervisor(s):
Daniel Russo

Copies: 1

Page Numbers: 73

Date of Completion:
April 11, 2023

Abstract:

This project is a continuation of the Bachelor project by Daniel Britze and Jacob Vejlin Jensen published on May 26, 2021 [17]. Educado is a mobile application developed primarily for a specific user group; Brazilian waste-pickers that want to improve their knowledge by exploring and completing educational courses that cover different topics, such as finance, health, sustainability and etc. This report documents our journey in developing a mobile application for those users. By utilizing the Agile Software Engineering practices combined with technologies such as React Native, Expo, Node and MongoDB, we strove to develop a working mobile application consisting of a complex front- and back-end that embed the needed functionalities to enable the users to create an account, explore various educational courses, completing them in a tailored user-friendly environment and storing the content locally on their mobile phones to ensure access in case of lack of internet. The mobile application is the result of work done by three individual project groups that gathered in one large team to combine their efforts in order to realize a solution, thus the report will also cover the disciplines of the Agile Software Engineering methods.

Contents

1 Introduction

Written in collaboration with all groups

One of the world's largest landfills lies in Brazil [10]. The landfill was closed in 2018 which is a positive development in regard to the current climate crisis and the general health of the people working and living in or near the landfill [31].
When the landfills were closed, the waste pickers who lived and worked there were instead provided with jobs at the newly created recycling centers to sort the waste. This has led to a lower income for the waste pickers in general, even though they were hired at the recycle facilities. Furthermore, the payment went from a daily to a weekly salary, which meant that the waste pickers had a harder time managing the money that was available to them. The waster pickers do not have many other options as many of them do not have any education so they can get another better paying job [18, p. 3].

This problem identification has led to the creation of the *Educado* project, a digital learning platform for waste-pickers in Brazil. It began in 2019 when students at Aalborg University collaborated with students at the University of Brasilia to create a mobile application that can be used to educate waste pickers.
In order to complete this, an app was created to provide basic education while completing courses. This is meant to help these people improve their standards of living. The app should be intuitive and contain audio and video clips instead of text when possible, to make it more available for people who cannot read very well.
A web application was also created, allowing "Content Creators" to build, modify and publish courses, all from a web browser [17].

This year, three groups at Aalborg University are collaborating to continue this project and its vision. The project's goal is to clean up and refactor the existing code and database, as well as redesign the mobile and web applications while adding new functionality.
As we are working on a shared project, some sections in the report have also been written in collaboration between two or more groups. These sections' beginning and end will be clearly marked throughout the report.

1.1 Important concepts

Throughout the report, we will use terminology from Scrum and Agile Software Development. These terms will be presented here so the reader has a basic knowledge of the different kinds of software development methods. In this project, we are working as members of a development team following (while learning) Agile practices. This section briefly describes key concepts of the Agile framework called Scrum that we are meant to work within. For in-depth explanations of these concepts, explore original resources on "*The Agile Manifesto*" [1] and Scrum [3].

1.2 General Concepts

The following concepts fall into a more general category:

- The **Product Backlog** is a list of all the items or work that needs to be done to achieve the desired state of the product.
- A **Sprint Backlog** is a subset of the Product Backlog containing the work required to fulfill a sprint's goal.
- An **Increment** represents the finished valuable work the Developers have completed during a sprint.
- A **Definition of Done** formally describes the state an Increment has when its quality is what is required for the product. It is about the activities needed to ensure a certain level of quality of the work being done on the backlog items before they can be considered an Increment.
- The **User Story** is not described in the Scrum Glossary because it is not a mandatory part of Scrum [5]. However, it is perfectly compatible with Scrum, as the user story aims to express requirements from a user's perspective. It is what the user needs from the system. In Scrum, it is the product owner's responsibility to relay this information to the developers [4]. In this project, we, the developers, have made the user stories based on our understanding of the system and then confirmed them with the product owner.

- The concept of **Scaling** becomes relevant when a developer team or software product grows in size and complexity, possibly becoming unmanageable. A scaling framework attempts to set up the rules or structure for managing these challenges. One such framework, Nexus [2], minimally builds upon Scrum to enable three to nine teams to work together on a single product.

1.2.1 Roles in Scrum

Key people and their roles in Scrum are listed below:

- The **Scrum Master** is the head of a Scrum team. First, among his peers of developers, he is responsible for the effectiveness of the team.
- The **Product Owner** (PO) ensures that the team produces a valuable product by being the link between developers and stakeholders.
- The **Developers** are the members of the scrum team creating the product.
- A **Scrum Team** consists of a Scrum Master, a PO, and the Developers.
- The **Stakeholders** are people outside of the scrum team, who have knowledge of or an interest in the product. They are represented by the PO and should play an active role at Sprint Review.

1.2.2 Events in Scrum

Defining activities in Scrum that have been central to this project.

- The **Sprint** is a time-restricted period during which all the other events take place.
- **Sprint Planning** marks the start of a sprint, where the scrum team chooses from the Product Backlog what is most valuable to work on during the coming sprint.
- **Daily Scrum** is a daily 15-minute event to inspect previous work and lay out the work plans for the following day.
- At **Sprint Review** the Increment is inspected by the Scrum Team and Stakeholders, its value assessed, and the Product Backlog updated.
- During **Sprint Retrospective** the Scrum Team evaluates the ending sprint and tries to find ways to improve future sprints.

2 Rich Picture

The previous developers of this project made a rich picture which can be seen in Figure 2.1.

It showcases how the waste pickers are encouraged by the government to make use of these proposed sorting facilities to help educate them on key areas for their well-being such as sanity, economy, and more. Hereby, we see a draft of a solution to the problem. They will be able to use their smartphone to access content that can help facilitate important learning goals.

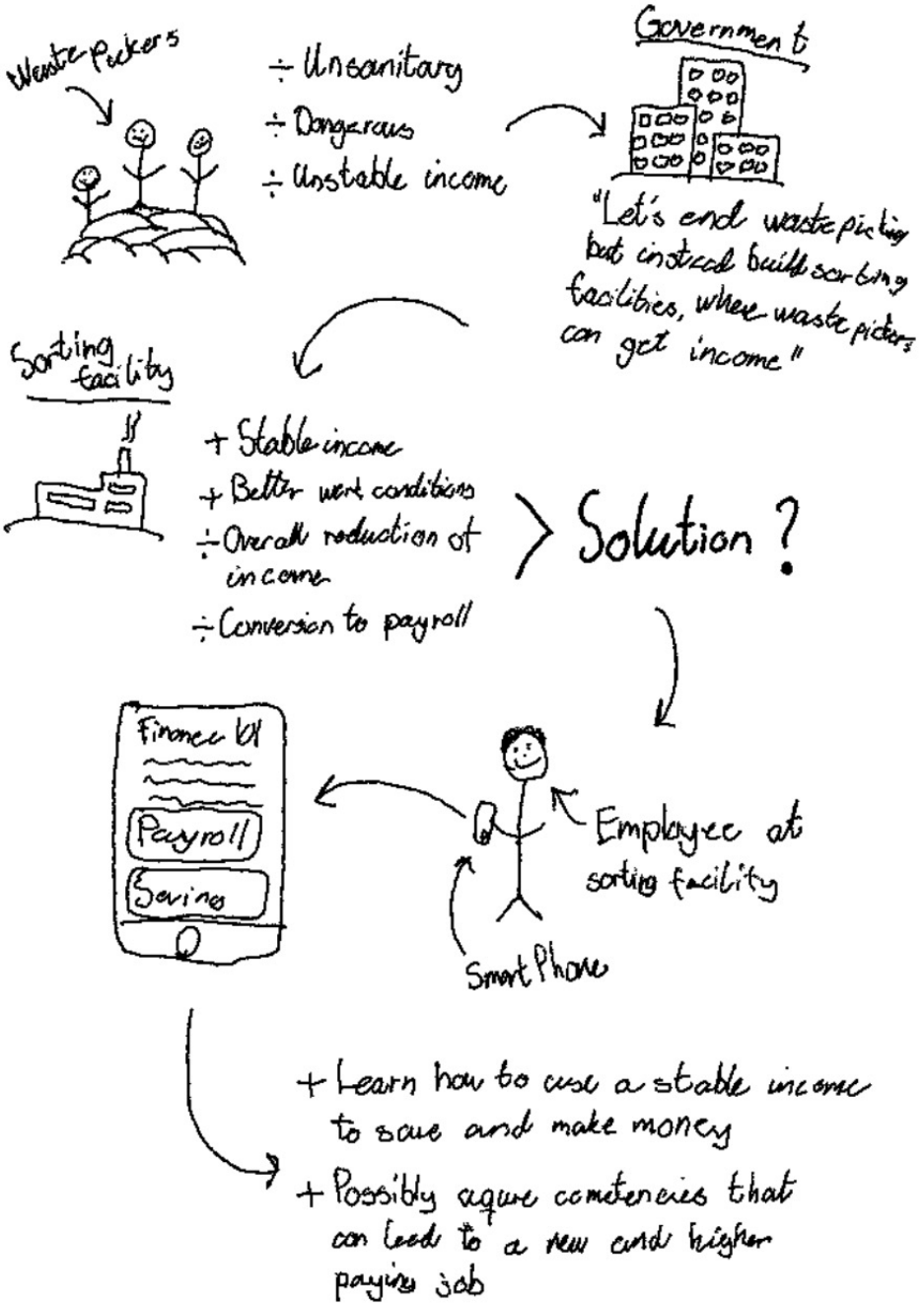

Figure 2.1: Rich Picture from the previous project group [17]

3 Initial state of Educado project

Currently, there are three major components of the project. The web application, the back-end, and the mobile application. There are three different users: The administrator, the content creator, and the waste picker in need of learning. The web application is a gateway for the administrators and the content creators, while the mobile application is the gateway for the waste pickers. The back-end is communicated with by both the web and mobile front-ends. The administrators (and content creators) create the courses for the platform on the web application and the waste pickers attend the courses through the mobile application. The administrator is the one controlling the *Educado* Mobile Learning Platform.

3.1 Web application

When the content creator logs into the web application, they do it through Google's authentication system, which requires a Google account. So far, this is the only form of authentication implemented on the platform. Thus, storing the user data has not been dealt with yet. After logging in, the content creator can see the courses. They are each represented as a square tile with a title and an image (if uploaded). For each course, there is a button with a title that says "edit". Furthermore, on the initial page, the employee has a button to log out in the upper right corner. There are two buttons on the left side - one that takes you to the course overview and one that lets you create a new course.

The Web Application has a somewhat simple design so far, as not a significant amount of time has been put into it.

When creating a new course, you can add a title, add a description, choose a category (such as *Health*, *Finance*, *Technology* and *Language*) and upload a cover image. Each course is divided into sections and each section can be further divided into different components, such as text, images, videos, or audio clips. Each of these components can be added as desired. In total, these elements make up what is essential for a full-fledged course. Furthermore, each section can be moved around, such that the course-creator can structure their course, down to each element. Fundamentally, a course is made up of n number of sections, and each section consists of m number of components, that are either text, videos, images, or audio.

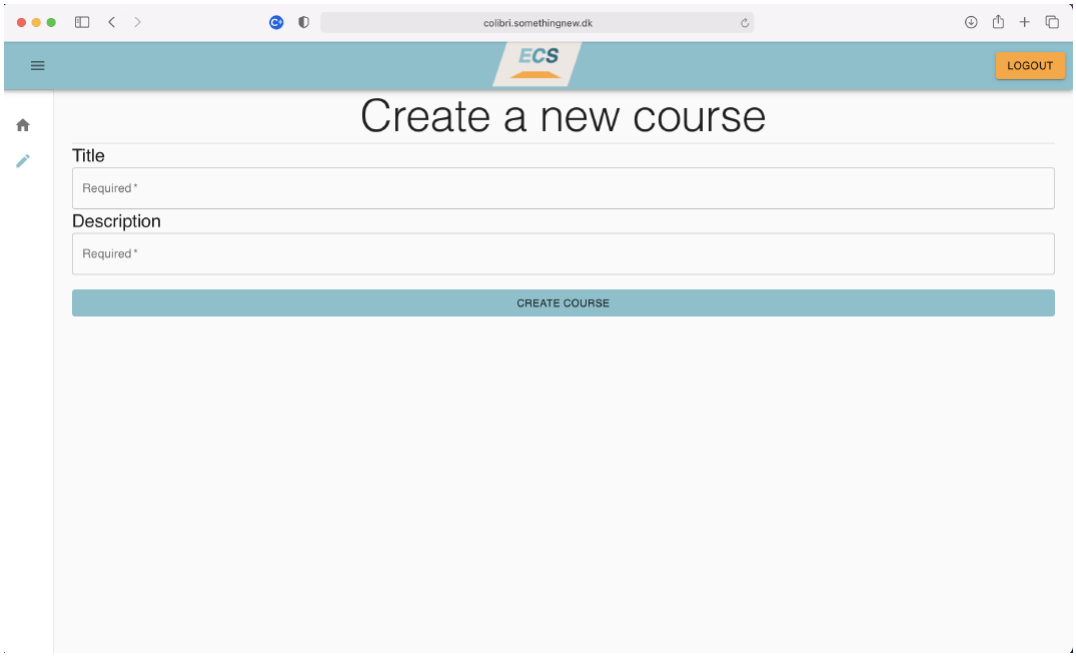

Figure 3.1: Previous state of the front-end web platform

3.2 Mobile Application

Currently, the mobile application is a simple React Native application that gives you access to courses that are created on the web platform. When launching the mobile application, you are presented with a page that lists all the courses you are actively participating in. Currently, all elements are presented in a similar fashion as the web application.
Upon opening the mobile application, we encounter the following main tab as seen in Figure 3.2, where the user is able to see the active courses that they are currently following and how far they have progressed overall. There is a demo course available where the user can see what a course looks like so they get more familiar with the functionalities of the app.

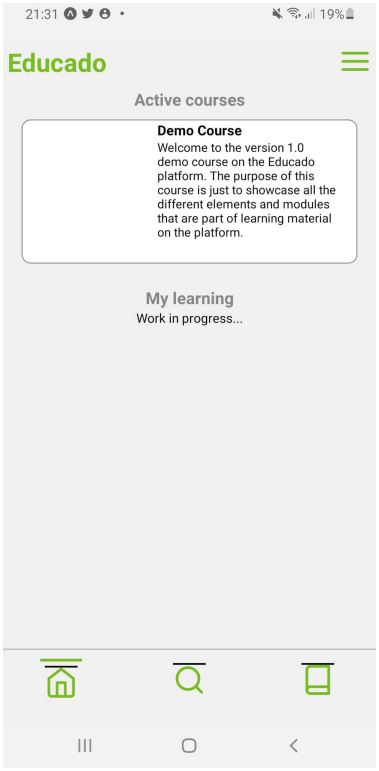

Figure 3.2: Main Tab

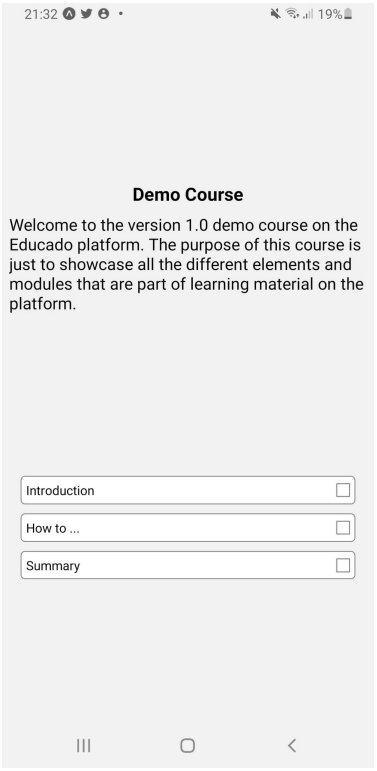

Figure 3.3: Course Tab

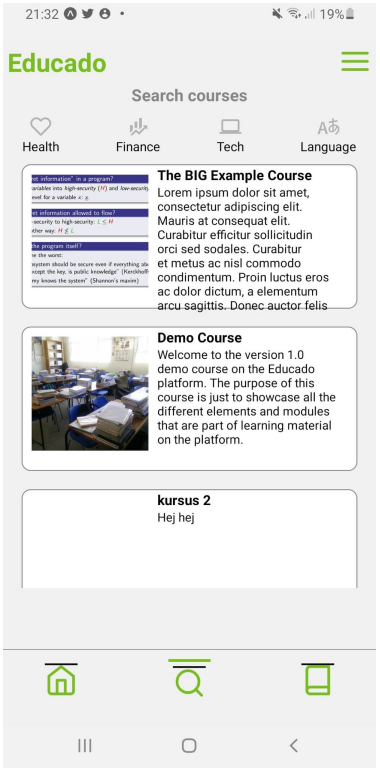

Figure 3.4: Search Tab

Since we know that the users of the mobile application are people with little to no educational backgrounds, it is important to limit the amount of text in order to make it easier for the users to interact with the app. This is not the case with this initial solution as it contains a lot of text, which might cause trouble for the user and can result in them not using it. Besides that, there is a button at the top-right corner where it supposedly should bring down a menu to navigate to different parts of the application, but it has not yet been implemented. Moreover, there are three buttons at the bottom:

- Home Page
- Search
- Library

By clicking on the Demo Course in Figure 3.2, we can see an example of how a course could look.
In Figure 3.3 the user can see the introduction to the course, learn how to complete and then get a summary of the results. By clicking on the back-button the user returns to the main page on Figure 3.2. From there, if the user clicks on the search-button, the page from Figure 3.4 will show up:
On this page, it is possible for users to browse between available courses from different categories.

Class Diagram

This section will give an overview of which classes were featured in the project when the project was handed over. In the project's source code, all three classes have additional variables that are never used in code and thus are not represented in the class diagram.

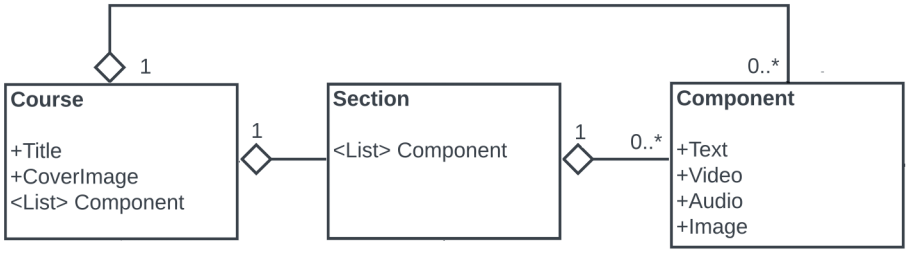

Figure 3.5: Class diagram of the current Educado mobile app

Classes

- **Course:** *Course* is a class that represents a course.
- **Section:** *Section* is a class that represents a section within a course.
- **Component:** *Component* is a class that represents a component in a section, this can be audio, video, images, or text.

Events

The events below provide an overview of the different functionalities that are implemented by the previous project group. It visualizes how different classes are involved in different events. This may give a better understanding of how the classes are related to each other and what they are able to do.

- **Open application on Android Smartphone**

  This is an event where a waste picker is able to open the app on their Android smartphone. As of now, Expo Go can be used to simulate the app.

- **See list of available courses**

  This is an event where a waste picker is able to see a list of courses they can participate in.

- **See overview of course content**

  This is an event where a waste picker is able to see all the sections of a course.

- **Load video, audio, images, and text**

  This is an event where a waste picker is able to load the content that is located in the sections in the courses.

Events	**User**	**Course**	**Section**
Open application on Android Smartphone	+		
See list of available courses	*	*	*
See overview of content	*	*	*
Load video, audio, images and text	*	*	*

Table 3.1: Event Table: “+” Indicates 0 - 1 time. “*” indicates 0 - several times

End of collaboration

4 Limitations

Written in collaboration with all groups

In this section, we describe the limitations that dictate the scope of the project. The project description has set a baseline for the project limitations:

> *Improve the Educado platform and transform it into a great functional Mobile Education solution for Waste Pickers to be tested in the field by the end of the semester in Brazil.*

Working from an existing project

Since this project is based on an existing solution, we are limited to working with the same underlying technologies as the original solution. This means that all further implementation will be built with:

- MongoDB
- Express (Node)
- React
- React Native

The microservice architecture from the original report also represents how the different technologies were implemented in the received project:

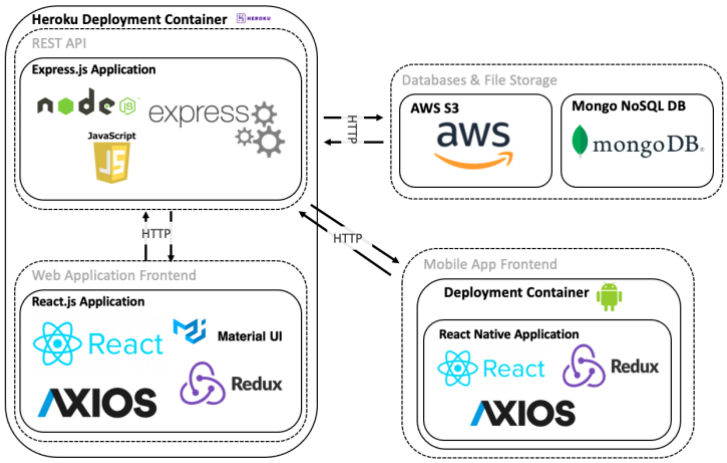

Figure 4.1: Microservice architecture from the Educado Bachelor project [17, p. 47]

The only differentiation in the technology stack that has been made, is adding type support for the React web application with Typescript, as well as some minor changes concerning the styling of the user interface in the mobile application.

Continuous learning

Our course on agile software development ran concurrently with our project, so some sprints will introduce agile development methods that do not appear in previous sprints. As such, this project involved a steep learning curve regarding many new concepts and methods of agile development for multiple groups.

Deployment

Since the project is already an existing, working platform, we are not going to work with the deployment of the system, and as such, further discussion about the current deployment choices will not appear in the report.

Dependencies

As described in the project description, the entire class of Software 5 is going to work on the same project during this semester. This means that the improvement of the existing platform is divided between the three teams, where each group is going to focus on a specific domain of the project. Group 1 will work on both front-end and back-end of the mobile application and assist with creating a link of information flow between the other two groups' implementations. Group 2 will focus both back-end and front-end for the web application, for content creators who want to create educational courses for the waste pickers in Brasilia. Group 3 will work on creating an engaging mobile application for the waste pickers, and will primarily be in charge of the front-end side of the App.
Therefore, some of our main concerns in this project are to balance the work between fulfilling stakeholder wishes and resolving cross-team dependencies. This means that having a functionally aligned platform in every team is more important then a single group providing new features for a single aspect of the platform.

End of collaboration

5 Development Process

During this project, we will learn and apply different skills and competencies taught during the semester. One of these competencies is "collaboration in a multi-project across groups" [9]. The agile methodology has been applied to the project to facilitate cooperation with other groups; specifically, we have been using the Scrum agile management methodology.

The code for the project is found at https://gitlab.com/A-tech/educado-mobile-application for the Educado repository and https://gitlab.com/A-tech/colibri for the Colibri repository.

5.1 Applying Scrum

In section 1.1, some important concepts used in agile were presented. Here, we will provide a brief overview of how they were used by this project group.

Every group had its own approach, but some parts of it were the same. All groups had a common PO. Each group also consisted of a Scrum Team with a Scrum Master that would change every sprint. There were also common Scrum events such as Sprint planning, the actual sprint; which lasted for 2 weeks, sprint review, and sprint retrospective. Another Scrum event is the daily stand-up. In the first sprint, we had a stand-up each day. This did not work well with the way we worked on the project, so it was changed to being twice a week, every Tuesday and Friday at 9 am. The reasoning can be found in the first sprint retrospective, section 5.3.2.

In the last few sprints, we set up a meeting each Friday at 10 am, where all the Scrum Masters would meet up to share updates about each team's progression, what was pending from other teams, and what we could work together on and help each other with.

Some of the Scrum artifacts were also applied. Each group had its own product- and sprint backlog. Every group also tried to have something for the product increment, even though it may not always have been ready and merged with the other group's code. This will be discussed in section 7 [13].
For each sprint, the sprint backlog will be presented in this report. They have been split into tasks and each task has a reference that marks which sprint it is from and its task number (S[*]-T[*]). Every task has also been color-coded. A green task is a standard user story that can easily be verified by the PO or through usability testing. The blue task can be technical and harder to showcase and verify by using the front-end app. It is instead an important task for us to complete to ensure a better functioning code-base and application.

Another important artifact is the Definition of Done (DoD). Several times, a common DoD for all groups were discussed, but in the end, only the basic version of manual testing was used.
This group also tried to set up our own DoD, which was expanded upon along the sprints, which are presented below. A description of it and how it was actually used can be found in section 7.1.

- Code is written
- Manual testing
- Write tests when it makes sense
- Write documentation for any new APIs
- Peer code review with someone internally
- Code review with someone from another group when possible

Our implementation of Scrum evolved throughout the whole process as we learned and applied it. The above presents the final way of working with Scrum. Each of the following sprint sections illustrates how our understanding and implementation of Scrum grew.

Estimation
Beginning from sprint 3, we began to try to estimate the effort expected from each task. The idea behind estimating the effort is to gain a better understanding of what can be expected to be achieved during the sprint.

There are different ways to decide the effort for each task. A common way is to estimate it using hours or story points [27].

In our case, we went with a modified version of magic estimation. In magic estimation, you can use t-shirt sizes or the Fibonacci sequence to estimate each task. Our version was more simplified, the categorization went from low, medium, high, and critical. During the sprint planning, each team member would come up with an estimation. If there were a disagreement, the people would explain their reasoning and we would all agree on the final estimation based on this.
A discussion about how it was implemented in the project can be found in section 10.

5.2 Combined User Stories Diagram

Written in collaboration with all groups

A diagram was made to give the reader a better understanding of what has happened in the project as a whole. Figure A.1 in the appendix gives an overview of the user stories that have been completed during the six sprints. The blue 'Team COWS' have been contributing to the front-end app. The green 'Team Half Full Stack' has been contributing to both the front app and back-end capabilities inside the app, while also handling the back-end specifics for the app. The orange 'Team Sharp Deluxe's main focus was content creation and they have worked on both back-end and front-end capabilities.

End of collaboration

5.3 Sprint One

Sprint one lasted from the 22nd of September to the 6th of October. The time was mostly spent setting up the project, navigating the existing code base and developing new features for the Educado mobile platform.
To organize the different tasks related to the development of the project we had several meetings with the product owner and came up with a set of user stories which can be seen below.

S1-T1: Front-end Login	
Task	As a waste-picker, I want to have the ability to log in to the application, so that I can use the application.
Acceptance Criteria:	• Design a minimal login page • Login needs two fields (phone, password) • Password has to be not visible to read

S1-T2: Registration	
Task:	As a new waste-picker, I want to register using my personal information, so that I can use the application as a WP.
Acceptance Criteria:	• Account should be added to the database • Password needs to be stored securely

S1-T3: Persistent Login	
Task:	As a waste-picker, I want the application to remember my credentials the next time I try to log in, so that I do not have to write my credentials multiple times.

S1-T4: User Validation	
Task:	As a system administrator, I want to validate the users' login information, such that they use proper formatting when signing in
Acceptance Criteria:	• Username field needs validation

S1-T5: Validate with database	
Task:	As a system administrator, I want to validate with the database that the user exists, so we don't let any non-registered waste pickers log in.

S1-T6: Timestamp	
Task:	As a system administrator, I want to save a snapshot of the time the user signed in, so we know when they last logged in successfully.

Start Date: 22/09-2022

End Date: 06/10-2022

S1-T1: Front-end Login

The existing Educado mobile platform featured a very simple front-end with little functionality connected to the back-end as seen in section 3.

To be able to distinguish between each user, we decided to build a login system. From the meetings with the PO and the previous project, we understood that many of the waste pickers have trouble reading, therefore it is most suitable to not have much text in the front-end, but as of now, we decided to include it to get a better feeling of how the front-end should look and be organized. We started working on the login screen by designing it in Figma [8]. The intention of using Figma, was to be able to generate code from the designs to use as a starting point to experiment and develop on, but unfortunately, it did not work as expected right away. Therefore we decided to just design the login screen from scratch again by coding it using React Native [12].

The following two figures are the screens we designed in Figma. Figure 5.1 was intended as a startup screen for the mobile application, so this would be the first screen the user sees when opening the app on their phone.

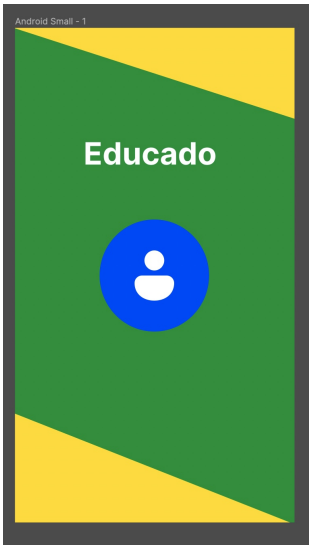

Figure 5.1: Start-up screen designed in Figma

Figure 5.2 is also designed in Figma and is an interpretation of how the login screen could look. From the figure, we can see that there are two possible ways to log in to the platform, either by using a username and password or by using their google account.

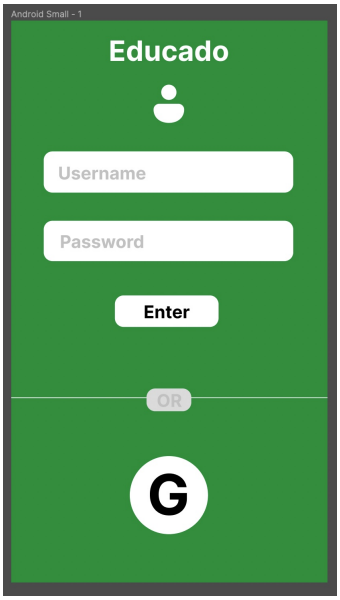

Figure 5.2: Login screen designed in Figma

Later, we decided to have a look at the existing code base and get an understanding of the structure. From here on we decided to code the login screen by creating a file called `LoginForm.js` in the components directory. This file includes the entire design of the login screen. We understood that the project was set up in such a way that you would reuse different designs in the components directory. As we would most likely not be reusing any of the elements of the login screen later we decided to place all the different elements in this single file. To display the contents of `LoginForm.js`, we return it in the screens directory file called `Login.js`. The screens directory's purpose is to display different views and navigate between them in the app. To run the app we used Expo [7], which allows us to use our laptops or smartphones as simulator. Each change in the code will be reflected on the device almost instantly, which makes it easy and responsive to develop the front-end.

Figure 5.3 showcase the login screen that was coded and designed in React Native. Here, the functionality of using google to login has been removed after discussions with the back-end team.

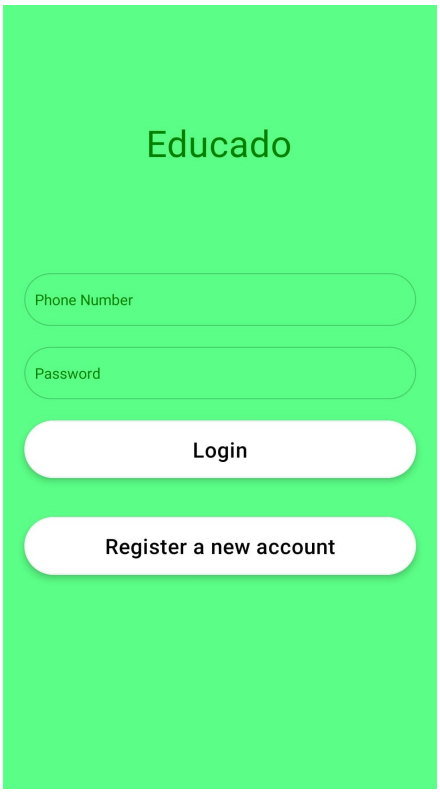

Figure 5.3: Login Screen in 1st. sprint

S1-T2: Back-end Register New User

For this sprint, only the back-end functionality for user registration is in scope, which can also be noticed from the acceptance criterion seen in the user story.

Before being able to implement this in the back-end repository of this project, we first had to read and understand the already existing code and how their database had been structured. As mentioned in section 1, the current solution consists of two platforms that users can interact with. One is a website only meant to be used for content creators, etc. while the other is the Android app that is the focus of this report.

Currently, there already exists a way of login and registration for users of the web platform. This is done through the use of email/google account [17, p. 64]. It was discussed if this should also be implemented as the login method for the mobile application but was ultimately decided against. The reasoning for not using this is firstly because the original project learned that the target group has little to no education when it comes to reading and writing [17, p. 8]. This may prevent them from using an email account.
Secondly, in 2022, a statistic for email usage, from October 2021 to March 2022, in Brazil was published by *Statista*. Their research department found that 60% are sending and receiving emails [29]. In the same time frame, 93% had been using instant messaging, which is deemed the main internet activity in Brazil [30]. This does not mean that the people who have not sent and received emails in that time frame do not have an email account. But, most of them certainly have a phone to be able to use instant messaging, which is why this was deemed the better first choice.

MongoDB
As we wish for the app users to be able to register themselves, we would need a way to store their information so they can continue to use the login they created the first time accessing the app. In order to do this, the project uses MongoDB as its database.

MongoDB is known as a NoSQL database (“Not only SQL” or “Non-SQL”) and is useful for storing large amounts

of data in an easily scalable way, but it stores it in an unstructured way, which means it does not conform to a specific format[1].

MongoDB were already used as the database when the project was handed to us. Their arguments for originally using MongoDB are as follows:

- "*Maintainable and effective NoSQL data structure*
- *Effective integration with Express*
- *Simplicity of use*" [17, p. 42]

Implementation
It was discussed how to best approach app user creation. There already exists a model for users in the directory, which is the one used for the web application. After discussing it internally and with Team Sharp Deluxe, it was decided that the best solution was to create a new model for the app user.

The first step for this was to create the model. To do this a schema was defined. Every time this model is called with a valid input, a new document will be added to the `AppUser` collection in the database. In Figure 5.4 the fields used in each document for `AppUser` can be seen. `phone`, `password` and `timeOfLogin` are all fields that have been specified in the schema. `timeOfLogin` will be discussed in section 5.3. In this schema, the `_id` has not been applied but is automatically added as a field for the `ObjectId`.

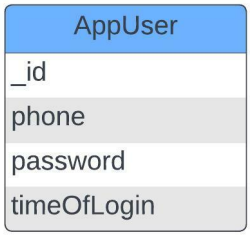

Figure 5.4: Schema for an AppUser record in MongoDB

To be able to use the model, an endpoint was needed. The endpoint contains an URL and whenever the user interacts with the front-end part, the router can detect this change and render a new view that matches the route.
Inside the `appAuthRoutes.js` file, a POST request was created for user registration. A few things are happening inside the actual endpoint. First of all, a new instance of the `AppUser` model is created. It collects the data input and saves it in the database. It also runs checks to see if the user was created successfully. Otherwise, it will catch the error.

For security reasons, the password should not be stored as it is in the database. This is because there is a risk that the application can be hacked and that would leave all the users' login information completely available to hackers. The passwords, therefore, need to be encrypted before being stored [28].
To begin solving this problem the library `bcrypt` was added to the project. This library uses a bcrypt algorithm to hash passwords. Hashing works by taking the input from the user and then turning it into a different string using encryption. But there is some risk with only using hashing. If two users have the same password their password would be hashed to the same string. And, even though hashing is a one-way function it is still possible for hackers to figure out the password with a dictionary attack [28].
Salting can be used to make passwords even more secure. Salting adds a random and unique string to the user's password before it is hashed. This means that the hashed password is now even more secure than without

[1] https://www.mongodb.com/nosql-explained

salting, and even though several users may have the same password, it will be hashed into different strings.

An application called Postman[2] was utilized to test the POST call. It is known as an API development tool, and through this, it is possible to make HTTP(S) requests to test the REST API that has been created for the project. A POST request was created and provided with the specified HTTP from the code. Then the required input, which is a phone number and password, was written into the body of the request. If it is successful as in the example from Figure 5.5, it will return with a 201 status code and provide an overview of the record that will be stored in the MongoDB `appusers` collection.

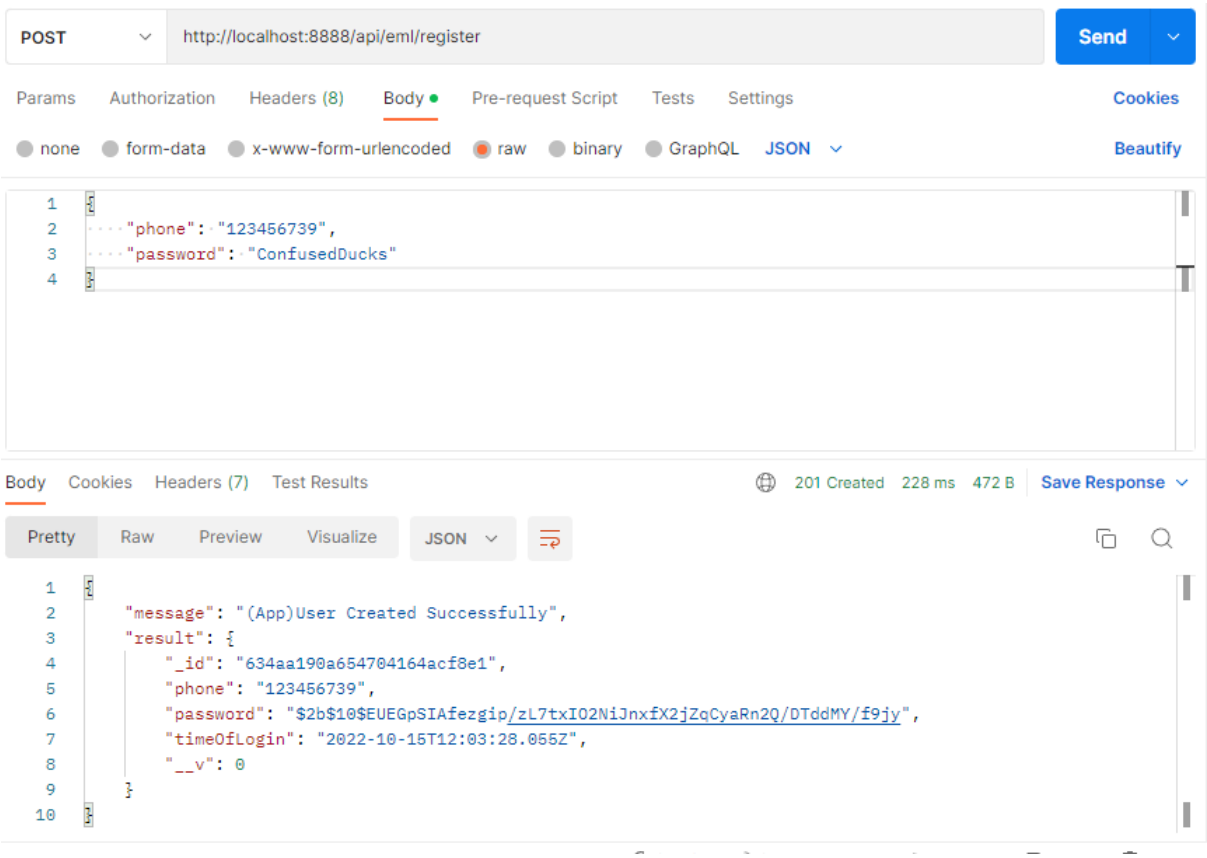

Figure 5.5: An example from Postman showing a POST request and how the record will be stored in MongoDB

S1-T3: Persistent Login

This was not a user story we were able to work on in this sprint. It will be placed in the product backlog until the back-end login implementation has been completed.

S1-T4: User Validation

In section 5.3 the registration for the app user was created. In the back-end schema for the user, the required validation for each field was added. Below is an example of the validation for the phone number used to register the user.

```
phone: {
  type: String,
  require: [true, "Please provide a phone number"],
  unique: [true, "Phone number already exists"],
  minLength: [8, 'Must be at least 8, got {VALUE}'], // For Brazil it needs to be 10
  maxLength: [11, 'Must not be longer than 11, got {VALUE}']
}
```

[2] https://www.postman.com

If a user tries to register using improper formatting from what has been specified here, they will not be able to register their account and will receive the matching error depending on their validation issue.

S1-T5: Validate with database

This user story focuses on actually getting the information about the user from the application when they try to register/login, and then add or compare it to the database.
The issue here was that we were only aware of how to save the information if the back-end and front-end code were in the same repository, but here they were shared over two. We did not complete the increment in this sprint.

S1-T6: Timestamp

This is based on the user story with the reference S1-T6. In Brazil, they have a data protection law called LGPD (Lei Geral de Protecao de Dados), which is somewhat based on the data protection law from the EU, GDPR [19]. The focus here is how to handle personal information, which is something that can be used to identify a person. The database is currently only storing a person's phone number. But, at some point, it should at least also contain the worksite of the waste picker. Since this project may at some point be used in other countries aside from Brazil, it was decided as a good practice to try to implement features that could help future developers easier comply with LGPD and GDPR.

One thing that should be considered is how long to store the user's information. According to GDPR users' data should be stored "no longer than is necessary" (General Data Protection Regulation, article 5, 1.e, 23.5.2018). Even though there is no specific time limit for how long you may store the user's data, it is not supposed to be stored if the user no longer accesses the account. That is why a snapshot of the last time the user was logged in should be stored in the database. Then it can be decided that if a user has not been accessing their account for a certain amount of time, they are no longer using the application and their data should be deleted.

To implement the functionality, the login time was added to the app user schema and given the type `Date`[3], see Figure 5.4. The next step is to get the actual time of login. In the API requests for registration and log in a variable was created and was set to be equal to the Date, which will be the exact time the user last logged in, and then it updates the user's record.

5.3.1 Sprint Review

The first sprint review was held as a meeting in the group room where all three groups were gathered. For each group's presentation and review, the PO was present as well as our semester coordinator to facilitate communication. During the review, each user story from the backlog was presented and examined. The items, that were completed and thus successfully showcased were validated and marked 'complete'.

The user stories we worked on for this sprint can be seen at the beginning of this section. Four increments were developed successfully and were validated by the PO (S1-T1, S1-T2, S1-T4, and S1-T6). There were some issues with the second user story, which we failed to showcase successfully due to a small error. This error was quickly fixed, showcased, and validated right after the review.

The S1-T1 was presented on the computer using an iOS emulator. The group was instructed to showcase on an android smartphone from this point on, but the item was validated.

The second item was validated concurrently with the fourth item. This was done using the software 'Postman' and showing a snapshot of a user schema in the database on MongoDB.

The third user story was not fulfilled and thus moved to the sprint log for sprint #2. Same for the fifth item that was moved to the product backlog.

[3] "A Number that represents milliseconds since 1 January 1970" (mdn web docs, https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date)

5.3.2 Sprint Retrospective

For this sprint, we have been grateful for the workflow the SCRUM framework provides, especially since we are a relatively large group of 7 people and must cooperate with 2 equally large groups. Of course, this first sprint has also revolved around understanding and incorporating the framework into our own workflow and group structure.

The following are some concrete thoughts on how the first sprint went, and what we want to do differently for the second one.

For the first sprint, we had a daily stand-up at 10:30 am. This was changed to Tuesdays and Fridays at 09:00 am.
This change was made to raise the quality of each stand-up meeting. We found that having a daily stand-up disrupted most people's workflow, and since one does not necessarily work on their issue every single day, there would more than often be nothing of relevance to report or discuss. The new time slots have been selected under the assumption that everyone will always have new increments to report.

We have appointed a new Scrum master, as planned. We will continue to switch Scrum master after each sprint.

As we progressed in sprint 1, we gained a deeper understanding of the code base that was handed to us. We have familiarized ourselves with the project structure and the technologies used for the mobile- and web application.

Something that might be relevant to exercise for our next project where we inherit a code base, is to have a 'sprint 0' wherein we have little to no user stories to work on, but simply familiarize ourselves with the code. Perhaps even do refactoring and plan and discuss requirements with the other groups.

5.4 Sprint Two

The main focus for this sprint was to finish implementing the important parts of the CRUD-operations[4]. It was also important for us to learn how to access the logic from the back-end repository in the front-end repository. The tasks for this sprint can be found below:

S2-T1: Delete account	
Task	As a waste-picker, I want to have the ability to delete my account, so that I know that the application is LGPD/GDPR compliant.

S2-T2: Log out	
Task	As a waste-picker, I want the option to log out of the application once logged in, so that I do not automatically sign in on devices that are shared.

S2-T3: Login	
Task	As a waste-picker, I want to have the ability to log in to the application, so that I can use it.

S2-T4: Registration	
Task	As a waste-picker, I want to have the ability to register as a new user, so that I can log in to the application going forward
Acceptance Criteria:	• Add name (nickname) field • Add Phone number field • Add password field

[4]Create, Read, Update, Delete

S2-T5: Smart Caching	
Task	As a waste picker, I want the App to Download and Store the content temporarily whenever I have access to internet, such that I can access a certain amount of the content when offline
Acceptance Criteria:	• Download sessions ahead when wifi is available • Delete previously completed sessions simultaneously • Persistent storage of access token on phone

S2-T6: Validate with database	
Task	As a system administrator, I want to validate with the database that the user exists, So we don't let any non-registered waste pickers login (cont. of S1-T5)
Acceptance Criteria:	• User model in DB • Signin/signup API routes • Persistent storage of access token on phone

Start Date: 06/10-2022

End Date: 20/10-2022

S2-T1: Delete Account

As discussed in section 5.3, there was a focus on preparing to keep the application LGPD and GDPR compliant. This also included giving the users an option to delete their account if they no longer wish to use it.

Back-end Implementation
Beginning at this increment, the way the API calls in the back-end are created will be different from sprint 1. In sprint 1, the new features were created following the setup that already existed. But in future sprints, the principle of the MVC model should be used when creating features in the back-end.

MVC stands for Model, View, and Controller and is a design pattern primarily used when working on websites. This design pattern was chosen for a few different reasons. First of all, it was a pattern that some members of the team already had implemented in previous projects. It was also chosen since the line between front-end and back-end had to be divided as clearly as possible. The benefit of this pattern is that if structured correctly, all the logic of the program will be split into easily manageable parts for both front-end and back-end. This is also known as Separation of Concerns (SoC) [11].

Figure 5.6 presents an overview of how each part of the MVC pattern works together. Shortly, the View is what the users of the app will be able to see and interact with. This is the front-end part of the application. When a user interacts with the app the controller will either update the view or the model. The controller is basically what contains all the logic of the application. The model contains the data of the app, and it should update the view if there are any changes to it [11].

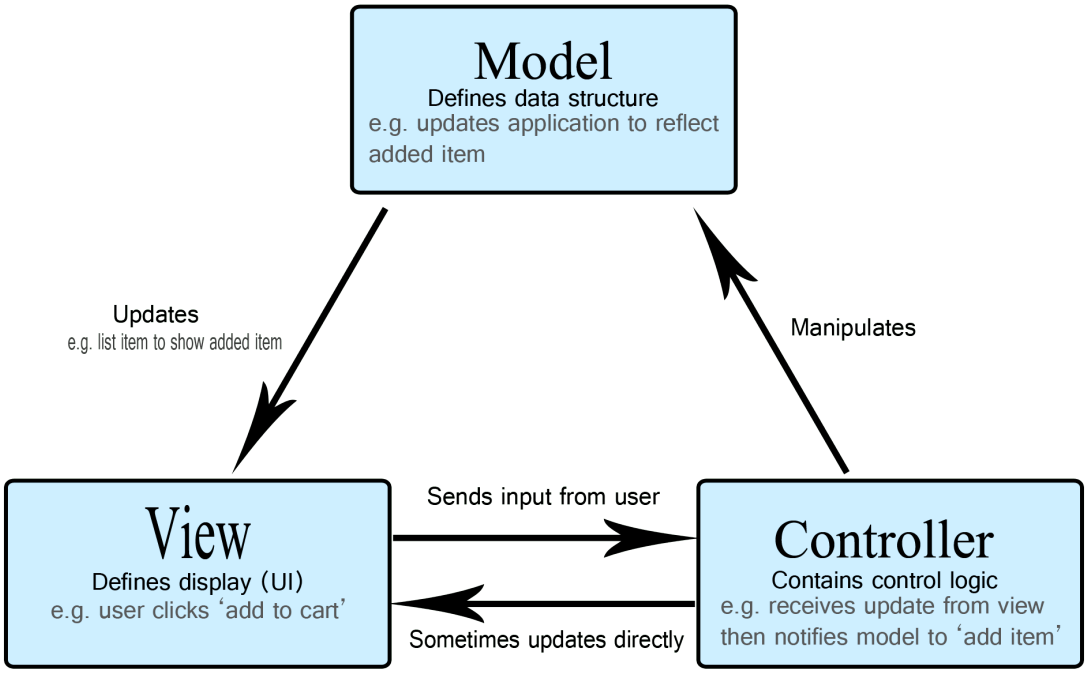

Figure 5.6: An overview of how the MVC-model functions from mdn web docs [11]

To begin implementing the MVC model, we added a controller file. This controller is working together with the `AppUsers` model described in section 5.3. The delete user feature is supposed to work such that when a user interacts with the view and requests their account to be deleted, it will go through an API DELETE request (Another HTTP Rest API method like the Post request). The route for this has been set up in the back-end of the project. When this router is called it will call the function `deleteUser` from the `AppUserController`.
It is important that the id of the user is also sent to the controller through the delete request. This id is used to find the user in the database. If the user is found it will return a 200 success code. If the user id does not match anything in the database it will return a 400 status code. The controller will then update the model which should affect the view of the user in the app.

To test the functionality of the program another Postman API test was created. A unit test was also written in order to be able to confirm its functionality.

S2-T2: Log out

Front-end Implementation
The logout button was implemented such that the user is able to log out of their account. This will ensure that if the user hands over their device to another person, this person will not be able to see anything personal on the app, and they will have to register or log in with their own account.

First off the button was created to get a feeling of where it should be placed and what it should look like. This meant that we first introduced some basic functionality, such as redirecting the user to the login screen if they tapped on the logout button. This only means that the screen that is shown will change, the state of the user object in the database will not be changed. The button was added to a simplified profile screen, such that when the user may want to change another setting they will be shown the opportunity to log out as well.

S2-T3: Login

Following S1-T1 in the previous sprint, we wished to finalize this increment with the needed back-end implementation. The actual connection and validation with the database can be read about in section 5.4.
For the back-end login of the mobile, the same structure as with the registration was followed. A POST endpoint was created. When the endpoint is called it should return a body that contains a phone number and password. First, the phone number is compared with the phone numbers in the database. If no phone number is found, a 404 status code is returned. Otherwise, it will continue and use a bcrypt function called `compare`. It takes the password from the body and compares it with the hashed password saved in the database.

If all comparisons complete successfully, the next step will be to find a way to get persistent login for the user. This is a key feature, if it is not implemented it could potentially be a reason for the waste-pickers to stop using the app. The idea behind this feature is that the user should only log in once, and should only be logged out when they themselves press the log-out button.

It was decided the make use of a JSON Web Token (JWT). We chose this option as it is the one where we found the most documentation on proper implementation. It is also this approach the back-end web group uses for their login. The people behind JWT describe it as follows:

> "*JSON Web Token (JWT) is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object.*" (auth0 by Okta [16])

A JWT contains three parts; a header, a payload, and a signature. The header normally contains the type of token and which signing algorithm it uses. The payload contains something called claims. In this case, the claim contains information about the user. Lastly, the signature is a way to make the token secure and takes some encoded information and also a secret that will be specified somewhere not publicly published. Then we can use this signature to confirm that the token that is generated and received is actually from the correct user [16].

When a user writes in the correct phone number and password we sign a JWT token. Since the payload contains some user information, we need to inform the token of what it should contain. In this case, it contains the user's id and phone number.
To fulfill the signature we need a secret as well. At this stage, a hard-coded and not random secret was used to ease the process of testing. This is not how it should be in the final implementation, but the other back-end group works on setting up the implementation for a secure token secret.

To actually confirm the token is correct before the user can log in, a middleware file was created. It requests the token, decodes it, and tests if it matches the expected token and secret. Then it will confirm if the decrypted token matches the user that is logged in. Lastly, an API endpoint that uses this middleware was implemented, and then it was possible to test it in Postman.

Even though the above implementation could be used for persistent login, there are some security flaws. First of all, there is no expiration time on the authentication token. Ideally, this token should expire 10 minutes after the user logs in. Then a new refresh token should be created for the user as long as they are not logged out. This token's life span can be longer than the authentication token [15].
As before mentioned, the secret for the token is available in part of the code that is being pushed to the repository. This means there is a risk that somebody with malicious intent can find the secret code quite easily.
These two issues are important to fix from a security standpoint, but for functionality, this implementation will work. As already mentioned, one of the other groups is implementing a more secure setup for tokens, and it is possible to use part of their code for the mobile application. There were two reasons why we started working on persistent login before they finished their part. It was one of the PO's main feature requests. And, the logic behind how we handle the token in a login should not change when they finish their code, so we can just merge their work in with our work afterward.

The token will, in the end, be sent back to the app user and then in the front-end implementation, they can store the token to allow the user to have persistent login.

S2-T4: Front-end Register New User

After implementing the login screen in the previous sprint, we decided to implement the register screen in this sprint. Similar to the login screen this screen features a couple of text fields and buttons. But what is different is that this screen adds an extra text field for the user to type in a username they want to be referred to while using the app. Secondly, the buttons below the text fields have different functionality than the ones on the login screen. The upper one will let you register a new user on the app after the user has typed their phone number and password in the text fields above, the username is not required for registering in the app. The lower button will let the user go back to the login screen, in case they accidentally went to the register screen, even though they already have a user profile they want to log into.

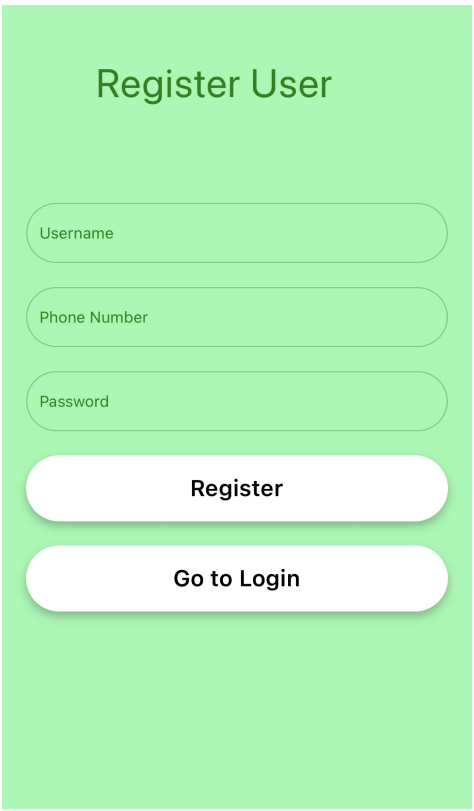

Figure 5.7: Register screen in mobile app

S2-T5: Smart Caching System

The exact functionality of the smart caching system was not set in stone at this point. The general idea was to find a way to store content locally and later decide how much content would be stored at a time and for how long. One idea was to have a download button in a course, allowing users to download an entire course manually. Another idea was to download the entire course automatically once a user had started it.

S2-T6: Validate with Database

From what we had from the previous sprint we wanted to work on the front-end login, so that, it is not just a set of forms and buttons without any functionality behind it. We wanted to connect the front-end of the login screen to the back-end. This meant we needed to set up various routes and make API calls to the back-end for communicating any changes or actions done by the user in the front-end. Below we can see how the different routes in the `userApi` are set up. We can see the URL used in each of the routes, which is set to be the IP address of the local network of the computer it is running on. In the future, it should point to a domain hosting the back-end.

```
import axios from "axios";

const prod = 'http://educado.somethingnew.dk'
const testOld = 'https://ancient-basin-06516.herokuapp.com'
const test = 'http://172.20.10.8:8888' //Change this to your LOCAL IP address when testing.
const local = 'http://localhost:8888'

const url = test;

export const registerUser = async(obj) => {

    const res = await axios.post( url: url + "/api/eml/register", obj);
    return res.data;

}

export const loginUser = async(obj) => {

    const res = await axios.post( url: url + "/api/eml/login", obj);
    return res.data;

}

export const deleteUser = async(id) => {
    const res = await axios.delete( url: url + "/api/eml/delete/" + id);
    return res.data;

}
```

Figure 5.8: userApi.js

5.4.1 Sprint Review

Just as in sprint 1 (section 5.3.1) the sprint review was completed with all groups present. The user stories that were being worked on in this sprint can be seen at the beginning of this section.

Task S2-T3 were completed during this sprint. The back-end implementation had been added for these user stories, but they were deemed not to be secure enough for proper use, but this will be added by another group in a later sprint. This was discussed in section 5.4.
The user story for S2-T1 has only been implemented in the back-end part of the project as discussed in section 5.4. But, it does currently not have any functionality in the front-end part of the project and could therefore not be validated.
S2-T5 is a very important user story for this whole project, but it took more research than originally thought. There also needs to be very close cooperation between the front-end and back-end teams in order to get it working properly.

The tasks S2-T2 and S2-T4 were both tested and validated with the PO. It was validated by letting the PO sit with the application and test the functionalities. Implicitly S2-T6 was also validated as the registered user that was created in the sprint review was also visible in the database.

5.4.2 Sprint Retrospective

After completing this sprint, we had a sprint retrospective with the other groups once again. A general observation was that there was not a lot of teamwork between the groups. There had previously been some small meetings among the groups, but they were often somewhat spontaneous and only to get a brief understanding of what the other teams worked on.
It was agreed upon that moving forward all groups would have a meeting once a week. 2-3 people from each group should be present for these meetings. This number was decided because there is a risk that there will be too many different conversations going on at once if everybody was present. But, if only 1 person from each group is participating they may forget some important information or not be able to relay the information properly to the rest of their own group.

Another thing that was brought up in the retrospective was that it took a long time to get through the sprint review. During the first two sprints, all groups had different ways to structure their backlog. It was agreed that everyone will begin to structure it the same way beginning at the next sprint. This will make it easier for the PO to get an overview of what needs to be validated and what still requires work.

Lastly, the groups all agreed on a new Definition of Done (DoD). Currently, each group has been working on their own branches with their own structure and design. From here on a new branch will be created and this will work as the main branch for this project. When working on a feature we can then branch out from there, but before being able to merge it back the other group should check the code for a peer review and confirm it is working and following an agreed-upon structure that will be presented in section 5.5.

Internal Retrospective
Apart from the retrospective that was done with all the groups, our group also had a small retrospective about how the team had worked together during the sprint and what could be done differently or what had worked well.

We agreed that it worked well with having stand-up meetings each Tuesday and Friday as these are the days that are the best fit for everybody's work- and lecture schedules.
The group sadly suffered from sickness during this sprint which affected nearly half of the group, including the designated Scrum master. Because of the loss of labor, the workload that was planned for this sprint ended up being too much to complete as seen in the sprint review.

Another thing we noticed was that the way our backlog and workflow were structured did not work efficiently if somebody was sick. There was a high risk that nobody would take their assigned task because it was not communicated clearly what they were working on since they could not participate in stand-up meetings. With the new backlog structure it should be easier to follow along and the people that are sick or not able to participate need to communicate clearly what they have completed and what still needs to be done.

Apart from the update of the DoD with the other groups, it was decided internally that we would have some extra DoD. This is especially done because the group both have front-end and back-end work, so we need to make sure it is easy to follow along on what is happening. The agreed-upon DoD is as follows:

- Manual testing
- Write tests when it makes sense
- Write documentation for any new APIs
- Peer code review with someone internally.

5.5 Sprint Three

The sprint lasted from 21/10 to 01/11, which meant it was two days shorter than our other sprints. The main focus in this sprint was to begin merging the increments that all groups had done and make certain the application were still functional.

S3-T1: Profile Page	
Task	As a waste picker I would like to be able to see my profile shown on a separate page, in order to see what data is connected to my profile and edit in case something is incorrect or not up to date
Acceptance Criteria:	• Choose a profile picture • Change user info • Choose a username
Effort	Low

S3-T2: Persistent Login	
Task:	As a waste picker, I want the application to remember my credentials the next time I try to log in, so that I do not have to write my credentials multiple times.

S3-T3: Password Strength	
Task	As a waste picker I want the app to provide feedback on my password's strength and show requirements for a safe password when registering as a new user in the app.
Acceptance Criteria:	• Length: minimum 8 characters • Needs to contain at least one letter
Effort	Low

S3-T4: Delete account	
Task	As a waste picker, I want to have the ability to delete my account, so that I know that the application is LGPD/GDPR compliant. (continuation of (S2-T3)
Effort	Low

S3-T5: Merge Front-end	
Task	Merge our code in the Educado repository with the other group's code.
Effort	Medium

S3-T6: Merge Back-end	
Task	Merge our code in the Colibri repository with the other group's code.
Effort	Medium

S3-T7: Organize Back-end Schema	
Task	Organize back-end schema for users so that we can have a way of checking what courses that are currently active for the current user, which they have already done, and which to download next when wifi is available
Effort	Medium

Start Date: 21/10-2022

End Date: 01/11-2022

5.5.1 Stand-up

In section 5.1 some of the events for using Scrum were discussed, and it also included a daily stand-up event. This has been used in all sprints, even though it was decided to only be done twice a week (please refer to retrospective from sprint one in section 5.3.2). From this sprint and onward, there will be a section with a short summary of what was discussed.

- **21/10/2022:** This was the first day after a new sprint had begun. The focus was delegating tasks. These tasks included the user stories and tasks from the start of sprint 3, but also who was going to assist in merging our code branch with the other group and also refactoring the already written code.
- **25/10/2022:** There has not been much progress since last stand-up. Since Friday 2 more people in the group have fallen ill, which means we are now missing 3 people from the group.
  The merging of back-end is almost complete. There will be a focus on proper database schema design after this has been completed successfully. Front-end is focusing on fixing the bugs on the login screen and user profile.
- **28/10/2022:** Since the last stand-up meeting, the two branches for the back-end for team sharp deluxe and half full stack have been merged into one, to be able to combine the work that is done in the two groups into one product.
  We are still missing 3 people that have fallen ill, which has made it challenging to complete the different user stories that were assigned to our group at the last sprint planning.
- **01/11/2022:** Due to the sprint review being this same day at the time slot we normally have the stand-up meeting, we did not feel the need to have one as we just updated each other on the progress done in the project.

S3-T1: Profile Page

Front-end Implementation
As the user story describes, we wanted to give the possibility for the user to see their progression, and account details along with functionalities such as logging out and deleting their account, in a separate screen called the Profile Screen. The ability to see the progression was not prioritized for this sprint, since the PO had explicitly asked us to focus on merging the work of the other groups into one "working" application. So we decided to focus on the other aspects of the Profile Screen for this sprint. By continuing work done in the previous sprint, we added functionality to the Logout and Delete Account buttons, so the user has the possibility to log out or completely erase their data from both the local phone storage and the database in the back-end.

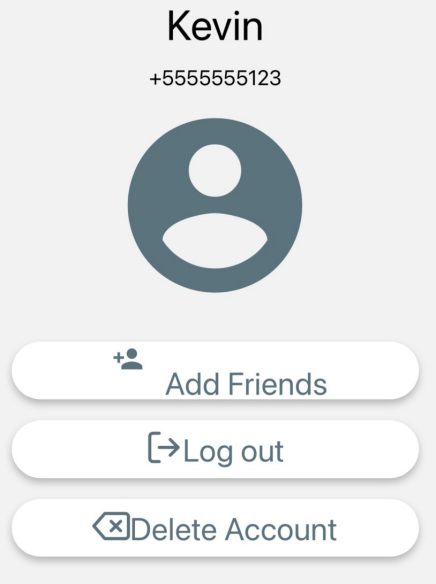

Figure 5.9: Logout Button on the Profile screen

S3-T2: Persistent Login

In the previous sprints, we made the functionality for the user to be able to register a new account and then login. But what we did not manage to create was the functionality that allows users to stay logged in, until they choose to log out. This was made possible in this sprint by actually saving a unique token generated in the back-end, at the local storage in the front-end. So now, when a user logs into the app, as always they get back a unique token from the database. This token is then saved in the local storage of their phone and whenever they close and reopen the app, at the login screen the app will look after the token to see if that exists in the local storage. If it does, it will not ask the user to log in again and redirects the user to the home screen. If the user goes to the Profile Screen, they have the possibility to log out. And only when they click and accept to log out, the saved token gets erased from their local memory. Then they will have to log in again upon next time they open the application.

S3-T4: Front-end Delete Account

The Delete Account functionality has been worked on in this sprint. Partly because of the GDPR rules, but also in order to give the possibility for the user to erase any data saved on the database or their own local phone storage. When a user registers on the mobile application, their username, phone number, and password gets saved on the database. A user state will be created in the front-end that holds all user information including account data and their current progress in the courses they are enrolled in. This user state along with all the data on the database is deleted when the user taps the `Delete Account` button.

S3-T5: Merge Front-end

After reaching a certain point in the development of the mobile application, it was requested that the work done by our group along with the work done by the COWS group should be merged into a branch called "`frontend-merge`" on Gitlab. The process of merging the work was fulfilled without issues. One person from

each group was responsible for organizing and meeting with the other group's representative. The code from each of the group's branches got merged into the merge branch. Upon merging there were about 16 conflicts in the code. These were manually resolved by going through each of them and refactoring the code. Overall, the merging of the increments done by both groups helped everyone get a better understanding of the application and each group's coding styles. Also, it helped both groups to align the increments and make it easier to work together on shared user stories and future merges.

S3-T6: Merge back-end

As agreed in the last sprint, the main shared goal for this sprint was merging the increments from each group together. Just as described for merging the front-end. The main branch used for this group was merged into a common branch called `dev`. All merge conflicts were fixed and afterward, both groups tested and confirmed all functionalities still worked. By merging the code together we now have access to the proper handling of JWT tokens described in section 5.4, which can now be used for refactoring in an upcoming sprint.

S3-T7: Organize back-end schema

The plan for this user story is that we needed to research how the app user schema could be structured in order to be able to save the user progression for a course. It was again quickly discussed with the web team group if we should make a common user profile and give each user different permissions depending on the user type. Because app users at the moment can only create an account using a phone number this was put on hold and can be used in future works.
Next, it was discussed if the progression should be saved inside the app user schema or if they should instead create a new schema with the progression, which has an `objectId` type that refers to the user id of the user that is enrolled in the course. Then in the app user schema, there could be an array of `activeCourses` ids. It was decided against using this option as there would be created a new document each time a user enrolls in a new course.

It was decided to keep the active courses tracked inside the app users' own document instead of creating a new schema. Each user will have an array of `activeCourses`. Each course they enroll in should contain a reference to the course and it also should have a `boolean` for whether or not the course is complete. This is illustrated in figure 5.10. It can be expanded so that each course inside the active courses has an array of sections and their completion. Then each section should also have an array with all its exercises and their completion status.

The actual implementation of the schema was not completed in this sprint but instead put back in the product backlog for a later sprint. But these initial thoughts will be beneficial for completing this increment when it is put back in the sprint backlog.

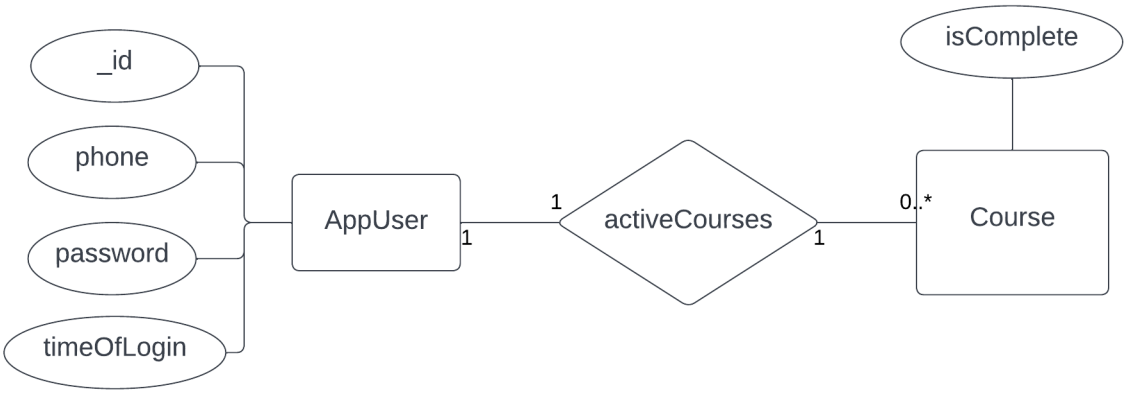

Figure 5.10: A diagram showcasing an example of how the tracking of progression in courses could be organized.

5.5.2 Sprint Review

The third sprint review was done together with all groups present and the previous developers of the project instead of our PO. The stories that were reviewed can be seen at the beginning of the section.

The main theme for this sprint was to merge the different groups' work on the web- and mobile applications, so we are left with two products consisting of all three groups' work. Tasks S3-T5 and S3-T6 were completed in this sprint, which means we are meeting the expectations of this sprint's overall goal judging by the main theme.
Apart from finishing the main theme task, we also implemented a profile page for the user (S3-T1). This was accepted despite us not fulfilling the acceptance criteria, as these were deemed not feasible for this project. The deletion of an app user (S3-T4), and persistent login (S3-T2) was also accepted.

The remaining tasks were not fulfilled. One of the remaining tasks will continue into the next sprint, which is the user story S3-T3. S3-T7 will be put back into the product backlog and will be up for consideration in the upcoming sprints.

5.5.3 Sprint Retrospective

After completing this sprint, we have had different conversations between the groups and with Daniel and Jacob, and have come to the conclusion that moving forward there will be a bigger focus on functionality rather than security and front-end design details. From these talks, we have planned to hold meetings on how we want to store and retrieve data in the app from the web application, so the user can retrieve them while being offline on their smartphones. Also, we have planned to align our front-end design principles between our group and the fully dedicated front-end group. It was also mentioned to keep on holding meetings between the teams where the scrum master from each team will be present and will pass details to their respective groups afterward.

Internal Retrospective
Internally in our group we still had sickness and people being absent because of work outside of the project. This meant we effectively only had two people working on some of the major tasks that were being put on for this sprint, which met the overall theme for the sprint we agreed upon between all the groups and PO. We have also agreed to have a bigger focus on the report, as this has been de-prioritized quite a bit this sprint. So we would like to update the report, describing the different tasks we have been working on for this sprint and the previous one as well.

5.6 Sprint Four

One of the main overall goals for all groups was to align and refactor the merged code in order to try and give the code base a more streamlined design. Also, from the last sprint review, we learned that we should not focus on the security side of the project, but just on functionalities. Our own main priority is to try to research and generate a proof of concept on how to download a course.

S4-T1: Download	
Task	As a waste picker, I want to able to download a single course, so that I am able to see it even though I am not connected to the internet
Acceptance Criteria:	• Research and design proof of concept
Effort	Medium

S4-T2: Password Strength	
Task	As a waste picker I want the app to provide feedback on my password's strength and show requirements for a safe password when registering a user in the app (cont. of S3-T3).
Acceptance Criteria:	• Length: minimum 8 characters • Needs to contain at least one letter
Effort	Low

S4-T3: Caching	
Task	As a waste picker, I want the App to Download and Store the content temporarily whenever I have access to the internet, such that I can access a certain amount of the content when offline.
Acceptance Criteria:	• Download sessions ahead when wifi is available • Delete previously completed sessions simultaneously
Effort	High

S4-T4: Align Front-end	
Task	Align front-end to match frameworks and design.
Effort	Medium

S4-T5: Refactor Back-end	
Task	Refactoring of back-end so all code has a common architecture.
Effort	Medium

Start Date: 01/11-2022

End Date: 15/11-2022

5.6.1 Stand-up

- **04/11/2022:** The first stand-up of the sprint focused on delegating the different tasks and making sure everybody had a common understanding of the goals for this sprint. The tasks that were delegated can be seen above.
- **08/11/2022:** It was a quick meeting for catching up on how far each person was with their respective tasks. There has been done a lot of research for some of the tasks, such as S4-T1. The next step is working on a functional implementation for these.
- **11/11/2022:** Everybody has worked on their assigned tasks since the last time. A lot of process have been done especially on S4-T1 as it has been a big focus since COWS the front-end group needs this to be done as soon as possible. But after a not code reviewed merge from the other back-end group, there are some issues with getting course data from the database, which is an important step to actually implement the feature.
  The functionality of S4-T2 has been implemented and just needs to be tested.
  S4-T4 turns out to be a bigger challenge than expected as there are some issues refactoring new design patterns and still saving the app users' info in the MongoDB database.
  It was agreed that an extra meeting should be held on Monday the 14th to prepare what should be shown in the sprint review the day after.
- **14/11/2022:** We had a quick meeting just to catch up on what could be presented at tomorrow's sprint review. The only thing we have that can be presented is the password strength. Everything else we have worked on is refactoring or back-end programming, which should not be presented during a sprint review. The downloading and storing of a video in a course have some issues. There are some bugs in the implementation that returns an undefined value, so currently it is not ready for presentation. In order to try to get ready, everybody who did not have another highly important task to work on was assigned to bug-fix this task.

5.6.2 Cross-Team meetings

In this sprint, we began having more focus on cross-team meetings. This section will provide a quick recap of the main points discussed during cross-team meetings.

- **02/11/2022:** On the 2nd of November we held a meeting with the dedicated app front-end team. We discussed how they would receive data from us in the form of JSON. To be able to implement this we came to the conclusion to make a storage controller. After some research, we decided on using the library called `async_storage`[5].

[5] https://react-native-async-storage.github.io/async-storage/docs/usage

- **11/11/2022:** There were some issues with the dependencies for courses, this was fixed during the meeting so we are able to access them.
  In order to save videos we are using something called Expo File System and we have a storage service for saving JSON files.
  The other back-end group has set up an S3 bucket with some dummy data for videos we can use. They are working on uploading images, videos, and links that will in the future be changed to AWS bucket routes.
  It was agreed that each Friday at 10 am, the Scrum master from each team will join a meeting with the other Scrum masters to discuss dependencies, etc.

S4-T1: Download

This task was included as a "proof of concept" task to see if we could set up functionality to fetch and store a course locally. We wanted to explore several options for storing files locally to see if we could get the functionality implemented in this sprint.
The steps were to first retrieve the entire course from the database through the back-end, including links to the course content. All the metadata would be stored using `AsyncStorage`, since it handles small files efficiently. Larger files, such as videos would be stored locally using expo-file-system, since it is designed to better handle larger files.

S3-T3/S4-T2: Password Strength

This task was initially placed in the sprint backlog in sprint 3 but was completed in sprint 4. The user story describes that the user should be encouraged to choose a password for their account consisting of a minimum of 8 characters and it needs to contain at least one letter. The implementation was done using the library called `zxcvbn`[6], which returns a score from 0 to 4, with 0 being the lowest score, resulting in a very weak password and 4 being the highest score resulting in a very strong password. This score is then used to display the password strength meter. The meter consists of a colored bar, which will change length and color depending on how weak or strong the current written password is. For example, if the user types in a password only consisting of 5 numbers, the bar will change to the color red and be extended a bit in length, and here it will state what the password strength is.

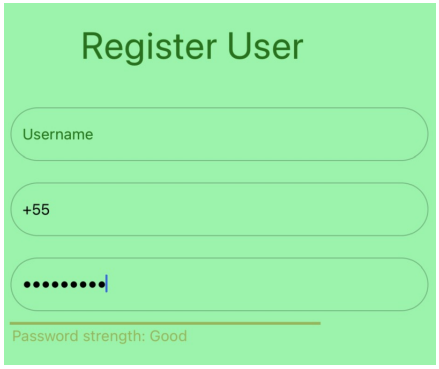

Figure 5.11: password strength meter

S4-T3: Caching

We started the research on how we could bring caching functionality to the app. However, no profound discoveries were made and the task was ultimately terminated by the PO at the end of the sprint since sprint 5 was going to refocus the project towards aligning and merging between the project groups.

S4-T5: Refactor Back-end

In section 5.4, it was discussed how MVC was used as the design pattern. Then in section 5.5.1, the work of the two back-end groups was merged together. During the meetings of the merging and planning of this sprint, it

[6] https://github.com/dropbox/zxcvbn

was discovered that the other group has used Domain-Driven-Development (DDD) and Clean Architecture for their design pattern and general structure. This meant that two different design patterns were used, which we ideally would have preferred to avoid. Their approach has some similarities to MVC but is splitting the logic and domain behavior up even further. DDD and Clean Architecture will be discussed in the "Software Quality Management"-section 7.2.
To attempt to align our work and make the repository easier to navigate for future developers, we agreed to try and continue with DDD and Clean Architecture. This meant we had to set aside extra work to refactor code that was already completed instead of focusing on new features. This could have been avoided if we had been better at planning alignment meetings across groups.

The effort of this user story was set to a medium. This was because we both had to learn about the design patterns and then refactor the code to match it. In the end, understanding not only how these work but also how it was interpreted in this project, took longer than originally thought. It ended up requiring two back-end developers and a few meetings with the other web application group before we were able to refactor the registration and deletion of the app user. These two were completed in this sprint.
The effort for this task in this sprint should have been higher, as we did not have time to refactor the login feature since this required several extra changes compared to registration and deletion. We, therefore, had to set this task into the next sprint backlog.

5.6.3 Sprint Review

S4-T2 has been implemented and tested by the PO who confirmed that it working and can be marked as completed in this sprint.

The important functionality was to download and store videos and courses. This is very dependent on the other groups. S4-T1 focused on researching and designing a proof of concept on how this could be done. Through this, we did find a way this could be implemented, so we could continue working on S4-T3. A lot of code has been implemented and tested, but it still needs to be merged with the front-end groups' code before it can be marked as completed. This will be moved to the next sprint, but it should not have a high effort as most are already implemented. The smart caching part of this will be placed back in the product backlog, and we will instead focus on downloading when the user manually requests it.
S4-T4 and S4-T5 will also both be carried over into the next sprint backlog.

5.6.4 Sprint Retrospective

Written in collaboration with all groups

During the sprint retrospective, at least one member from each group formed a team and discussed what they felt did not go well in this sprint and how we could improve it. A summarized list of some of these discussions has been collected below:

- **Sprint review:** One of the things discussed was the flow of the sprint review. It has happened that during one group's review they have been interrupted with questions or comments. The groups should be allowed to finish their showcases before questions are asked, and ideally, further discussions about future approaches should be done after the review.
  There has also been a noticeable lack of engagement from other teams during the presentations. Everybody should follow along during the review.
- **Communication:** As per request, the number of meetings between groups has increased slightly. However, the meetings have been quite informal as they are usually not scheduled ahead of time and occur in a more spontaneous manner. There has only been held one official meeting in which all three groups partook.
  In order to get a better communication flow, a fixed meeting between all Scrum Masters has been set. For the rest of the project, the groups will have at least one meeting every Friday at 10:00.
- **Backlog:** A topic that has been discussed several times during this project is having a shared backlog between all groups so that every group always knows what the other groups are working on. The PO initially promised to create this with the Trello boards that were shared with him, but we have not yet received them. A person from each group should get together to set up a backlog. Ideally, the Scrum Masters will do it at their weekly meeting.

- **Alignment:** Each group has been focusing on their respective user stories and goals. As a result of this, the common end goal has not received a lot of attention. This also means that everyone has made their own design choices for both the code structure, as well as the front-end design.
  This has led to extra work, as the design patterns should ideally be consistent in a repository so future developers can easily understand the flow of the code.

  There also needs to be better alignment between front-end and back-end developers. If changes need to be made to one of the routes in the back-end repository, a front-end developer has to be informed about this, since it can affect their code as well. Another contingency that will be used from now on is to have shared documentation for all REST APIs, so front-end developers can easily check existing APIs and how they are to be utilized.

- **DoD:** One way to fix our alignment issues would be using a common definition of done (DoD). There has not been any agreed-upon set of rules for this yet. One DoD that will be added is that each repository should have a branch that will mock the master branch of the project. Only the working code should be pushed to this branch. In order to maintain this, each group should appoint a code review master. Their job will be to code review and accept the other group's code before it will be pushed to the master branch. When the code is reviewed it should also be checked to see if it follows the newly agreed upon design pattern, so our code base is aligned.

End of collaboration

5.7 Sprint Five

The goal of this sprint is to integrate and make sure everything we have works and can be delivered.

S5-T1: Persistent Login	
Task	As a waste-picker, I want the application to remember my credentials the next time I try to login, so that I don't have to write my credentials multiple times.
Acceptance Criteria:	• Generate login token • Store login token with the user
Effort	Low

S5-T2: Download	
Task	As a waste-picker, I want to be able to download a whole course So I can access it even when I am offline.
Effort	Medium

S5-T3: Align Front-end	
Task	Align front-end to match frameworks and design (cont. of S4-T4).
Effort	Medium

S5-T4: Refactor Back-end	
Task	Refactoring of back-end so all code has a common architecture (cont. of S4-T5).
Effort	Medium

S5-T5: Debugging	
Task	As a developer, I want proper debugging tools that allow me to set breakpoints, step through code and examine the state of the mobile application while running.
Effort	Low

S5-T6: Dynamic view of course list	
Task	As a front-end developer I want to have a function in the `StorageService` for accessing the locally cached `courselist`
Effort	Low

Start Date: 15/11-2022

End Date: 29/11-2022

5.7.1 Stand-up

- **18/11/2022:** We had a short meeting delegating tasks for the upcoming sprint.
- **22/11/2022:** The other back-end group has fixed the SW3 bucket links, so we can continue our work with downloading links for the app.
  We are also pending the other front-end group, as we need them to review our code before we merge it into the shared branch.
  The focus until the next sprint meeting is to continue to get the other two teams' code to work together and refactor existing code, so we follow the same design patterns.
- **25/11/2022:** The newest front-end implementations have been merged with the other group and are functional. Before when retrieving courses, all courses with all sections were retrieved. Now, only a short overview of each course is fetched. To get the sections a specific course has to be called. Therefore, the code has to be refactored to match the back-end changes, so the app user gets a correct view.
  The back-end code is now following the same design patterns for both groups. There are some issues with getting the login function to work again after the refactoring.
- **28/11/2022:** The back-end refactor has been completed and tested with unit tests. Before merging it into our main branch, we agreed to test the app later today to make sure everything is still functioning.

  The "download a course" functionality is working and will be merged into the developer branch so the other front-end group has access to it.

5.7.2 Cross-Team meetings

- **18/11/2022:** At the sprint planning session on the 15th of November, there was mutual agreement among the project groups to start using each other to do code reviews before pull requests can be accepted. Another important point was to figure out how to do a shared backlog to be better aligned throughout the sprint. However, since this is very late in the project we decided to simply invite each other to our boards, allowing each team and especially each scrum-master to simply check up on the other teams' progress.
  We further started discussing finding a date when we can align our project reports as well as making sure that we can make a final merge of our code branches, so everyone has the same code base to refer to towards the end of the project and everything works.

  After the meeting was completed and the information about what was discussed was relayed to every team, one of the teams decided against using code review from other groups as it would take too long.
- **26/11/2022:** At the meeting we first went through what each group had done since last time and what is expected to be showcased at the sprint review.
  The front-end group and our group have completed the merge of our code and our repositories are working together. Apart from this, they are working on the UI of the application and will be able to showcase it. As most of our efforts have been placed on ensuring the connection between the work done by the other groups, we had discussed whether team COWS should showcase the work done by both groups.
  The other back-end group focuses on user role security and expects to be able to showcase it as well.

As there is only one sprint left we began to align what each group's focus would be. The other front-end group is planning a code freeze for the next sprint, but they are willing to take on tasks if they are needed in regard to aligning functionalities, and if the PO agrees to it.
It was discussed that no new tasks should be accepted at the next sprint if the PO agrees. The focus should only be on existing functionalities and refactoring and improvements.

S5-T1: Persistent Login

In the increment with the reference S2-T3, 'Login' (section 5.4), it was discussed how the implementation was not yet deemed secure. As the proper functionality for handling tokens has been implemented it can now be refactored in our login process. This was completed and validated in this sprint. As this user story is dependent on S5-T4 it will be further described in section 5.7.2.

S5-T2: Download

This task required both front- and back-end functionality to be added. In order for the app to store content locally so the users were not in need of constant internet access we wanted to make a download button. This was also a natural first step toward making a caching system.

Back-end Implementation
First, we made the logic for retrieving the given course from the database based on a course id and made a function in the storage service called `downloadCourse`. This calls the `createDirectory` and `downloadAndStoreContent` functions we made in the directory service. The functions in the directory service does exactly what the name suggests. The `downloadCourse` function first gets the contents and icon from the database and stores it with the help of `AsyncStorage` and afterward creates a new directory for each exercise locally in order to save the video content that belongs to each exercise. For this, we used the expo file system framework. Lastly, we updated the course to be active for the user locally, meaning we could show the user visually that the course had now in fact been downloaded by changing the color of the course and the sections from blue to green.

Front-end
To accommodate the back-end functionality, we created a download button placed inside of a course, just above the sections. The button is made such that it will change its look based on what download state the course is in. So if the course has not been downloaded, it will show a download button in the form of an arrow pointing down. If the button is tapped, it will begin to download the course and a spinning wheel will be shown as the course is being downloaded. When the course has been downloaded, a checkmark will be shown to indicate that the course has successfully been downloaded to the user's storage.

S5-T4: Refactor Back-end

This task is a continuation of the task presented in section 5.6.2. The login functionality still needed to be refactored to match the Clean Architecture design pattern. In an attempt of securing the login process compared to the previous implementation, a new middleware called `passport` was introduced. `Passport` is used for authenticating a user and there are several existing strategies for this authentication. As it was agreed upon to use JWT as the strategy in sprint 2, the `JWTStrategy` was chosen as the most optimal [24].

A strategy for the mobile app was created. It finds the user through their id. Then it serializes the user to determine what data from the received user object should be stored in the token. Then whenever we request the token in the future we deserialize the token, find the user's id and compare it with the user that made the request.
A function for restricting endpoints was also made in an attempt to create secure handling of the mobile app. When the function is called in an endpoint a user will not be allowed to access this page without a valid token.

In section 5.4 the library bcrypt was mentioned. When the other team implemented their login function they decided to use the library crypto which handles the hashing and salting in a very different way. Again, as we wanted to align our work, we also changed the password handling to match with crypto. But apart from changing the library, the app user schema itself had to be changed to store the hashed and salted version of the password so that can be used for comparison in the future.

After finishing the refactoring, it was tested and validated both internally in the group and externally by the other back-end group.

S5-T5: Debugging

Previous sprints' debugging was mainly achieved with print statements i.e. `console.log`. This is generally considered a bad practice and further did not allow us to set breakpoints and examine the state of the application. We were experiencing problems while diagnosing failed unit tests and verifying content and correctness of the Storage Service implementation. We needed proper debugging tools and an effort was required to become familiar with this part of the React ecosystem. After having read the Debugging section of the Expo documentation page, it seemed a good fit for the task would be the React Native Debugger [21]. Setting it up and getting it to work proved to be a frustratingly hard experience, as the debugger itself apparently has an unsolved issue preventing it from working out of the box unless given a specific parameter [22]. The fix was easy but finding it, was not so much. Another issue was that the debugger did not behave as expected, often requiring multiple reloads of the app before the debugger would properly attach to the running app. Eventually, after having dealt with these issues, it worked and allowed us to step through code, examine `AsyncStorage` content and fix issues namely with `StorageService` and provided a more enjoyable debugging experience moving forward.

S5-T6: Dynamic view of course list

Back-end Implementation
In the app, we have an explorer page used to show all the courses you can enroll in. For this reason, we needed to add functionality for updating this list, either if new courses were added/deleted in the database or if one of the courses was downloaded since there was a requirement from the other front-end group to visually show which courses were active. This meant that if it was active, we also needed to update the course list with a new attribute for the course, which was now available locally for the user. To do this we added a function that the other group could then implement and call from our Storage Service. The function is called `getCourseList` and works in the following way:
First, it gets all courses from the database through the API controller. It then iterates over all courses and pushes the relevant data from each course for display on the explorer page. The data gets pushed to a newly created list that is then returned after the iteration is complete. In this process, it also adds an attribute, as before mentioned, `isActive`, which is used to keep track of what is local and what is not.
If the user is not connected to the internet, the `getCourseList` function returns the latest downloaded list from the local storage, so in this way, the user is always able to have some courses available to enroll in.

5.7.3 Sprint Review

The main focus for this sprint was to get the download functionality to work. The PO was able to test and verify that this was now a possibility in the app (S5-T2 and S5-T6).

S5-T1 and S5-T4 are dependent on each other and are not something the PO can easily validate. The PO could validate the login part and that everything still worked after the refactoring of the back-end. The other back-end group could confirm the design pattern and use of tokens were consistent with theirs.

S5-T5 is very technical and was not confirmed by the PO as it is something related to our development process and debugging of the application, but it was finished in this sprint as well.

The alignment of front-end (S5-T3) was still not completed and was instead moved to the last sprints backlog.

5.7.4 Sprint Retrospective

A lot of key functionalities were completed in this sprint, which helped us align our project with the other teams, but it also opens up for the other teams to be able to finish some of their pending tasks.
Compared to the other sprints, we were better at estimating the effort for the tasks compared to the amount of time we were able to set aside to solve them, as we only had one user story we had to carry over. Better estimation was helped along by for the first time since the beginning of sprint 2, having the whole group available for completing tasks.

5.8 Sprint Six

The goal for this sprint was primarily to finalize the previously implemented features. Specifically, we needed to add adequate error handling, and we had to align the front-end design. The only features we needed to implement from scratch in this sprint were the option to delete a downloaded course and track progression. We also wanted to do some tests in regard to static code analysis in this sprint.

S6-T1: Delete download	
Task	As a waste-picker, I want to be able to delete downloaded courses from my phone, so that they do not take up storage when they are completed.
Effort	Medium

S6-T2: Refresh Explore screen	
Task	As a waste-picker, I would like to have the ability to refresh the explore screen, so that I can see the newly added course from the database.
Effort	Low

S6-T3: Progression	
Task	As a waste picker, I want to be able to save my progression in a course, so that I can continue it if a get another phone.
Acceptance Criteria:	• New accessed course should be saved for the user • Completion of exercise, section, and course should be updated and saved • Reflect their progression when accessing the course again
Effort	Medium

S6-T4: Organize Back-end Schema	
Task	Organize back-end schema for users, so that we can have a way of checking what courses that are currently active for the current user, which they have already done, and which to download next when Wi-Fi is available (cont. of S3-T6).
Effort	Low

S6-T5: Align Front-end	
Task	Refactor front-end to match frameworks and design (cont. of S5-T3).
Effort	Medium

S6-T6: Error handling	
Task	Refactor the download function so it also has error handling to avoid crashes.
Effort	Low

S6-T7: Static Code Analysis	
Task	Run static code analysis on the projects. Add a pipeline with ESLint.
Effort	Medium

Start Date: 29/11-2022

End Date: 13/12-2022

5.8.1 Stand-up

- **02/12/2022:** We have begun aligning front-end design.
  ESLint has been set up with a pipeline on the `pipeline-eslint` branch. When used on the storage service branch, there are over 3000 errors, but most can be fixed with 1 command. Before fixing them we want to merge everything together and then we can look into fixing the issues.

  We are working on the delete button for downloaded courses and error handling in the storage service. Lastly, we are also planning on making an update feature for the course list to finish the CRUD operations on courses for the user's view.

- **06/12/2022:** The functionality for S6-T2 with refreshing the explore screen has been created, but is not implemented into the actual view for the app user. The delete download and alignment of front-end have also been completed.

  There have been some issues with figuring out how to best implement the reorganization of the app

user schema to handle the courses the user has downloaded.
Some work has been completed on the error handling, but there were some issues with a commit into one of our main branches, so we had to revert which did cost us some time.

- **09/12/2022:** After the meeting with the other groups on the 2nd of December we have stopped working on the static code analysis and have not implemented a pipeline in the dedicated back-end repository.

  Adding the courses a user downloads into the user document has been added. There is still work to be done on updating the completion status, but should be done later today.

- **12/12/2022:** The progression is all done and reviewed by other team members, but has not been implemented in the front-end.
  The focus for today is to make sure all code works together among all three groups. Then we are going to prepare one demo for all three groups instead of a demo per group for the sprint review.

5.8.2 Cross-Team meetings

- **02/12/2022:** COWS are merging back-end and front-end but should be done today with the changes from the `HFS` branch.
  The exercises are set up to always have 4 answers on the front-end, but from the back-end they can send 2 to 4, so this needs to be aligned.
  Team Sharp Deluxe works on refining the features that are already implemented.

  We discussed static code analysis. Currently, we have ESLint in the pipeline, but we still want to use CodeScene as ESLint is more focused on coding style, whereas CodeScene will catch bad code, and code smells and output a detailed overview of possible technical debt.

- **09/12/2022:** The three Scrum masters from each team held a meeting. The conclusion of this meeting was that we decided to make a full demo for the sixth/last sprint review. This meant that the flow between each group's work had to work harmoniously together, so we could show how a course was made on the web application, and then how one enrolls to take that course. We will make a test-demo on Monday the day before the sprint review. We also planned to delegate all of the chapters that must be the same in every report after the last sprint review.

S6-T1: Delete Download

After implementing the download button we had to equally make a way of deleting the content that was now stored locally on the user's phone. This task consisted of both a bit of front-end and UI work in the design of how the user would interact with the app. We decided on simply pressing the download button again which would now be shown as a checkmark and warn the user if they are sure they want to delete the downloaded course. In an effort to make the design more intuitive, the other front-end group changed this icon to a trash can. When the course is deleted the color of the sections in the course is changed back to blue from green, so it was intuitive that it was no longer stored. This button calls a function in our back-end storage service, and on the front-end, it then calls a function from the directory service, which we had implemented earlier that allowed us to delete entire directories at a time. The directory of course only held the videos stored and the rest we delete through `AsyncStorage`.

S6-T2: Refresh Explore Screen

In order to give the user the ability to fetch the newest courses from the database, we created a function called `refreshCourseList` that is embedded in the `getCourseList` function. So when the user navigates to the explore screen, the `getCourseList` function is called which then first calls the `refreshCourseList` function. In `refreshCourseList` function a call is made to an API which returns the current `courseList` available in the database i.e. the newest list. If the API does not respond to the call because of a lack of internet or other issues, the `refreshCourseList` function returns the latest downloaded course list that should be available in the local storage (`AsyncStorage`). All in all, we managed to create the functionality which refreshes the explore screen to get the newest courses, but we did not manage to implement the function as a UI component. This is something that could easily be implemented as a Future Work; a button component in the explore screen that calls the `refreshCourseList` function upon pressing and re-renders the explore screen.

S6-T3: Progression

Back-end Implementation
Saving the user's progression in a course has been a requested feature from the PO. Since it is possible for the user to download a course and access it again even after closing the app, because of local storage, we needed to find a solution for actually saving the progression in the database, so they can access it in case they get a new phone or delete the app.
The final schema design can be seen in section 5.8.2.

To get the information needed from the user when they interact with a course, the endpoint is created such that it takes the user id and the course id they are enrolling in and places it in the parameters.

When the app user id and course id are found and confirmed, a course object is created that sets the completion to false for the course and also creates an empty array for the sections in the course. In order to populate the sections and exercises that belong to each course we first needed to understand how the schemas for a course were defined.
They are created in a way such that each course, section, and exercise has its own schema. A course has an array where all the sections with a matching `parentCourse` are being added. The same goes for each section. They have an exercise array for all exercises that contain their id as the `parentSection`.

To get all the sections from a specific course, we utilized our knowledge of the array of sections with a reference, so we were able to use the mongoose functionality called `populate`, to find all the references. We could then go through all the sections and create a section object with a false `isComplete` and an exercise array. These objects are placed into the course section array. The same procedure was followed for populating the exercise array for each section.
When all sections and exercises have been found, it will save the course and all the before-mentioned information into the `activeCourses` array in the specified user's document.

If it finds the course is already in the `activeCourses` array when the user tries to download the course, it will return the corresponding ids for the course, sections, and exercises, and if they are completed or not. This can be used by the front-end team to update the view with the correct progression for the user.

In order to update the completion status, an endpoint for both course, section, and exercise was created. When they are called they will just change the `isComplete` from false to true in the user document.

S6-T4: Organize Back-end Schema

In section 5.5.1, which took place in sprint 3, we decided on how to organize the app user schema. This was still deemed the best solution, and the implementation of tracking both sections and exercises and their completion status was also implemented.

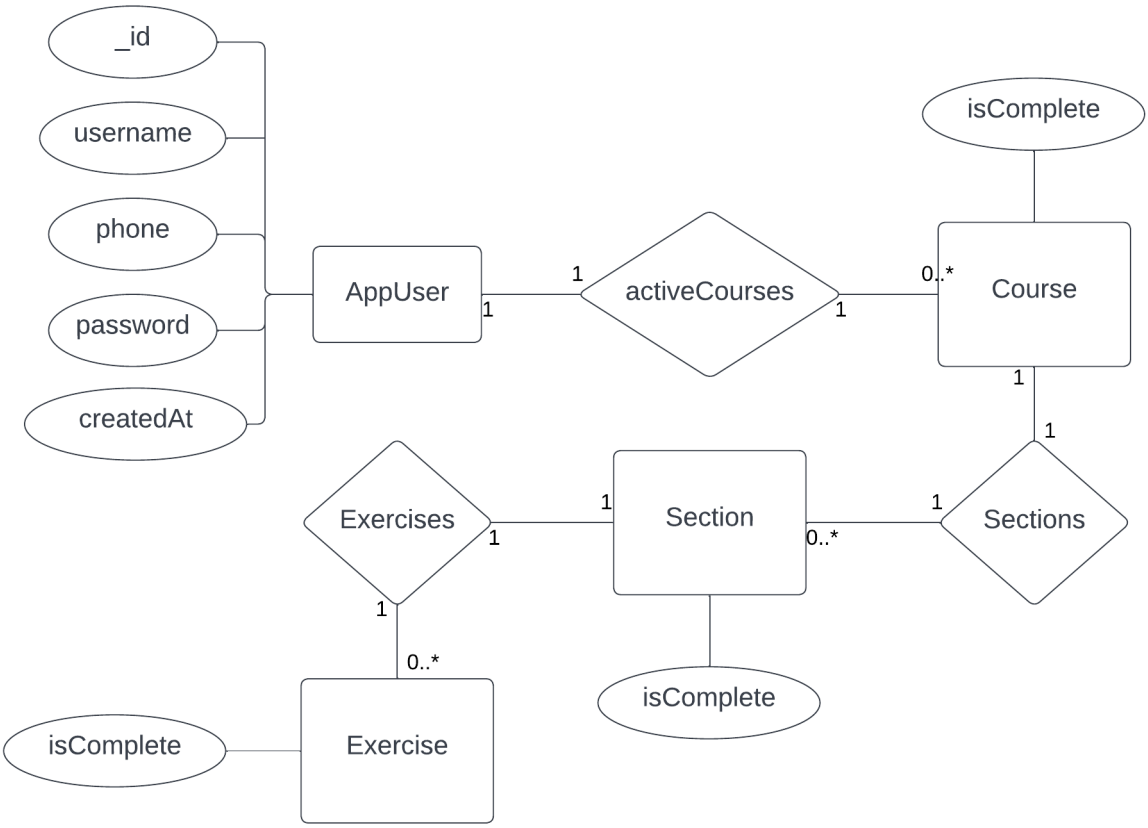

Figure 5.12: An updated version of the diagram showcasing an example of how the tracking of progression in courses could be organized.

S6-T5: Align Front-end

This task is a continuation of task S5-T3, which is derived from task S4-T4. The task has been postponed for quite some time, as we waited to align the front-end until near the end of the project as we felt it was not a very time-consuming task. The task was to match frameworks and design, but as we proceeded we did not feel the need for matching the frameworks, as we were able to create a more aligned design without the need for the same frameworks used by the other groups. We made sure that the colors used in the app matched with what was used by the other front-end group, to create a more coherent look. We have mainly been focusing on the login and register screens throughout the project, these screens are a bit more separated from the rest of the screens you see in the app, as you will only have to login/register once per new device you want to use, and will not have to interact as much with it once you are registered and logged in. As they are more separated we did not feel it was important to use our time for learning a new framework, just to change the look of the buttons to a similar look.

S6-T6: Error Handling

While implementing new features and testing the app, we experienced a number of crashes when performing certain actions. We realized that some of the new functions lacked error handling. One hotspot for this was the `StorageService` component, and thus, we focused on adding error handling to the functions of this component. Upon completion of this task, most of the problems we had run into had been fixed.

S6-T7: Static Code Analysis

In an effort to increase code readability and quality, we decided to set up and configure an ESLint pipeline. After some discussion with the other groups, we adopted the Airbnb JavaScript Style Guide [14]. The result of this was thousands of errors because the code was not written by one developer, and JavaScript has many interesting

language features. Naturally, this meant violating the rules of the style guide. Many of the errors could be auto-fixed by ESLint but a sizeable amount would require manual refactoring, and all of it would require extensive testing and verification that nothing broke as a result of it. We realized it was unfeasible to use ESLint's auto-fix during the sprint, as with thousands of lines changed slightly across files, it would undoubtedly result in many merge conflicts just before the sprint review.

5.8.3 Events

Following up on the events that were presented earlier in 3.2, we have decided to include the final events here at the end of sprint 6. This was done as an attempt to help provide a clear indication of how the mobile application has developed and which new features the user is able to see and interact with.
There are other events such as answering correctly in an exercise, but these are not something we have been a part of implementing, and they are therefore not included here.

- **Open application on Android Smartphone**

  This is an event where the waste picker is able to open the app on their android smartphone.

- **Register user profile**

  This is an event where the waste picker can register a new user profile on the app, where they will be required to enter a phone number and a password, and optionally a name they want to be referred to in the app.

- **Log in as user**

  This is an event where the waste picker is able to log into their account with their credentials once they have opened the app.

- **Logout**

  This is an event where the waste picker is able to log out of the app if they are logged in.

- **See profile screen**

  This is an event where the waste picker can see information related to their user profile. They will be able to see their name if they have chosen one and the phone number they entered.

- **See list of available courses**

  This is an event where the waste picker is able to see a list of courses in which they can enroll.

- **Start course**
  This is an event where the waste picker starts a course of their choosing. Ideally, this should only be possible to do 0-1 time, but as the progression functionality from the back-end has yet to be implemented in front-end, every time the user downloads the course they will start over.

- **Download course**

  This is an event where the waste picker is able to download a course. The course is then accessible locally on their phone and the waste picker does not need Wi-Fi to access the course.

- **Delete Course**

  This is an event where the waste picker is able to delete a course once it is already downloaded locally on their smartphone.

- **Overview of course content**

  This is an event where the waste picker is able to see all the sections of a course.

- **Load video, audio, images, and text**

  This is an event where the waste picker is able to load the content that is located in the sections in the courses.

- **Delete user profile**

  This is an event where the waste picker can delete their user profile from the app, in case they do not want to use the app anymore or would like to register with a different phone number.

Events	User	Course	Section	Exercise
Open application on Android Smartphone	+			
Register user profile	+			
Login as user	*			
Logout	*			
See profile screen	*			
See list of available courses	*	*		
Start course	*	*	*	*
Download course	*	*	*	*
Delete course	*	*	*	*
See overview of content	*	*		
Load video, audio, images and text	*	*	*	*
Delete user profile	+			

Table 5.1: Event Table: "+" Indicates 0 - 1 time. "*" indicates 0 - several times

5.8.4 Sprint Review

S6-T1 for deleting a downloaded course, S6-T4 for organizing back-end schema, S6-T5 which is refactoring front-end, and S6-T7 for static code analysis were fully completed this sprint.

The important functionalities behind S6-T2 with Refresh Explore Screen and S6-T3 for saving user progression have both been implemented, but they are missing the final parts of the implementation for the users to be able to benefit from them.

5.8.5 Sprint Retrospective

Written in collaboration with all groups

We began the sprint retrospective with an internal group discussion about sprint 6. Afterward, each group presented their thoughts on this sprint. This included discussions about what did not go well, but also about improvements from previous sprints.

- **Sprint review:** In the sprint review, it felt more like one product compared to previous reviews. We had a flow where not all groups presented. Instead, we began showcasing the web application for the content creators, and afterward, we presented the shared work of the two mobile app groups together. Here, we could also see the new course created during the web presentation.
- **Workflow:** It has been a very stressful sprint. Not only did we want to try and get a working MVP before usability testing in Brazil, but we also had to try to fix the issues that were presented from the Static Code Analysis. We did have technical debt from previous sprints, which ideally should have been continuously fixed, but at the beginning of the project, the focus was on implementing new features and learning to work together in order to ensure a good product.
- **Communication:** The communication in this sprint was a lot better compared to previous sprints, but there is still room for improvement.

5.9 Final Retrospective

After we had the retrospective for sprint 6 we completed the development process with a final retrospective that covered the whole project from beginning to end.

- **Progress:** One thing we discussed was how we as students had evolved since a similar project in the 3rd semester. Here, we had to work as a single group that solved a real-life problem for a company.
  During the 3rd semester, we had to spend a lot of time trying to figure out how to use the skills we learned in courses such as system development. Whereas in this semester these skills were simpler to apply, which was a great indication of our own development.
- **Sprint 0:** It was discussed that one way to start with a stronger foundation for common goals and better communication would be with a sprint 0. This was not something we were aware could be done before the end of sprint 1. In sprint 0, we should not touch any of the code. The focus should be on communicating with the other groups and getting everyone to agree on a common set of requirements, based on the project we received and the wishes of the stakeholders and PO.
- **Communication:** One of the common topics throughout the development process was the lack of communication. This includes communication among the three groups, but also communication internally.
  Each group's sprint backlogs were not shared with the other two groups. This caused issues as we did not always know what the other groups were working on. Especially in the beginning when we had yet to establish a good communication flow.

  The lack of communication lead to issues where there were risks of groups working on similar tasks, but there were also examples of different coding approaches in the same repository. In order to make it easier for future developers, the naming conventions and approaches should have been discussed before we began the development.
  But, in the last two sprints, the groups started to work more as one unit with one common goal. People from different groups would start sitting together to fix common issues. Also, the communication between front-end and back-end became clearer. If there was something front-end or back-end needed from each other, they would talk together about it to try and find a good solution.
- **Scaling:** In continuation of 'Communication'. We did not have any good tools that could help ease the process of working together across teams. This is of course one of the learning goals for this project, and we did in the final 3 sprints have a weekly meeting with the Scrum Masters from each team. But, one thing that was talked about was that we had hoped there where more information earlier about how we could have used a scaling framework to assist us.
- **PO:** When the project first began, it felt as if the PO was learning his role along with us. Especially in the first few sprints, we were still not clear about a lot of fundamental knowledge from agile, so we had hoped that the PO could be of assistance. At the end of the 3rd sprint, we had a guest lecturer join us for the sprint review. The guest lecturer came with a lot of useful feedback, that helped us, but also was of assistance to the PO. After this, the PO seemed more confident and aware of his role. This along with our expanded agile knowledge made sprint reviews and sprint planning easier to complete.

  Another thing that was discussed in the retrospective was that we would have had great benefits from the PO writing the user stories or setting clear sprint goals for each sprint. We wrote our own user stories

and for the first few sprints, there was no agreed-upon goal for what should be achieved. This is one of the reasons why we started to accumulate technical debt.
It was sometimes not easy to get proper approval for the user stories we had written for each sprint and at times it felt as if the PO was not aware of what was in the sprint backlog.

- **Sprint review:** Just as with the scaling framework, it was discussed that it would have been beneficial to have learned how an actual sprint review is normally handled. There was confusion about how to actually proceed with the sprint review, and it was not until the final sprint that it felt as if the review was about one product and not 3 different products.
- **Pipelines:** It was not until the last two sprints that we learned about pipelines and Continuous Integration. It is normally considered good practice to create a pull request instead of merging code directly into the main branch of the project. It was discussed that it would have been very beneficial for us if we were provided with an option and guideline to set up a pipeline that forced us to do pull requests.

Overall, there have been a lot of frustrations during the project for every group. There was a very steep learning curve, as we not only had to take over an existing project but also had to work together across teams.
In the end, our way of working together still has room for improvement and the product may still have some issues, but through the frustrations and failures, we experienced throughout the project we also learned valuable skills that we could slowly start applying while working on the project.

End of collaboration

6 Architecture

Written in collaboration with all groups

This section focuses on the system's overall architecture which will be handed over and used for the next 5th semester's project. The ISO/IEC/IEEE 42010:2011 standards define architecture as follows:

> "*(system) fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution* [ISO/IEC/IEEE 42010:2011(E), p. 2]"

Architecture diagrams were used to properly represent the architecture of the system. They provide a visual representation of how the system works and which components are interacting with each other.

A representation of a high-level architecture diagram can be seen in figure 6.1. The final three repositories of the Educado system are all represented and the components use HTTP protocols to communicate over different repositories. All communication with the MongoDB database happens in the content platform, but they send and receive this information to/from the dedicated web and mobile repositories.

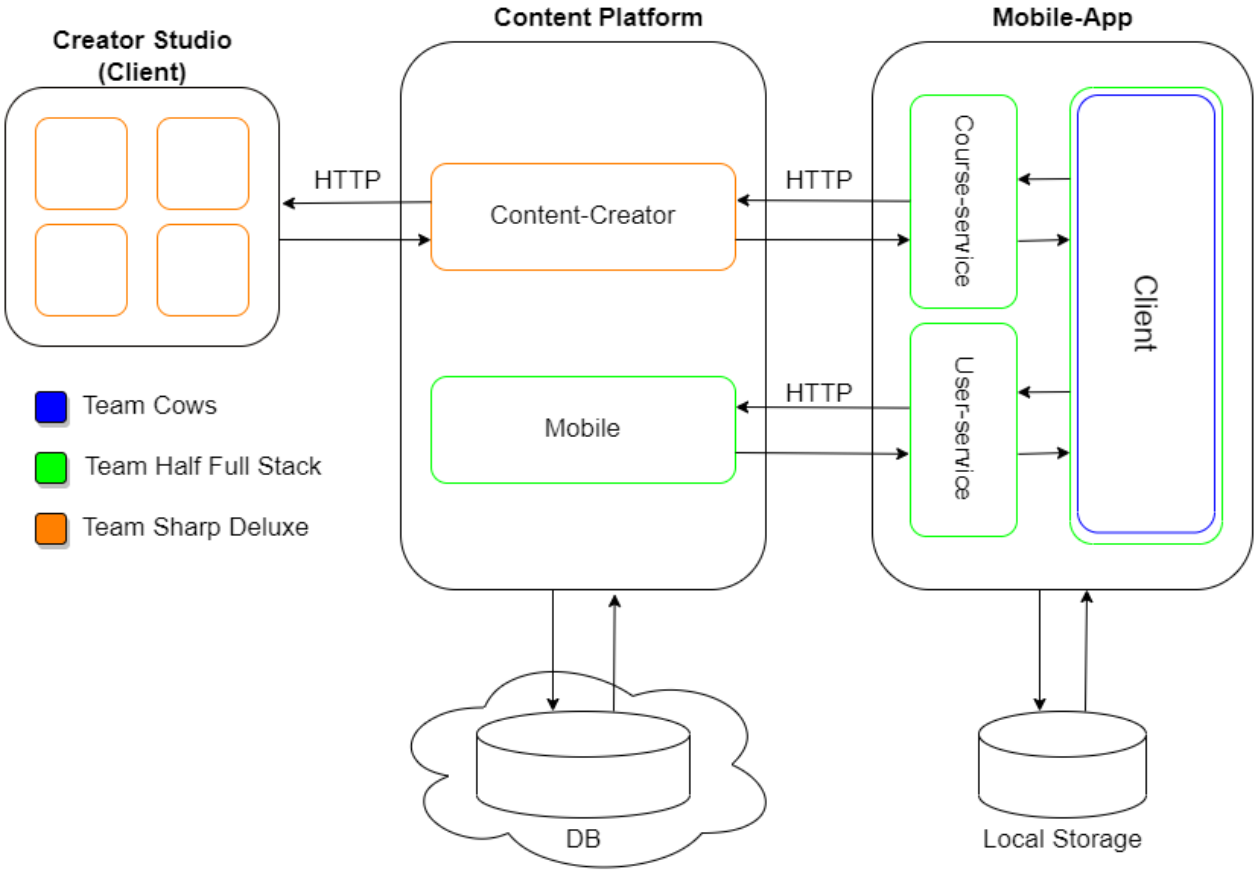

Figure 6.1: A high-level architecture diagram of the Educado system

App-mobile overview

In the following section, a more detailed diagram of the structure of the mobile side of Educado is provided.

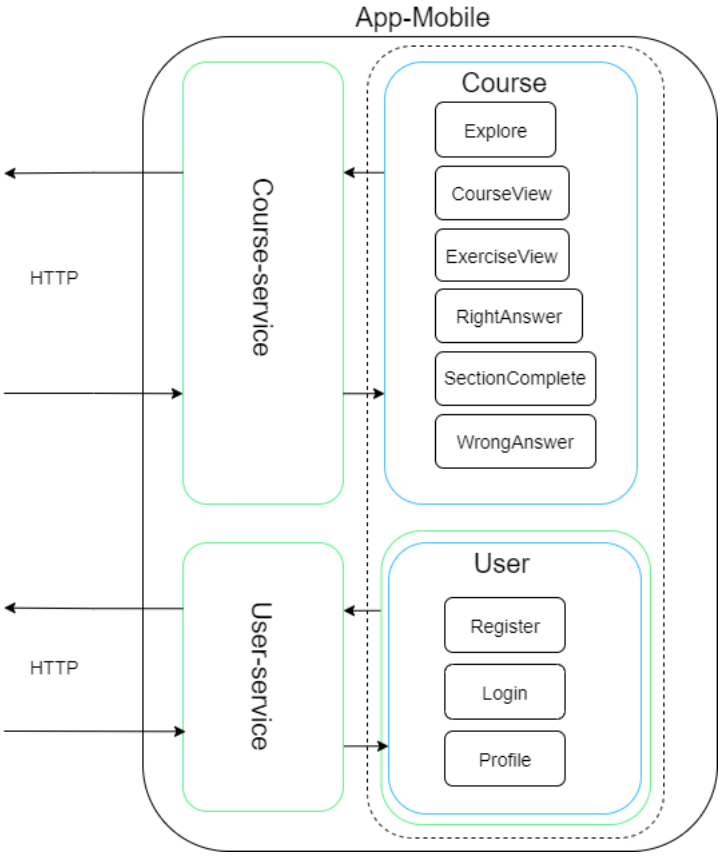

Figure 6.2: App-mobile overview

Here, the Course- and User-related aspects are coupled together in conceptual frames to better create an easy-to-understand overview. The Course-part is everything that is related to the courses themselves, that be the structure, sections, exercises, and content, whereas the User-part relates more to the Brazilian waste picker and their use of the application. Internally, both frames are connected to the back-end through a service layer that handles the requests made by the user.

Back-end overview

The web-application back-end is made up of a few components that each have their own set of responsibilities within the system. Each of these components is subdivided into several layers. These layers help split up the concerns by focusing on a single aspect of the application. The following is a short description of the purpose of each layer.

- **Routes**: The routing layer is responsible for the outermost interaction with the web and therefore contains all the valid endpoints of the API.
- **Controllers**: The controllers are responsible for validating the data of incoming requests and passing that data to the corresponding use-cases or services.
- **Use-cases**: The use-cases have the responsibility of carrying out the actual steps in an operation. For example, a use-case might be `approveMotivation(motivationId)` in the case of successfully signing up as a new content creator. The use-case should find the existing motivation record with the given id, then update the status to approved, and finally, send an email containing a one-time password to that applicant. The use-case would contain the information for carrying out the steps.
- **Domain**: The domain layer is where the main entities in the system reside. In the case of content creation, examples of an entity might be the Course, Section, Category, and so on. The entities are themselves responsible for doing the state changes that always ensure that they are in a valid state. For example, an applicant's motivation should not be able to go from an accepted state to being in a rejected state as this

would rarely be the case in the real world. The motivation should also always include a first name and a last name of the person applying. These types of rules would be enforced inside the motivation entity itself.

- **Gateways**: The gateways have the responsibility of transforming and persisting data to the database while providing a simple interface to client code.

The below diagram shows how the components of the back-end each store these layers as well as the direction of dependencies between them.

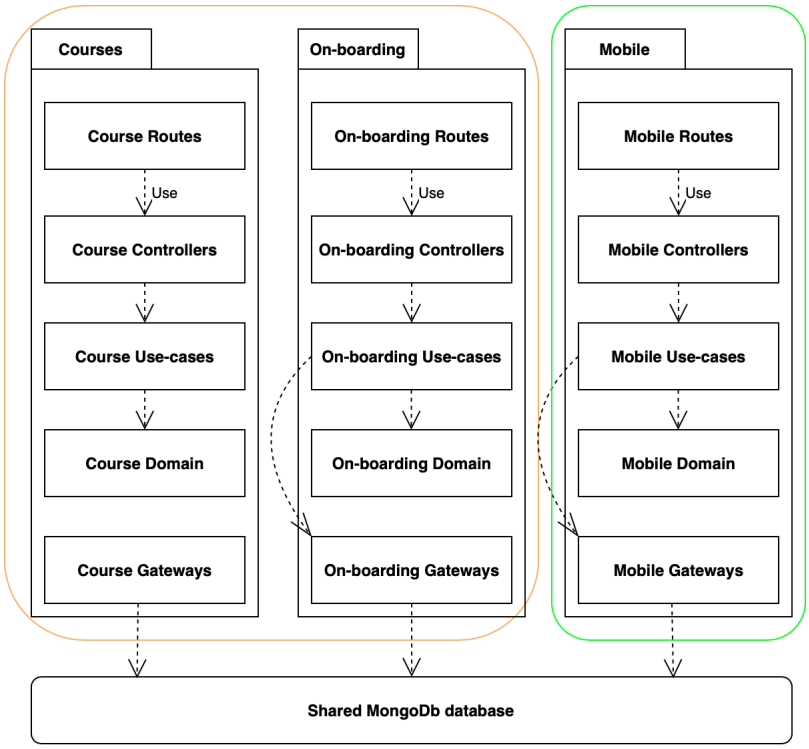

Figure 6.3: Component diagram of the back-end

End of collaboration

7 Software Quality Management

Throughout the development process of this project, our understanding of software quality management grew. In the beginning, our focus was just to produce new features, but later in the project, we shifted our focus to refactoring previously written code in order to ensure higher software quality. Software Quality Management (SQM) can be defined as follows:

> "*An effective software process applied in a manner that creates a a useful product that provides measurable value for those who produce it and those who use it*[Pressman [26]]."

In the following section, we have defined the steps that were taken during the project in order to try and ensure quality and some of the reflections we did about it.

7.1 Definition of Done

An efficient way of preparing for quality assurance is creating a Definition of Done. The DoD is there to help the development team have an agreed-upon set of standards and processes that should be satisfied in order to guarantee the quality of a product.

Ideally, all groups would have defined the same DoD and followed those practices. Then, if needed, each group could have expanded on them to suit their needs. In the first two sprints, we had not talked about a DoD, but when looking back there was a general consensus that these two needed to be done before a task could be marked as completed.

- Code is written
- Manual testing

At the end of sprint 2, we decided to expand on our DoD. This was done in an effort to secure a better code standard and quality. The newly written DoD for our group is as follows:

- Code is written
- Manual testing
- Write tests when it makes sense
- Write documentation for any new APIs
- Peer code review with someone internally
- Code review with someone from another group when possible

We added the testing when it made sense as we should not strive to have a 100% test coverage as this does not guarantee a high coding quality. Instead, we should make sure to debug while we code. When a new feature was added we would write unit- and integration tests when we deemed it necessary.

In the end, our workflow in GitLab consisted of a feature branch made within each group per feature developed. Later in the sprint, these feature branches would be merged internally and then merged together with the shared main branch at the end of each sprint.

Below we can see a figure which illustrates the generic workflow in Gitlab that has been used by the teams. We can see how the code from the *Master* branch is pulled and used to create the new branches *Develop* and *Feature*, as this is more of a generic illustration of the *Feature* branch can symbolize multiple branches with an individual feature on each. In sprint two there may be introduced more features, this is shown by adding an additional *Feature* branch. From sprint two and forward we can see that *Feature* branches are merged into the *Merge* branch. In sprint four we can then see that the *Merge* branch is merged into the *Develop* branch, and in sprints five to six we can see how teams are beginning to use the *Merge* branch more exclusively and lastly will be pushing to the *Develop* branch in sprint six. This way of working ensures we have version control.

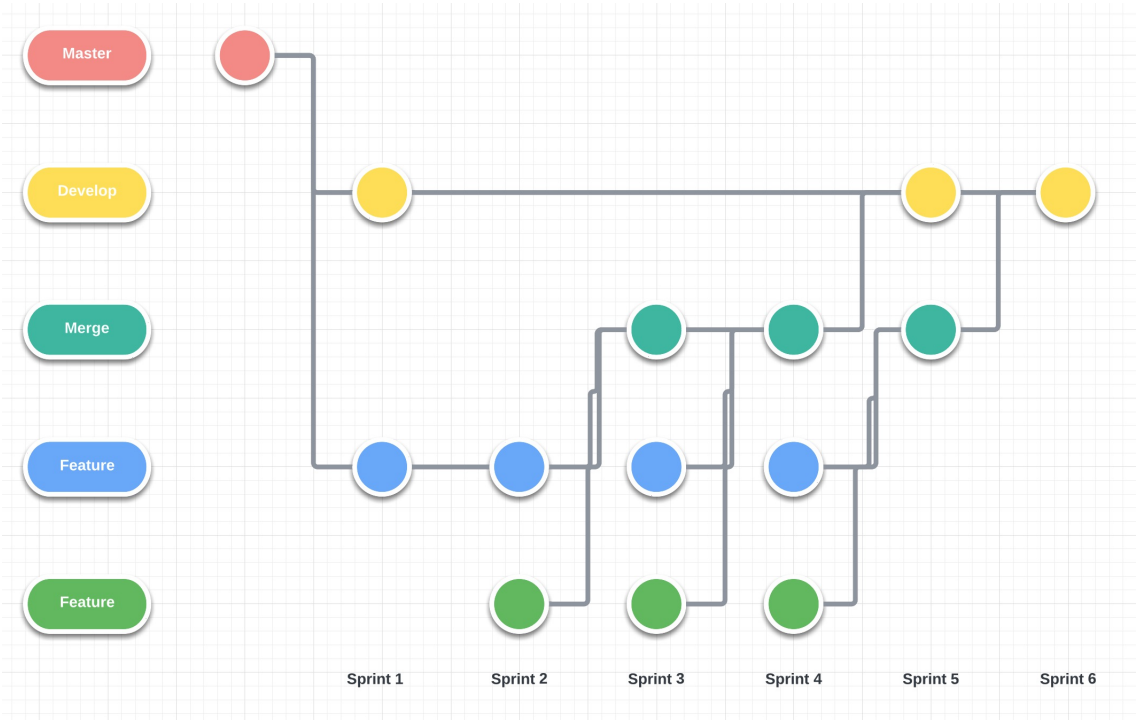

Figure 7.1: Generic GitLab workflow

Next, we added the code review, both internally and ideally also by someone from another group. Code review is a common way to ensure code quality. Another person is able to go through the code and test that it is working or if anything is missing. They might be able to provoke bugs that the original developer did not anticipate. When pushing a feature such as presented in figure 7.1, we were able to create a merge request that should be approved before being accepted into the branch.
Another person can also look through it to confirm that the naming and code style match existing code or predefined agreements.
We included getting the code reviewed by another group when possible for a few different reasons. Firstly, in the Colibri repository, we agreed to follow the same design pattern. Code review helped us adhere to the coding standards we agreed upon. It also allowed the other group to check for merge conflicts and make sure their features didn't break. It also made it easier to align progress between the groups when we were depending on each other work to be completed.

It was essential for us to have documentation for our REST APIs. This helped us understand the connection between front-end and back-end, and was essential for the front-end developers to know what the various functions returned and what input they required in order to use them. The documentation would also be helpful for other people working on the project in the future. Currently, the documentation for the REST APIs for our implementation can be found here: https://documenter.getpostman.com/view/22133458/2s8YzTT2By .

At some point during the development, we did not actually follow our own DoD. The DoD that was followed each time was the writing of the code, manual testing, and writing documentation for any new APIs.
The DoD for writing tests was mostly followed in the Colibri repository (section 7.2). See section 7.3.1 for how testing was used in the project.
In regards to code review, these were not always followed as well. The process for when we requested a code review was commonly done through the use of a pull request on GitLab. Then we assigned a reviewer to the code and awaited feedback. If the reviewer had something to add, they would answer with a comment. Then when a new version that fixed these issues was pushed, they could look through it again. When they confirmed everything was in order, they would accept the pull request and merge changes into the branch. It was helpful for finding missing error handling events and other places for improvement when the process was followed.
But as this was not always followed, there were situations where something with a bug might be pushed into the branch, and in the worst-case scenario, we had to revert a commit. By following the guidelines for user story approval, we could increase the probability of a software implementation of higher quality with less technical

debt.

7.2 Design Pattern

When we were first handed over the project, it did not follow any set design patterns. Design patterns are not the written code, but focus more on the structure of how it should be set up. Depending on what your final goal is, a design pattern can assist in setting up a project and solving a design problem.

In the Educado repository that contains our front-end code and some back-end features, we did not set up any consistent design pattern. Here, we focused on getting important new functionality to work and getting a matching design.

In the Colibri repository, our group began using the Model-View-Controller (MVC) design pattern (see section 5.4) in sprint 2. In sprint 3, the collaboration between the groups started to increase, and we realized that we had used two different design patterns. It was agreed that instead of using MVC we would try to implement Domain-Driven-Development (DDD) and Clean Architecture on this repository with a high focus on Clean Architecture. DDD and Clean Architecture have many similarities, but DDD primarily focuses on understanding the core domain of the project and separates it from technical implementations [6].

Clean Architecture has some similarities to MVC, but it is designed for the separation of concerns. In figure 7.2 there is a recap of the general structure of Clean Architecture as presented in section 6. The general idea behind it was that everything is separated and the information flow only goes one way. This means that even if the database at some point was moved from MongoDB, we would only have to change the Frameworks and Drivers layer, and the rest of the code would stay unchanged. This makes the code easier to maintain, but it also makes it more flexible as we can change implementations without affecting the already decided entities of the program [23].

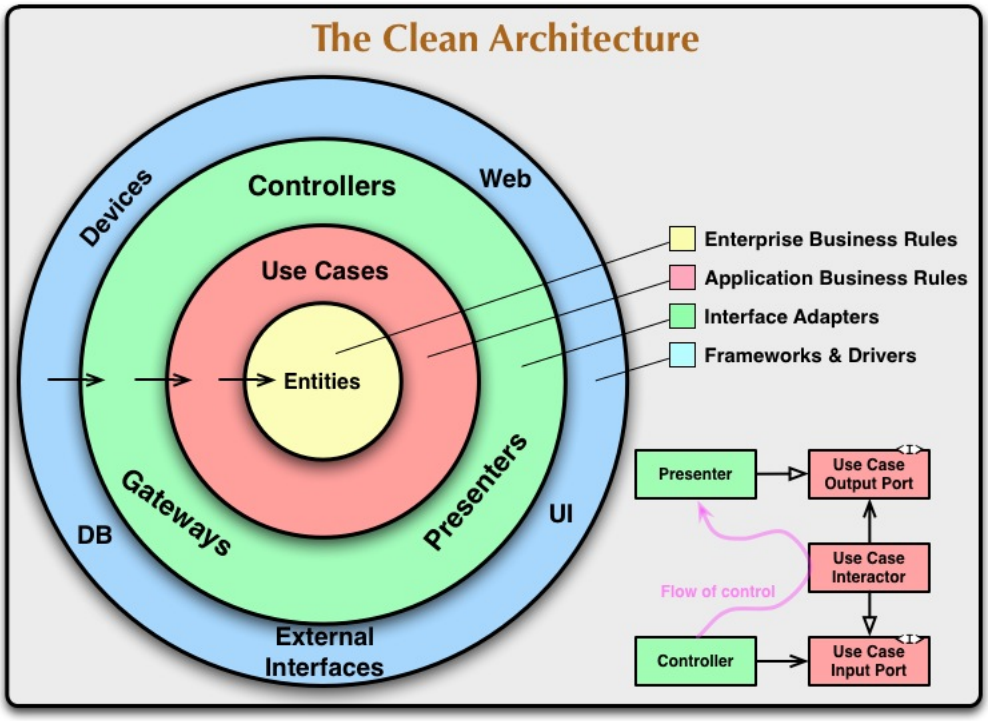

Figure 7.2: An illustration of the different layers in Clean Architecture [23].

After going through a steep learning curve for implementing these features, it ended up being much easier to go back and change/update functions without getting a lot of errors and having to change several different files. In order to make it even simpler, the handling of password and phone validation was placed in a helpers folder. If the requirement for these changes, or the hashing of passwords is going through bcrypt instead of crypto, it is

possible to change it in these files and not touch the entities. Overall using these design patterns gives a higher software quality (see section 7.4.1), but it should also be easier for future groups to expand on the project.

7.3 Testing

An important part of securing good software quality is through the use of testing, even though it should not be thought of as good test coverage equals high software quality. That is also why one part of our DoD was formulated such that we should only test when it makes sense. Below is a presentation of how testing was applied.

7.3.1 Unit Testing

We performed the bulk of our unit tests using Jest, which is one of the most commonly used testing libraries for JavaScript. For the mobile application, we used a Jest preset known as jest-expo, which automatically sets up 'mocks' for many Expo functions, such as the expo-file-system. This allows functions to return a default or custom mocked value, allowing for easier testing with functions that rely on local or external systems, such as the device's local storage or interaction with a back-end through API calls. Testing of API calls will be covered in the next section, 7.3.2. It should be noted that despite there not being many tests written in the dedicated mobile repository, this does not mean we have not worked on trying to implement it. There have been many issues with getting debugging and testing to work with react native, and most of them were only solved at the end of the project which provided us with a very short time frame to write the tests.

In the Colibri repository for the back-end, whenever a new file was created using Clean Architecture, a matching Jest-test file was created, when it made sense to test the functionalities. Although the test files were for testing the single unit in the specified file, the way the architecture for the project is created with for example the controller, it would have to use other functionalities from other files in order to return the correct status. It, therefore, helped with testing the integration of some of the functionalities.

7.3.2 API Endpoint Testing

To test the various API endpoints present in both the mobile application and its back-end, we used Postman, which allows us to send and receive data using API calls. An example of such a request can be seen in the image below.

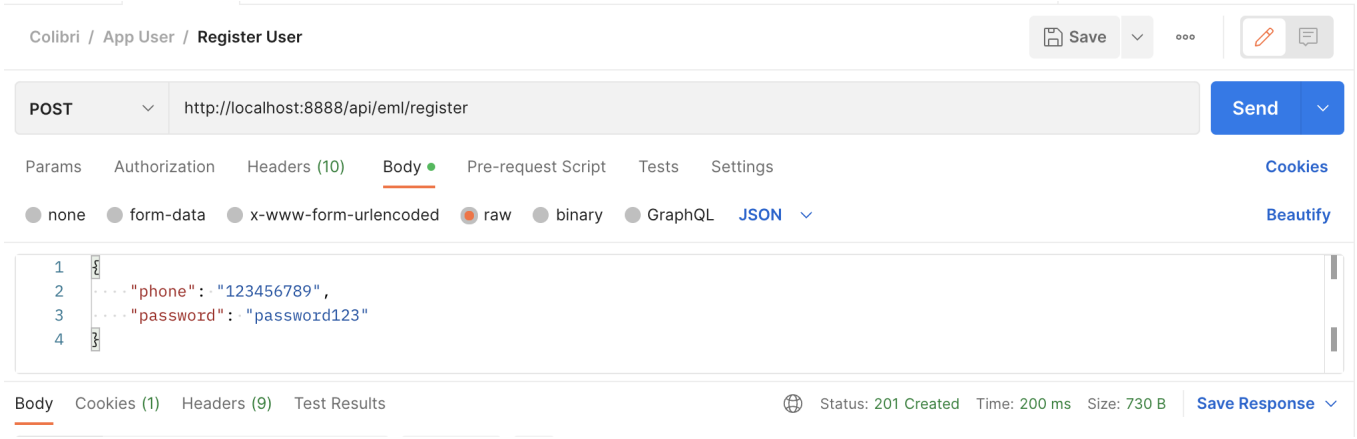

Figure 7.3: API request setup using Postman

This request interfaces with the endpoint we use for registering new users with a phone number and a password. It will respond with either an error or a successful login response. In this case, the phone number and password were valid and the user was registered successfully and stored in the database.

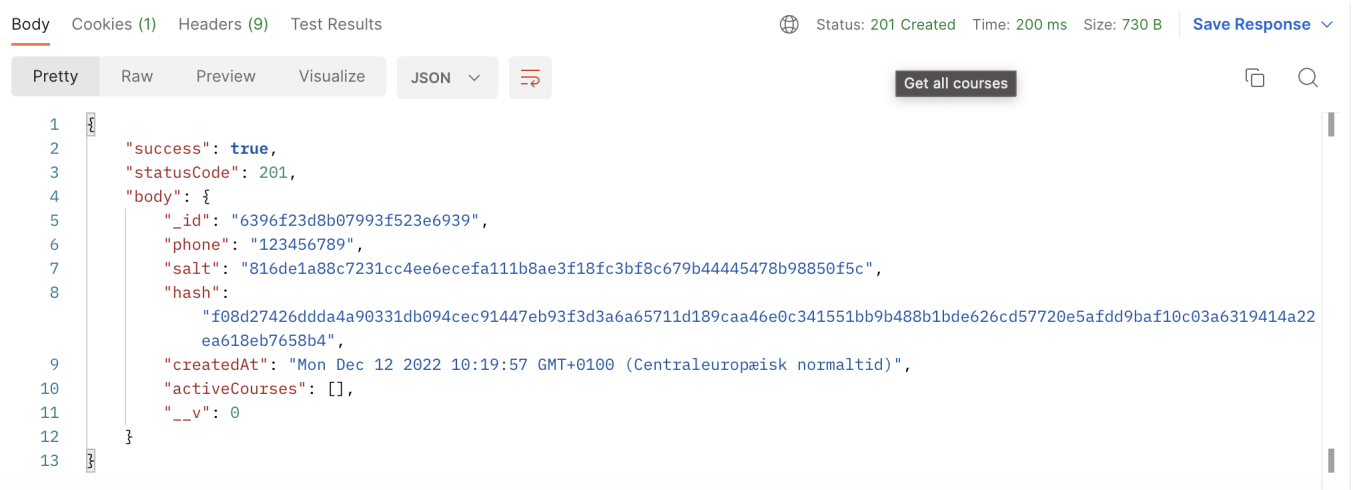

Figure 7.4: API response in Postman's response body viewer

These tests were performed regularly during active development to ensure that newly written API endpoints were functional and behaving as intended.

These tests provide us with an indication of whether or not the functionalities are working as intended. Postman does have a feature for creating actual API tests that runs every time Postman is being used to check an endpoint. This was not something that we made much use of in the project, but it has been created for registering a new app user. Here we just test if we get the expected response and that an id was created correctly.

Figure 7.5: API test for registering a new app user

7.4 Static Code Analysis

To help enforce the common coding style, we found several tools that are able to assess entire repositories and check for inconsistencies in formatting and coding practices. These tools can be loaded with a custom profile to specify everything from indentation sizes to variable naming schemes. Another type of tool is one that can detect needlessly complex functions, technical debt, and numerous other issues related to code. Two tools that were utilized in the project were CodeScene and ESLint.

7.4.1 CodeScene

CodeScene is a software analysis and visualization tool that helps teams to identify and prioritize technical debt, improve code quality, and optimize the performance of their development process. CodeScene uses advanced analytics and machine learning algorithms to automatically analyze the source code of a project and identify potential issues, hotspots, and trends in the codebase. The tool provides a range of visualizations and reports that help teams to understand the structure, complexity, and evolution of their codebase, and identify areas where refactoring, testing, or other improvements may be needed.

We performed two of these analyses on the dedicated back-end repository - one for the old version before the work done this semester, and one for the new version after completing our increments. Upon analyzing the older version, we saw that it had deemed the project's code to be quite healthy according to CodeScene standards. The code did contain one "bad performer", meaning a file with one or more issues. This file is "`EditCourse.js`", which contains two functions with lengths past CodeScene's standard threshold. However, the same file was also deemed to have low cyclomatic complexity, meaning easily understandable logic.

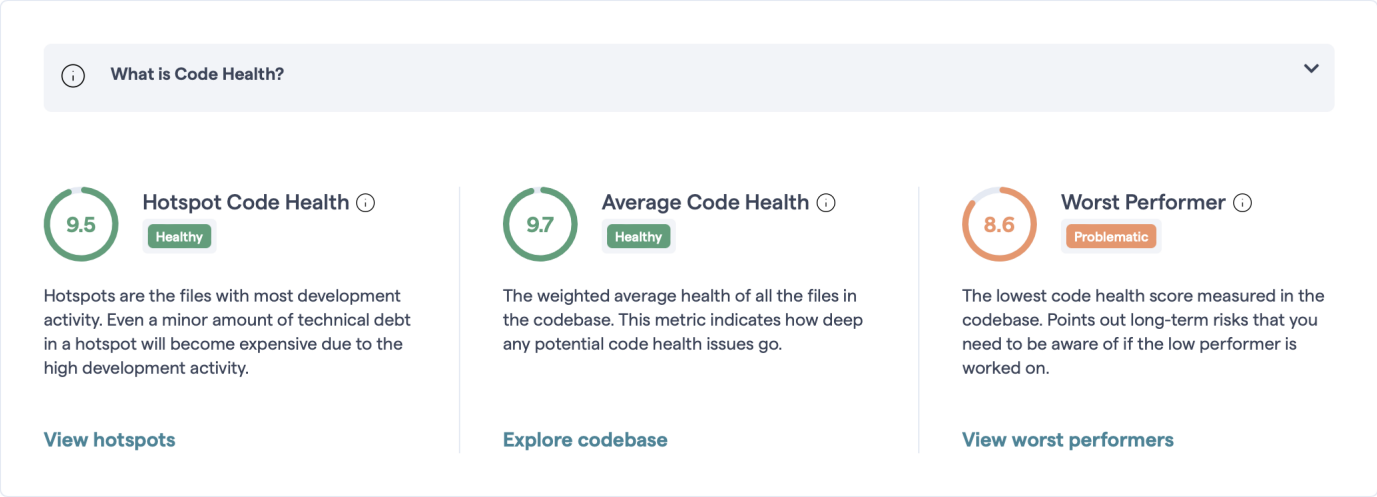

Figure 7.6: Summary of CodeScene code analysis on old code

We then ran an analysis on the new code, which resulted in almost perfect metrics across the board. The previously problematic "Worst Performer" metric had risen to a nearly perfect score, which was a very good sign.

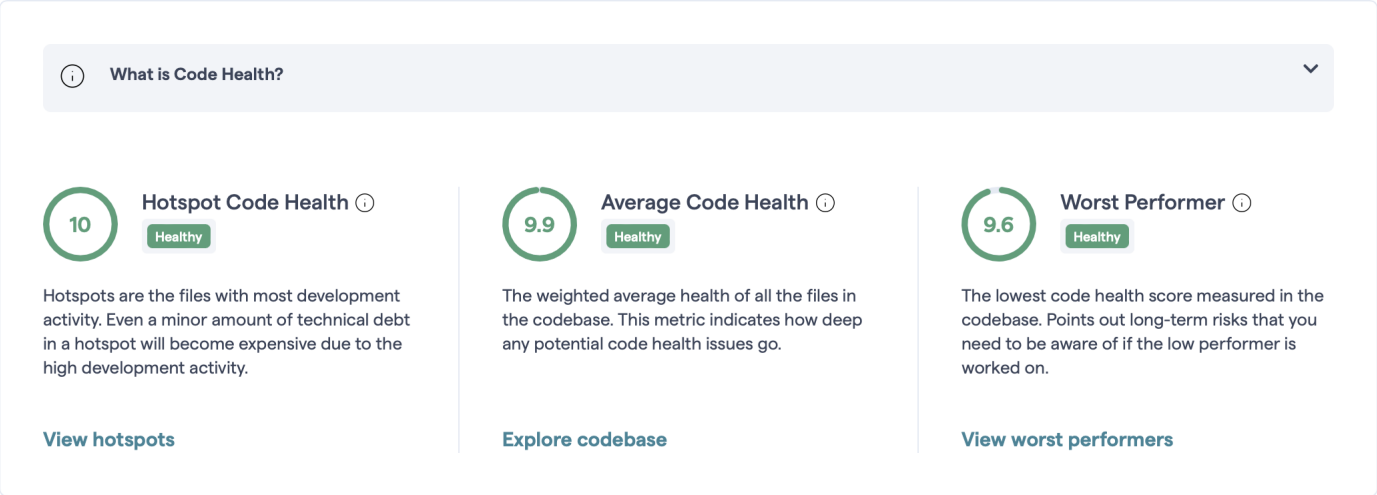

Figure 7.7: Summary of CodeScene code analysis on new code

Only a few files containing complex functions are contributing to the non-perfect code health score, which is indicative of a well-written program. It would appear that the focus on refactoring the existing code and committing to "Clean Architecture" bore positive results in regard to the health of the code.
Although CodeScene is considered reliable and gives a very good indication of the coding standards, there is most likely still some technical debt that it has not caught.

7.4.2 ESLint

ESLint is "*a tool for identifying and reporting on patterns found in ECMAScript/JavaScript code, with the goal of making code more consistent and avoiding bugs.*" it uses an 'Abstract Syntax Tree' together with a set of rules to evaluate patterns of code [20]. With the adoption of the Airbnb JavaScript Style Guide in (S6-T7) we had a good baseline set of rules that could help everyone write more consistent code.
We set up a GitLab-ci pipeline which would run the linting tool on the codebase and provide a detailed output of where the code violated the style guide's rules, see Figure 7.8.

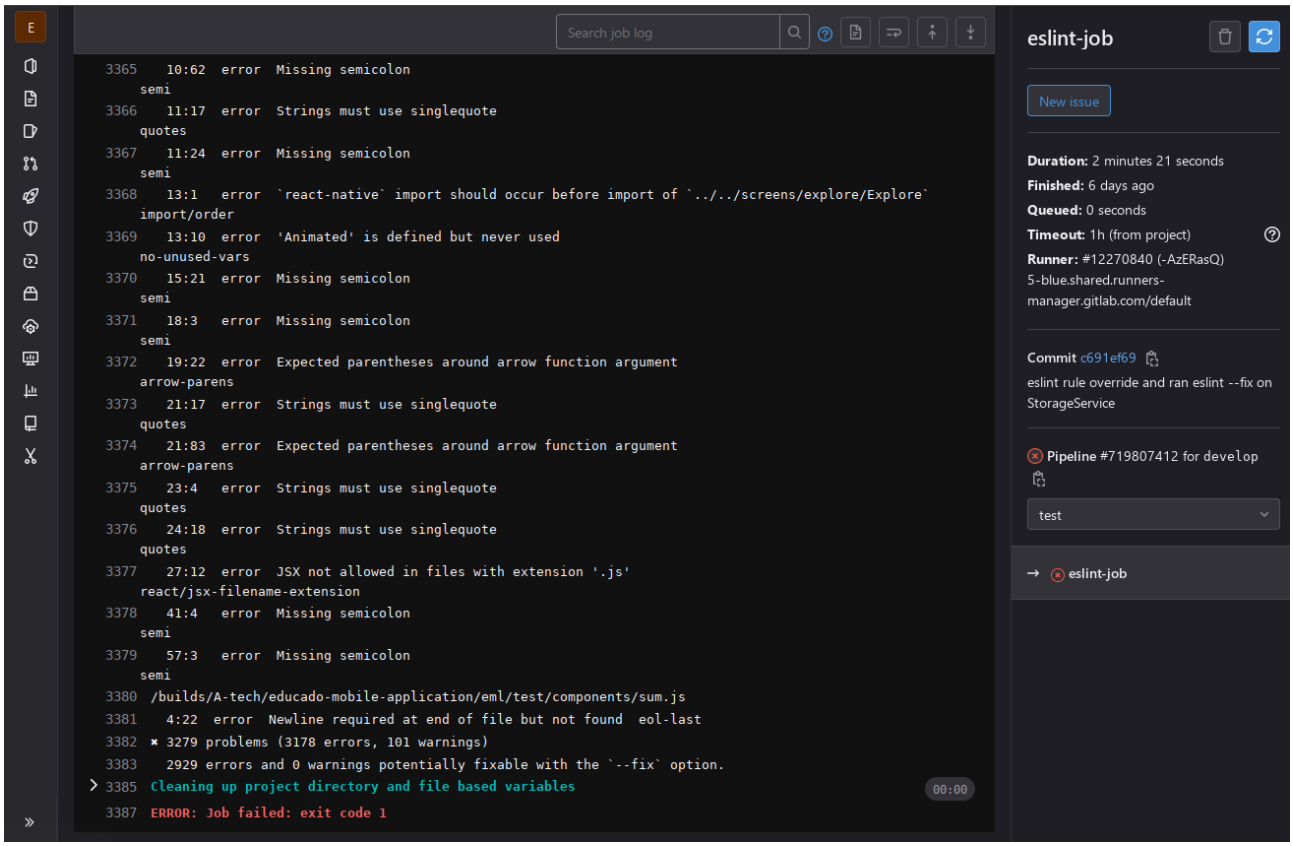

Figure 7.8: Result of latest ESLint Gitlab CI/CD Job on develop branch

This is only a snippet of the output Each line of output contains information about `line:column` number associated with a filename and a description of the rule violated. The full output can be seen in the educado-mobile-application GitLab repository. The result states that there are thousands of errors i.e. rule violations and around 90% of them can potentially be fixed automatically with one command. This was not done during Sprint 6 for reasons explained in (S6-T7).
Another perhaps better way to get this feedback during development is to configure ESLint with one's editor of choice as this can provide feedback in real-time and auto-fix problems when the file is saved. We did not explore this too much in-depth as it would require everyone across groups to make an effort to configure it for their own editor and there was a great number of different editors and IDEs used across the groups.
For the sake of maintainability and consistency, we recommend any future maintainers and contributors continue using the style guide and take the time to configure their editor for a better development experience and higher-quality code.

7.5 Software Quality Summary

The result of the analysis of the software quality management of this project has not helped to give a concise answer. In general, following the DoD has most likely been a great assistance in lowering the amount of technical debt produced by this project group. Toward the end of the project, we started skipping code reviews and tests.

Using a design pattern has certainly been beneficial, but there are features, such as the user course progression (**S6-T3**), that do not follow the convention yet and should be refactored to not leave too much technical debt.
As the analysis from the ESLint pipeline indicated, there are many small inconsistencies in our codebase, which indicates there might be some issues with the quality.

8 Data Model

Written in collaboration with all groups

MongoDB is neither an object database nor a relational database [25]. This means the methods we have learned for Entity Relationship Diagrams do not apply here, as the database is not relational. Below is a best-effort attempt to model the database using a UML class diagram.

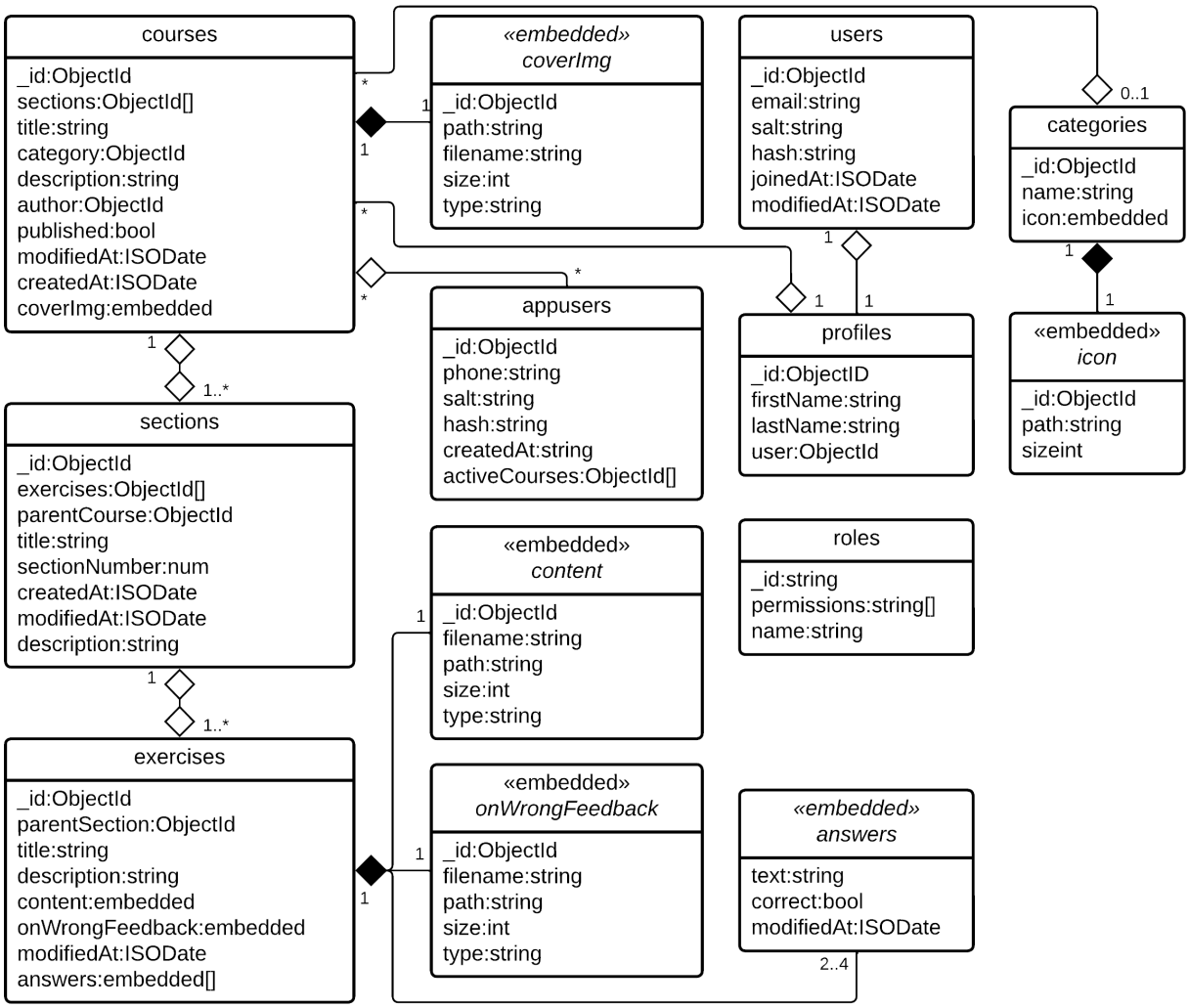

Figure 8.1: Class Diagram depicting relationships in MongoDB collections.

Note the non-standard usage of composition and aggregation. Composition (black diamond) denotes objects embedded inside other objects in a collection, and aggregation (white diamond) denotes referenced objects from another collection.
Class fields are separated by a colon where the right-hand side denotes the type.
Any field other than _id with type `ObjectId` references another object by id. `ObjectId[]` type denotes an array of referenced ids, similar rules apply to embedded and embedded`[]` with the exception that those objects are embedded in the parent object and not part of a collection.
Class names in figure 8.1 refer to the collection name in MongoDB, the collection contains objects, or documents to be exact, with a structure as depicted in the figure.

End of collaboration

9 User Evaluation

Written in collaboration with all groups

During the project, the group discussed multiple times how it could make sense to perform usability testing of the application. One key aspect that caused some frustration was the fact that we did not have any opportunities to get in contact with the real end-users, the Brazilian waste pickers. We could still derive features that the end-user would probably want based on the PO's understanding of the problem domain and its users. However, coming up with reasonable ways of validating these features, was very troublesome.
The group considered the possibility of finding users that represented the waste pickers, to perform usability tests on. However, to accurately represent the waste pickers. Those users would have to be from low educational backgrounds with limited ability to read and write. These criteria limited the field of possible candidates severely. At one point, elderly people or young children were on the table of possibilities, if it were possible to get in touch with someone that could not read very well.
However, due to several issues, the group eventually decided to hold off on performing the test. Some issues were due to the state of the app which was still being developed into a coherent state by the very last sprint of the project, while other concerns revolved around the fact that even if we got in contact with similar users to the waste pickers, there would still be significant differences.
Among others, the fact that the waste pickers know a limited amount of English would mean that the intuitive way of navigating through the application would be different than the test subject that we would be able to perform the testing on.
In the end, the group wanted to perform the usability test, but only in a way that made sense. Due to the reasons listed above, we concluded that the test ought to be performed with some of the actual Brazilian waste pickers if the test could be considered to be anything other than performance. Luckily, two of the group members were enrolled in the trip to Brasilia planned by Aalborg University. Therefore, the plan became to have the two group members perform the usability testing while visiting the waste pickers. They will report the findings of the test back to Aalborg University in order for future generations of software students to be able to make use of it.
The following usability test was developed by the group, for the purpose of performing it with waste pickers in Brazil. First off, the test user should be asked some formal questions before the actual testing begins. These include, but are not limited to asking if the test may be recorded as well as the test user's name. Afterward, the Educado application will be installed on the user's phone if possible. Alternatively, the user will use a provided Android with the application preinstalled.
Following this, the user will be asked to perform several different tasks, each of which focuses on a specific feature of the application. The tasks will be given as a case, where each case has a setup, something that the user wants to do or achieve in the application, and an open-ended question so that the user has a somewhat free interpretation of what to do. For each task, the group member that will act as an observer takes notes about what behavior and steps are taken in order to try to complete the given task, whether or not it has been completed, as well as the time to complete it, if successful. An important thing to note is that the following tasks will be done as a think-aloud test, where the user says aloud what he/she thinks at any given time, what their conclusions about the task are, as well as any reasoning for doing what he/she is about to do.

Tasks:	**Observations:**	**Completed:**	**Time to complete:**
1) You want to access the application. What do you do?			
2) You want to have an overview of all the available courses in the application. What do you do?			
3) You want to see the content of a course. What do you do?			
4) You want to try another course. What do you do?			
5) You decide that this course looks interesting. How do you proceed?			
6) You want to start the course. What do you do?			
7) You want to change what part of the course, you are focusing on. What do you do?			
8) You want to get an overview the course you are currently doing. What do you do?			
9) You want to see the information the app has on you. What do you do?			
10) You don't want your account anymore. What do you do?			

Table 9.1: Usability test

When the test is actually being carried out, several obstacles might arise, which we have given some thought to beforehand.
A possible issue that could affect the testing is the possible language barrier that might occur between the test user and the group members. It should be noted though, that the PO of Educado, Mateus Halbe Torrés, will be

attending the part of the trip to Brazil where the group will interact with the waste pickers. Mateus is native-speaking as well as English-speaking, so he could step in and act as a translator if needed.
Furthermore, several aspects of the test user might affect the outcome of the testing. The understanding of what it is to be a waste picker is only based on what the PO has told us. At one point, he mentioned that the waste pickers were unlike anyone we have ever met since the closest waste picker in Denmark (a tin deposit gatherer) would still be considered miles apart from the Brazilian one.
In the worst-case scenario, the waste pickers could prove unable to perform the testing due to issues that the group would have a hard time predicting, such as the waste pickers' ability to understand the given task, their ability to focus on the task at hand, their ability to verbalize their thought process and so on. We hope to minimize this aspect by having the waste pickers speak in their native tongue while recording the test, which could then be transcribed afterward.

Lastly, there are some technical aspects to consider when performing a usability test on the other side of the Earth. For once, the back-end should be deployed on a server closer to Brazil to make the content available there. This will cost money, and therefore befall the funding of the project to get this aspect implemented before the trip in January.
Additionally, the group members who are traveling to Brazil will need a way to get the application onto the waste pickers' phones, if possible. Currently, the plan is to have the application installed by entering *developer mode* on their phone, giving access to install the Android Package Kit (APK) via a link to the application stored in an AWS-bucket in the cloud. The backup plan could be that the group members who will travel to Brazil will use their own phone and hand it out when the test is ready to be performed, and help when issues might arise. There might also be an option where we bring phones from the department to Brazil in order to have something to perform the test on.

End of collaboration

10 Discussion

This section's focus is on discussing the process of this project. Mainly, the discussion will focus on the teamwork of the group and how we handled working with agile during the project. A part of the project was also to work with other groups, and this collaboration will also be reflected in this section.

Internal Collaboration

As previously mentioned in section 4, we learned more about agile throughout the whole project. This meant that for each sprint we gained a better understanding of how to work with agile, so even though we tried to apply the agile events and artifacts they did not provide much support for us in the beginning.

Scrum Master
A new Scrum master was chosen in each sprint. This was chosen to give most people in the group a chance to try this role. In the end, the Scrum master title was mostly just a title and did not have many other functionalities. If the Scrum master was present during the stand-up they would be in charge of beginning the meeting and make sure everybody was heard, but that, for the most part, was the extent of their roles.
When we got to sprint 4, the Scrum master also had to participate in a weekly meeting with the other Scrum masters, but on a few occasions, it was another person than the Scrum master that attended.

Stand-up
As mentioned at the sprint retrospective of sprint 1, we changed from a daily stand-up to having them twice a week. This is still considered a good decision in regard to the workflow of the group. As we had several other lectures and exercises apart from this project that we needed to attend, it would happen that there was no process of a user story that could be presented. We placed the two stand-ups so they matched our weekly calendar and gave us a chance to work on our tasks before the next meeting. This meant that we more often than not were able to present actual progression for the assigned task.
But, even though we tried to keep the stand-up meeting short, they would often lead to a technical discussion about the progression of a task. Here, it would have been beneficial if the Scrum master had tried to keep the conversation on track so the meetings were just a short formality and everybody knew what was being worked on.
Because the stand-up might sometimes become more technical it was not always easy for everybody in the group to be completely up-to-date on what each team member was working on. This feeling was reinforced by our lack of managing the sprint backlog. New tasks were added to the sprint backlog during each sprint planning, but it was not always moved to the correct column when it was being worked on, and sometimes it was not moved to done either, so we regularly had to sit as a group and try to go through it in order to place the tasks in the correct columns.

Back-log
Regarding the sprint backlog, we had some issues estimating how many tasks we could complete during one sprint. This is also discussed a bit in '*Effort*'. After each sprint, we had a small retrospective. These were really helpful in order to get an understanding of how the whole group felt the last sprint had gone. A common topic during these retrospectives was that we often had missing group members because of sickness. This has been a benefactor to our issues with estimating.
But, even without sickness, many of our user stories turned out to be bigger than originally assumed, and it was therefore not possible for us to set aside the required time each sprint to solve them, which caused a lot of the stories to stretch over 2 sprints.
In sprint 4, we did try splitting similar user stories using their acceptance criteria. For example, we knew a user needed to be able to download a course. Then, in sprint 4, the user story for the download functionality was included (**S4-T1**), but the acceptance criteria were to research and design a proof of concept for it. That meant we could accept this user story at the end of this sprint and then make a duplicate user story next sprint that would focus on the actual implementation instead (**S5-T2**). This was something that might have been beneficial for us to do more often, but in general, the acceptance criteria were mostly used for us as a general guideline of what the group had discussed would be needed for the user story to be considered done, but if not all criteria were completed we would sometimes still considered accepted.

Effort
In section 5.1, the handling of assigning effort for the project was discussed. The modified version that was being applied to estimate the effort of each task did not provide any benefit to our project workflow. It would be beneficial to have more sprints in order to properly be able to use estimations. For each sprint, the members of the group get a better understanding of the project and how long it takes to implement features.
This can also be seen if we look back at the sprint reviews. In sprints 3 and 4, several increments had to be carried over into the next sprint backlog or moved back into the product backlog. In sprint 5 we only had one item that needed to be carried over.

The estimations were normally done at the beginning of the sprint, but we did not have any agreed-upon limit of the amount of work we could accept in one sprint. This generally led to sprint planning where we would accept a certain amount of user stories before estimation. Then even if we had estimated several tasks to be of medium or high effort, we would have kept all user stories and tried to get as far as we were able to in that sprint.
It would have been helpful for us to put more focus on estimations in order to be able to finish a higher amount of the user stories we accepted.

External Collaboration

In section 5.9 many of the frustrations all the groups had were presented. This discussion will work as an extension of this, but only from the focus of this project group.

Product Owner
During the sprint reviews, we were generally happy with our PO. We would get useful feedback which we could try and implement in the next sprint. But we did sometimes have some concerns during the sprint planning. We often had issues getting our user stories approved. Especially in the first few sprints, we were still waiting on the final approval of the user stories the day before the sprint review. This could lead to a risk of us implementing a user story that had no benefit for the project.

Beginning at the 3rd sprint, we started getting sprint goals, which did help provide an overall idea of where our focus should be for the sprint. Having a shared goal across all teams also felt like it sparked collaboration between all groups, as the common goal felt more tangible than before.
Instead of getting each user story approved, we began sending a list of the overall goals we as a group wanted to achieve during the sprint, and if no comment was given on it from the PO we considered it accepted. It still seemed futile to send the list of goals as it was not brought up again during the sprint review. Here, we should have been better at mentioning our goals during the presentation and then showcasing how we fulfilled them.

Cross-team Collaboration
What felt like the most important learning goal in this project was to not only work together within your own project group but also across several other teams. This also turned out as one of the hardest goals to achieve.

When 21 people work together, there is a high risk of miscommunication if we are not very careful. This can for example be noticed in regard to conversations about a common DoD. In section 5.4.2 we discussed that we should start merging into the same branch, but this required a merge request approved from another group. This was a misunderstanding on our part, as no other group followed this step going forward.
Several times during the weekly meetings we tried to get the “code review by another group” added as a common requirement. It was a big focus for us to try to get it implemented as we had several times experienced that there was a push to a common branch that we were not aware of. Sometimes these newly added implementations/refactoring caused an issue with some of our previously completed increments, as the shared functionality they depended on was changed. We then had to go back and figure out what the changes were and how to fix them.
It started becoming less of an issue after unit tests were added to the core functionalities. Then if another group changed some of the shared functionalities and it caused the test to fail, they would either inform us about it or if it was a small change just fix it themselves.
But, there were still issues with another team changing the functionalities of a course without informing the front-end team. They would only discover the changes when they tested the mobile app.

11 Future Works

As a result of us using an Agile framework while developing our project, we were regularly implementing and brainstorming new features in the form of User Stories. While a fair amount of these features were implemented, many were also dropped for one reason or another. This section will cover the more notable features that were either not finished in time or dropped altogether before work ever began on them.

Phone Number Authentication

When we were initially implementing our phone number registration system, considerations were made as to whether to add an authentication system to verify that the phone number was valid. This system would work by sending a text message containing a code to the phone number, which would then need to be typed into the Educado app's user interface to verify that the user was in possession of the phone number and had typed it correctly. We looked into a few systems that made it possible to send text messages, but all of these required us to set up API keys that would charge a small amount of money each time a message was sent. This meant that we would either have to pay out of our own pockets while testing and implementing this feature or that we would have to request funding from the university, which could potentially be a very time-consuming process. We felt that this feature would not be necessary for what we wanted to accomplish and when discussed with the PO, it was decided that it would be beyond the scope of the project.

Course Progression

While course progression was successfully implemented in the database, this data is never used in the mobile app. Optimally, the user interface would show the user's progress in some way. This could be represented by a progress bar in the course overview or some other way. This feature would give users a better idea of which courses they have completed and which they are in the process of completing. This might incentivize the completion of more courses, leading to a higher level of engagement.

Notification System

A notification system would make it possible to send notifications to the user's smartphone, such that they will be notified of new content that is available in the app, and tell them how they have progressed to improve their motivation for completing the courses they have begun participating in.

App Preferences

To better suit, the user's needs and expectations of the app, a set of preferences could be implemented in the app such that the user decides how the app should behave when using it. Examples of preferences could be the opportunity to choose between a light theme and a dark theme, such that they do not feel eye strain if they are using the app in a dark environment.

Animations

To make the app more user-friendly and fluid, it could feature more animations. Examples of animations could be an animation when the user opens the app, such that they feel the app is responsive and is not frozen. Another example could be more animations when changing between different screens in the app, such as going from the initial login screen to the register screen.

Caching System

One of our initial goals with this project was to implement some sort of smart caching system that would automatically download and store content from the app locally, such that users would not need to have entire courses downloaded at once. This feature ended up being scaled back to simply being able to manually download and delete content, which is a little less user-friendly. A smart caching system could work by downloading the first few exercises of a course when the user starts that course. This content would be stored locally for some time until an algorithm would determine that the content could safely be deleted. With this feature, users could spend less time worrying about storage space and manually downloading/deleting content, and instead spend more time using the app for its intended purpose.

Reward System

To keep the users feeling more motivated to keep using the app and completing courses, a reward system could be implemented. A reward system could include various badges for different achievements. This could be badges for when they have proceeded in a course every day for a week, when they have completed their first course, or when they have completed all courses in a specific category.

Profile Screen

Additions to the profile screen could further improve the personalization of the user experience. For example, the user could have the opportunity to upload a profile picture, such that they can communicate who they are to other users when sharing their progress in courses. They could also be able to add a biography to their user profile, such that they have a space to tell more about themselves to other users. It can also be set up to show the achievements they have earned. And lastly, they could be able to see their progress on the profile screen, such that it is easy to glance over all the information in the app that is related to them at once.

Restricted Routing

In section 5.7.2 we discussed having implemented restricted routing for endpoints. This feature has only been implemented back-end, but the implementation is still missing in the front-end part of the application. It is therefore not possible to add the routing as it will cause the endpoint it is added to, to not function for the app user.

Roles

At the moment, only an app user can register for using the mobile application. If a content-creator wants access, they need to register again with their phone number. Ideally, in the database there would only be one user, then depending on how and where they register they should be assigned roles, such as app user, content-creator, etc. Then the role they are given at registration should affect what they can interact with and what their view is.
By having one main user collection where we assign roles, a content-creator should be able to log into the application using the same login as they do on the website.

Responsive front-end

While using the mobile app on various devices among team members, we experienced that the front-end would look a bit different on each of them. There were two main issues that were apparent while testing, the first being that the lower button on the login- and register screens would be shifted up a bit once one of the text fields was tapped and the keyboard opened. The second issue was that the two lower buttons on the login- and register screens would have more space between them depending on the device's screen.

More intuitive front-end

In the report for the project we were handed over, it read that most waste pickers are not able to read, so to work around this, the design of the front-end would likely need to feature less text. Instead of text, the design could feature icons that are familiar to them on the buttons such that they can tell the function of the button just by glancing at it.

Translate text in app

To make the design easier to understand, the text in the app could be translated from English into Portuguese, such that the waste pickers who are able to read Portuguese will be able to read what the buttons do.

Testing

Despite trying to make the mobile as functional as possible, there were some issues with buttons that did not work as expected, even though they had been working before. If there had been more focus on testing important functionalities in the Educado repository we might have been able to catch the issue before the review.

12 Conclusion

In section 4, the following goal was stated that we wanted to work on.

> ***Improve the Educado platform and transform it into a great functional Mobile Education solution for Waste Pickers to be tested in the field by the end of the semester in Brazil***.

This was a common goal for every group. However, in this section, we strive to conclude how our group contributed to this.

The first part of the problem we wish to solve is to "*Improve the Educado platform*". We are only able to give a partly subjective answer on this, as the application has yet to be user tested. We are instead comparing it to what we received versus the product we showcased at the final sprint review and the SQM, etc., which were explored in the previous sections.
Generally, a lot of new functionalities have been implemented during this project. As seen in section 3, in the original version everyone could access the application. It contained a screen for the courses and you could start a course and see the sections.
After the increments in the six sprints the user now has the possibility to register through the app, they can log in and stay logged in, and they also have the option to delete their account. In collaboration with the other dedicated front-end team, the courses created on the web application can be seen instantly on the explore screen. Each user can enroll in a course, and they can download it on their phone in order to continue their learning, even when they are not connected to Wi-Fi. It can be deleted again afterward.

The next part was formulated as so "*and transform it into a great functional Mobile Education solution for Waste Pickers*". Even though they do not cover the exact same information, the class diagram from section 3 and the data model from section 8 / the architecture from section 6 assist with giving an idea of the development of the application. The mobile application is still simple, but the design and the way course and functionality are being handled have been updated so it ideally is more intuitive for the user to utilize as a learning platform.

As we do not have a user evaluation, the best indication we have for concluding if the mobile application is 'great' can be found in our software quality management section, 7. Even though it took a long time to refactor parts of the application, that time is considered well spent. If bigger changes are needed, such as implementing something else than `express`, nearly all files and many functions related to routing would have to be rewritten. Using the new design is more stable for future maintenance of the code.
Testing of key features has also been more common, and through the use of static code analysis, we can see that the average code health has increased, despite there still being some issues with common coding standards.

Despite our attempts to develop a learning application for waste pickers in Brazil, this was not the main objective from an educational standpoint. As mentioned several times throughout the report there were a lot of issues with figuring out how to properly collaborate with several groups. Even though we slowly learned the basics of good teamwork using the agile methodology, there were still many issues that remained unresolved at the end of the project. For example, when we tried to suggest code review across teams, it was quickly voted down as it was not time efficient.
From our perspective, it might have taken longer to merge into the branch, but nearly, every time we had a merge request the reviewer either found an unhandled error event or instances where the code could be optimized. On the other hand, after another group merged directly into the shared branch, our group ran all the tests (by chance) and realized that their latest push caused some of their own test to fail. This was quickly relayed back to the team so they could fix it before the sprint review.

Despite the minor issues remaining in the final product, and unresolved collaborative challenges, the semester has still been quite educational. It has yielded some insight into how it might be to work in the industry. Even with the steep learning curve, this project has without doubt, contributed to us becoming better software developers. Especially regarding working together as a team.

Bibliography

[1] The agile manifesto. URL https://agilemanifesto.org/principles.html.

[2] Scaling scrum with nexus. URL https://www.scrum.org/resources/scaling-scrum.

[3] What is scrum?, . URL https://www.scrum.org/resources/what-is-scrum/.

[4] User stories are needs described from the business perspective, . URL https://www.scrum.org/resources/blog/user-stories-are-needs-described-business-perspective.

[5] [myth busting] what is a user story?, . URL https://www.scrum.org/resources/blog/myth-busting-what-user-story.

[6] What is ddd, Mar 2007. URL https://www.dddcommunity.org/learning-ddd/what_is_ddd/. Accessed: 12/12/2022.

[7] Expo docs, 2022. URL https://docs.expo.dev. Accessed: 10/10/2022.

[8] About figma, the collaborative interface design tool, 2022. URL https://www.figma.com/about/. Accessed: 10/10/2022.

[9] Kompleks back-end software 2022/2023, 2022. URL https://moduler.aau.dk/course/2022-2023/DSNSWB521?lang=da-DK. Accessed: 19/09/2022.

[10] 2022. URL https://www.statista.com/statistics/530481/largest-dump-sites-worldwide/. Accessed: 05/12/2022.

[11] Mvc, Sep 2022. URL https://developer.mozilla.org/en-US/docs/Glossary/MVC. Accessed: 12/10/2022.

[12] React native, 2022. URL https://reactnative.dev. Accessed: 10/10/2022.

[13] Glossary of scrum terms, 2022. URL https://www.scrum.org/resources/scrum-glossary. Accessed: 17/12/2022.

[14] Airbnb. Airbnb React/JSX Style Guide | Airbnb JavaScript Style Guide, 2022. URL https://airbnb.io/javascript/react/. Accessed: 17/12/2022.

[15] Auth0. What are refresh tokens and how to use them securely, Oct 2015. URL https://auth0.com/blog/refresh-tokens-what-are-they-and-when-to-use-them/. Accessed: 07/11/2022.

[16] auth0.com. Json web tokens - jwt.io. *Jwt.io*, 2013. URL https://jwt.io/introduction. Accessed: 02/11/2022.

[17] D. Britze and J. V. Jensen. Digital learning platform for waste-pickers in brazil. Bachelor thesis, Aalborg University, May 2021.

[18] D. Britze and R. N. Nielsen. Mobile education platform - smart caching learning materials, 2019. AAU Student Report.

[19] D. by OneTrust. *Comparing privacy laws*. URL https://ec.europa.eu/futurium/en/system/files/ged/dataguidance-gpdr-lgpd-for-print.pdf.

[20] ESLint. Getting Started with ESLint - ESLint - Pluggable JavaScript Linter, 2022. URL https://eslint.org/docs/latest/user-guide/getting-started. Accessed: 18/12/2022.

[21] Expo. Debugging - Expo Documentation, 2022. URL https://docs.expo.dev/workflow/debugging/#react-native-debugger. Accessed: 18/12/2022.

[22] iArchin. toggle developer tools doesn't work - ubuntu 22.04 LTS · Issue #700 · jhen0409/react-native-debugger, 2022. URL https://github.com/jhen0409/react-native-debugger/issues/700. Accessed: 18/12/2022.

[23] R. C. Martin. The clean architecture, 2021. URL https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html. Accessed: 16/12/2022.

[24] Passport. passport-jwt, 2015. URL https://www.passportjs.org/packages/passport-jwt/. Accessed: 15/12/2022.

[25] Phillipp. MongoDB Schema Diagram - StackOverflow. URL https://stackoverflow.com/a/34369991. Accessed: 13/12/20220.

[26] R. S. Pressman. *Software Engineering: A Practitioner's Approach*. McGraw-Hill Education, 8th edition, 2017.

[27] D. Radigan. What are story points and how do you estimate them?, 2022. URL https://www.atlassian.com/agile/project-management/estimation. Accessed: 18/12/2022.

[28] P. Sriramya and R. A. Karthika. Providing password security by salted password hashing using bcrypt algorithm. *ARPN Journal of Engineering and Applied Sciences*, 10(13), july 2015. ISSN 1819-6608. URL https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.1072.20&rep=rep1&type=pdf. Accessed: 12/10/2022.

[29] Statista. Penetration rate of sending and receiving emails among internet users in brazil from 2017 to 2022, 2022. URL https://www.statista.com/statistics/1083373/brazil-email-usage-rate/. Accessed: 10/10/2022.

[30] Statista. Leading internet activities in brazil in 2022, 2022. URL https://www.statista.com/statistics/1052520/brazil-internet-activities/. Accessed: 10/10/2022.

[31] K. Vasarhelyi. The hidden damage of landfills. *Environmental Center*, Apr 2021. URL https://www.colorado.edu/ecenter/2021/04/15/hidden-damage-landfills. Accessed: 05/12/2022.

Appendices

A Combine User Story Diagram

Written in collaboration with all groups

Figure A.1: Combined User stories diagram

End of collaboration