
Educado

Redefining education, one initiative at a time

5th Semester Project Report
Social-Gamified Learning Team

Aalborg University
Software



AALBORG UNIVERSITY
STUDENT REPORT

Electronics and IT

Aalborg University

<http://www.aau.dk>

Title:

Educado - Redefining education, one initiative at a time

Theme:

Complex Frontend Software and Complex Backend Software

Project Period:

Fall Semester 2023

Project Group:

Social-Gamified Learning Team

Participant(s):

Amalie Pernille Dilling

Anders Mazen Youssef

Bence Szabo

Freja Lüders Rasmussen

Louise Foldøy Steffens

Magnus Peetz Holt

Supervisor(s):

Daniel Russo

Copies: 1

Page Numbers: 135

Date of Completion:

December 15, 2023

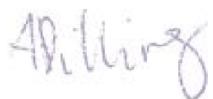
Abstract:

In collaboration with the University of Brasilia and Brazilian waste-pickers, Aalborg University's 5th-semester software students further developed the Educado platform through six sprints in a mock-realistic agile environment. This report provides an overview, emphasising a social-gamified learning approach. Evaluation includes challenges in Agile implementation, repository management, database handling, covering misinterpretations, communication bottlenecks, and acceptance criteria issues. Repository challenges affected backend and mobile development, encompassing code and database issues. Integration team challenges and the product owner's role are explored. The conclusion highlights project achievements in stability, usability, and gamification, addressing technical debt and cross-team communication challenges, and offers insights for future projects.

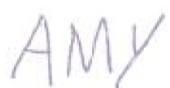
The content of this report is freely available, but publication (with reference) may only be pursued due to agreement with the author.

Preface

Aalborg University, December 15, 2023



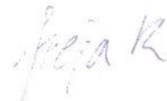
Amalie Pernille Dilling
adilli21@student.aau.dk



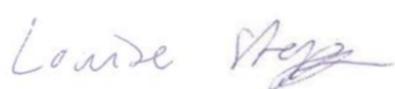
Anders Mazen Youssef
amyo21@student.aau.dk



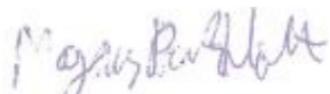
Bence Szabo
bszabo21@student.aau.dk



Freja Lüders Rasmussen
flra21@student.aau.dk



Louise Foldøy Steffens
lfst21@student.aau.dk



Magnus Peetz Holt
mph21@student.aau.dk

We want to thank the individuals and groups who played an important role in this project. As some of the following individuals are mentioned throughout the report, this section serves as an overview of the people involved.

First, we want to thank Luiza Cardoso Queiroz Melo, who contributed as the product owner of the Educado project.

We also want to extend our gratitude to Frederik Bode Thorbensen, who acted as the repository manager.

Additionally, we want to acknowledge Daniel Russio, our semester coordinator who also acted as a stakeholder fulfilling the management role in the project.

Finally, we would like to express our gratitude to the development teams that worked alongside us:

- Group 1: Certificate Issuance Team
- Group 2: Video Streaming Team
- Group 4: Virtual Tutor Team
- Group 5: Onboarding Team
- Group 6: Offline Accessibility Team

Contents

Preface	v
1 Introduction	2
1.1 Educado	3
1.2 The Nexus Framework	5
1.2.1 Roles	8
1.3 Definition of done	11
1.4 AI-powered tools	12
2 State of the Project	13
2.1 Initial Database Diagram	14
2.2 Adjustments before the first sprint	14
2.2.1 Backend	14
2.2.2 Frontend for mobile	15
2.2.3 Preparing the development environment	15
2.2.4 GitHub actions, templates, and static code analysis	16
3 Sprints 1-2: Stabilisation and Enhancement	17
3.1 Sprint 1	17
3.1.1 Sprint Planning	18
3.1.2 Sprint Review	25
3.1.3 Sprint Retrospective	26

3.2 Sprint 2	27
3.2.1 Sprint Planning	27
3.2.2 Sprint Review	34
3.2.3 Sprint Retrospective	35
4 Research on social-gamified learning experience	37
4.1 Theory	37
4.1.1 What is social gamification?	37
4.1.2 Why is social gamification a good thing?	38
4.1.3 4 Phases of the gamification journey	39
4.1.4 Framework for social gamification	43
4.2 State of the art	44
4.2.1 Duolingo	45
4.2.2 Brilliant	50
4.2.3 Sub-conclusion	52
5 Sprints 3-6: Gamification	53
5.1 Sprint 3	53
5.1.1 Sprint Planning	54
5.1.2 Sprint Review	67
5.1.3 Sprint Retrospective	68
5.2 Sprint 4	68
5.2.1 Sprint Planning	69
5.2.2 Validation test	74
5.2.3 Sprint Review	75
5.2.4 Sprint Retrospective	76
5.3 Sprint 5	77
5.3.1 Sprint planning	78
5.3.2 Sprint Review	91
5.3.3 Sprint Retrospective	93

Contents	1
5.4 Sprint 6	94
5.4.1 Sprint planning	95
5.4.2 Sprint Review	104
5.4.3 Sprint Retrospective	106
6 System Architecture	108
6.1 The Educado Platform Architecture	108
6.2 CI/CD	110
6.2.1 CI/CD for this project	111
6.3 Database Model	113
7 Quality Management	116
7.1 Unit Testing	116
7.1.1 Mobile (Frontend) Unit Testing	117
7.1.2 Backend Unit Testing	117
7.2 End-to-End Testing	119
7.3 User validation	120
7.4 Version Control	120
8 Discussion	123
8.1 Issues with implementation of Agile	123
8.2 Issues with the received repositories	125
8.3 Issues with code and database	126
8.4 Evaluation of degree of Gamification	126
8.5 Integration team issues	127
8.6 Product owner	128
9 Conclusion	130

Chapter 1

Introduction

This paper documents the process of taking on and further developing a complex full-stack software solution. The project is a part of the Educado Initiative, a free educational platform targeting Brazilian waste pickers, consisting of a cross-platform mobile application for the aforementioned users, a web application for the creators, and a backend connecting them and the database [13].

The main challenges this project proposes are taking on a complex software system, analysing it, and achieving further goals in individual teams following an agile framework.

Select sections of this paper were written in collaboration with other teams working alongside the team of this paper. The start and end points of collaborative sections are indicated with disclaimers.

Collaborative writing begins

In this semester's project, multiple teams work together on the same project. In total, there are six full-stack teams and each team is between five and six students. Each team has its theme for the project, and its product backlog items

(PBIs) that they are responsible for, but the final product will be the same for all groups. Though all the teams are collaborating on the same project, some parts of the project are more relevant for some teams than others. The final product will be represented as both a website and an Android application. Therefore, some teams mainly focus on the website, while others focus on the Android application. To coordinate and work with multiple teams on the same project, finding a good development methodology is essential. The Agile software development methodology, with the use of the Scrum framework scaled with Nexus, has been used to ensure high-quality software development. In addition to working in a larger development environment, an important part of this project is to inherit code written by other students and use this for further development. The code written in this project is based on a four-year-old code base from the students of AAU in Aalborg in 2019. Last year, in 2022, it was rewritten and further developed by 5th-semester software students in Copenhagen. Even though the initial sentiment was to avoid modifying the existing code base where possible, a lot of changes and reworks of the inherited code were needed. This year UI designers, who are students from the University of Brasília created a completely new design. In addition, the code had not been maintained for over a year and did not follow the guidelines for this year's project.

Collaborative writing ends

1.1 Educado

Collaborative writing begins

The project is part of the Mobile Education Project (MEP), which was formed in collaboration with Aalborg University (AAU) and the University of Brasília (UNB) in 2019 for the SGDC initiative. Educado is a part of the MEP with a focus on improving the life situation of waste pickers in Brazil. At UNB the students of this semester and the last semester have been working on user journey mapping,

user interface design, content creation, as well as product management in terms of users' needs and their journeys. The project is made in close collaboration with the product owner, the stakeholders, and the intended users of the product in Brazil. AAU will be responsible for the development of the Educado mobile application and the web application, while UNB will be responsible for the design and requirement criteria.

As part of the Educado project, this collaborative effort was dedicated to enhancing the educational prospects of waste pickers through digital learning. Educado serves as a platform designed to connect with waste pickers and address their educational needs. The primary objective is to offer free access to educational resources and improve the quality of life for the Brazilian population. To maintain user engagement, the app will also offer education through a gamified experience. The layout will be created to provide users with visual designs and animations that will reward them for learning and provide the maximum amount of educational content possible. For the content to be accessible to the target group, there will be a specific emphasis on the development of a web application for qualified content creators. The content creators should, in the web application, be able to provide a variety of courses, which should be appealing to the users.

The project also takes part in both the Sustainable Development Goals (SDG) and the European Region Action Scheme for the Mobility of University Students (ERASMUS). ERASMUS is the European Union's program to support education, training, youth, and sport in Europe [14]. Educado was one of the projects that received funding this year (2023). The SDG is the United Nations' 17 goals for sustainable development [27]. A meaningful part of the work done by software developers and engineers is developing solutions that contribute to a better future. A way of making sure of that is to have the SDGs in mind. In Brazil,

waste pickers and other low-income groups lack access to basic and professional education and have poor social conditions. Therefore, as stated by the projects' product owner (PO), Luiza Cardoso Queiroz Melo, this project aims to provide access to a tailored educational experience in an easy, quick, and dynamic way, to keep the waste pickers engaged and interested in learning [4]. The project's main focus will be to achieve this by providing a mobile education platform for waste pickers in Brazil. However, the project can be up-scaled and help not only waste pickers but also other vulnerable groups around the world. Therefore, if the project is successfully implemented, one could argue that it will affect multiple SDGs. Such as goal number four: *quality education* and goal number eight: *decent work and economic growth* [27].

When it comes to the education part, there are multiple ways the Educado project can have an effect. On one hand, one can use the Educado app as inspiration and a stepping stone into applying to different educations. On the other hand - if there are quality professors that contribute with different courses on the app, and if Educado could collaborate with universities as well as both governmental and private organizations, the app could create certificates, that would be approved as a form of alternative education. This way, Educado can work as an alternative education itself. Either way, it could be argued that all forms of education are great, and especially for vulnerable people even a basic education can have a significant positive effect on their quality of life.

Collaborative writing ends

1.2 The Nexus Framework

The project is structured in accordance with the Nexus Framework, which leverages Scrum principles and employs an iterative and incremental approach to scaling software and product development methods [24]. Applying a scaled version

of Scrum in this project is logical since 6 teams are working on the same product in parallel, introducing a unique set of problems. These problems include increased difficulty in coordination, collaboration, and communication. Tackling the issues effectively will mean delivering value to the customer with a satisfactory frequency.

The following paragraphs are based on Scrum.org's introductory video [23]. Any changes made to the original Nexus Framework are clearly stated.

The Nexus Scaled Scrum Framework consists of roles, events, artefacts, and the rules that bind these elements together [23]. In this project, the Scrum teams work from a single product backlog. Following the Nexus guidelines, unlike in Scrum, product backlog refinement is strongly encouraged. Here, the Scrum teams decide which team delivers which product backlog items, and identify dependencies across teams.

During Nexus sprint planning, the activities of all of the teams during the current sprint are coordinated. Furthermore, the product owner provides domain knowledge and guides selection and priority decisions regarding product backlog items. Each team validates and/or makes adjustments to the ordering of the PBI's, and sprint planning is complete when all teams have finished their individual sprint planning events. Sprint planning also results in a sprint backlog: a composite of a team's product backlog items, highlighting dependencies and flow of work. It should be updated daily - often as part of the daily Scrum [23].

Following the Nexus Framework principles, a daily Scrum meeting is to be attended by representatives from each team. During these meetings, the state of each team's respective increment, and any integration or dependency-related issues are discussed. The new information and developments are subsequently

transferred to each group by their representative and are discussed [23]. For a smoother workflow, this project works with a slightly modified Nexus Framework. Daily Scrum meetings are held internally in the team, and semi-regular integration team meetings are held every 2-3 workdays. The handling of dependency and integration-related issues and transparency are secured by documented and open communication through representatives from each team. This choice was made due to time constraints.

At the end of each sprint, which in this case is every 2 weeks, a sprint review is held. During the sprint review, the product owner (and potentially other stakeholders) provides feedback on the integrated increment [23]. The product backlog is adapted accordingly if needed.

A sprint is concluded with a Nexus sprint retrospective, which is a formal opportunity for a Nexus team to inspect itself and create a plan for improvements to be enacted during the next sprint. This ensures continuous improvement. During the sprint retrospective, the integration team meets and makes issues transparent to other teams. Furthermore, each Scrum team holds their own retrospective meeting and proposes actions to address the potential issues. Lastly, the representatives come together again and agree on how to visualise and track the identified actions to improve [23].

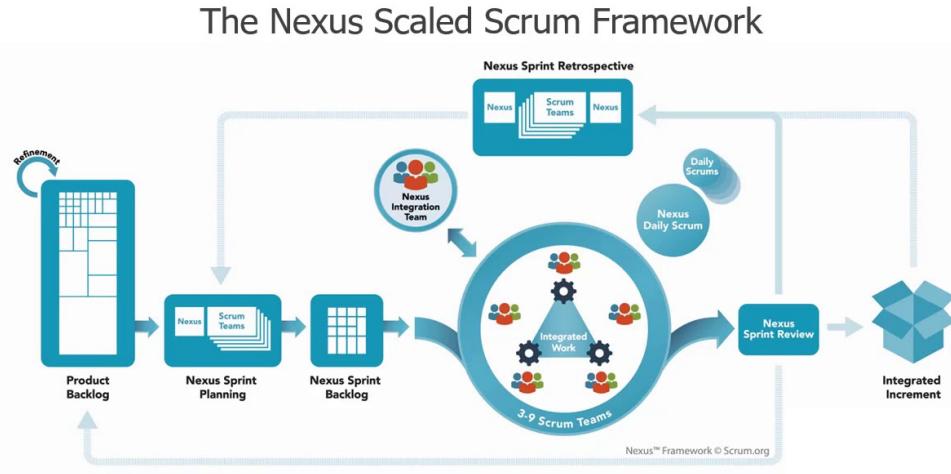


Figure 1.1: Nexus Scaled Scrum Framework [23]

1.2.1 Roles

The roles used in the framework are product owner, repository manager, Scrum team (developers) including a Scrum master, and integration team.

The product owner of this project is Luiza Melo; a Strategy and Project Management Analyst at Nubank and a Production Engineering student at the University of Brasília. The product owner is a domain expert whose skills are used to translate user needs to product backlog items, formulated as user stories. Since the Scrum team does not have direct contact with the users themselves, product backlog items must be accepted by the product owner to be considered done. It is therefore also important to frequently consult the product owner. It is also the product owner's responsibility to manage the product backlog and ensure value maximisation so that all the work done contributes to the business objectives. This is done by making refinements to the product backlog and correctly prioritising PBIs based on size and value [23].

The repository manager of this project is Frederik Bode Thorbensen, who is a Software Engineering student and Research Assistant at Aalborg University Copenhagen. Their responsibility is handling code that is accepted into production.

The Scrum team is responsible for self-organisation according to the standards of the applied Scrum framework, and delivering done increments at the end of each sprint [23]. See more about the definition of done (DoD) in section 1.3. Every Scrum team also has a Scrum master, who ensures that the Nexus framework is correctly applied in practice.

The Integration team consists of a representative from each Scrum team. This team is responsible for ensuring that a done and integrated increment is produced at least once every sprint.. This also includes the aforementioned processes described in section 1.2. The integration team also fulfils a crucial role in interpreting and addressing issues related to the Agile process during sprint retrospectives.

Stakeholders

Before the first sprint, the main stakeholders of the project are identified, discussed, and placed on a stakeholder map. The map placements are based on two axes, that represent the stakeholders' stake in the product and their influence over the product respectively.

Below is a digitalised version of the stakeholder map done by the Nexus Team. This mapping was done before the first sprint and was not adjusted afterwards,

despite the conflicting opinions of the team in this paper.

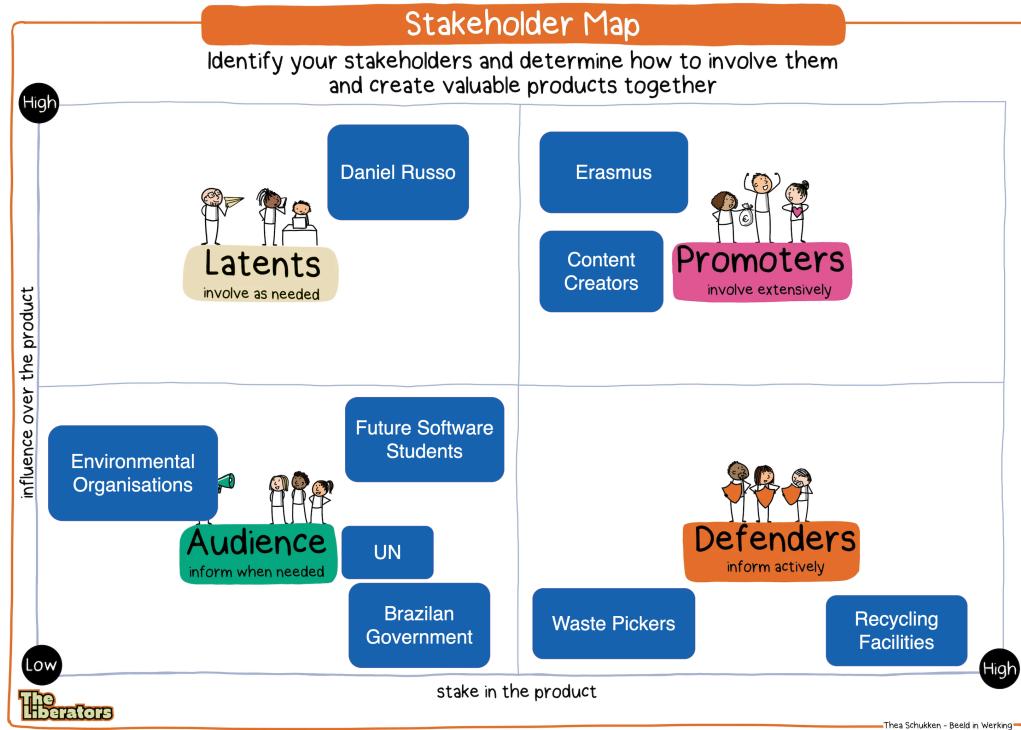


Figure 1.2: Stakeholders of the project [23]

Stakeholders of the project are categorised depending on their level of influence over the product and their stake in the product. At the highest level of influence and stake, Promoters are identified as the group that requires extensive involvement. Content creators and Erasmus are included in this category, with the group's subjective opinion favouring Erasmus as a better fit for the defender category at this time. Erasmus is not directly involved, nor do they have any substantial influence over the product. Without content creators, the application is virtually useless, so taking their needs seriously and consulting them as often as possible is essential. Defenders are to be engaged actively if possible since they have a lot of stake in the product. According to the group, waste pickers should also be considered to have greater influence over the product, because they are the core users, and will be involved in validation tests. The Latents category

includes actors with major influence, but less stake in the product. This category includes Daniel Russo, who practically fulfils the management role in the Agile process. The team in this paper would most likely place Daniel Russo as someone with more stake in the product. The Audience category, which includes Environmental Organisations, Future Software Students, the UN, and the Brazilian Government, is positioned to be informed when needed, indicating a lower level of influence and stakes in the product. The team would place future software students considerably lower on the scale of influence, however, developers should consider future tasks throughout the project.

1.3 Definition of done

In the context of agile methodology, the definition of done represents a consensus reached by a product team regarding the specific criteria that must be met to categorise product backlog items as fully completed [22]. The integration team has formulated these points as an improved version following an initial attempt that was less than satisfactory. Any changes to the DoD should be done with extreme care since all previously accepted items must be re-evaluated if a change is made. If this results in decreased value delivered for the customer, it is not an advisable decision.

The DoD for this project consists of the following points:

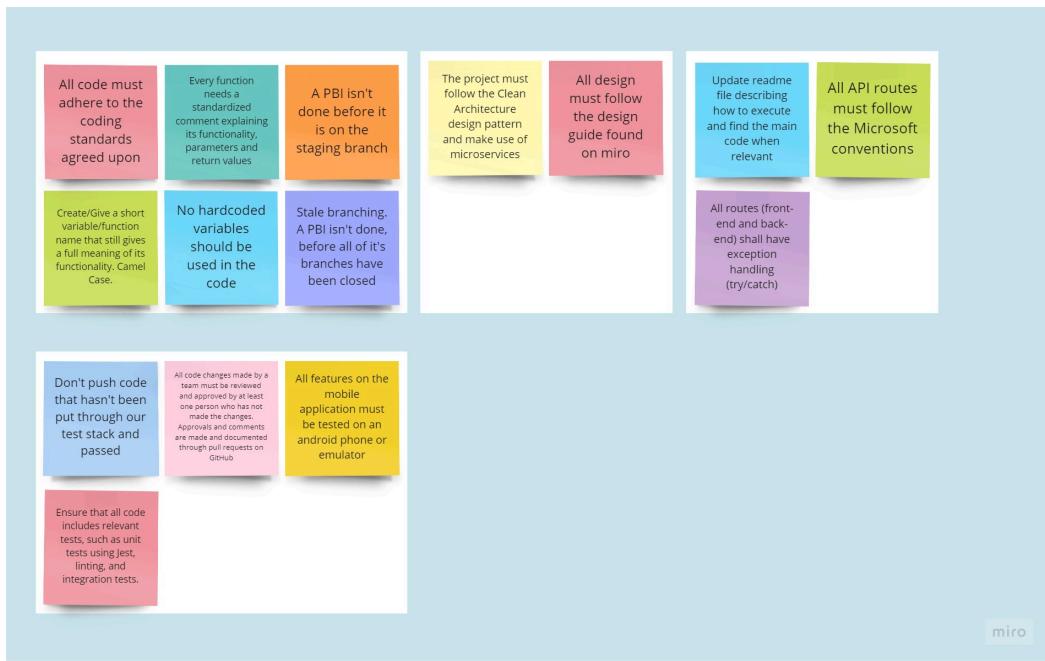


Figure 1.3: Definition of done

1.4 AI-powered tools

In this project, GitHub Copilot [15] and ChatGPT-3.5 [3] are used to assist the team's productivity and workflow. ChatGPT is primarily used to give suggestions for fixing relatively simple issues with JavaScript code and CSS. ChatGPT is not treated as a reliable source, but rather a helping hand. GitHub Copilot is applied as a code completion tool, providing contextually aware code suggestions, thereby boosting overall productivity.

Chapter 2

State of the Project

Since this project has been worked on by several teams beforehand, the first step is to understand and document its current state.

The relevant repositories for the Nexus Team are educado-frontend, educado-mobile, and educado-backend. This paper focuses on the educado-mobile and educado-backend, since the social-gamified learning aspect only directly works with these repositories. The frontend repository is not relevant as it contains the code for the web application.

The mobile app uses the Expo framework, a software library extending on the React Native open-source UI software framework. For styling, Nativewind (Tailwind for React Native) is used, although it is only partially implemented, so the app still uses vanilla CSS in some instances.

The backend follows the clean architecture design principles and uses Amazon Web service as its cloud infrastructure provider (AWS). It employs Node.js and Express for a streamlined backend server, Axios for HTTP requests, and MongoDB for the database.

2.1 Initial Database Diagram

The initial database diagram is made in collaboration with the other teams and can be seen in figure 2.1. The diagram is too simple compared to what the application should be able to do. Compared to the mobile repository and its endpoints, the two repositories do not align, and as a result, a lot has to be resolved before any enhancement work can be started.

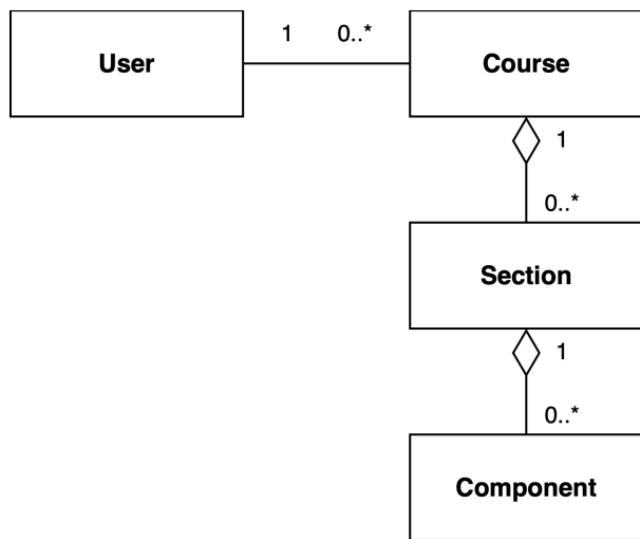


Figure 3.6: Initial data model

Figure 2.1: Figure of the initial database diagram

2.2 Adjustments before the first sprint

2.2.1 Backend

The first and most important thing to be done before the project is ready for further development is to change the cloud storage solution for the backend from

AWS to Google Cloud Platform (GCP). This means that the keys for communication with the cloud and database have to be changed, which delays the ability to make further changes and actually see what the app is supposed to do in action. The keys are published to the developers one week after the initial start of the first sprint. After the keys are acquired, it is possible to create a link between the mobile application and the backend. This has to be done almost from pure intuition since the README files in the two projects are either outdated or lack a lot of necessary information. See more about how the README issue is tackled in section 2.2.3.

Testing the backend repository is made possible with the Jest framework and setting up a MongoDB memory server. The Jest framework handles the setup of the test environment including mocking data, assertions, testing utilities and a lot more to ensure efficient and reliable testing of Javascript code [21]. The MongoDB memory server serves as an in-memory database for testing and therefore makes sure that the test cases do not affect the production database.

2.2.2 Frontend for mobile

The frontend for the mobile application does not currently have a shared stylesheet for the different screens and components. To add to that, the different components and screens use vanilla CSS at the bottom of each file, even though the project is using Nativewind CSS. This is changed so that the components and screens share one styling defined in a Tailwind configuration file, to centralise the different fonts and text sizes, as well as simplify them.

2.2.3 Preparing the development environment

The environment needed for the mobile application includes Xcode for IOS users and Android Studio for Android users. These tools are used to emulate the mo-

bile application on a computer. Before the emulators can be used, a part of the project's dependencies have to be updated.

Another essential step in setting the environment up is to get the frontend and backend to run simultaneously so that communication with the database is possible. This setup phase includes preparing Docker, MongoDB Compass, and the backend repository with updated keys.

The README files for the respective repositories were not updated by previous contributors either, so one of the first steps in the project is to update these files including setup instructions and fixes for specific errors that may arise during the setup process. The group published a README for both frontend setup and getting the frontend and backend to communicate in a local environment. These README files are made available for all of the other teams as well.

2.2.4 GitHub actions, templates, and static code analysis

The Scrum Teams are provided with Issue and Pull Request templates to streamline collaboration and give a unified look to dependencies and commits. A static code analysis tool suggested for the developers is CodeScene, which is a tool used to enhance the quality of the project's code [8]. It gives helpful suggestions for problematic functions and possible ways to reduce complexity by extracting code into their own functions, etc. Unfortunately, monitoring changes over time is not available to the team, since it requires a pro account to access an organisation with multiple repositories. As a workaround, the staging branches are cloned at the end of each sprint, and their current state is inspected. Lastly, the backend is meant to seamlessly integrate microservices by integrating Docker into the project, to decentralise some of the core features of the project.

Chapter 3

Sprints 1-2: Stabilisation and Enhancement

The sprints documented in this section are the structured and iterative development cycles approached in the project, where specific tasks and goals from the product backlog are addressed. Each sprint includes the aforementioned key activities such as Sprint Planning, Sprint Review, and Sprint Retrospective. They are documented to offer a comprehensive overview of the project's progress and insights gained from each iteration, promoting flexibility and adaptability.

The first two sprints of the project focus on the stabilisation and enhancement of already existing features of the application, backend, and web platform.

3.1 Sprint 1

The overarching sprint goal is the following: *"Stabilise and enhance existing features of the Educado platform, focusing on design improvements, usability fixes, and performance optimisation. Prepare the foundation for future feature integrations while ensuring a robust and user-friendly experience for both content creators and learners."*

In the following sections, the sprint backlog items are discussed individually. They are planned and developed in pairs, and additional reviews are conducted internally by other pairs from the group (before external code reviews). Furthermore, since this is the very first sprint of the team, many mistakes are expected and is generally regarded as a valuable learning experience rather than a fully viable development period compared to later sprints.

3.1.1 Sprint Planning

As all sprints do, this sprint starts with sprint planning. As described in chapter 1.2, the forthcoming activities and objectives of the team are clarified, adjusted and coordinated, resulting in a sprint backlog. This backlog includes the product backlog items that the team strives to deliver during the sprint.

Sprint backlog:

- App Login: As a non-logged in user, I want to have the ability to preview the app, so that I can understand the application before register/login
- App Home: As a waste picker, I want to view when I don't have an active course (empty state)
- App Profile: As a waste picker, I would like to be able to see and edit my profile, in order to see what personal information is connected to my profile and edit in case something is incorrect or not up to date
- App Profile: As a waste-picker, I want to have the ability to delete my account

Since the first couple of sprints focus on stabilising and enhancing already existing features of the app, the group's resources are mainly used for implementation rather than research. Following the Nexus Framework, progress and any

potential dependencies were discussed in daily Scrum meetings, providing better transparency internally and with other teams if necessary.

App Login: As a non-logged-in user, I want to have the ability to preview the app, so that I can understand the application before register/login

The PBI is in the mobile section of the product backlog and is concerned with creating a state in which a non-logged-in user is able to read about the app before registering or logging in. The acceptance criteria for this PBI are as follows:

- Brief explanation about the application in 3 steps
- User needs to have the ability to enter OR register

Firstly, the user is shown a loading screen seen in figure 3.1



Figure 3.1: Loading Screen

While the screen is loading, the app checks if the user has opened the app before. If they have opened the app before, the user is shown the login screen where they can also register. Otherwise, the preview section is displayed before the login screen as shown in figure 3.2. The previous section can either be navigated by swiping or pressing on the arrows next to the text.

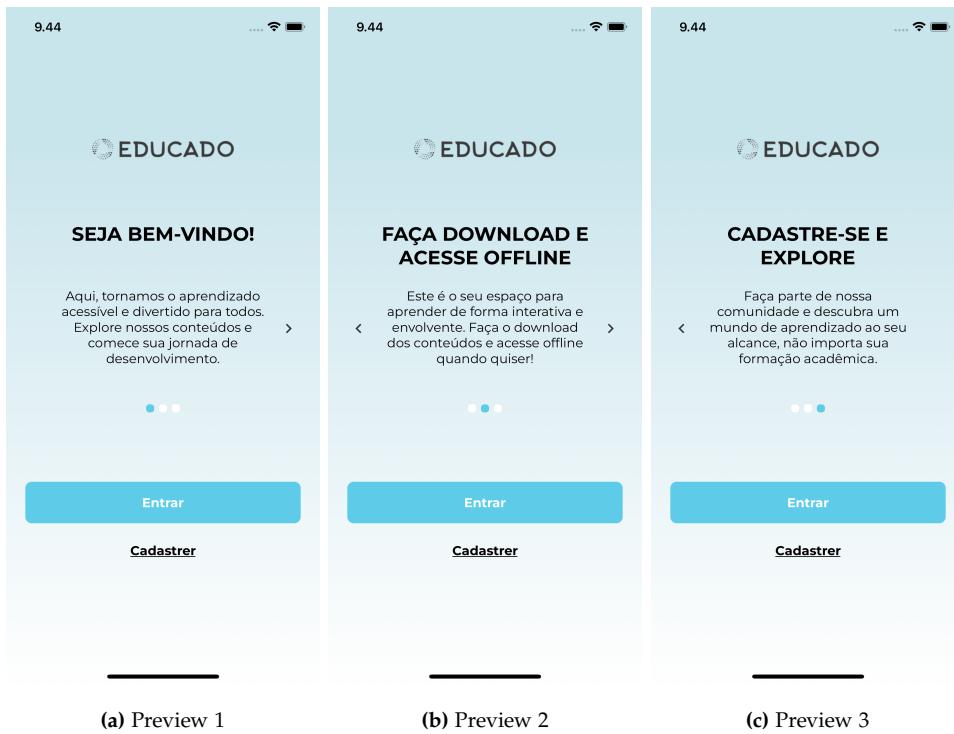


Figure 3.2: Preview Screens

App Home: As a waste picker, I want to view when I don't have an active course (empty state)

This PBI is purely concerned with frontend since the objective here is to create a view that informs the learner, that they have no active courses. The screen can be seen in the mobile app and was developed following the style guide and design layouts from the Figma files provided by the product owner. The acceptance

criteria for this PBI are as follows:

- Notify the user that it is necessary to subscribe to a course to begin studying.
- Show the user how to subscribe to a course.

The text on the screen gives the user the necessary information (see figure 3.3), thus fulfilling the acceptance criteria. Overall, the creation of this view is relatively straightforward; however, certain challenges arise when translating some of the CSS code to Nativewind CSS due to the use of plain CSS in the existing code. There are also a couple of dependencies that must be solved during integration, namely the explore courses button redirecting to the correct page, which at this point is non-existent, and correctly fetching the user's active subscriptions to check if there are any. The PBI was accepted by the product owner at the following sprint meeting. The resulting view can be seen below:



Figure 3.3: No courses view (Empty state)

App Profile: As a waste picker, I would like to be able to see and edit my profile, in order to see what personal information is connected to my profile and edit in case something is incorrect or not up to date

This PBI needs implementation in both the mobile and backend repositories, as a result of the mobile application using APIs provided by the backend such as "`..../api/user/update-email`". The PBI deals with editing the user's information like name, e-mail and profile picture. If the user wants to edit their profile, they can click a button which will take them to the edit profile screen. There is no design mock-up provided for this PBI, so the team decided to make a relatively flexible draft and is expected to refine it based on the product owner's feedback.

The acceptance criteria for this PBI are as follows:

- Name, e-mail, photo or default image
- Users have the ability to edit their personal information

In the first sprint, the edit profile screen looks like this:

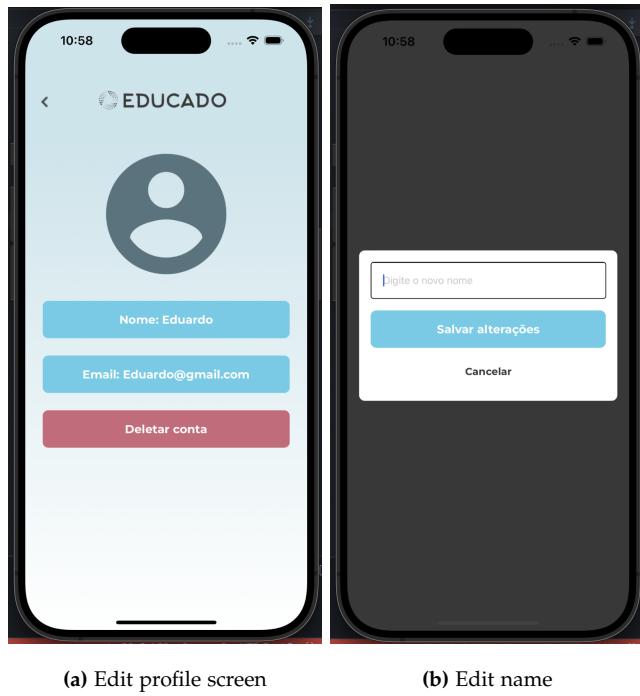


Figure 3.4: Edit profile modal

If the user presses the name or e-mail buttons on figure 3.4a a modal box pops up. In this modal, the user can write a new name and save it as seen in figure 3.4b. This triggers an API call to the backend which is responsible for updating the database. If the API call returns with a code 200, the updated field is also updated in the async storage.

While implementing the design of the edit profile screen, necessary unit tests are made for this PBI. This mostly concerns the backend, testing if the name and e-mail are updated properly in the database.

App Profile: As a waste-picker, I want to have the ability to delete my account

This PBI is already implemented in the mobile application, but the API endpoint it tries to reach is non-existent. This means that this PBI is easily implemented because the backend just needs to make the endpoint available and delete the user when activated. The code for the API looks like this:

Listing 3.1: Delete User Route

```
1 router.delete("/delete/:id", requireLogin, async (req, res)
2   => {
3     try {
4       const { id } = req.params;
5
6       const deletedUser = await User.findByIdAndDelete(id);
7
8       if (!deletedUser) {
9         return res.status(404).json({ error: "User not found"
10           });
11      }
12
13      res.status(200).json({ message: "User deleted
14        successfully" });
15    } catch (error) {
16      console.error("Error deleting user:", error);
17      res.status(500).json({ error: "An error occurred while
18        deleting the user" });
19    }
20  });
21
```

After the setup of the API endpoint has succeeded, some unit tests are made for deleting a user from the MongoDB memory server via the API endpoint.

3.1.2 Sprint Review

Due to a lack of coordination, there was no increment to be presented by the integration team. Therefore, the individual groups discussed their progress with the product owner instead.

Out of the aforementioned PBIs, the second one was accepted by the product owner, which was the empty-state page (no courses). This means that it was ready to be pushed from the team's feature branch to the development branch. At this time, the DoD does not include that a feature must be in the staging branch before being considered done but is discussed and swiftly added following the sprint review.

As for the delete-user and welcome-screen PBIs, additional unit tests are requested before being considered for acceptance.

The edit-profile PBI was considered incomplete because it requires further refinements in its design, the first and last name must have their respective fields, and the change phone number field must be changed to e-mail. The new e-mail field must have validation as well to ensure that the user does not enter an invalid e-mail address, nor one that matches with their current one. Lastly, the feature of uploading a profile picture is missing.

The CodeScene static code analysis tool is set up and the most problematic files are immediately fixed using CodeScene's own suggestions. Overall, code health is vastly improved and is currently in the 8+ range out of 10. The team of this paper implemented code health in its acceptance criteria for the individual PBIs, as it wasn't agreed upon to include it in DoD.

3.1.3 Sprint Retrospective

During the retrospective, the integration team established that the teams need to conduct better and more frequent stakeholder communication with the product owner. The acceptance criteria can therefore become more refined, and aligned with the expectations of the product owner.

Furthermore, a central element of Scrum is to consistently deliver value to the product owner, which was not done in this sprint. Therefore the teams should strive to push their increments into staging before the next sprint review, and GitHub Actions and static code analysis must be implemented into the production pipeline for improved delivery.

Improving the communication between teams (cross-functionality) is vital, both considering dependencies and code merging. Therefore, communication should be conducted through open channels, such as the team's Discord channel or GitHub issues to create better transparency. Before merging, additional code reviews must be done by other teams to ensure a smoother process.

To improve team autonomy, dividing larger PBIs into smaller ones with matching acceptance criteria can be a good option, and creating "dummy data" as a temporary stand-in for database communication is advisable if needed. Working with "dummy data" is used to work around dependencies, to not stop the implementation of some PBIs.

Looking at the overarching sprint goal quoted at the start of the section, this increment resulted in considerable progress regarding enhancing or at least fixing existing features but was kneecapped by the time required to understand the system at hand.

Integration team developments

Concluding the first sprint, the integration team established that automation must be improved, and the definition of done must be revised to better reflect the expected quality of the product, as well as being more clearly defined. To achieve this, GitHub actions are to be implemented for linting and type-checking, and all tests must pass before a pull request can be considered for a review. As for the integration team itself, meetings should be held with higher frequency and everyone is reminded that dependencies and issues are to be documented on GitHub rather than on communication channels.

3.2 Sprint 2

The overarching sprint goal is the following: *"Stabilise and enhance the Educado platform by enabling web course creation and mobile app access. Focus on design, usability, and performance improvements while ensuring future feature integration and user-friendliness."* The key difference between this and the previous sprint goal is that this sprint focuses on further development and improvements to be done to the old system, rather than fixing what is already there. Nevertheless, the remaining PBIs related to fixing the old system which have yet to be accepted, must be completed, adhere to the project's DoD, and be pushed into staging. Since only minor fixes are required for most of the leftover PBIs, they do not get their own items in the paper's sprint backlog.

3.2.1 Sprint Planning

Sprint backlog:

- App Profile: As a waste picker, I would like to be able to see and edit my profile, in order to see what personal information is connected to my

profile and edit in case something is incorrect or not up to date (cont. from Sprint 1)

- App Course [Frontend]: As a waste picker, I want to be able to answer exercises
- App Course [Backend]: As a waste picker, I want to be able to answer exercises
- App Course: As a waste picker, I want to be able to review the exercises and get feedback from it

Originally, answering exercises and getting feedback was a single PBI, but was chosen to be broken up into backend and frontend PBIs for better clarity.

Minor fixes to the first sprint's PBIs

The delete account button is, based on product owner feedback, moved to profile settings, see figure 3.5. Furthermore, unit tests are made for the API handler to make sure that the URL works properly. Unit tests are also written for the welcome page, and all translation errors are corrected.

App Profile: As a waste picker, I would like to be able to see and edit my profile, in order to see what personal information is connected to my profile and edit in case something is incorrect or not up to date (cont. from Sprint 1)

The revised acceptance criteria for this PBI are as follows:

- Have a default profile picture
- Users have the ability to edit their personal information (first name, last name, email)

- Edit name(s) and email in database (backend)
- Validate name(s) and email
- Have relevant error messages in Portuguese
- Do not create new issues in CodeScene (static code analysis tool)

Based on feedback from the first sprint review, the looks and logic of the profile settings are updated. Additionally, the possibility of uploading a profile image is separated into a new PBI due to its size and was subsequently passed onto the video streaming scrum team due to the technical relevancy of file buckets. It was formulated as the following user story: "[App Profile]: As a waste picker, I want to upload a new profile picture or choose the default profile picture in my profile settings."

The functionality of the UI stays the same, but styling is made more uniform with the rest of the application. See the updated version here:



Figure 3.5: Revised profile settings screen

The logic of the email model is updated with semantics for validating the email, and a confirmation field is added, which is used to ensure that the user enters the correct email. The semantics of the email are integrated in coordination with the other team working on the login screen to ensure a unified semantic check.

The singular name field is changed to separate first and last name fields. The modal checks if the new first or last name matches with the current first or last name, and gives the proper warning before updating. Generally, all warnings and text fields are translated into Portuguese.

Lastly, the field 'name' in the database is separated into first name and last name, because of the update of the edit profile screen. This means that new unit tests must be made for updating the first and last names in the database. In the first sprint unit tests were made to update the MongoDB memory server, but the API endpoints were not tested. Therefore in this sprint unit tests are made to test the

API endpoints, for updating email, first name and last name.

App Course [Frontend]: As a waste picker, I want to be able to answer exercises

This covers the visual part of letting the user answer a given question within an exercise. The acceptance criteria for this PBI are as follows:

- Question with 4 options (1 right and 3 wrong)
- User can only move to the next one when choosing one of the alternatives
- User should be able to go the review page after finishing the exercise

This screen is designed using the styling guides provided by the product owner. There is already a file in the repository for this screen, which was leftover from the previous developers. It is generally agreed upon, that if leftover code is found, it should be analysed for usable code that could be reused instead of deleted. In an effort to reuse the old code, it was quickly discovered that it did not follow the current vision of the product, and was therefore largely discarded. As minor exceptions, the "leavebutton" component and progress bar seen at the top of the finished PBI are kept using the old code. Everything else presented in this section is newly created during this sprint, such as the information bar at the bottom of the screen, which is now a component to make it reusable on other screens.

The screen works by fetching the relevant exercise from the database and displays the question within the exercise object. Afterwards, the array within the exercise object which contains every possible answer is mapped through and displayed. This solution is made to be dynamic for every exercise.

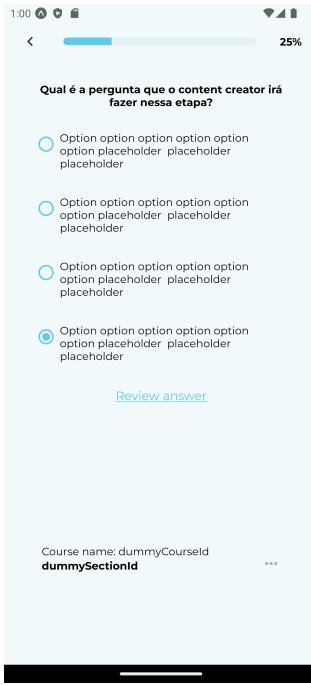


Figure 3.6: Answering exercises

App Course [Backend]: As a waste picker, I want to be able to answer exercises

This PBI covers the backend part of letting a user answer exercises and the acceptance criteria for this PBI are as follows:

- Get the exercise from the database
- Update the user's progress after completing an exercise
- Have unit tests for get and patch routes

Firstly, the model for the user in the database has to be updated with a completedCourses array. Inside each completedCourse is a new array with the courses' sections, and inside each section is its completedExercises for the specific user. The structure of the completedCourse field in the user model looks like this:

```
1 completedCourses: [
2   {
3     courseId: {
4       type: Schema.Types.ObjectId,
5       ref: 'Courses'
6     },
7     completedSections: [
8       {
9         sectionId: {
10          type: Schema.Types.ObjectId,
11          ref: 'Sections'
12        },
13        completedExercises: [
14          {
15            exerciseId: {
16              type: Schema.Types.ObjectId,
17              ref: 'Exercises'
18            },
19            isComplete: {
20              type: Boolean,
21              default: true
22            }
23          }
24        ],
25        isComplete: {
26          type: Boolean,
27          default: false
28        }
29      }
30    ],
31    isComplete: {
32      type: Boolean,
```

```
33         default: false
34     }
35 }
36 ]
```

The users of the application are using mobile devices which for the most part have less computing power than computers, which the content creators use. The users, therefore, need fewer computations to get their completed courses, sections and exercises, making the application more responsive. Based on the location of the users, it is reasonable to assume that the internet connection is not reliable, and therefore they have to use the application offline. When they are offline and complete exercises, it is easier to simply update the logged-in user locally, and then when the user gets an internet connection they can update their user in the database. If a new collection is made for completed courses, sections and exercises, the user would need to store more collections locally while offline, thus using more memory on the device. Taking this into consideration, it makes more sense to add the new fields inside the user model instead of creating a new collection.

3.2.2 Sprint Review

During this sprint review, the integration team presented only the app-login PBI, since it was the only one on staging-branch. Following that, each team presented their progress individually.

The team of this paper had all PBIs from the first sprint accepted, which were the welcome screen, delete-account, and edit profile.

As for the frontend part of answering exercises, some adjustments must still be made to adhere to the current definition of done. The backend and feedback

parts are missing most items but were presented to show progress to the product owner.

Lastly, the overall code health determined in the static code analysis is neither unchanged or improved.

3.2.3 Sprint Retrospective

During the sprint retrospective, it was assessed that the management support is to be utilised more efficiently, so the integration team should hold a weekly meeting with the repository manager, where all questions are addressed rather than sending direct messages.

As for team autonomy, other than using "dummy data", one can also mock dependencies rather than being reliant on another group's work. Furthermore, database models are to be updated, so every team can work in a uniform manner. Daily stand-up meetings should also receive more attention if needed, to ensure internal transparency and efficiency through frequent communication.

A major issue of this sprint was the lack of value delivered despite the progress made by individual teams. Therefore, more time should be set aside to tackle merging conflicts and related challenges. For cross-team functionality, merging smaller increments instead of processing numerous PBIs at a time is preferable, so it should be the standard in future sprints. Furthermore, CodeScene static code analysis is to be added to the production pipeline to ensure improved code quality.

Lastly, it is strongly advised that everything that can be automated should be automated.

Integration team developments

It was concluded that more resources must be spent merging and reviewing merged code, while tests, linting, type-checking, and static code analysis should all be automatically done when opening a pull request. Furthermore, all teams should beware of only pushing relevant changes when creating pull requests. A good practice is to clone dev and merge one's one branch into this temporary branch before merging into dev, so only the new files are shown on the pull request that is to be reviewed by other developers.

Chapter 4

Research on social-gamified learning experience

Now that the initial enhancement and stabilisation part of the project is finished, the primary focus of the group can finally come into the spotlight. Unlike previous PBIs, where a clear plan and design were laid out for the design and development processes, the upcoming sprint also requires a more thorough analysis of the domain of social gamification and planning on the group's part. Since the product owner is not a domain expert in this specific field, numerous discussions are also planned to map out the most beneficial features to be implemented.

4.1 Theory

4.1.1 What is social gamification?

Before delving into any specifics of social gamification, one must understand what the term represents. There is no one clear-cut commonly agreed-upon definition, as it is a relatively new, and continuously evolving field. The definition for gamification that this paper works with is the following: "*Gamified learning*

approaches focus on augmenting or altering an existing learning process to create a revised version of this process that users experience as game-like. Because gamification is usually not used to replace instruction, but rather to improve it, effective instructional content is a prerequisite for successful gamification.” [18]. Given this definition, the “social” adjective means that it is to be interpreted in, or scaled to function in a social setting.

4.1.2 Why is social gamification a good thing?

To create something of real value, one must also understand the relevancy of social gamification and its strong suits. In this paper [18], the authors discuss the results of their research and unanimously conclude that social gamification is a promising prospect; *“Evoking social interactions via gamification in the form of combinations of collaboration and competition was most promising for behavioural learning outcomes. This result for behavioural learning outcomes is in line with evidence from the context of games, showing that combinations of competition and collaboration in games are promising for learning.”* [18]. The paper emphasises social interactions evoked by collaboration or competition. These can be achieved in many ways, and open up numerous possibilities. Even individual work can be socially gamified by providing competition in the form of leaderboards for instance. Nevertheless, these two factors should be considered as the fundamental ideas or overarching goals of all social gamification-related PBIs.

The paper further elaborates on the relevancy of social gamification; “The present meta-analysis supports the claim that gamification of learning works because we found significant, positive effects of gamification on cognitive, motivational, and behavioural learning outcomes.” [18].

Deciding between collaboration or competition does not seem important accord-

ing to the data: “*For the cognitive and motivational learning outcomes, no significant difference in effect sizes were found between the different types of social interaction.*” [18].

4.1.3 4 Phases of the gamification journey

Although the relevancy and high-level idea of social gamification have been clarified, more specific guiding principles to achieve these results are also helpful when crafting and discussing individual ideas. For this purpose, the book Actionable Gamification: Beyond Points, Badges, and Leaderboards by Yu-kai Chou, a gamification expert and pioneer, is used as the primary source [5].

The main point of social gamification is to provide motivation in exchange for more engagement. Note, that more engagement can mean scaling the number of users, as well as the longevity of user engagement. A social-gamified experience consists of 4 experience phases; discovery, onboarding, scaffolding, and endgame [5].

Discovery

The Discovery phase is about getting introduced to and gaining product awareness. Here, the potential customer learns WHY they should become a user. A first impression is crucial; see for instance the immaculate detail of the packaging of a new iPhone or MacBook. The presentation should be brief, motivating, and appealing to the specific target group of the product [5].

The welcome page crafted during the first sprint could certainly be considered a part of the packaging of the product, which means that the group has already contributed to improving this factor of gamification in a meaningful manner. Packaging, of course, does not directly influence the competition or collaboration aspects but is more of a core element in the user experience. Without a proper

introduction, the lifetime of a user could be over before it begins, thus largely decreasing the value provided in terms of user longevity and engagement.

Onboarding

Identity is a fundamental element of onboarding. Identity covers the process of creating an account and choosing a profile picture, username, etc. With this, the user has a perceived ownership of their progress. Identity is worth considering later on, implementing personalised messages, for instance, boosting a product's capability of highlighting individuality. Based on the Octalysis Framework [9], the core drive, Ownership & Possession is very relevant here. It refers to the desire to own and control things, and the satisfaction that comes from possessing or being responsible for something [5].

Seeing personal development on one's profile rather than just on a course page could contribute to the mental image of tying one's progress to a user. Furthermore, depicting win-states, e.g. completed courses, level, points, and badges on one's personal profile can contribute to the interpretation of progress as personal progress rather than just progress on an app.

Other than gaining identity, onboarding in a social-gamified setting is about figuring out what the "game" is and where the user stands in the "game". Familiarising the user with the options, mechanics, and "win-states" is crucial.

This could be done using a tutorial without proposed risk, such as in a practice run of a board game where the players play with open cards.

Using the Octalysis Framework, it is worth mentioning three drives in the onboarding phase. For the sake of clarity, a leaderboard example is used to exemplify these drives. A leaderboard presented to new users is something already

discussed, so it makes perfect sense to discuss the core drives in this specific context.

Development & Accomplishment: It encompasses the drive to make progress, achieve goals, and overcome challenges. Displaying a leaderboard can tap into the user's desire for achievement and progress. By showcasing the accomplishments of top performers, new users can be motivated to strive for recognition and progress on the leaderboard themselves.

Social Influence & Relatedness: This drive involves the desire for social interaction, influence, and a sense of belonging within a community. A leaderboard can foster a sense of community and social interaction among users. New users may feel encouraged to engage with the platform or community, as they observe the achievements and activities of other participants, thus creating a sense of relatedness and social influence.

Scarcity & Impatience: It relates to the motivation derived from the perception of limited availability or the fear of missing out on something valuable. Introducing a leaderboard during the onboarding stage can create a sense of urgency and drive among users, especially if the leaderboard highlights limited spots or rewards for top performers. This can encourage new users to actively participate and compete to secure their position on the leaderboard before it's too late.

Scaffolding

Now that the user is familiar with the “rules” of the “game”, their objective is to achieve as many “win-states” as possible. To not lose the user, they should be motivated to come back on a regular or even daily basis, with daily rewards or streaks for instance [5].

All 8 core drives of the Octalysis Framework are crucial in this phase to achieve success. Other than the drives described above, it is worth elaborating on the development and accomplishment drive, which encompasses the drive to make progress, achieve goals, and overcome challenges. Progression should have a point, otherwise "win-states" become meaningless.

"Win states" in the context of the current state of Educado encompass correct answers, completed sections, completed courses, and levelling up. These could, in concept, be extended with badges, a leaderboard, statistics such as answer accuracy, and another team's focus, certificates.

The leaderboard example from above is arguably a good way to retain several of the core drives, as long as healthy competition is present. In some clicker games, to achieve an illusion of competition, some bots are also present on the leaderboard, posing as real competitors, their scores reacting to the user's own score. This could, however, be classified as black-hat gamification, since it is arguably deceiving the user. In a real-life setting, many such practices are applied to create a feel of community and competition despite the potential lack of userbase.

Endgame

The endgame represents the phase where the user has done everything there is to do and is starting to feel like there are no longer any unexplored "win-states". A possible solution is regular updates enriching the content of the product [5].

If Educado's idea of an ever-evolving creator community comes to fruition, the issue is virtually solved. Still, providing new challenges and ways of succeeding to the user is a common practice worth considering to keep so-called long-time "veteran users" hooked.

Continuing with the leaderboard example, if a leaderboard is reset frequently, or is changed to be based on a different statistic, there are suddenly new “win-states”, furthermore, this plays on several other core drives, as it implements a sense of unpredictability and scarcity into the system. Appreciating “veteran users” is also an important aspect of the endgame.

4.1.4 Framework for social gamification

Yu-Kai Chou’s book [5] also provides a walkthrough of universally applicable guidelines to enable and enhance the social gamification aspect of a product, and common pairs of game mechanics and dynamics.

Guidelines [5]:

1. **Allow Repeated Experimentation:** Learning activities, like games, should allow repeated experimentation to reach a goal.
2. **Include Rapid Feedback Cycles:** Immediate feedback helps students improve their strategy and increases the chances of success in the next attempt.
3. **Adapt Tasks to Skill Levels:** Good games help players realistically believe in their chances of success. Different levels of goals adapted to students’ skills improve motivation.
4. **Increase Tasks’ Difficulty as Students’ Skills Improve:** Adapting tasks to the skill level of each student improves their expectations of completing the task successfully.
5. **Break Complex Tasks into Shorter and Simple Sub-Tasks:** Allowing students to complete small sub-tasks within a larger task helps them deal with complexity in a divide-and-conquer approach.

6. **Allow Different Routes to Success:** Each student should be able to choose a different sequence of sub-tasks, following their own route to complete the task.
7. **Allow Recognition and Reward by Teachers, Parents, and Other Students:** Being rewarded and appraised promotes students' social status.

Most common game mechanics and dynamics [5]:

Game elements	
Game mechanics	Game dynamics
Points	Reward
Levels	Status
Trophies, badges, achievements	Achievement
Virtual goods	Self expression
Leaderboards	Competition
Virtual gifts	Altruism

Figure 4.1: Game Mechanisms and Dynamics [5]

This information should be kept in mind and used in discussions when brainstorming and developing features related to gamification.

4.2 State of the art

Following the exploration of the theoretics of social gamification, it is worth taking a closer look at a couple of real-life examples of how it is implemented. In the following section, research regarding current uses of gamification that encourage learning is documented and presented. This is done to confirm the validity of the previously established knowledge regarding gamification, and also to document

where the inspiration came from which inspired some of the elements seen in the Educado app.

4.2.1 Duolingo

Duolingo is a popular and widely used education platform and is number 2 on Apple's App Store under the education category as of writing this paper [1]. It got popular by taking a gamified approach to teaching its users different languages, and they have since then expanded to other categories such as math and music theory.

The app makes it very clear to the user that it uses gamification to ease their learning journey by making it more fun.

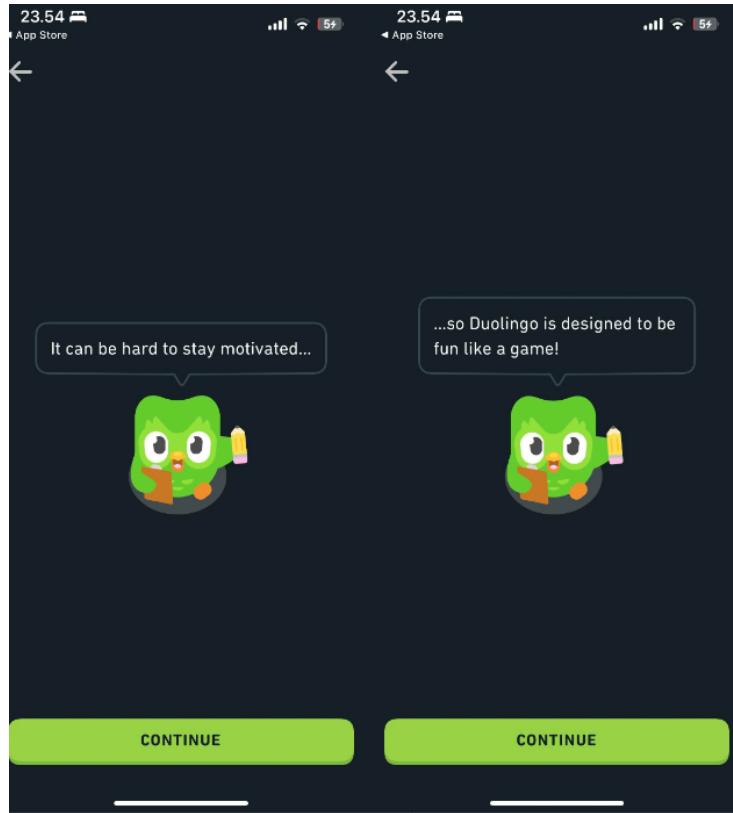


Figure 4.2: The screen shown to new users of Duolingo

The introduction to Duolingo explains to the user what they can expect from the application, while also encouraging the user to participate. Subsequently, the user is asked to choose their personal goals, which is done by selecting a language and a milestone of how much they want to learn within a given time frame. This is important to mention as it is a key step in the phases of the gamification journey, which is the onboarding phase. The user is introduced to all the core mechanics in a playful manner, and the introduction does not overstay its welcome.



Figure 4.3: Encouraging messages that give the user a sense of community

In the onboarding phase, the user is allowed to create their own avatar. This otherwise meaningless feature allows the user to get a sense of individuality, which makes the learning experience more personal. The avatar becomes the face of the user's progress, as it also shows their statistics underneath their avatar on their profile page.



Figure 4.4: The profile page in Duolingo

On the profile page, some of the previously mentioned game elements are seen. It is a summary of the user's journey in the app and shows things such as points and achievements. These are key elements in game design, but in the case of gamification, they help the user track their progress and motivate them to unlock achievements and levels by working towards their goals which they made during the discovery phase. Furthermore, these elements affect multiple core drives, as they present numerous win-states for the user.

In terms of social gamification, Duolingo also uses a leaderboard. This works by enabling users to compete against each other in leagues, which are weekly leaderboards. This ties into the development & accomplishment phase, as it encourages the user to engage with the application more as their progress has a possibility of being showcased among other top performers. In a blog post made

by Duolingo, it is mentioned that the leaderboards were introduced because they found that "*a little competition worked for a *lot* of learners!*" [26].

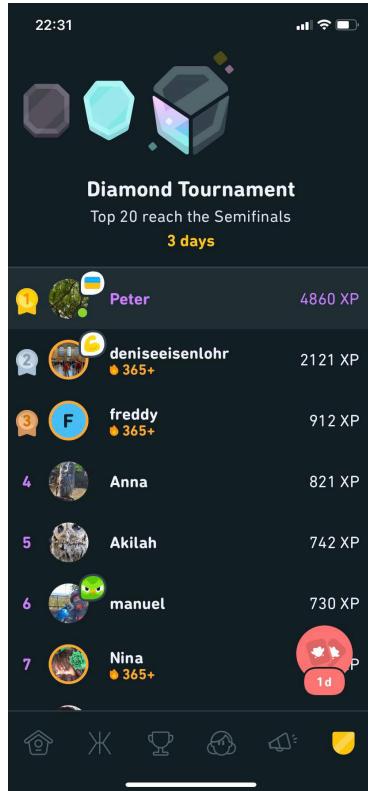


Figure 4.5: Leaderboard in Duolingo

In addition to leaderboards, Friends Quests is another feature that uses social gamification to ease the barrier of entry by allowing the user to learn with their friends [9]. Every week users are paired up with a random friend of theirs on Duolingo, and then they are assigned a random challenge. These challenges can be everything from gaining a certain number of points to completing a certain number of lessons. "*Everyone needs something different to motivate them, but we've seen that Duolingo's social features, like adding friends and congratulating them on learning milestones, can actually boost your learning!*" [9]. Duolingo tries to expand the sources of encouragement to the user by enabling users to hold each other accountable to a shared goal and stay committed to learning.

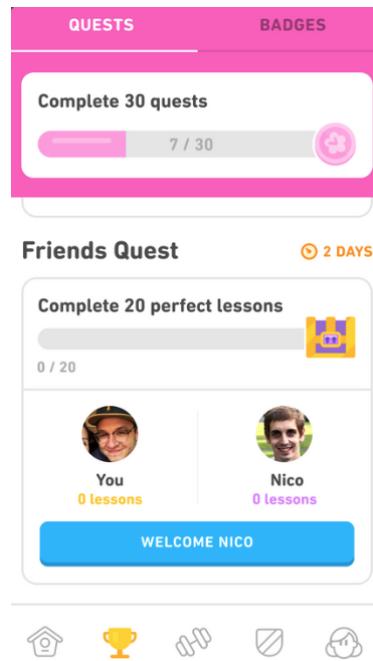


Figure 4.6: Friend Quests in Duolingo

Duolingo's gamification strategies serve as a validation of the effectiveness of incorporating game elements into educational platforms. The success of Duolingo, reflected in its high ranking on Apple's App Store and widespread user engagement, underscores the potential impact of gamification on learning outcomes.

4.2.2 Brilliant

[H] Brilliant is another online learning platform that focuses on interactive and problem-solving-based education, which also incorporates elements of gamification to ease the learning experience by making it fun [2]. The two applications are compared here as they are very similar in what they are trying to achieve, and the methods they use to do so.

Brilliant lets itself inspire a lot from Duolingo. Everything from when points are given to the user during an exercise, to the way their courses are navigated

through like stages from a video game.

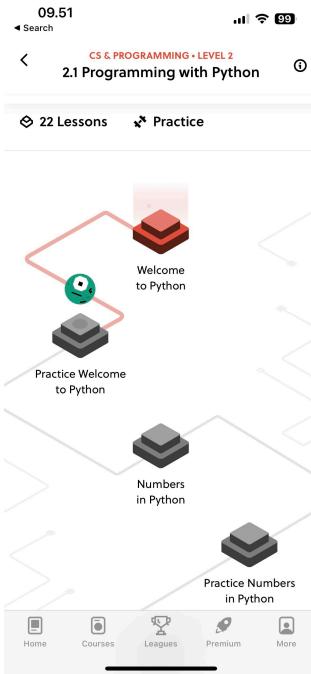


Figure 4.7: An example of how courses are navigated through in Brilliant

One feature that is used in both apps, but is not as fleshed out in Brilliant is the leaderboard system. It works the same in both apps, but since the friends system is absent in Brilliant, the leaderboard is not as effective. Part of Duolingo's research concluded that learning together and motivating each other by displaying progress to each other, can make the user engage with the application more [9]. The user might not feel inclined to engage with these social features, if there are no other users to display these statistics and achievements to.

Brilliant and Duolingo share common ground in their pursuit of interactive and gamified learning experiences. Brilliant draws inspiration from Duolingo's successful gamification elements, incorporating features like streaks and course navigation akin to video game stages. However, the absence of a robust friends system in Brilliant somewhat diminishes the effectiveness of its leaderboard, a

feature that Duolingo leverages to enhance user engagement through social interaction and friendly competition. The gamification of learning appears to be a promising approach, but the effectiveness may vary based on the execution of social features and the depth of the gamified elements within each platform.

4.2.3 Sub-conclusion

Having taken a look at the inner workings of social gamification and concrete examples, the team is now better prepared for a valuable discussion with the product owner. Based on core drives and maximising engagement longevity, the items the team presents at the upcoming meeting are the following; highlighting win-states on the profile page, providing a better onboarding experience using an introductory course before real ones can be accessed, achievements or badges for providing new win-states, already existing win-states e.g. correct answers, section completion, course completion receiving unique congratulatory messages and/or statistics, points and levels, streaks, and a leaderboard based on the aforementioned point system. Although not all of these features are expected to be developed due to time constraints, those that do could take inspiration from already existing solutions - the team should provide value with good velocity rather than reinventing the wheel.

Chapter 5

Sprints 3-6: Gamification

The primary focus of the sprints described in this section is to enhance the user experience by improving the social-gamification aspect of the Educado app. Other than describing the development process of the product backlog items, they are also related to the research and theoretical background of social gamification.

5.1 Sprint 3

The overarching sprint goal is the following: *"Enhance the user experience on the web and app, including optimising course editing on the web and improving the social-gamified learning experience on the app. Prepare for stakeholder validation and ensure deployment readiness."* Other than finishing up every previous PBI, improving the social-gamified learning experience was a deciding factor in selecting the forthcoming PBIs for this sprint.

Starting the gamification process with points provides a solid groundwork for numerous other features that appeal to the core drives of gamification. Despite being a simple concept it unlocks the potential for levels, leaderboards, rewards,

and several other items that can create competition amongst users, and have a positive effect on progress ownership, empowerment and epic meaning.

5.1.1 Sprint Planning

Sprint backlog:

- App Course: As a waste picker, I want to be able to review the exercises and get feedback from it (cont. from sprint 2)
- App Course [Backend]: As a waste picker, I want to be able to answer exercises (cont. from Sprint 2)
- App Course: As a waste picker, I want to see me getting points after completing lectures/exercises on a course
- App Course [Backend]: Implement points in Database

Minor fixes to the first sprint's PBIs

The frontend of the exercise screen is finalised and pushed into the development branch following the sprint review. It is first merged and pushed together with the corresponding backend code during this sprint to ensure correctness. Smaller issues in the development branch, such as incorrect redirection, and the use of icons instead of SVG types for arrows are also fixed. Additional unit tests are also added, due to being requested during code review.

App Course: As a waste picker, I want to be able to review the exercises and get feedback from it (cont. from sprint 2)

This PBI is an enhancement of the previously passed-on prototype from sprint 3.2; the team is further developing the existing exercise answers view (see figure

3.6). Providing an instant feedback loop also provides value towards gamification. The acceptance criteria for this PBI are as follows:

- The app should provide feedback on the user's performance after completing an exercise
- The feedback should be clear and concise, highlighting areas of strength and weakness
- All options should have justifications for being correct or wrong that must be presented to the user

By using conditional statements, the team applies conditional styling, thus changing the view according to the review of the answer. When the "Confirmar resposta"-button is clicked, the answer is evaluated to true or false, which is then used to determine whether the colour of the scroll-view should be green (the answer is correct) or red (the answer is incorrect). When the button is clicked the feedback text for each option is also shown and is also colour-coded. The resulting view from this PBI can be seen below 5.1:

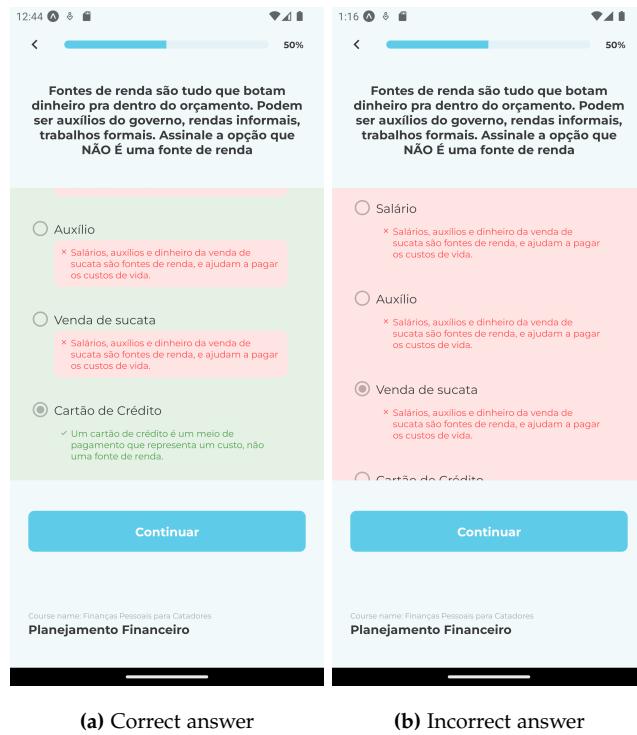


Figure 5.1: Answer feedback

In addition to the changes made to the frontend, some changes were also made to how the data was fetched and displayed to the user. In the previous iteration of the PBI, the data displayed was not fetched from a database but rather from dummy data, which is a set of information or data created for testing, development, or demonstration purposes. It is not real or actual data but is designed to mimic the format and structure of real data. In this iteration of the PBI however, the data displayed is actually fetched from a database. This is done by finding an exercise in the database using a given ID, and then displaying the relevant data in the view. In addition, the course title and section title, which the given exercise is a part of, are also displayed in the bottom component of the view. This is also gathered from the database. This is done with the following HTTP requests to an API:

Listing 5.1: api.js

```
1 export const getCourseByid = async (courseId) => {
2   const res = await axios.get(url + "/api/courses/" +
3     courseId);
4   return res.data;
5 }
6 export const getSectionByid = async (sectionId) => {
7   const res = await axios.get(url + "/api/sections/" +
8     sectionId);
9   return res.data;
10 }
11 export const getExerciseByid = async (exerciseId) => {
12   const res = await axios.get(url + "/api/exercises/" +
13     exerciseId);
14   return res.data;
}
```

These are asynchronous functions that send HTTP GET requests to retrieve information about specific exercises, courses, and sections by their ID.

App Course [Backend]: As a waste picker, I want to be able to answer exercises (cont. from Sprint 2)

The acceptance criteria in this sprint are the same as in sprint 2 [3.2.1](#). In this sprint, the user model is implemented correctly from sprint 2, and the functionality has to be made. As a start, an API endpoint is made for adding an exercise to the user's completedCourses field. The logic for doing so can be found in this code:

Listing 5.2: markExerciseAsCompleted method.

```
1 async function markExerciseAsCompleted(user, courseId,
2   sectionId, exerciseId) {
3
4   const completedCourseIndex =
5     user.completedCourses.findIndex(completedCourse =>
6       completedCourse.courseId.equals(courseId));
7
8   if (completedCourseIndex === -1) {
9     await UserModel.findByIdAndUpdate(
10       user._id,
11       {
12         $ push: {
13           completedCourses: {
14             courseId,
15             completedSections: [{ sectionId,
16               completedExercises: [{ exerciseId }] } ],
17             isComplete: false
18           }
19         }
20       }
21     );
22   } else {
23     const completedSectionIndex =
24       user.completedCourses[completedCourseIndex].completedSections.
25       findIndex(completedSection =>
26         completedSection.sectionId.equals(sectionId));
27
28     if (completedSectionIndex === -1) {
29       await UserModel.findByIdAndUpdate(
30         user._id,
31         {
32           $ push: [
33             `completedCourses.$`
```

```
        {completedCourseIndex}.completedSections `]:  
        {  
          sectionId,  
          completedExercises: [{ exerciseId  
            }],  
          isComplete: false  
        }  
      }  
    }  
  }  
);  
} else {  
  const isExerciseAlreadyCompleted =  
    user.completedCourses[completedCourseIndex].  
    completedSections.some(completedSection =>  
      completedSection.sectionId.equals(sectionId) &&  
      completedSection.completedExercises.some(completedExercise  
        =>  
        completedExercise.exerciseId.equals(exerciseId))  
    );  
  if (isExerciseAlreadyCompleted) {  
    throw errorCodes['E0801'];  
  }  
}  
  
await UserModel.findByIdAndUpdate(  
  user._id,  
  {  
    $addToSet: {  
      [`completedCourses.$  
        {completedCourseIndex}.completedSections.$  
        [section].completedExercises`]: {  
          exerciseId }  
    }  
  }  
);
```

```
49         }
50     } ,
51     {
52         arrayFilters: [{ 'section.sectionId':
53             sectionId }]
54     }
55 }
56 }
57 }
```

The first if-statement checks if the course is found in the database. If it is not, it will be added with an array called completedSections with an array called completedExercises inside. The two arrays will have the section- and exerciseId inside them as seen on line 11. This can be seen from lines 4 to 16. In the else part on lines 18 to 32, it checks if the section is in the found course's array. If not, it will be added to the given course's array seen by the variable called completedCourseIndex, which is the index of the course found in the completedCourse field of the user from line 2. Again, an array for the section's exercises is added together with the sectionId. Lastly, on lines 44 to 54, if the course and section already are in the user's completedCourse array, then it simply adds the exerciseId to the section array. Right before it gets added, it is checked if the exercise is already completed (in the exercise array). If so, the error code E0801 will be thrown as seen on line 40.

This function is used in the function called markAsCompleted(). When the markExerciseAsCompleted() function has finished running, the course and section for the given exercise get checked if all their children's sections and exercises have been completed. If they are all completed, the course's and or section's field isComplete is set to true. The function markAsCompleted() is finally used in the

route on endpoint "api/users/:id/compeleted" where the id is the id of the user.

The route looks like this:

Listing 5.3: Route for api/users/:id/completed.

```
1 // Mark courses, sections, and exercises as completed for a
  user
2 router.patch('/:id/completed', requireLogin, async (req, res)
3   => {
4     try {
5       const { id } = req.params;
6       const { exerciseId } = req.body;
7
8       let user = await UserModel.findById(id);
9
10      if (!user) {
11        throw errorCodes['E0004'];
12      }
13
14      const updatedUser = await markAsCompleted(user,
15          exerciseId);
16
17      res.status(200).send(updatedUser);
18    } catch (error) {
19      if (error === errorCodes['E0004'] || error ===
20          errorCodes['E0008'] || error === errorCodes['E0012']) {
21        res.status(404);
22      } else {
23        res.status(400);
24      }
25
26      res.send({
27        error: error
28      });
29    }
30  }
31
32  res.send({
33    error: error
34  });
35}
```

```
26     }  
27 }) ;
```

In the code above, the ID for the user and exercise is retrieved from the request's parameters and body. Then it tries to find the user by ID in the database, and if it is not found, it will return the error code E0004. On line 13 it calls markAsCompleted() with the given user and exercise ID. If no errors are caught it will return the updatedUser returned by the markAsCompleted together with status code 200 [19]. In the catch block, it checks for error codes E0004, E0008 and E0012. If any of these error codes are present in the error object, the status code 404 will be returned meaning that either user, section or exercise was not found.

App Course: As a waste picker, I want to see me getting points after completing lectures/exercises on a course

This PBI is an important one for the sprint goal, considering improving the social-gamified learning experience on the app is a main focus point. Receiving points in exchange for completing exercises gives the user an incentive to work on their skills. It is also the first win-state the user is presented with. The acceptance criteria for this PBI are as follows:

- Visual sign that the user got points (e.g. when answering an exercise)
- Some form of animation
- Documented point system

Following an initial meeting with the product owner, it was determined that the right course of action is splitting it up into two parts; defining the logic for points, and creating an exercise screen animation.

As for now, the points for the exercises are defined as 2 multiplied by the difficulty of the section or lecture. The product owner informed the group about

a future meeting considering the points, so it was put on the sidelines for the duration of the sprint, working with this simplified logic.

Since there are no mock-ups for the animation, the group decided to create drafts in Figma and consult with the product owner. The presented drafts are mainly inspired by Duolingo. This was a fruitful activity, since other than having a bottom-screen popup accepted, the other full-screen mock-up was taken into consideration for a future PBI, a popup for a completed section. During this meeting, it was also determined that the product owner wanted to extend the popup animation with the point amount floating up the screen towards the progress meter. This is quickly implemented but is not considered release-ready, but rather a functional mock-up to be reviewed before the next sprint.

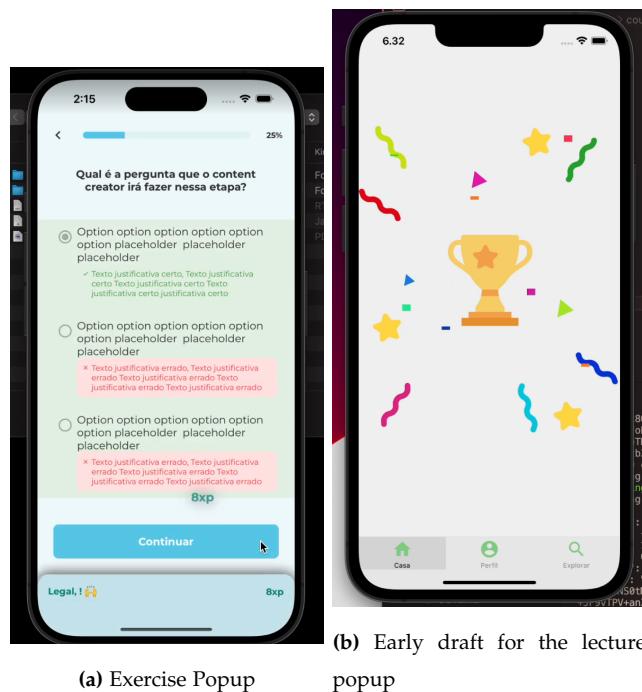


Figure 5.2: Exercise and Lecture popups

The bottom-screen popup either takes a random phrase of affirmation (in case of

a correct answer) or support (in case of an incorrect answer) from a pre-defined collection. To enhance personalisation and progress ownership, an important aspect of social gamification, the user's first name is added to some of the phrases, such as "Good Job, Carlos!". Also, emojis are used for a more fun experience. This popup is, as previously mentioned, loosely based on Duolingo's popups. The popup eases in from the bottom of the screen and eases out with a smooth animation.

The collection of phrases used for affirmation or encouragement is stored in a file named `popUpPhrases` located inside the `constants` folder. This is because the file encapsulates two functions that return an array of "hard-coded" strings, namely `generateSuccessPhrases()` and `generateEncouragementPhrases()`. These functions take a name in the form of a string as an input, which, as previously described, is inserted into a phrase for improved personalisation. These functions are used inside the `PopUp` component.

The `PopUp` component takes two arguments; a boolean `isCorrectAnswer`, which is used to determine what type of animation is to be shown, and an integer `xpAmount`, used to display the amount of points shown in the animation. The component consists of five functions: `fetchUserName`, `getRandomPhrase`, `startPopUpAnimation`, `startXpAnimation`, and `customEasing`. The code snippet below [5.4] fetches the user's first name, sets a random phrase using that name, and then triggers animations on the page. Since fetching the username is an asynchronous operation, it is crucial to use the "then" method to ensure that the code inside it is executed only when the promise returned by `fetchUserFirstName()` is resolved, i.e., when the first name is successfully retrieved. The `customEasing` function is used inside the `startXpAnimation` to achieve smoother movement.

Listing 5.4: The sequence of execution inside the `PopUp` component

```
1  useEffect(() => {
```

```
2     fetchUserFirstName().then((firstName) => {
3         setRandomPhrase(getRandomPhrase(firstName));
4     });
5 }, [];
6
7 startPopUpAnimation();
8 startXpAnimation();
```

Generating a random phrase uses the aforementioned collection of strings, taking firstName as an input. Depending on the correctness of the answer, the appropriate imported function is called, and a random index of the array of phrases is selected. For usability reasons, there is a character limit on the length of the phrases, but it is only a fail-safe in case the semantic check during registering or updating the username is not functioning correctly. Although such hard-coded variables do not adhere to the definition of done, it is still accepted, since it means minimal technical debt.

Listing 5.5: Generating random phrases for the popup

```
1 const getRandomPhrase = (firstName) => {
2     let randomIndex = 0;
3
4     const phrases = isCorrectAnswer
5         ? generateSuccessPhrases(firstName)
6         : generateEncouragementPhrases(firstName);
7
8     randomIndex = Math.floor(Math.random() * phrases.length);
9     let randomPhrase = phrases[randomIndex];
10    if (randomPhrase.length > 69) {
11        randomPhrase = randomPhrase.substring(0, 69) + "...";
12    }
13
14    return randomPhrase;
```

15 };

The animations themselves use the Animated library from React Native, and are fairly simple easing animations with customised timers, text content, and opacity curves.

App Course [Backend]: Implement points in Database

This PBI is extracted from the previous one (I want to see me getting points...), to distinguish between front and backend aspects of the same feature. The acceptance criteria for this PBI are as follows:

- Each user has a point field
- Each exercise gives x amount of points when answered correctly
- Each lecture gives x amount of points
- Each section gives x amount of points done

To implement points in the database, the user model in the backend must first be updated. This is done by adding these two fields:

Listing 5.6: Fields added to the user model to implement points.

```
1  points: {
2    type: Number,
3    default: 0
4  },
5  level: {
6    type: Number,
7    default: 1
8 }
```

Now the user has a field called 'points' and 'level'. The field 'level' is added after a quick discussion with the product owner, and has very simple logic to level up. The function for updating the user's points and level is shown below [5.7](#)

Listing 5.7: Function for updating the user's points and level.

```
1 function updateUserLevel(student) {
2     const pointsToNextLevel = student.level * 100;
3     if (student.points >= pointsToNextLevel) {
4         student.level++;
5     }
6
7     return student;
8 }
```

The function `updateUserLevel` is used to add the given points to the user and check if the user is supposed to level up.

5.1.2 Sprint Review

During this sprint, the team successfully finished and integrated the exercise system, and the popup animation was approved by the product owner but is still yet to be fully integrated, due to the user info not being correctly saved upon logging into the application. With that, the gamification of the user experience has arguably been improved, although the point system, as mentioned before, needs revision before a full implementation can be done. This issue is expected to be fixed fairly shortly, as soon as other teams have also merged their code into the development branch.

The CodeScene code analysis is now considered a part of the DoD. Therefore, improved code health is expected during the next sprints, since everything pushed should have a satisfactory score. As of now, the code health has slightly increased due to the new PBIs pushed, that are all unproblematic.

5.1.3 Sprint Retrospective

The team itself has been good at focusing on the overall sprint goal and providing value to the product owner despite unresolved dependencies. Overall, the expectations for the work to be done, considering the team itself, have been realistic, but considering future PBIs also require research and analysis of other competitors' solutions, the workload could require some adjustments.

Considering the bigger picture, to increase productivity, accepting some technical debt in exchange for improved delivery speeds seems to be the way forward. This is supported by the idea that frontend testing will be discarded in the future. Furthermore, GitHub actions can be extended to include CodeScene static code analysis to ensure low complexity on future pushes without manual checks.

Integration team developments

The integration team strongly suggests relying on more metric data to evaluate improvements and weak points in the team's workflow. This metric data is generated by the Liberators questionnaire that is used during sprint retrospectives. Additionally, keeping the sprint goal in mind is crucial, and delivering more finished PBIs in smaller and faster increments is preferred both considering the time it requires to conduct code review and to deliver more value in shorter periods.

5.2 Sprint 4

The overarching sprint goal is the same as the last sprint: "*Enhance the user experience on the web and app, including optimising course editing on the web and improving the social-gamified learning experience on the app. Prepare for stakeholder validation and ensure deployment readiness.*" Hence the unchanged sprint goal, the main focus is to finish enhancements to the already existing application and to continue the

research and implementation of the team's specific field, which is providing a social-gamified learning experience. In this sprint, the point system is further fleshed out, providing an improved user experience. Furthermore, a new win-state is created, in the form of a section completion animation.

5.2.1 Sprint Planning

Sprint backlog:

- App Course: As a waste picker, I want to be able to view the scored points on the whole course when I'm in a lecture or exercise.
- App Course: As a waste picker, I want exercises, lectures and sections to give points based on intricate logic.
- App Course: As a waste picker, I want to see me getting points after completing sections (section animation).

Minor fixes to previous PBIs

A large part of this sprint is spent resolving issues and dependencies to get the app into a deployment-ready state, so a user test can be conducted. This includes the backend and frontend of answering exercises and popup animation for the exercise screen. All of these PBIs are finalised, accepted, and deployed during the start of the sprint. In conclusion, all PBIs preceding this sprint are considered done.

App Course: As a waste picker, I want to be able to view the scored points on the whole course when I'm in a lecture or exercise.

The acceptance criteria for this PBI are as follows:

- The app should have a feature that displays the scored points on the whole course.
- The feature should be accessible to waste pickers.
- The feature should be available during lectures or exercises.
- The displayed points should be accurate and up-to-date.
- The feature should be user-friendly and easy to navigate.
- The feature should not interfere with the overall functionality of the app.

Since there is no Figma or any mock-up presented for this feature, the initial step is to come up with some design suggestions, from which the product owner can choose. The different mock-ups created using Figma can be seen in figure 5.3.

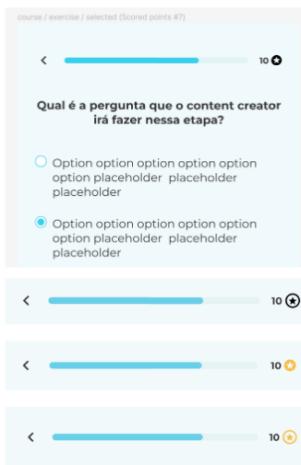


Figure 5.3: Designs for the PBI awaiting approval

Beyond the missing design, there is also a dependency on both the backend and the application itself. The pop-up that is displayed when the user answers a question correctly is being fixed during this sprint and is necessary to have finished before working on this PBI.

App Course: As a waste picker, I want exercises, lectures and sections to give points based on intricate logic.

The acceptance criteria for this PBI are as follows:

- The points system should be based on intricate logic.
- The app should accurately calculate and display the points earned by the user.

After a discussion with the product owner, this PBI is changed to only give points for exercises. Further details taken from the meeting with the product owner are that the exercises upon answering correctly on the user's first try should give 10 points. If the user answers incorrectly, but in another iteration of the section answers correctly, the exercise gives 5 points, no matter the number of tries it takes. The user model's field 'completedCourses' is updated to include both the date of completion for the specific exercise and the amount of points given to the user. Also, the fields 'isComplete' for 'completedCourses' and 'completedSections' are removed. The field looks like this now:

Listing 5.8: The field 'completedCourses' in the user model.

```
1 completedCourses: [
2   {
3     courseId: {
4       type: Schema.Types.ObjectId,
5       ref: 'Courses'
6     },
7     completedSections: [
8       {
9         sectionId: {
10           type: Schema.Types.ObjectId,
11           ref: 'Sections'
12         },
13       }
14     ]
15   }
16 ]
```

```
13     completedExercises: [
14     {
15       exerciseId: {
16         type: Schema.Types.ObjectId,
17         ref: 'Exercises'
18       },
19       isComplete: {
20         type: Boolean,
21         default: true
22       },
23       completionDate: {
24         type: Date,
25         default: Date.now
26       },
27       pointsGiven: Number
28     }
29   ]
30 ]
31 ]
32 ]
33 ]
```

The logic for the amount of points given is based on whether or not the exercise is present in the given user's model when answering the exercise. There are three different outcomes:

- If it is not, and the exercise is answered correctly, the user gets ten points.
- If the exercise is present and the exercise is answered correctly, the user gets 5 points.
- If the exercise is answered incorrectly, the user gets 0 points, but the exercise is added to the user's data, to show that the user has answered it at least once.

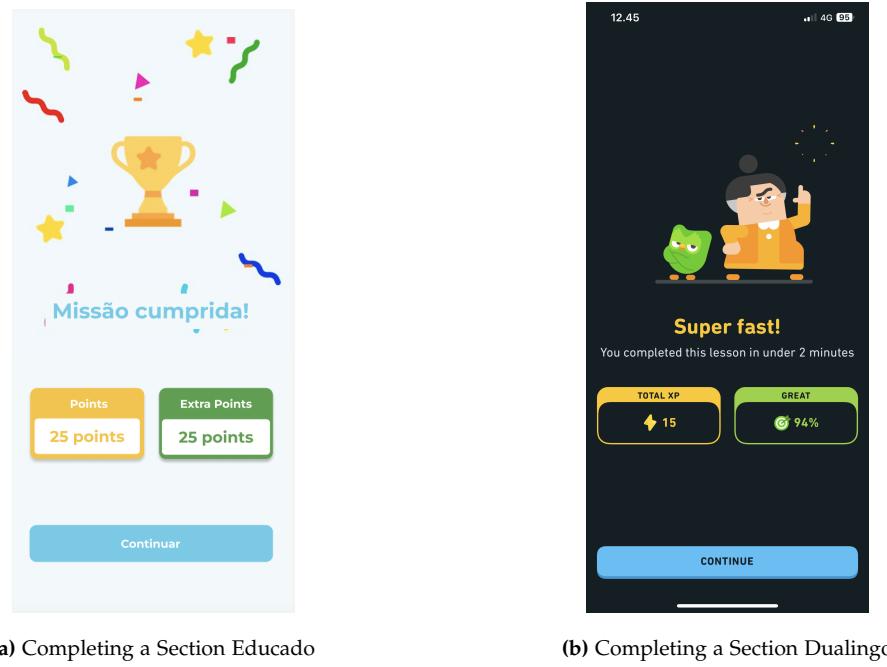
Further point logic is discussed with the product owner, considering activity streaks for extra points, but as of now, the currently implemented logic is considered sufficient for this PBI.

App Course: As a waste picker, I want to see me getting points after completing sections (section animation).

The acceptance criteria for this PBI are as follows:

- Visual sign that the user got points (e.g. when completing a section)
- Some form of animation
- Documented point system (difficulty times 2 for now, revised later)
- Points are saved in DB (code is there, but not connected to the frontend)

There is no mock-up provided for this feature. Therefore, the first step is to create one in Figma and have it accepted by the product owner. The accepted rendition can be seen in figure 5.4a. As previously noted, during sprint 3, the royalty-free floating trophy with confetti GIF is kept as a part of the design. The main focus, therefore, lies on displaying the gained points in a gamified, yet easily understandable and systematic manner. The main inspiration for this PBI is drawn from another educational mobile platform, Duolingo; see figure 5.4b.



(a) Completing a Section Educado

(b) Completing a Section Dualingo

Figure 5.4: Section completion screens

In conclusion, the design is successfully being implemented in the mobile app. The screen is not connected to the database. As a result, the points and extra points earned in the section for the specific user are hard coded. The displayed message is also hardcoded, but it should be a random message from a list of congratulatory messages.

The screen comprises a royalty-free trophy animation, a randomly selected congratulation message, and two boxes to display points earned in the section and extra points obtained from various activities such as daily streaks, etc.

5.2.2 Validation test

During the fourth sprint, the product owner set up a meeting with the end users, so that a validation test could be conducted. The six teams created some tasks that the users should attempt to do. After the users failed or succeeded in com-

pleting the individual tasks, they were asked questions regarding the design and functionality. The validation test provided valuable gatherings regarding the design. For example did the users not understand the logout button, which therefore was changed to text instead of an icon. Some of the other test issues which were called out were due to the application not working properly, for instance, to get to an exercise they had to click on a video. This was of course not the intended behaviour, so the developers were already aware of such issues. Following the validation test, it was the product owner's responsibility to formulate PBIs or refine already existing ones to better reflect user needs.

5.2.3 Sprint Review

A considerable part of this sprint has been spent on figuring out the fundamentals of social-gamification ideas of the app and preparing upcoming PBIs in collaboration with the product owner. Because of this, all of the new PBIs remain unfinished. Due to the same reason, the team also scored lower on self-management and team responsiveness in the questionnaire.

As of now, the section animation simply needs to be hooked up with other components. The point system is clearly defined and implemented, only needing to be merged into the staging branch. Viewing scored points during exercises and lectures is currently in the design phase. This means that 2/3 PBIs are expected to be finished very shortly. Despite the progress, the point of Agile is to continuously deploy valuable increments, which has now consistently not been the case partially due to a lengthy integration process, but also a lack of general understanding of Agile principles. This should be urgently addressed.

The presentation to the stakeholders is done using a single shared Google Slides file this time, ensuring a uniform presentation and smoother flow.

The mobile repository has a very solid average code health of 9.8, with the only problematic performer being StorageService.js, still acquiring a solid score of 7.9. This file is manually scanned and is determined to have appropriate complexity, with all functions being very easily readable and self-explanatory.

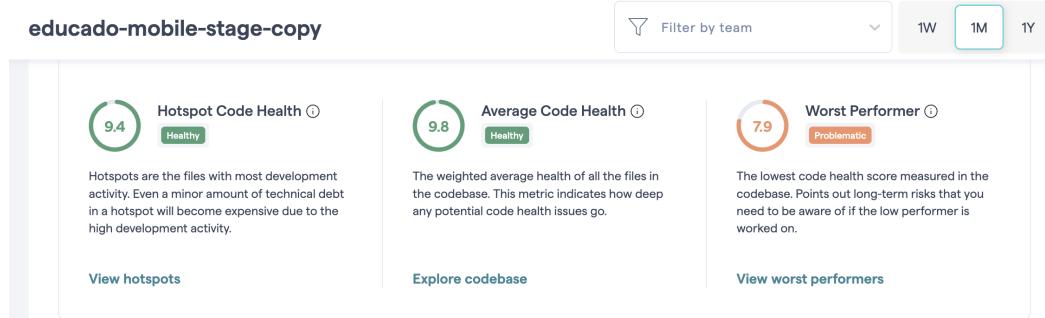


Figure 5.5: CodeScene for Mobile repository in sprint 4

The backend repository has an average code health of 9.6, with the weakest performer, courseRoutesTest acquiring a score of 8.6. After a brief look at this file, it is considered acceptable.

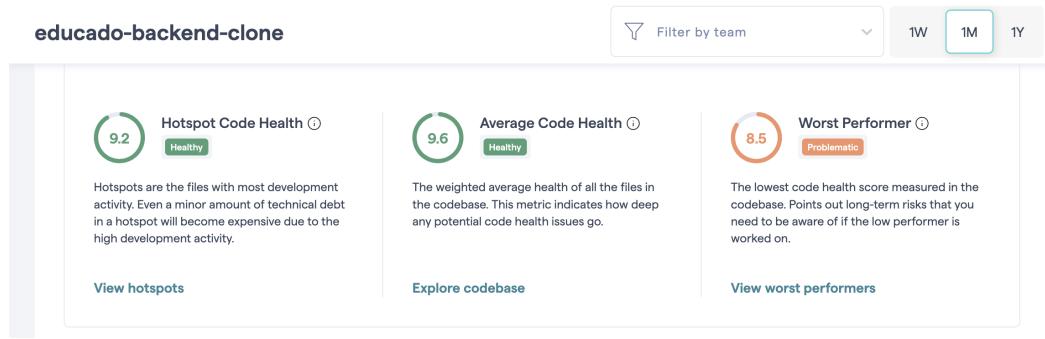


Figure 5.6: CodeScene for Backend repository in sprint 4

5.2.4 Sprint Retrospective

Based on the questionnaire results, it is determined that metrics should be more actively used to evaluate the team's work. Therefore, the results of the static

code analysis done using CodeScene will be explored more in-depth at the end of each sprint from now on. The health and quality of the mobile staging branch and backend staging branch are discussed individually, and the progressive changes/lack thereof are reflected.

Integration team developments

The integration team recognised the success of using GitHub issues and encouraged the continuation to achieve better visibility and tackle problems as soon as they are spotted. A new problem that halted the progress of some PBIs was the lack of communication regarding larger structural changes in the database. Specifically, the courses' data models.

To further enhance automation, automatic deployment is the next step, and integrating CodeScene into the GitHub actions is aborted. Instead, the results of the static code analysis are discussed in more depth. Although it would be better to automate as many manual processes as possible to achieve better velocity, the difficulty of implementation does not seem reasonable at the moment. The main goal is to maximise value, and implementing CodeScene into the production pipeline does not seem like a reasonable tradeoff considering the time lost on other PBIs. It must be noted, that the team has used CodeScene from the start of the project to check for any issues before opening pull requests.

5.3 Sprint 5

The sprint goal of this sprint is the following: "*Deliver a stable production release for both the app and web. Implement a comprehensive points and levels system in the app. Releasable of the highest priority PBIs of offline access, video streaming, certificates, and institutional onboarding features. Address design issues on the web platform.*" The

focal point of this sprint goal for the team is the comprehensive points and levels system. This means that the prioritisation of the numerous new PBIs defined in collaboration with the product owner must follow the notion of supporting this goal.

Other than developing new features, a portion of this sprint is also spent further researching gamification, so that the features and course of action are logical and based on confirmed empirical evidence. Other than enhancing and stabilising the points system and section completion, a new win-state is added, the course completion animation. This screen is also planned to incorporate an excerpt from a leaderboard, statistics on answer accuracy, and an indication of receiving a certification. In this way, several other win-states are also presented to the user, visualising additional core drives such as social influence and relatedness, epic meaning, and empowerment through feedback.

5.3.1 Sprint planning

Sprint backlog:

- App Course: As a waste picker, I want to be able to view the scored points on the whole course when I'm in a lecture or exercise (cont. from sprint 4)
- App Course: As a waste picker, I want to see an animation with congratulations messages after completing a section (cont. from sprint 4)
- App Course: As a waste picker, I want to see me getting points after completing sections (cont. from sprint 4)
- App Course: Completing course animation

Minor fixes to previous PBIs

Based on user feedback from the validation test, the following PBI had to be finished up during the start of the sprint: App Course: As a waste picker, I want exercises, lectures and sections to give points based on intricate logic.

All instances of "xp" are changed to "pts", and the getting points animation during exercises is adjusted to ease up to the point counter. This is presented to the product owner, who approves the changes, but an additional user validation test is most likely needed to confirm the success of the fixes.

App Course: As a waste picker, I want to be able to view the scored points on the whole course when I'm in a lecture or exercise (cont. from sprint 4)

The acceptance criteria are unchanged from the previous sprint 5.2.

There are a few dependencies that have to be resolved before this PBI can be completed. One of the dependencies is a remake of the progress bar seen at the top of the screen, which is a PBI for another group. Another dependency is fixing the flow of the lectures and exercises in a section, which is currently a PBI that was assigned to another group by the product owner. These dependencies have arisen since the point tracker should be placed next to the progress bar, and the functionality of the counting feature should work with the flow fix. The logic of the feature does however work on its own and only needs to be integrated with the other PBIs once they are ready and have been pushed to the development branch. In the meantime, the usage of data in the exercise screen is going to be reconsidered. Currently, the data used is fetched from the database and causes unnecessary loading. This data should instead be passed from the previous screen which displays the lectures a user can pick from. This will increase the

performance and responsiveness of the screen and will ensure the point tracker is up to date.

App Course: As a waste picker, I want to see an animation with congratulations messages after completing a section (cont. from sprint 4), App Course: As a waste picker, I want to see me getting points after completing sections (cont. from sprint 4)

These two PBIs are described in the same subsection, as their development was largely intertwined and happened linearly with no notable challenges in implementation. The acceptance criteria are updated from the last sprint, and they are as follows:

- Visual sign that the user got points (e.g. when completing a section)
- Some form of animation
- The animation should be triggered automatically after the user completes a section.
- The animation should display a congratulatory message to the user for completing the section.

Navigation to and from the screen has been added, so the animation is triggered after a user completes a section. When navigating away from the screen, the user goes back to the section overview and is ready to start the next section. The points in the point box are fetched from the database, making it a dynamic value. The points have an animation that counts from zero to the points earned in the section. Lastly, congratulatory messages are shown randomly every time a section is completed. The PBI is fully implemented in this sprint and fulfils the DoD. Therefore, the PR for the PBI is accepted to be merged into the staging branch.

App Course: Completing course animation

The gamification of answering exercises starts with implementing popups for single answers, continues with a congratulatory screen for completing sections, and concludes with a summary at the end of a course. Unlike the first two, the course completion should not only emphasise the "win-state" of getting points, but also include other important driving factors, such as a leaderboard, statistics, and, most importantly, the certification that the user receives by completing the course. Following research on the theory behind social gamification, and several meetings with the product owner, the acceptance criteria were formed, providing a clearer picture of the expectations for this PBI. The acceptance criteria for this PBI are as follows:

- Design approval (dependent on designer feedback)
- Screen provides statistics for the user's performance on the entire course
- Clearly indicates user certification
- User redirected to course overview after certification
- Shows the user's current position on the leaderboard
- Screen provides the user's stats over correct first tries
- Screen is shown after completing the last section

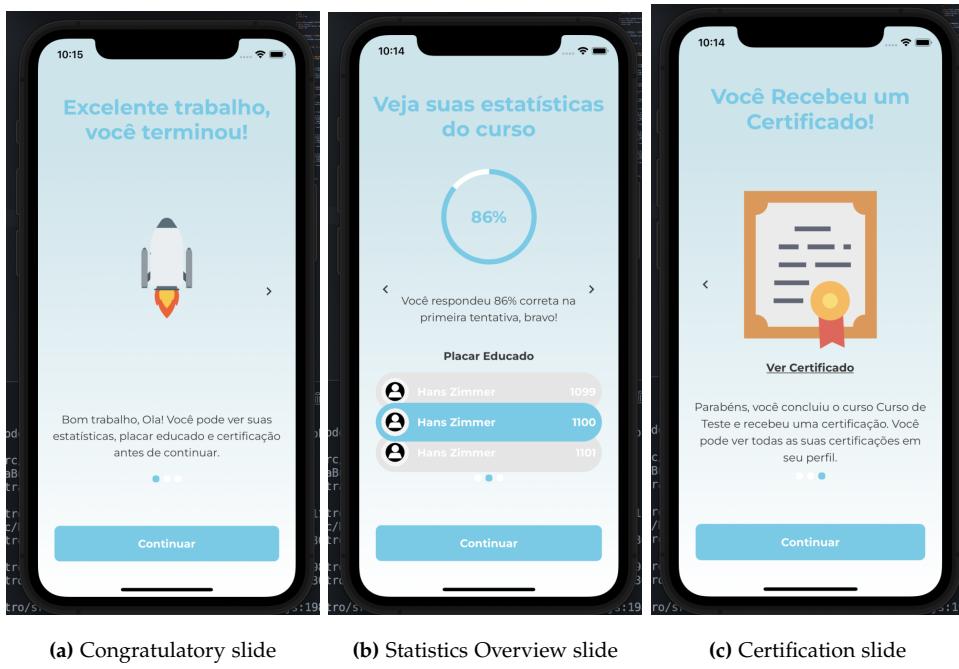
The screen, which follows the completion of any course, consists of 3 slides; a congratulatory slide, a slide containing a snippet of the leaderboard and statistics (a circular bar displaying the percentage of correct answers as of now), and a slide for the certification.

Firstly, each slide is designed in Figma and is presented to the product owner. As of now, they are accepted, but further design changes may be requested by

the assistant designer of the product owner.

Some important considerations before implementation were based on previous PBIs and technologies used in them. The slider library previously used in the welcome screen was lifted over, and high-quality royalty-free gifs are also inserted and styled using LottieFiles, in the same fashion as they are used in the section completion screen. Inserting the user's first name and a random choice from a collection of phrases are also implemented following code snippets from the exercise popups.

Getting the simplest elements out of the way, the overall layout is implemented, and using previously implemented methods, the congratulatory slide and certification slide are created. The new features to be tackled are creating the leaderboard and circular progression bar. As of now, the leaderboard PBI is not in progress yet, so placeholder text is used, which can later be swiftly replaced by real data. All that is missing is backend code for fetching the relevant information, and entering these into the correct text fields. The circular progression bar is first attempted without libraries to avoid further dependencies but is later changed for a library, whose codependencies are already in use, providing much cleaner and less code than building it from the ground up. The final version in this sprint can be seen in the screenshots below [5.7](#).



(a) Congratulatory slide

(b) Statistics Overview slide

(c) Certification slide

Figure 5.7: Slides of the Course Completion screen

The following paragraph provides a more technical, in-depth walkthrough of the implementation. Firstly, inside the `CompleteCourse` screen the course information itself is fetched, and the frame for the slides, as well as the continue button, are included. Other than resulting in more readable code, using the slider as its own component also helps with reducing complexity and increasing modularity.

In the `CompleteCourseSlider`, the slide components are imported (`Congratulation`, `StatsOverview`, and `Certification`), and are mapped to the indices of the slider. The slider changes slide every 10 seconds, or the user can manually go to the next slide by swiping or tapping the arrows next to the shown content. The slider is implemented as such:

Listing 5.9: Use of the components in the Slider of the Course Completion Screen

```

1 const screens = [
2   <Congratulation />,

```

```

3     <StatsOverview ref={statsOverviewRef}
4         courseObject={courseObject} />,
5     <Certification courseObject={courseObject} />,
6   ];
7   ...
8
9   {screens.map((screen, index) => (
10     <View key={index}>
11       {screen}
12     </View>
13   ))}
```

After importing the components, they are initialised as a part of an array called "screens". Subsequently, they are mapped to the indices of the slider, by setting the view key. When required, additional information is passed, namely the course object, and stats overview reference. The course object contains relevant data for the screen, since it has to display personalised information for each course, and the stats overview reference is used to initiate an animation. Without this, the animation would play before the correct slide is accessed.

The Congratulation and Certification components use the aforementioned LottieFiles library for inserting gifs into React Native code, and the following snippet ensures the inclusion of the user's first name, in order to increase the personalisation aspect of the product. A relevant snippet worth including for these screens is the implementation of the user's first name being included in the personalised messages:

Listing 5.10: Fetching username to be displayed on the screen

```

1 const [name, setName] = useState('');
2
3 const getName = async () => {
```

```
4     const userInfo = await getUserInfo();
5     setName(userInfo.firstName);
6   };
7
8   useEffect(() => {
9     getName();
10 }, []);
```

This code is a React functional component that uses the useState and useEffect hooks to manage the state of a variable called name. To start out, the useState hook is used to declare a state variable name with an initial state of an empty string (""). The setName function is a setter function that allows you to update the value of the name state.

The getName is a function declared using the arrow function syntax. It is marked as asynchronous (async) because it contains an await expression. Inside getName, there is an await getUserInfo(); statement. The getUserInfo function originates from the StorageService file, and as the syntax suggests, this function returns a promise. The await keyword is used to wait for the promise to be resolved, and then the result is stored in the userInfo variable.

The useEffect hook is used to perform side effects in functional components. In this case, it runs the getName function when the component mounts (specified by the empty dependency array []). This ensures that the getName function is called once when the component is initially rendered. In the view itself, the name is then inserted into the sentence in the following fashion:

Listing 5.11: Displaying the fetched username on the screen

```
1 <View>
2   <Text className="text-center text-base text-projectBlack
px-5 mt-12">
```

```
3     Bom trabalho, \texttt{\{name\}}! Voce pode ver suas
4         estatisticas, placar educado e certificacao antes
5         de continuar.

6     </Text>
7
8     </View>
```

Such as in the exercise popup, this is expected to be slightly changed, so the name will be passed to the sentence generator. Nevertheless, the logic stays the same.

The leaderboard part of the StatsOverview screen is hard-coded as of now, but is expected to represent real data when the backend of the leaderboard system, and user data models are updated. The circular progression bar uses a component from the react native circular progress library with some changes to its styling and animation. The following logic is implemented to find the percentage to be displayed:

Listing 5.12: Logic behind the displayed percentage on the circular progression bar

```
1 async function getPercentage() {
2     try {
3         const completedCourse = await getCompletedCourse();
4         let totalExercises = 0;
5         let totalExercisesWithFirstTry = 2;
6
7         if (completedCourse) {
8             completedCourse.completedSections.forEach((completedSection)
9                 => {
10                 completedSection.completedExercises.forEach((completedExercise)
11                     => {
12                         totalExercises++;
13
14                         if (completedExercise.firstTry === true) {
```

```
13         totalExercisesWithFirstTry++;
14     }
15   });
16 });
17 } else {
18   return 0;
19 }
20
21 return Math.round((totalExercisesWithFirstTry /
22   totalExercises) * 100);
23 } catch (error) {
24   console.error('Error fetching completed courses:',
25   error);
26   return 0;
27 }
```

This code snippet is of an async function `getPercentage()`: This defines an asynchronous function named `getPercentage`. The "await `getCompletedCourse()`" line calls the asynchronous function `getCompletedCourse` and waits for it to complete. The result is stored in the `completedCourse` variable. The function then initialises two variables, `totalExercises` and `totalExercisesWithFirstTry`, to count the total number of exercises and the number of exercises completed on the first try. The if-statement in line 7 then checks if `completedCourse` returns true (not false, null or undefined). If it is true, it iterates through the completed sections and exercises, updating the counters. If `completedCourse` returns false, null or undefined, the function returns 0, indicating 0% completion. The function calculates the percentage of exercises completed on the first try using the formula `(totalExercisesWithFirstTry / totalExercises) * 100`. Finally, the result is rounded using `Math.round`. If any errors occur during the execution of the function (e.g., an error in the asynchronous operation), it is caught in the catch block. An error

message is logged to the console, and the function returns 0.

App Course [Backend]: As a waste picker, I want to see a leaderboard ranking users by total points on a monthly basis

The acceptance criteria for this PBI are as follows:

- The leaderboard should display the total points of each user.
- The leaderboard should rank users based on their total points.
- The leaderboard should be reset on a monthly basis.
- The leaderboard should be accessible to all waste pickers using the app.

This PBI is made more general in the backend in the sense that a route for returning a leaderboard for the top 100 users is made. The route takes a time interval as input which can be either 'day', 'week', 'month' or 'all'. This way, a leaderboard for the day, the week and overall can be returned from the same endpoint by simply changing the input parameter. The functionality for the ranking of users can be seen below:

Listing 5.13: Functionality for sorting by amount of points the user have in the database.

```
1 const leaderboard = await StudentModel.aggregate([
2   {
3     $match: {
4       'completedCourses.completedSections.completedExercises
5         .completionDate': dateFilter
6     }
7   },
8   {
9     $unwind: '$completedCourses'
10  },
11  {
```

```
12         $unwind: '$completedCourses.completedSections'
13     },
14     {
15         $unwind: '$completedCourses.completedSections.
16             completedExercises'
17     },
18     {
19         $lookup: {
20             from: 'users',
21             localField: 'baseUser',
22             foreignField: '_id',
23             as: 'user'
24         }
25     },
26     {
27         $unwind: '$user'
28     },
29     {
30         $project: {
31             firstName: '$user.firstName',
32             lastName: '$user.lastName',
33             points: 1,
34             level: 1,
35             completedExercisesPoints:
36                 '$completedCourses.completedSections.completedExercises.
37                     pointsGiven'
38     },
39     {
40         $group: {
41             _id: '$_id',
42             firstName: { $first: '$firstName' },
43             lastName: { $first: '$lastName' },
```

```
44         points: { $first: '$points' },
45         level: { $first: '$level' },
46         completedExercisesPoints: {$sum:
47             '$completedExercisesPoints' }
48     },
49     {
50         $sort: {
51             completedExercisesPoints: -1
52         }
53     },
54     {
55         $limit: 100
56     }
57 ]);
```

Mongoose is used to filter the top 100 students which can be seen by the aggregate() function. The first thing that happens is the 'match' keyword, which filters the user's completedExercises by the dateFilter variable. The dateFilter variable is equal to the input parameter 'time interval' mentioned earlier. So if the time interval is set to month, only the completedExercises with a completion date in the current month are taken into consideration. After the filtering, the completedCourses, completedSections and completedExercises arrays are deconstructed with the 'unwind' keyword, so that each element in the array is outputted as its own document.

After that, the student model's base user is found in the database using the keyword 'lookup' so that the leaderboard also returns the user's first- and last name. Then the found user is deconstructed with the 'unwind' keyword to match the other structures.

Now all relevant data is found for each user, and the keyword 'project' is called to specify which fields should be passed along the pipeline. The first- and last-name fields in 'project' are specified to be the user model's fields and not the student model's. The number 1 specified for points and level implies that the fields points and level should be included in the output of the aggregate() function. The field completedExercisesPoints is set to be equal to points given by each completed exercise and is used to find the overall sum later.

The keyword 'group' is then called and it simply serves as a restructuring of the student model, so that the output of the aggregate() function is always the same. Here the completedExercisesPoints are set to be equal to the user's completedExercises in the given time interval.

After all the users have been restructured, the keyword 'sort' is called, which simply sorts the users by completedExercisesPoints. Lastly, the keyword 'limit' is called to limit the amount of outputted users to 100. This is how the users are ranked by total points, and the time interval can be set to 'month', which satisfies the two acceptance criteria.

5.3.2 Sprint Review

Considering the team's own efforts, it can be assessed that the team could have been more focused on contributing more to a stable release, but design issues were addressed, and the points system was refined. Overall, since there is only one sprint left, the main focus is to get everything in place, and larger PBIs are expected to be left on the sidelines.

Considering the larger picture, the notion of increment is not entirely there; some teams still push lots of PBIs at the end of a sprint rather than working with small

PBIs and pushing them individually. Although this has not been an issue for this specific team, the sprint goal should have been kept in mind during development.

The code quality is overall unchanged. The mobile repository has an average code health of 9.7, with the only problematic performer being StorageService.js, still acquiring a solid score of 7.7. Although the overall score is slightly lower, it is still completely acceptable, and the new components have a very solid score.

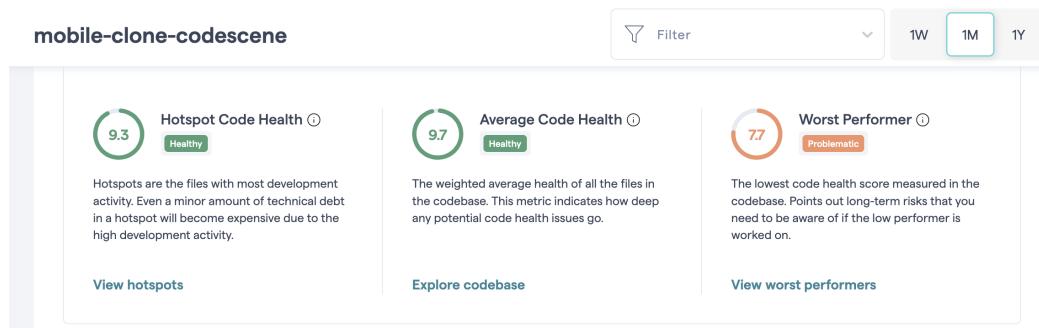


Figure 5.8: CodeScene for Mobile repository in sprint 5

The backend repository has an average code health of 9.6, with the weakest performer, courseRoutesTest acquiring a score of 8.5. The health is virtually unchanged and entirely acceptable.

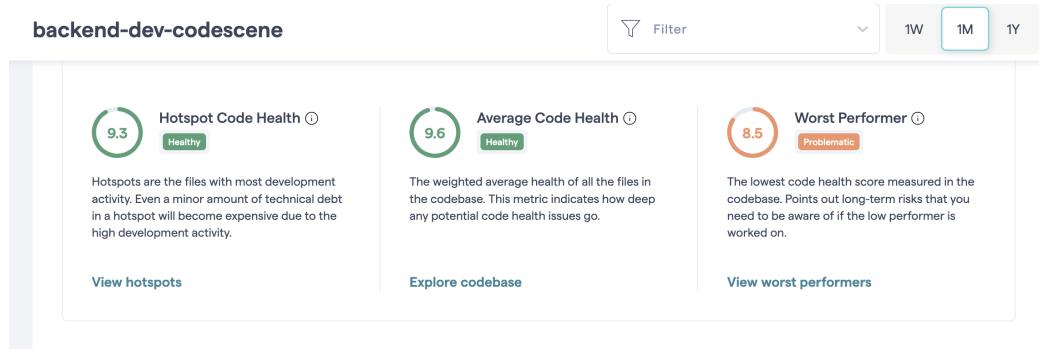


Figure 5.9: CodeScene for Backend repository in sprint 5

Even though this looks good on paper, sacrificing some code quality and gaining

technical debt is almost unavoidable for leveraging more value to the customer. Finding a fine balance is an improvement over having less code with higher quality. Also considering this point, the tradeoff between less testing and more deployment also seems like a good idea. For several sprints, this has been an issue, and should therefore definitely be taken into consideration for the last sprint.

5.3.3 Sprint Retrospective

Nearing the end of the project, the only aspects of the agile process that slightly decreased were responsiveness and team motivation. Morale could have been a major factor in the lacklustre performance considering the value delivered. The loss in morale could be possibly further induced by the slow integration process due to a lack of planning on the integration team's part. Thus, getting PBIs into staging was a painstakingly long and frustrating task with new undocumented dependencies being introduced to the codebase numerous times, resulting in further resources used on reworks of completed PBIs.

Integration team developments

The integration team emphasised the same points. As a last important upgrade to the pipeline, an automatic end-to-end test using Detox will be attempted to be implemented.

Furthermore, if time allows for it, there will be a test implemented that checks if the app can be built. This is because there have been issues with staging not being able to be built even though no tests failed or warnings were issued.

It is, however, worth highlighting again, that having all the crucial features in place should have reserved priority over new quality-of-life updates.

5.4 Sprint 6

The sprint goal for the sixth and final sprint is the following: "*Complete the development and ensure the integrated launch of the Educado platform to make it accessible to all users. In the app, ensure a gamified experience and offline access. On the web, implement institutional onboarding, enhance usability, and enable efficient certification issuance for activities on Educado.*" The most important aspect of this sprint is to ensure a stable release. Furthermore, the team themselves can ensure a gamified learning experience by pushing all of the most valuable PBIs that have been worked on.

The main focus of this sprint is to ensure the delivery of all of the most important PBIs from the previous sprint. These are, namely, fixing the flow of components, showing points during lectures and exercises, and section & course animation PBIs. All of these PBIs have received a lot of attention already, so they must be in staging during the first week of the sprint. If possible, some of the new PBIs may also receive attention during this first week of the sprint; namely extra points logic and showing level on profile. Work on the leaderboard feature is halted because it is not realistic to complete in the given time frame. To summarise, the first week of the sprint should focus on development and deployment. It is worth noting that the profile page is also receiving a large rework, making it a more personalised screen with space for highlighting "win-states" rather than a simple settings-page-like section where the user would otherwise not look.

The second week of the sprint should focus on fixing issues, tests, reducing technical debt, cleanup, writing readme files, and documentation. In short, the second week should ensure that everything is in its right place, documented and running in a stable manner.

5.4.1 Sprint planning

The sprint backlog items of this sprint are separated into previous and new items. The previous items receive priority, and as the title describes, they are already in progress.

Sprint backlog (previous items):

- App Components: Fix flow for components
- App Course: As a waste picker, I want to be able to view the scored points on the whole course when I'm in a lecture or exercise
- App Course: As a waste picker, I want to see me getting points after completing sections (section animation)
- App Course: Completing course animation

Sprint backlog (new items):

- App Exercise: Extra points logic
- App Profile: As a waste picker, I want to view my current level on my profile screen.
- App Course: As a waste picker, I want to see a leaderboard ranking users by total points on a monthly basis

App Components: Fix flow for components

This PBI was originally from sprint 5 and is continued in sprint 6 for two reasons; firstly, the lectures and exercises are not shown in the same order as the content creator made them, because all lectures are always shown before all the exercises. Secondly, the student can swipe on an exercise even though the student has not

submitted their answer yet. Since the task has not been completed correctly, the acceptance criteria have been updated to be as follows:

- Users should be able to easily navigate between different components.
- All exercises should be accessible and functional.
- The student should not be able to swipe on an exercise.
- The components should be in order as the content creator intended it.

The solution to this task is to refactor the database such the section model has a field "components" instead of two fields "exercises" and "lectures". This way the backend can extract the components in the right order and send it to the app. The components are also refactored in the student model such that when a student subscribes to a course, it gets the unfinished course saved in the database. This way when the student swipes on a lecture or answers an exercise, the component gets marked as completed and the progress on the course can be measured correctly. This means that when a student leaves in the middle of a section, they enter the last uncompleted component instead of the start of the section. If the database model had been more thoroughly discussed, mapped out, and kept up-to-date, this major issue would have not occurred.

App Course: As a waste picker, I want to be able to view the scored points on the whole course when I'm in a lecture or exercise

The acceptance criteria for this PBI are as follows:

- The app should have a feature that displays the scored points on the whole course.
- The feature should be accessible to waste pickers.
- The feature should be available during lectures or exercises.

- The displayed points should be accurate and up-to-date.
- The feature should be user-friendly and easy to navigate.
- The feature should not interfere with the overall functionality of the app.

Showing points during lectures and exercises PBI is cleaned up, reviewed and pushed shortly after the start of the sprint.

App Course: As a waste picker, I want to see me getting points after completing sections (section animation)

The acceptance criteria for this PBI are as follows:

- Visual sign that the user got points (e.g. when completing a section)
- Some form of animation
- Documented point system (difficulty times 2 for now, revised later)
- Points are saved in DB (code is there, but not connected to frontend)

The feature is reviewed shortly after the start of the sprint and is merged into the staging branch as it now fulfills the DoD. Following the dismissal of a full implementation of extra points logic, the extra points counter is removed from the screen. This simply means that a couple of lines are commented out but kept for future developers. This minor patch is pushed into staging following a brief review.



Figure 5.10: Updated section animation

App Course: Completing course animation

The acceptance criteria for this PBI are as follows:

- Design approved
- Screen provides a course summary
- Clearly indicates user certification
- User redirected to course overview after certification
- Screen provides the user's stats over correct first tries
- Screen is shown after completing the last section

The PBI was missing proper navigation and logic for when to navigate to the screen. Now, the screen is navigated to when all sections in the current course are completed. When the user has gone through all the slides on the course

completion screen, they are navigated back to the overview of courses to start a new course.

Listing 5.14: Handling all sections being completed on the section complete screen

```
1 async function handleAllSectionsCompleted() {
2     const studentInfo = await getStudentInfo();
3
4     if (isCourseCompleted(studentInfo,
5         parsedCourse.courseId)) {
6         navigation.reset({
7             index: 0,
8             routes: [
9                 {
10                    name:
11                      'CompleteCourse' },
12                ],
13            });
14        } else {
15            navigation.reset({
16                index: 1,
17                routes: [
18                    {
19                        name:
20                          'Section' ,
21                        params: {
22                            course:
23                                parsedCourse
24                            },
25                        },
26                    ],
27                });
28            }
29        }
30    }
```

In the code above, which is a function taken from the section completion screen, it is seen that navigation is reset to the complete course screen when the utility function `isCourseCompleted` returns true for the specific course that the user has just finished.

Due to the leaderboard PBI being put back in the product backlog, the leaderboard element is removed from the statistics screen, but the code is left there and commented out for future developers.



Figure 5.11: Updated course completion animation

App Exercise: Extra points logic

The extra points are rewards to the user when fulfilling different achievements such as answering all exercises on the first attempt or a daily streak of completing exercises or lectures. The extra points are only awarded when completing a section. The acceptance criteria for this PBI are as follows:

- The extra points should be stored on the finished section.

- When the user logs in and completes an exercise or lecture in multiple days, extra points should be awarded.
- The extra points awarded should be added to the user's total points.
- The current extra points should be stored on the user.

This PBI is not completed but the route in the backend is implemented. The route looks like this:

Listing 5.15: Route for giving extra points.

```
1 router.put('/:id/extraPoints', requireLogin, async (req, res)
2   => {
3     try {
4       const { id } = req.params;
5       const { extraPoints } = req.body;
6
7       if (!mongoose.Types.ObjectId.isValid(id)) {
8         return res.status(400).json({ error:
9           errorCodes['E0014'] });
10      }
11
12      if (isNaN(extraPoints)) {
13        return res.status(400).json({ error:
14          errorCodes['E0804'] });
15
16      const student = await
17        StudentModel.findOneAndUpdate(
18          { baseUser: id },
19          {
20            $inc: {
21              currentExtraPoints:
22                extraPoints
23            }
24          }
25        );
26
27      res.json({ student });
28    }
29  });
30
```

```
19         }
20     }
21 );
22
23     if (!student) {
24         return res.status(404).json({ error:
25             errorCodes['E0004'] });
26
27         res.status(200).json(student);
28     } catch (error) {
29         res.status(500).json({ error:
30             errorCodes['E0003'] });
31 }) ;
```

The route takes the ID of the student as a parameter, and in the body of the request, it takes a number value extraPoints. After extracting the values from the request, it checks if the ID is of type ObjectId from Mongoose's library. The next if statement ensures that extraPoints is a number and if not, it returns with status code 400 and error code E0804. Then it increments the student's field currentExtraPoints with the value extraPoints extracted from the request. Right before returning, it checks if the student is found in the database, if not it returns with status code 404, and error code E0004. Finally, it returns with a status code 200 and the student as the response. The entire functionality is encapsulated in a try-catch block, catching any unknown errors, and returns status code 500 (Internal Server Error) with error code E0003 [19].

App Profile: As a waste picker, I want to view my current level on my profile screen.

The acceptance criteria for this PBI are as follows:

- The profile screen should display the waste picker's current level
- The level system should update dynamically as the user receives points
- There should be a progress bar displaying the current progress to the next level
- The level system should have documented logic

In previous sprints, the custom progress bar always displayed the progress in a text field next to the progress bar, followed by a percentage symbol. This was not part of the Figma design, so it had to be changed. The custom progress bar has been modified to include an additional parameter called `displayLabel`, which is set to true by default. If it is set to false, the text next to the progress bar is not rendered.



Figure 5.12: First iteration of the profile screen with points displayed

The data used to display the user's current level and points are fetched from local

storage, and saved to the device when the user logs into the app. The profile page screen had to be adjusted from its current version to match the Figma design, and this modification has been completed. However, during the PBI progress review, the product owner expressed a desire to scale it back a bit, opting for a mix of the previous version and the one provided in the Figma design.

Lastly, some logic has been implemented to calculate the total points the user has gained throughout all their levels.



Figure 5.13: Updated profile screen with points and progress

5.4.2 Sprint Review

Considering the team of this paper, the sprint was considered a great success in light of the overarching sprint goal. All of the finished PBIs greatly contributed to improving the social-gamified learning experience on the app. Furthermore, all of the already existing gamification features were optimised and enhanced during the last week of the sprint, where the group focused on resolving issues.

Preparation for stakeholder validation and ensuring deployment readiness was also successful. The Nexus team presented a fully deployed and functional application at the end of the sprint delivering a lot of value for the customers in an incrementally integrated manner, thus better following agile principles. A full CI/CD pipeline was also finalised, greatly increasing the level of automation in the pipeline, and making future integration and deployment work much more streamlined. Using the new pipeline, any merge into the main branch automatically triggers a release to the Google Play store.

As expected, due to the less strict review and testing processes, the overall code health has slightly decreased. For the mobile repository, StorageService is problematic, receiving a decreased rating of 5,2 and the overall code health is evaluated to a slightly decreased 9,6, which is still good. Although this means that the project has received some code debt, the increased velocity is arguably worth the future cleanup work.

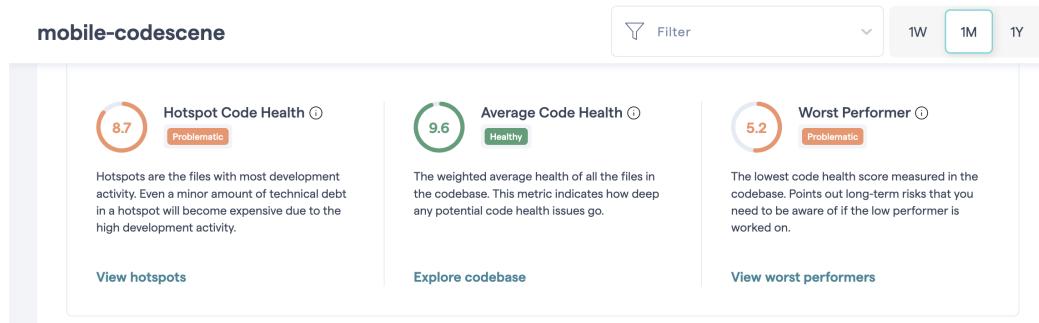


Figure 5.14: CodeScene for Mobile repository in sprint 6

In the backend repository, ProfileRoutes is most problematic, receiving a rating of 7,6 and overall code health is decreased to 9,4.

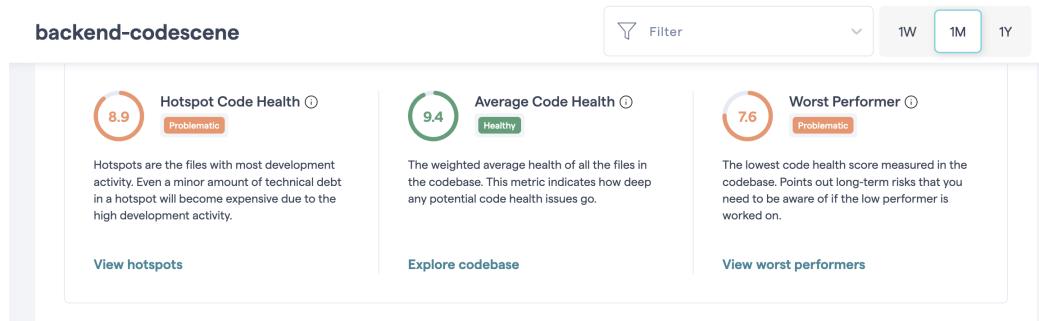


Figure 5.15: CodeScene for Backend repository in sprint 6

Despite the decreased code health, all repositories are left with improved documentation, guides, READMEs, and code that follows all standards. As formerly mentioned, the last week of the sprint mainly consisted of a coordinated cleanup of the project. This included removing all dead code, irrelevant comments and console logs, deleting all feature branches, deleting all unused files, removing all garbage from the database, updating database models, and documenting all feature progress. This is expected to improve the initial sprint of future developers, considering that one of the great challenges of this project was the team familiarising themselves with an insufficiently documented project with an outdated database model and no readmes.

Following the review, a couple of hotfixes were still pushed into production to be accepted into the Google Play store. In a real project, the end of the final sprint would mean a code freeze, but in this case, the minor fixes were accepted due to a couple of weeks still being left before the deadline.

5.4.3 Sprint Retrospective

In light of increased velocity, which was very much needed considering the last couple of sprints, prioritising features ahead of perfect code was considered a successful change to the workflow. Cross-team communication, on the other

hand, should have received more attention in the context of the more hectic last-minute merges. Furthermore, web and mobile teams should have conducted more frequent and thorough communication providing more transparency, as the PBIs of many teams presented more dependencies than previously observed.

Chapter 6

System Architecture

After restructuring the old codebase, the Educado project uses a microservice structure. This structure is not used by the group of this paper, and therefore it will not be discussed further. The theoretical system architecture used by this group is a 3-tier layered structure. Which means it has a presentation layer, an application layer, and a data layer. The presentation layer is the mobile repository, the application layer is the backend repository, and the data layer is MongoDB containing all the data for the project [17].

6.1 The Educado Platform Architecture

Collaborative writing begins

The Educado project attempts to follow the microservice architecture as seen in figure 6.1. This means that the project consists of many small and independent services, which is an ideal way of developing complex projects.

'Educado-mobile' contains the front end for the mobile application, to be used by the waste pickers, while 'Educado-frontend' contains the frontend for the web application to be used by content creators. These two microservices share a back-

end, which is the 'Educado-backend' microservice. This microservice contains the Student APIs, Content Creator APIs, and more general APIs. The Student and Content Creator APIs are specific to their platform (either mobile or web), while the general APIs are services that both the app and web use, such as login. Each platform has a way of storing information. The back end has access to the cloud database, while the web browser for the front end, and the smartphone for the mobile each have their local storage. This local storage allows the application to save data on the respective device for either authentication purposes, offline access, or fast data retrieval without communicating with the database.

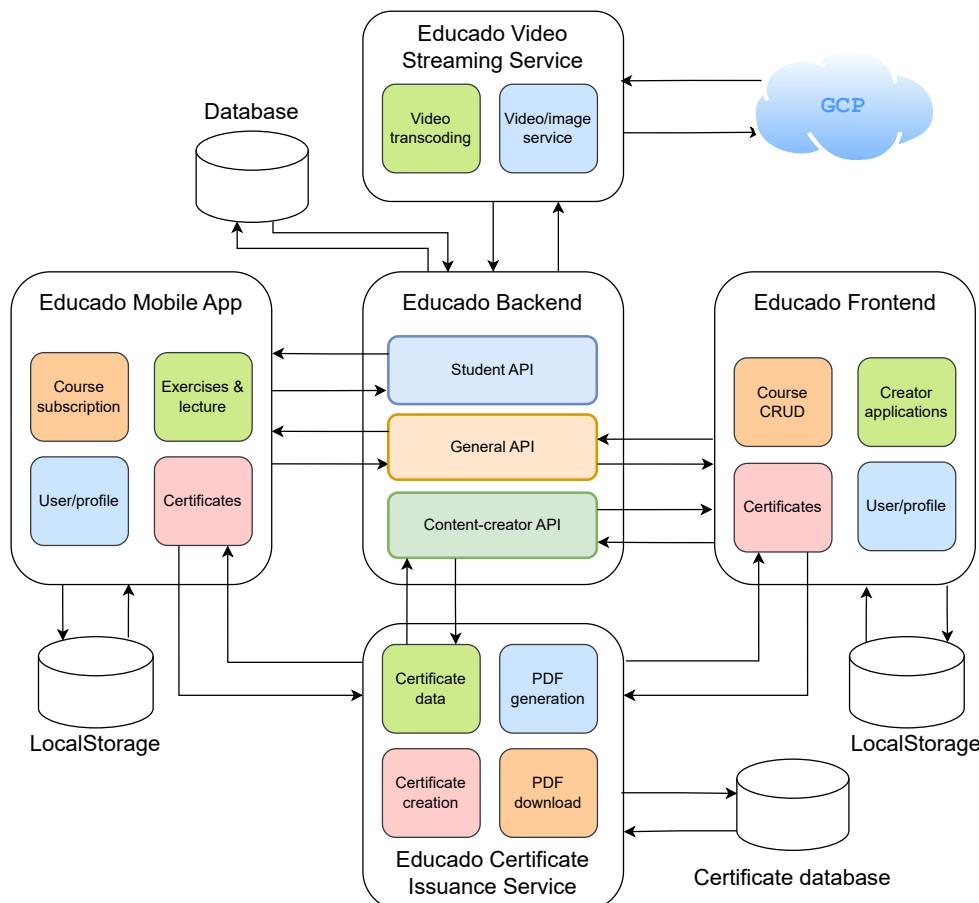


Figure 6.1: Architectural diagram of the Educado platform, consisting of three main repositories and two microservices.

Collaborative writing ends

6.2 CI/CD

CI/CD is the practice to combine continuous integration and continuous delivery, and falls under DevOps, which is "the joining of development and operations teams" [16]. The goal of CI/CD is to eliminate the manual human actions usually required to get new code to production by automating the processes. The process includes running a build, tests, and the deploy phases. A CI/CD pipeline can improve the efficiency of the organisation's workflows by the automation and testing of the code, thus maximising the development time. Another upside to CI/CD is when an organisation grows bigger, the pipeline can decrease development complexity [16].

The continuous integration part of CI/CD is the integration of the code changes into a main branch containing the source code. The idea is that the main branch should run the automated tests with every change and starting a build. The advantages of CI is when merging changes and triggering the tests, the potential bugs, breaking changes or code conflicts are highlighted earlier in the development process [16].

Continuous delivery is the second part of CI/CD. It begins after the CI has been completed, where the CD ensures the code is packaged to be able to deploy to its environment. The deployments are automated like the CI is [16].

Collaborative writing begins

6.2.1 CI/CD for this project

When it comes to the continuous integration and the continuous deployment of the project, 'GitHub Actions' were used for the educado-mobile and educado-frontend repository, while the educado-backend is set up with Google Cloud. Different GitHub Actions workflows were used for the different repositories.

The Educado mobile repository has four different workflows set up.

- **Run Linting on Pull Requests and Branches:** Automatically performs linting checks on pull requests into dev, ensuring correct code style and props validation.
- **Run Tests on Pull Requests and Branches:** Runs the tests created on the branch related to the pull requests into dev.
- **CI:** Runs when a pull request is made to the staging branch. It runs both linting and automated tests. In addition, if all the tests pass, it builds an APK file. By building the app on all pull requests into staging, one can make sure that everything on staging is possible to build. In addition to identifying problems early, one can also test the APK files on Android devices before deployment.
- **CI/CD:** This pipeline is run on all pull requests into the main branch. It runs both the linting and the automated tests. Then it builds the app, but as an AAB file. When the build has succeeded, it automatically deploys it into production on the Google Play Store. This pipeline can be seen in Figure 6.2

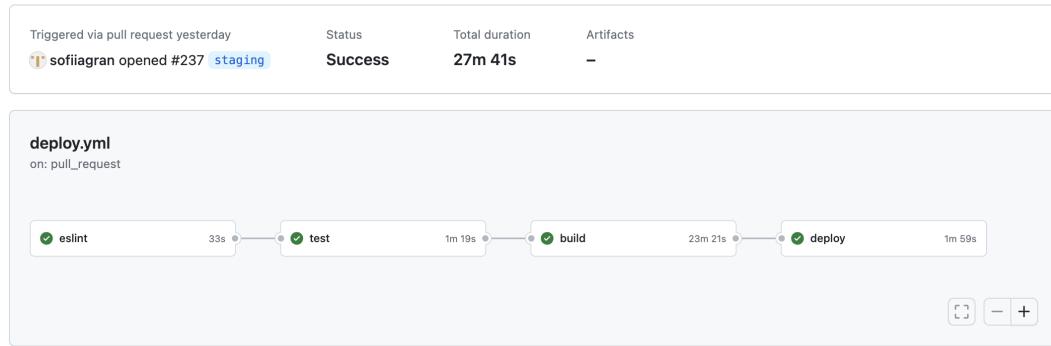


Figure 6.2: Model showing the CI/CD pipeline created for the mobile application on GitHub actions.

The Educado backend repository has two workflows: 'Run Linting on Pull Request and Branches' and 'Run Tests'. They are used for running linting and testing on all pull requests to dev, staging, and main. In addition, all pull requests that are merged into staging are automatically uploaded to the Google Cloud.

- **Run Linting on Pull Request:** Automatically performs linting on the Pull Request branch
- **Run tests:** Automatically runs the tests written in the branch related to the pull request on the development branch

Lastly, the Educado frontend repository had checks similar to the ones previously mentioned: type checking and testing. The frontend, however, also has additional checks for building and deploying the web app using Netlify.

- **Type Checking:** Automatically performs type checking to verify and enforce any desired constraints on variables used in the code
- **Run Tests:** Automatically runs the tests written in the branch related to the pull request on the development branch
- **Deployment and Building:** Automatically uploads and builds the website using Netlify build.

Collaborative writing ends

6.3 Database Model

Collaborative writing begins

The finalised data model for the project will be discussed in this section. The visualisation of the data model is made as an ER diagram, although MongoDB is a NoSQL document database and not a relational database. The database diagram has been split into two clusters for simplicity. The first cluster contains the Users, and the second contains Courses. The students connect the clusters using the Courses and the Content Creators create the Courses.

The first cluster shown in figure 6.3a is the user cluster showing the relations between the user model and its children. The first relation to the users is content creators, which must be approved by sending an application. A model for institutions of the content creators was also created, however, this is omitted in the diagram, as it does not have any references to other models at this stage.

The second relation of the users model is students, which is used to represent mobile users. The student model has a relation to courses, which is all the courses in which the student has either attended a lecture or answered an exercise. The course field within the Student model is an array that holds ObjectIds for each course that the student has taken. The management of course enrollments and student progress is handled within each course. Each index in the courses array has a courses object that also contains an array called sections. The indices of the sections array contain each section of the course. The sections contain a components field, which is an array of components for the specific section. This can be either exercises or lectures. When the student subscribes to a course, the course is added to the courses field in the student model. A component can either be an

exercise or a lecture. Lastly, the third child of the user model is the profile which is used for web users. Another collection connected to the users is the password reset token, which is the token used to reset the user's password.

The second cluster in figure 6.3b is an overview of the relations between the Courses and their children. The course model has a list of ObjectIds which is a reference to all the sections in the course. The same principle is applied in the section model, where the only difference is that components are a list of objects with the ObjectIds stored in compId, and its type is also stored in compType. The compType property is used to determine if a component of the section is an exercise or lecture.

Furthermore, we use a database for the certificate service, although this is very simple and does not contain any direct relations (although it does have the IDs for the user and course associated with the certificate). As such, a diagram for this database has not been included.

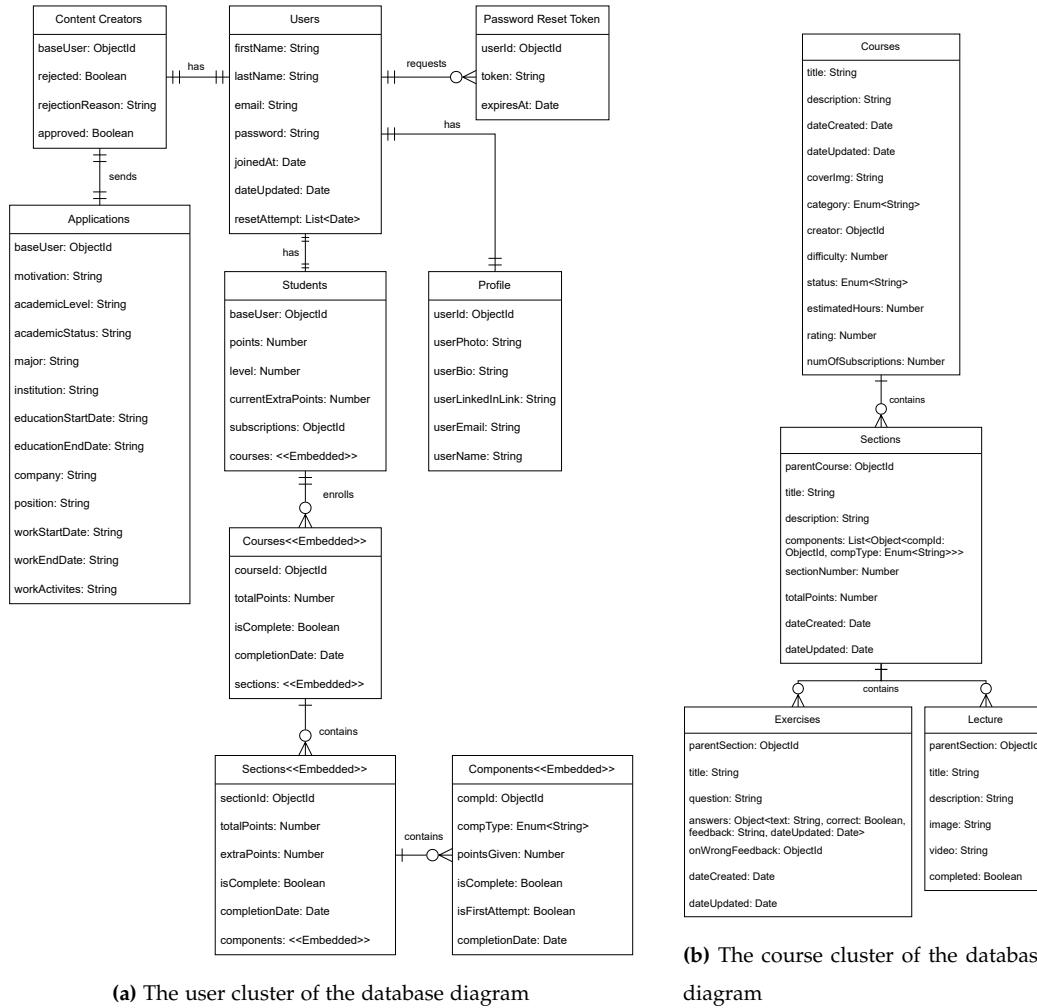


Figure 6.3: Two ER diagrams for users and courses respectively in the database. Note that the implementation of relations may not be exactly as shown in the diagram since we are using a NoSQL database.

Collaborative writing ends

Chapter 7

Quality Management

Quality Management is a crucial aspect in the development of software, encompassing strategies like unit testing, end-to-end testing, and version control. Furthermore, non-technical practices such as user validation are also relevant to value maximisation. This section explores the application of these methodologies in the project, highlighting their roles in ensuring the software's reliability and effectiveness. The chapter discusses the challenges and adaptations in the testing processes, both for frontend and backend components, and details the version control approach adopted for efficient project management and collaboration.

7.1 Unit Testing

Unit testing is a core strategy in assuring high quality of a software product. It is also an essential part of agile development and can lead to poor results if not used properly [12]. This section will delve deeper into how unit testing was done for this project.

7.1.1 Mobile (Frontend) Unit Testing

The Jest testing framework was used for unit testing in the mobile repository when receiving the project. The team's objective was to stabilise and enhance already existing features, so the framework scaled with newer tests and no other options were explored. The development teams found out that the utilisation of testing the frontend took too much time compared to the business value it brought to the stakeholders. A choice was made to increase the technical debt to prioritise increments and the fulfilment of the sprint goals. This was explained further in depth in sprints 5 and 6 [5.3] and [5.4].

7.1.2 Backend Unit Testing

When it comes to unit testing for the backend repository, almost everything new that is implemented must have a unit test dedicated to it. This is done to more easily validate the correctness of the backend, and to make sure that everything works as intended. The test cases that were present in the beginning of the project are outdated because they could not test the routing functionality that the backend provides. As a result of this, the entire test environment was changed, and restructured. Now, the test environment is in a single folder, and uses a library called Jest in combination with Supertest. Jest is used as the general test library, while Supertest is used to test HTTP requests. A test case for route "api/students/:userId/complete" can be seen below:

Listing 7.1: Test route for completing an exercise.

```
1 it('route should return 200 with the updated student
  (exercise)', async () => {
2
3   const response = await
  request(`http://localhost:${PORT}`)
```

```
4           .patch('/api/students/' + userId +
5                         '/complete')
6           .set('token', token)
7           .send({ comp: fakeExercise,
8                     isComplete: true, points: 10 })
9           .expect(200);
10
11
12
13
14
15
16       const student = response.body;
17
18
19
20
21
22   }) ;
```

Some of the "expect" functions are removed to make the code more readable. The first thing to notice is the use of the "it" keyword, which is part of the Jest library. The "it" keyword is used inside a "describe" block which is also taken from the Jest library. "describe" is not shown in this example but it serves as a way to group related tests together. "it" is used to define each individual test case [20]. Inside the test case a constant "response" is set to be equal to a request to a URL. The "request" function is imported in the beginning of the file from Supertest. The request is sent to the URL of the testing environment which

is "http://localhost:3000/api/students/:userId/complete", with a token and a body. The body contains two variables "isComplete" and "points" which are used in the route to modify the data model for the user specified in the URL. Lastly, the request expects a code 200 in the response, and if it receives anything else, the test will fail.

After the request has been verified to respond with a code 200 (success), the student which is also returned from the request is extracted from the body of the response. From line 11-21 the student returned in the response is checked to see if it matches the updatedStudent in the temporary database that is created for the test environment.

7.2 End-to-End Testing

End-to-end testing is a pivotal component of quality management, ensuring the robustness and reliability of software applications. This subsection explores the challenges encountered in implementing end-to-end testing for the mobile version of the Educado applications, addressing both technological intricacies and unique constraints imposed by the development environment.

Building must be done before conducting end-to-end testing in the quality management process for mobile applications, particularly when utilising frameworks like Detox in conjunction with Expo [10]. Detox is a powerful end-to-end testing framework for React Native applications, while Expo offers a convenient set of tools for developing React Native apps. However, the integration of Detox with Expo introduces a set of challenges that require careful consideration.

One challenge is that Expo support with Detox is entirely community-driven [11]. However, the primary challenge is that the Expo version used for the Edu-

cado mobile application is an older version and was never updated. This is a big oversight which will be discussed in [8]. As a result of this, end-to-end testing is not implemented for the mobile application and is only done manually.

7.3 User validation

Including stakeholders, such as users, in sprint review is a common and very useful practice. During the review meeting of the fourth sprint, the product owner assisted in facilitating a user validation test with some waste pickers, who were considered potential users of the app. A member from each team was also present to receive the information firsthand and note any usability-related issues while inspecting the users' interaction with the product. Following the test, the product owner collected the feedback and refined the product backlog based on the newfound information.

7.4 Version Control

Version control is integral to quality management, offering developers a comprehensive history of code changes. This historical record is crucial for tracking and managing modifications throughout a project's lifecycle.

Collaborative writing starts

For the project, Git with Github was used to manage versioning. The entire project consisted of five primary repositories:

- Educado-frontend: The web application used by content creators for creating and uploading content to the Educado Mobile App.
- Educado-backend: The primary back end, is used for handling most of the traffic between the different parts of the system, like the mobile app, the web application, and microservices.

- Educado-mobile: The mobile app is responsible for serving the content to mobile users.
- Educado-transcoding-service: The microservice is responsible for handling the transcoding of video and all the traffic regarding GCP storage buckets.
- Educado-certificate-service: The microservice handles the creation, retrieval, download, and deletion of certificates.

Each repository consisted of three key branches;

- dev: The Development branch used for pushing new features that have been tested thoroughly and reviewed.
- staging: The Staging branch had changes pushed to it once or twice per sprint. All of these features should be stable and work as intended. This branch was used to present the product to the product owner in each sprint review.
- main: The Main branch, which acted like a production branch.

During the development and implementation, there were several branches for the different features. These branches would all have a prefix, depending on the group to which they belonged. For instance, "video-" would imply that the branch was used by the video streaming group. Then the name of the feature being worked on would be added. An example of a branch name could be "video-upload-to-bucket", which then further would imply that the feature implemented in the branch had something to do with uploading data to the storage bucket. This way, one could keep better track of the different branches.

When merging a branch into one of the collective branches (dev, staging, and main) we would create a new branch specifically for merging. This branch used the destination branch as base, and the merged feature branch as additions. This

allowed the Nexus team to more easily solve merge conflicts, as well as allow fine-grained control over which files you wanted to merge from the feature branch.

Collaborative writing ends

Chapter 8

Discussion

This chapter critically analyses the challenges faced in implementing Agile methodologies, managing software repositories, and handling databases in a software development project. It focuses on the selection of an appropriate Agile scaling framework and the impact of key aspects such as communication and transparency on project efficiency. Additionally, the chapter assesses the initial state of received repositories, the balance between code quality and delivery speed, and the integration of gamification features within the project.

8.1 Issues with implementation of Agile

Shifting from a software development setting primarily resembling a waterfall model was a major challenge. The velocity of providing value to the product owner was often reduced because of this, and the small increments became larger than they should have been, resulting in integration problems. Integrating large chunks of code, namely several PBIs at once, requires more resources and creates bottlenecks in the delivery process. To summarise, a driving aspect of agile development was misunderstood by some teams, which ended up in a waterfall-type development process being forced into an agile setting.

Furthermore, selecting the scaled framework for the project was certainly not a sufficiently structured exercise. At the time of deciding, the teams did not have a sufficient understanding of any of the frameworks to make a qualified guess as to which framework would make the most sense to go with. Arguably, the Scrum of Scrums could have been an easier first step into the domain of Agile Software Engineering. Scrum of Scrums is a more lightweight framework that relies on the basic Scrum principles and practices. It is a natural extension of the existing framework, making it easier for teams to understand and implement. Scrum of Scrums typically involves representatives from each team participating in coordination meetings addressing impediments in comparison to the Nexus framework's integration team and numerous events, which add an extra layer of complexity [25].

Communication and transparency have also been critical bottlenecks for the productivity of the Nexus team. Changes to the database models, dependencies, and backend routes resulted in massive slowdowns in the merging process due to not being communicated properly. These issues should have received more attention from the integration team. A possible improvement could have been putting more resources into sprint planning, clarifying dependencies and planning out a logical way of coordinating tasks for the sprint as well as their integration. Possessing a more thorough understanding of others' contributions and timelines, and a more structured integration plan. The database models themselves should have been kept up to date whenever changes were made.

Another aspect of the sprint planning which has not received enough attention was the definition of acceptance criteria and user stories. These properties of a PBI should have given a clear understanding of the product owner's expectations and should have been more thoroughly discussed rather than the team defining

them and the product owner simply accepting the definition of the PBI including its user story and acceptance criteria as a whole.

The overall communication with the product owner, the domain expert representing crucial stakeholders, should have been more frequent and thorough. The team of this paper recognised this mistake during the start of the sprint and also communicated it via integration team meetings to share their knowledge, but the issue of misaligned understanding between some teams and the product owner persisted until the end of the project.

The team of this paper followed the advice of management on replacing the Scrum master and integration team representative for each sprint. This way, every team member got a taste of the work and responsibilities of the integration team, thus a better understanding. Some other teams, on the other hand, decided to have one representative throughout the entire project, which had some positive as well as negative effects. These people got a more thorough understanding of the big picture of the project, thus making qualified decisions affecting the whole Nexus Team and improving the effectiveness of communication with the repository manager. However, these people became much more dominant in common discussions and integration work, deciding on topics without hesitation that should have been further discussed, for instance, the prioritisation of implementing automation, end-to-end testing, and finalising the CI/CD pipeline.

8.2 Issues with the received repositories

It was discovered that the Nexus team was initially provided with a backend from 17. October 2022, due to a lack of sufficient version control system in last year's project. This explains the initial over-simplicity of the backend and the misalignment of the endpoints between mobile and backend repositories. The

work on the mobile repository was also largely affected by this oversight by the repository manager.

8.3 Issues with code and database

As mentioned in several sprint reviews, a major bottleneck in the delivery process was the unbalanced ratio of quality code and finished PBIs. The majority of the sprints did not result in enough completed PBIs because of this, but it was a valuable learning experience to understand the balancing of code debt and velocity of value delivery. The removal of frontend testing could have also happened earlier on since it is much more important to test logical components rather than rendering.

The database model was not updated regularly, which resulted in a lot of frustration from numerous groups. It was taken up at several integration team meetings but was not highly prioritised. The database model should never have been updated as regularly as the database itself did. To prevent this, the integration team should have assigned a single team to create the models at the beginning of the project, so they were well-defined before the individual Scrum teams started to work with the models.

8.4 Evaluation of degree of Gamification

The features related to gamification that were implemented are onboarding-screen, points, levels, exercise popups, section and course completion animations including statistics, and a more scalable profile rework that facilitates further gamification elements such as daily streaks or leaderboards. Although the implementation of leaderboards could have given a lot of value, a stable release was deemed a higher priority. Nevertheless, the implemented features were de-

veloped following competitors' examples and with the central aspects of gamification in mind, appealing to the user's core drives. At its current state, the Educado App follows numerous important guidelines 4.1.4 such as rapid feedback cycles, breaking down large tasks into simple sub-tasks for the user, and recognising progress with rewards. Common game mechanics and dynamics 4.1 could have been more widely implemented, providing rewards for collected points other than level-ups, badges or achievements, or facilitating healthy competition with leaderboards. As previously mentioned, the frontend part of the leaderboard and streaks were implemented, which makes it easier for future developers to finish the pbi.

Collaborative writing begins

8.5 Integration team issues

As stated before, one of the ways the different teams used to communicate with each other was through the integration team. Having an integration team helped with transparency by gaining a better overview of the different group's progress. It also contributed to better cross-functionality between groups by being a channel for communication between groups and solving dependency issues. However, at times, the integration team was more focused on the general overview and progress, than actual integration between teams. Especially, the web and mobile teams lacked communication with each other. This caused some problems with teams working on some code that other teams were dependent on, or by spending time structuring data that may cause problems for other teams. This caused multiple setbacks throughout the project. If one were to utilize the full potential of the integration team, one could avoid these problems caused by a lack of communication between groups.

A lot of the decisions made by the integration team were also communicated to the rest of the teams by their representative instead of written announcements to everyone. This caused a few problems with the decisions being communicated differently depending on the representative and sometimes getting forgotten without a written record accessible for everyone to easily see. In general, having a better structure for how to document the agreements and decisions discussed in the integration team would be beneficial.

8.6 Product owner

The PO is a domain expert, and their responsibility is to formulate user needs as product backlog items, refine these items, and ensure that the value created goes towards the overarching goal of the project. Their approval of any item is pivotal to consider an item done.

Throughout the project, the PO often had to remind the Nexus team of the sprint goals, which helped keep the bigger picture in mind. During sprint reviews, the PO was clear and concise about the expectations set for the team, and although they had their clear priorities, a lot of freedom was given to the developers in discussing the value, size, design, and clarification of acceptance criteria. This was also needed, since, as a domain expert, the PO is not necessarily aware of the technical difficulties of, e.g., implementing a Google Login or resetting a password via email.

The product backlog was always kept up to date, and PBIs were selected based on how much they contributed to the actual sprint goal and how much value they added to the product. The selection took place in correspondence with the developers for a good mutual understanding. The acceptance criteria were defined by the developers, and at times these were accepted by the PO without

further discussion. Since there was no discussion, it could at times, lead to the developers and the PO having different interpretations of the criteria.

In general, the PO was always open for meetings and easy to contact through Discord. She had frequent checkups with the groups to keep updated on their process, as well as realign expectations and priorities based on progress. At the end of Sprint 4, the PO moved back to Brazil, which led to communication being less frequent. However, to compensate, she then started to participate more in integration meetings, where she consulted with group representatives about progress, expectations, and priorities.

In the first couple of sprints, there were a lot of dependency problems between teams. Therefore, the PO took responsibility and started working on dependency management during sprint planning. In addition, she often reminded teams to deliver increments and thereby adhere to Scrum principles. Overall, the consensus surrounding the work with the PO is that it was satisfactory and enhanced our work and scrum process.

Collaborative writing ends

Chapter 9

Conclusion

The project comprised six sprints aimed at enhancing the Educado platform by applying a scaled Agile software development framework, Nexus, focusing on stability, usability, and, for this report, a gamified learning experience. Key accomplishments include implementing login features, refining profiles, enabling course creation, and addressing exercise-related challenges. Improvements in user experience, points systems, and backend logic were achieved, with a strong emphasis on stakeholder communication and acceptance criteria. Challenges were encountered in managing technical debt, unresolved dependencies, and maintaining code health. Each sprint contributed to a more robust platform, culminating in a successful final sprint with a deployed and functional application. The retrospective highlighted increased velocity but emphasised the need for improved cross-team communication. Despite slight decreases in code health, the project delivered value, leaving behind improved documentation and code standards for future development.

Bibliography

- [1] URL: <https://apps.apple.com/us/charts/iphone/education-apps/6017>. (accessed: 09.11.2023).
- [2] URL: <https://brilliant.org/>. (accessed: 30.11.2023).
- [3] Open AI. *All things about ChatGPT*. URL: <https://help.openai.com/en/collections/3742473-chatgpt>. (accessed: 02.10.2023).
- [4] Luiza Cardoso Queiroz Melo. *Educado project introduction*. English. Aalborg University in Copenhagen, Sept. 2023. URL: https://www.moodle.aau.dk/pluginfile.php/3137156/mod_resource/content/1/Software%20-%20Project%20Introduction%20%281%29.pdf (visited on 06/12/2023).
- [5] Yu kai Chou. *Gamification Design: 4 Phases of a Player's Journey*. URL: <https://yukaichou.com/gamification-examples/experience-phases-game/>. (accessed: 09.11.2023).
- [6] Yu kai Chou. *The Octalysis Framework for Gamification & Behavioral Design*. URL: <https://yukaichou.com/gamification-examples/octalysis-complete-gamification-framework/>. (accessed: 09.11.2023).
- [7] Yu kai Chou. *Yu-kai Chou LinkedIn Porfile*. URL: <https://www.linkedin.com/in/yukaichou/>. (accessed: 09.11.2023).

- [8] CodeScene. *CodeScene for Developers*. 2023. URL: <https://codescene.com/for-developers>. (accessed: 29.09.2023).
- [9] Nico Sacheri Courtney Leung and John Trivelli. *Introducing Friends Quests, a fun way to team up and learn!* URL: <https://blog.duolingo.com/friends-quests/>. (accessed: 09.11.2023).
- [10] Detox. *Detox*. URL: <https://wix.github.io/Detox/>. (accessed: 27.11.2023).
- [11] Detox. *Expo*. URL: <https://wix.github.io/Detox/docs/19.x/guide/expo/>. (accessed: 27.11.2023).
- [12] Kim Dikert, Maria Paasivaara, and Casper Lassenius. "Challenges and success factors for large-scale agile transformations: A systematic literature review". In: *Journal of Systems and Software* 119 (Sept. 2016), pp. 87–108. ISSN: 01641212. DOI: [10.1016/j.jss.2016.06.013](https://doi.org/10.1016/j.jss.2016.06.013). URL: <https://linkinghub.elsevier.com/retrieve/pii/S0164121216300826> (visited on 12/07/2023).
- [13] Educado. *The Educado Initiative*. URL: <https://github.com/Educado-App>. (accessed: 03.10.2023).
- [14] European Comission. *What is Erasmus+*? URL: <https://erasmus-plus.ec.europa.eu/about-erasmus/what-is-erasmus> (visited on 12/06/2023).
- [15] GitHub. *Your AI pair programmer*. URL: <https://github.com/features/copilot>. (accessed: 02.10.2023).
- [16] Gitlab. *What is CI/CD?* 2020. URL: <https://about.gitlab.com/topics/ci-cd/>. (accessed: 11.12.2023).

- [17] IBM. *What is three-tier architecture?* URL: <https://www.ibm.com/topics/three-tier-architecture>. (accessed: 11.12.2023).
- [18] Lisa Homner Michael Sailer. *The Gamification of Learning: a Meta-analysis.* URL: <https://link.springer.com/article/10.1007/s10648-019-09498-w>. (accessed: 09.11.2023).
- [19] Mozilla. *HTTP response status codes.* URL: <https://developer.mozilla.org/en-US/docs/Web/HTTP>Status>. (accessed: 14.12.2023).
- [20] Meta Platforms. URL: <https://jestjs.io/>. (accessed: 07.12.2023).
- [21] Meta Platforms. *JestJS.* URL: <https://jestjs.io/>. (accessed: 03.10.2023).
- [22] Productboard.com. *Agile Definition of Done.* 2019. URL: <https://www.productboard.com/glossary/agile-definition-of-done/#:~:text=In%20agile%2C%20the%20definition%20of,perform%20for%20every%20backlog%20item.>. (accessed: 28/09/2023).
- [23] Scrum.org. *Introduction to the Nexus Scaled Scrum Framework.* 2019. URL: <https://www.youtube.com/watch?v=29M-rw6DWd8>. (accessed: 27/09/2023).
- [24] Scrum.org. *Scaling Scrum with Nexus.* 2023. URL: <https://www.scrum.org/resources/scaling-scrum>. (accessed: 27/09/2023).
- [25] Chris Spanner. *Scrum of Scrums.* URL: <https://www.atlassian.com/agile/scrum/scrum-of-scrums>. (accessed: 14.12.2023).
- [26] Duolingo Team. *How do Duolingo Leaderboards work?* URL: <https://blog.duolingo.com/duolingo-leagues-leaderboards/>. (accessed: 09.11.2023).
- [27] United Nations. *The 17 goals.* URL: <https://sdgs.un.org/goals> (visited on 12/06/2023).

Appendices

The Octalysis Framework

The Octalysis Framework is a design framework created by Yu-kai Chou, a gamification expert and pioneer [7]. It is used to analyse and design experiences that effectively engage and motivate people. The framework is based on the idea that human motivation is driven by eight core drives, which are categorised into two main groups: the Left Brain and the Right Brain Core Drives [6].

The eight core drives identified in the Octalysis Framework are as follows:

1. Epic Meaning & Calling: This drive involves the desire to be part of something bigger than oneself and to work towards a meaningful goal or purpose.
2. Development & Accomplishment: It encompasses the drive to make progress, achieve goals, and overcome challenges.
3. Empowerment of Creativity & Feedback: This drive is about the satisfaction derived from expressing creativity, experiencing positive feedback, and feeling a sense of control over one's environment.
4. Ownership & Possession: It refers to the desire to own and control things, and the satisfaction that comes from possessing or being responsible for something.

5. Social Influence & Relatedness: This drive involves the desire for social interaction, influence, and a sense of belonging within a community.
6. Scarcity & Impatience: It relates to the motivation derived from the perception of limited availability or the fear of missing out on something valuable.
7. Unpredictability & Curiosity: This drive revolves around the excitement and engagement that comes from unpredictability, curiosity, and the exploration of the unknown.
8. Loss & Avoidance: This drive is associated with the desire to avoid negative outcomes or losses, and the motivation to protect oneself from potential harm.