Studienarbeit
Documentation

Photochef

Semester: Fall 2024

Date: 2024-12-20

Project team: Leonardo Ravani, Simon Peier, Tobias Kistler

Project advisors: Mitra Purandare, Clemens Meier

Departement of Computer Science
OST Eastern Switzerland University of Applied sciences

Abstract

The Smart Eating application is a web application that allows users to manage recipes in a smart way. The application is capable of analyzing recipe pictures into text and suggesting meal plans depending on the user's nutritional needs, but it lacks essential user-centric features and an intuitive user experience.

The objective of this project is to implement additional features that improve the usability of the Smart Eating application. These features include editing, saving and sharing of recipes, user authentication and user management as well as a dashboard. Furthermore, the user interface should be improved to make the application more user-friendly and appealing.

To approach said features, wide-spread software engineering practices are applied. First requirements engineering is conducted and then the problem domain is analyzed to find out which parts of the application are in need of modification and extension. After understanding how the application works and analysing the UI, mockups are created. Furthermore, additional features were discussed and planned for during this elaboration phase. Then, it is time to implement these features. In the course of implementation, it gradually became apparent, that the application possesses an exceptionally complex structure with virtually no documentation available. This significantly delayed the implementation, requiring frequent meetings with the advisors.

Due to these delays and the inclusion of additional features, the full range of planned features is not implemented. Nevertheless, the improvements introduced are expected to significantly enhance the application's usability and functionality.

Management Summary

Initial situation

Smart Eating is an AI powered web application that is used to analyze recipe texts and images. It can also create customized menu plans based on the users eating and nutrition preferences and their biometric data. The goal of this project is to improve the existing Smart Eating application. Adding extra features and refining the user interface aims to improve the general user experience.
These new features include editing, saving and sharing of recipes, user authentication with GitLab OAuth, an admin panel for user management and a new dashboard.

Procedure and Technologies

The approach is to extend the Smart Eating application by applying wide-spread software engineering practices. Initially, requirements engineering is conducted to analyze, identify and document the needs of the stakeholders. Then the problem domain is analyzed to find out which parts of the application are in need of modification and extension. Afterwards, the features are implemented. After the first milestone, usability testing is conducted with different end users. The generated feedback about the user experience and about is then implemented in the next iteration and usability testing is conducted again. The purpose of this is to gather feedback about the previously proposed improvements and check for bugs again.

The frontend is implemented with the React framework, using the MUI library for styling. It communicates with the backend using an API, which is generated with NSwag. The backend in turn is implemented with .NET framework. It uses Microsoft's entity framework which is used to generate the database schema and to access the database. Together, the frontend and backend build the web application container of the smarteating project. It is not in the scope of this project to modify and extend the other containers.

Result

The final result improves the user experience and enhances the functionality. Only the implemented extensions are explained, talking about already existing features is omitted. A user can log in with an OST Gitlab account. He can upload and analyze a recipe (with

AI) and then correct possible mistakes in result. Saving a recipe is still not possible. If a user has the administrator role, he can access the admin panel. This allows him to view all users, edit details of them and grant/revoke admin rights of other users. Navigating the application is easier, as thematically close topics are grouped together.

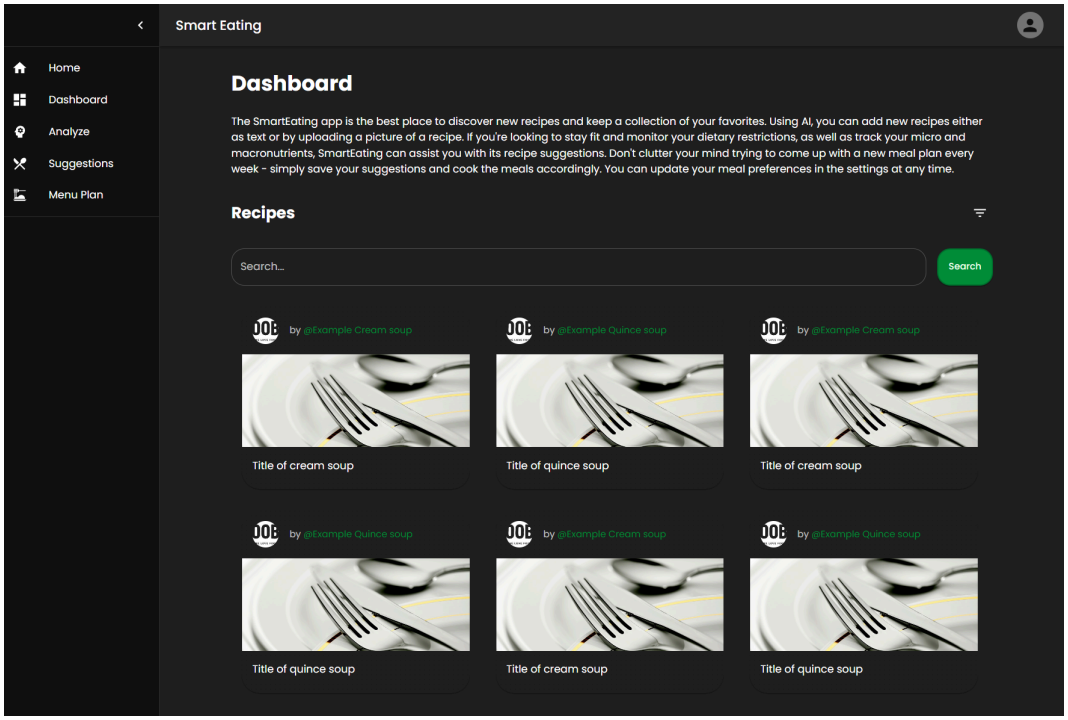


Figure 1: Smart Eating now

Despite this, the application remains incomplete. For a person to fully utilize the Smart Eating application, some features need to be implemented. Not being able to save recipes makes the whole analyze page essentially useless. Future development will depend on upcoming projects, which are beyond our control. The decision is up to the Smart Eating team.

Contents

1 Glossary

- **AI**: Artifical Intelligence is a computer process that simulates human-like thinking or behaviours.
- **API**: An Application Programming Interface is a set of rules through which different parts of an application can communicate with each other.
- **Agile**: In software development agile is a common methodology that focuses on iterative development, collaboration and adaptability.
- **C4-Model**: C4 stands for context, container, component, and code. This software architecture model is utilised to visualise IT systems.
- **Container diagram**: This diagram of the C4-model focuses on the containers of a system and how they interact (applications, databases, etc.).
- **Context diagram**: Another diagram of the C4-model that focuses on the data flow between the system and external entities like users or other systems.
- **CRUD**: Create, Read, Update, Delete. These are the 4 operations used when working with databases.
- **Docker**: A platform that allows developing, shipping and running applications in portable containers.
- **DTO**: Data Transfer Object, is used to transfers data between processes.
- **Elaboration phase**: The second phase of the Rational Unified Process that focuses on addressing and refining key system architecture and risks.
- **Fooby**: A website that has a vast collection of recipes.
- **FR**: Functional Requirements show what features need to be implemented during a project. These are normally interactive features.
- **GitLab**: A tool that allows for managing and versioning code and for coding among multiple people such as a team.
- **High fidelity**: A design or a prototype of a feature at a high level of detail that resembles the final product very closely.
- **IFS**: Iteration Feature Set is utilised in agile development to group features that are to be developed during a sprint.
- **Jira**: A project management tool used to track time on issues as well as plan sprints and issues.
- **Low fidelity mockup**: A design or a prototype of a feature at a low level of detail showing only generally what must be implemented and giving a general idea of what it should look like.

- **NFR**: Non Functional Requirements are less visible in the end product but are equally important addressing topics such as test-coverage, error handling, etc.
- **OAuth**: Open Authorization is an open standard for access delegation that is used to allow websites or applications to access user information.
- **OST**: The Ostschweizer Fachhochschule is a university of applied sciences in Switzerland with locations in Rapperswil and St. Gallen
- **PostgreSQL**: A powerful open-source relational database system utilising SQL.
- **Rational Unified Process**: A development process framework used in software engineering, designed for iterative development.
- **Risk Matrix**: A matrix that plots the probability of a risk occurring against its damage potential if it were to occur.
- **Scrum**: A framework that utilises sprints as fixed-length iterations to promote an agile software development.
- **Scrum+** : An enhanced version of regular scrum which in addition adds more practices and tools for productivity.
- **Scrum-Master**: A role in Scrum. The scrum-master is responsible for removing impediments as well as ensuring the team adheres to Scrum practices.
- **SEProject**: Software Engineering Project is a module which is usually completed by students before conducting the Studienarbeit. Both are very similar in structure.
- **Sequence diagram**: Provides a step by step overview over a given process in a sequence from the top to the bottom.
- **Sprint**: A fixed time period during which agreed upon features must be developed. Before and during sprints meetings are held to discuss the current progress as well as plan the next sprint.
- **Sqlite**: A lightweight serverless relational database.
- **UC**: A Use Case describes how users interact with a system to achieve a specific goal.
- **UI**: The User Interface is the visual part of an application that users interact with.
- **UX**: User Experience. The overall experience that a user has with an application. This refers more to things like the visual appeal or maneuverability of an application.

2 Introduction

The introduction first describes the initial situation of the Smart Eating project, the goal of our thesis and the framework conditions.

2.1 Initial Situation

The motivation of this project is to improve the user experience and extend the functionality of the Smart Eating project. The current application lacks certain key features, e.g. user authentication or editing a recipe after analysis.
Widely used software engineering practices will be used to plan for and implement these extensions and improvements.
The Smart Eating application was initially created with the intention to create an AI with the ability to analyze images of recipes and extract text from them. This was done at the IFS and in previous semester and bachelor theses. While previous theses focussed on the implementation of AI models, our theses focuses on improving the user experience and extending the user interface with key features.

2.2 Study thesis objective

This project aims to improve the user experience and extend the existing functionality. The key features to be implemented are authentication and user management, creating and editing of recipes and sharing of recipes. Optionally the generation of a shopping list, using the recipes from the menu plan, can be implemented. In addition, it might be a good place to mention, that during elaboration were discussed and planned for.
Furthermore, the User experience should be improved, including already existing components of the application.

3 Requirements

This section describes the functional and non-functional requirements.

3.1 Functional Requirements

Below are the functional requirements to meet the features described in the “Aufgabenstellung” (task definition).

3.1.1 Actors

In our system under development (SUD) we have identified two actors, a user and an admin.

User

Goal: Wants to use the Smart-eating application to capture, edit and save recipes. Furthermore, he wants to share recipes.

Admin

Goal: Wants to manage the users by reading, updating and deleting user data.

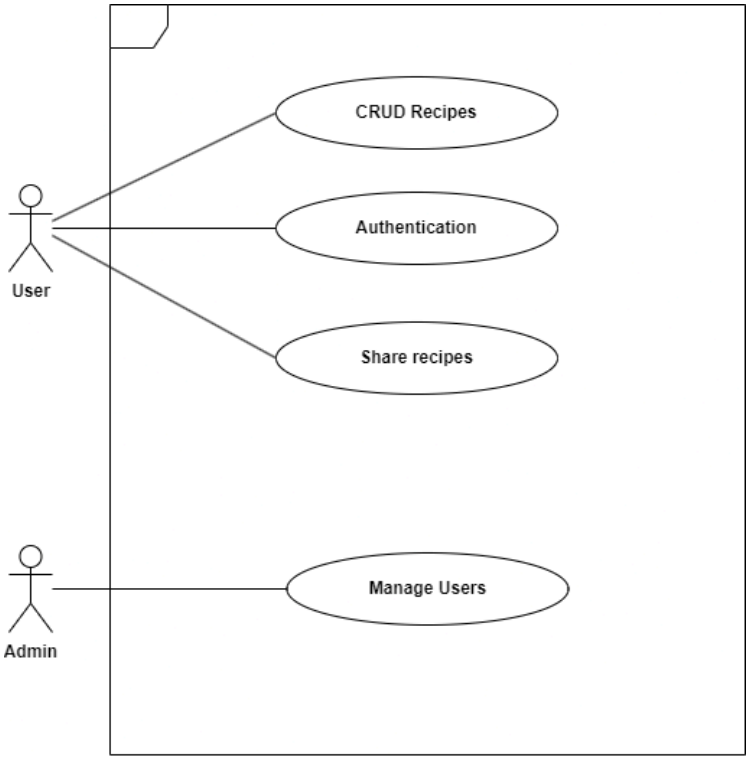


Figure 2: Jira Workflow

3.1.2 Use Cases

UC1: CRUD Recipes

The user wants to add new recipes to the application. When adding, he wants to be able to edit any mistakes of the AI. He also wants to be able to read, update or delete them at a later point.

UC2: Authentication

The user wants to access the application. For this, he either needs to enter his existing login credentials. Otherwise, he should be able to create a new account.

UC3: Share Recipes

The user wants to be able to share his recipes with other users on the platform.

UC4: Manage (CRUD) users

The admin wants to manage users. He should be able to execute create, read, update and delete operations on users.

Because the authentication (incl. registration) is implemented using OAuth, it does not make sense for the admin to have the ability to create new users. Therefore the admin should only be able to read, update and delete users.

3.2 Non Functional Requirements

As it is not in the scope of our project to deploy the application, we do not have any NFR's regarding Performance.

ID	NFR-1
Subject	The application is easy to understand and use
Requirement	Usability - Learnability
Measures	A user can open the app and use its functions without the need of a tutorial.
Priority	Medium
Creation Date	07.10.2024
Verification Date	Beta release
Due Date	22.12.2024

Table 1: NFR-1

ID	NFR-2
Subject	A user is protected from errors
Requirement	Usability - Error Handling
Measures	Errors are properly handled within the application preventing users from being confronted with error codes or other issues. Only simplified understandable messages should be displayed to users.
Priority	Medium
Creation Date	07.10.2024
Verification Date	Beta release
Due Date	20.12.2024

Table 2: NFR-2

ID	NFR-3
Subject	A user may only interact with data he has access to
Requirement	Security - Access Control
Measures	A user can only view the content he has been granted access to or created by himself.
Priority	High
Creation Date	07.10.2024
Verification Date	Beta release
Due Date	20.12.2024

Table 3: NFR-3

ID	NFR-4
Subject	The app is capable of handling errors without crashing
Requirement	Reliability - Fault tolerance
Measures	When errors occur in the application they are caught and dealt with instead of making the entire application freeze or crash.
Priority	High
Creation Date	07.10.2024
Verification Date	Beta release
Due Date	20.12.2024

Table 4: NFR-4

ID	NFR-5
Subject	The app must be visually appealing
Requirement	Usability - Aesthetics
Measures	The application's design must be visually appealing to the customers as well as the product owner.
Priority	Medium
Creation Date	07.10.2024
Verification Date	Beta release
Due Date	20.12.2024

Table 5: NFR-5

NFR-1

During the usability tests, the testers will create an environment for the user that will simulate them trying the application for the first time at home. During this process, the users will demonstrate how understandable the application is. If a new user is able to complete the usability test with only 2 or less mistakes the NFR will be approved.

NFR-2

Because errors only occur when unexpected data or processes are being handled, it will be impossible to think of every case for every field in the application. However, the application will be tested with various erroneous input and procedures to trigger errors which should then be caught internally and if necessary show only a message that would be understandable to a person outside of the IT world. If during this testing an error triggers an event that is not covered in this way the NFR would be denied.

NFR-3

New recipes will be created on two different users. Then, while utilizing the search function, the recipe of the other user should not show up unless it was shared. In this case, the NFR would be approved.

NFR-4

During the procedure of verifying the NFR-2, it will become clear if the application crashes at any induced errors or if they are all handled properly. If the application does not crash or freez during the verification process of NFR-2, the NFR-4 will be approved.

NFR-5

As part of the usability test, the users must be questioned on the design. Should the average result yield a rating less than 7/10 the NFR would be denied. Important to note is that on a scale of 1 to 10, the 7 will be excluded as studies have shown that it is more likely to be picked in order to give a neutral opinion. Omitting the 7 would result in more 6 and 8 showing a more clear line of who likes or dislikes the design despite being polite.

4 Architecture

This project is about extending an existing application and not about creating a new one. Therefore this section focuses on our extensions to the project. The application in its entirety, including the existing parts, is only superficially documented. Parts of the existing application that are deemed important are documented more precisely nevertheless.

4.1 C4 Model Architecture

The following architecture diagrams are based on the C4 model. They depict how the application is built in different levels of detail. Only the first two Cs, Context and Container, are used since Components and Code will be subject to many changes. Having a high-level overview of the architecture is enough.
The parts of the diagrams in green color represent our additions, the rest of the architecture already exists.

4.1.1 C4 Context

Figure 3 is the most abstract depiction of the architecture showing the big picture. It shows our system surrounded by its users and all the external systems it interacts with.

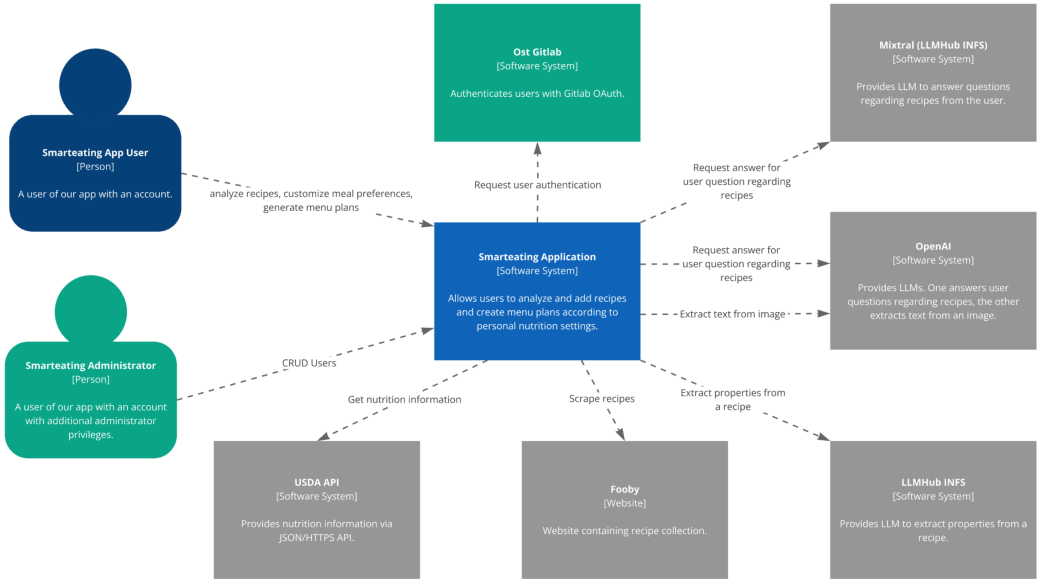
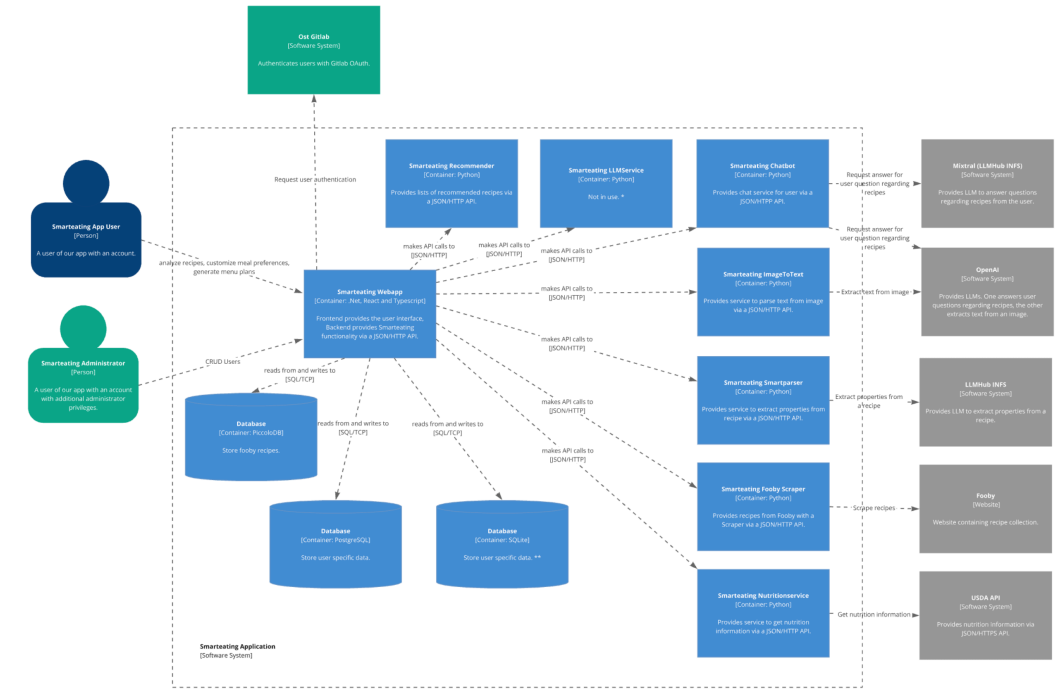


Figure 3: C4 Context Diagram

4.1.2 C4 Container

The container diagram (Figure 4) shows how the application is split up and how the containers interact with each other.



* No team member knows what the LLMService is used for.

** The SQLite db is currently contained inside the Smarteating Webapp container. For clarity reasons and as it is planned to decouple this db in the future, it is depicted as a separate container.

Figure 4: C4 Container Diagram

4.2 Deployment

The Smart Eating Application is deployed using docker. These docker files already exist and it is not in the scope of this project to create a deployment pipeline (which would use those scripts) or to deploy the application to the live system.
During development, the app is deployed by running the application locally with docker.

4.3 Database

The application has multiple databases. It incorporates a PostgreSQL database and a SQLite database, each with distinct roles. Additionally, there is a Python service named PiccoloDB, which, despite its name, is not an actual database but a service designed for enhancing prompts with AI.

4.3.1 PostgreSQL

The PostgreSQL database is run in a separate docker container. It is the main database, where data about recipes, nutrients, etc. is saved. This database contains the schemas *recipe*, *scraper_fooby* and *usda_data*. Its main purpose is to contain the recipes that have been retrieved from the fooby website. The *scraper_fooby* contains the raw recipes gathered from fooby and the *recipe* schema contains the recipes as well but altered to fit the project structure. Lastly, the *usda_data* holds all information regarding the weight and measurement-specific information. On the database, mainly read operations are performed in order to keep the performance on a high level.

4.3.2 SQLite

The sole purpose of this database is to store all data related to a user in any way. Prominent data related to the user are stored here which include:

- User roles: Information about user permissions.
- Personal information: Data such as names, email addresses, and settings.

To scale this database, multiple instances can be initialized.

4.4 Authentication

The previous authentication/login for the default user Max Muster persists for development purposes. In addition, the application also offers an OAuth login to authenticate users. This allows users with an OST GitLab account to authenticate or register for the Smart Eating application. The sequence seen in Figure 5 sequence describes the entire authentication process with OAuth:

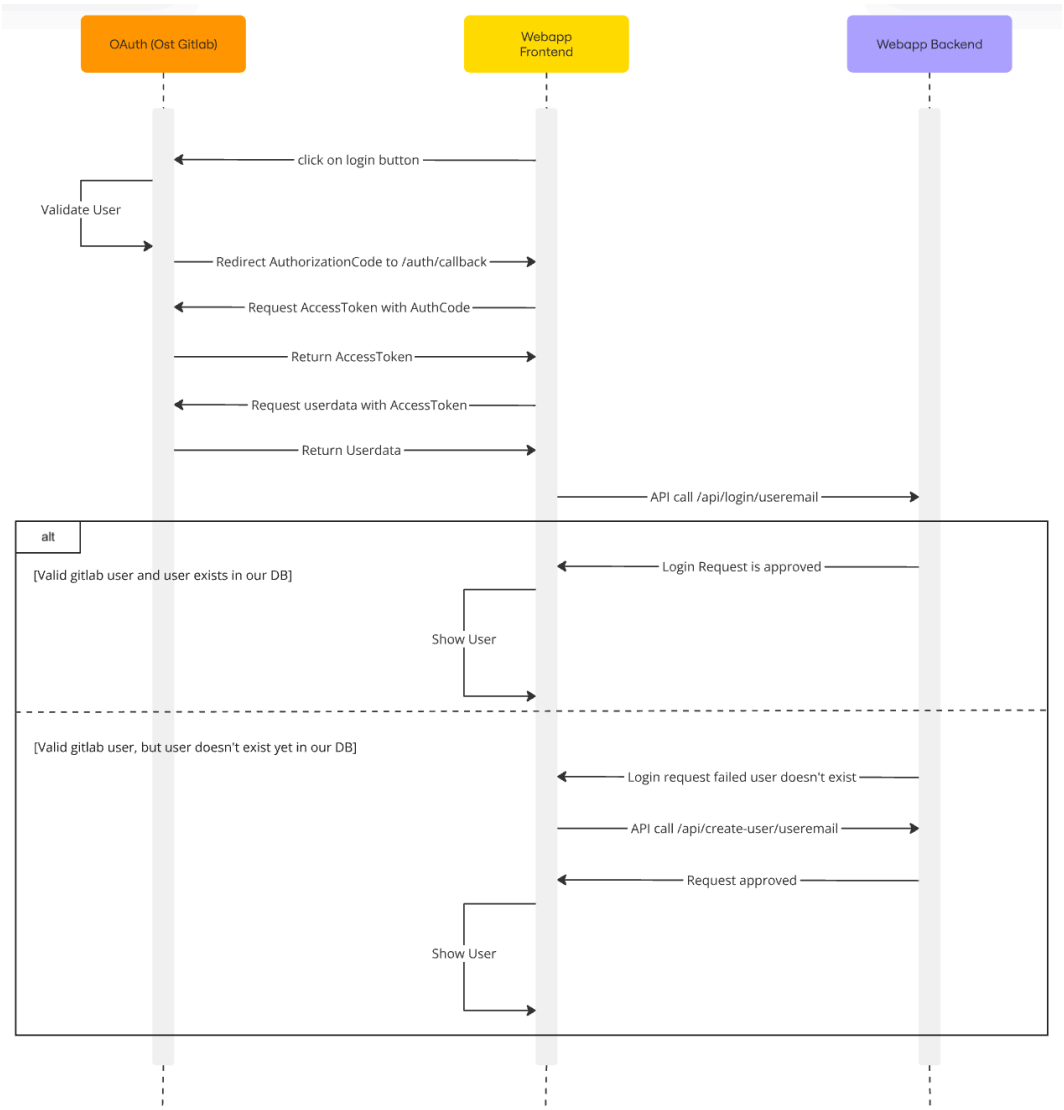


Figure 5: Seqeunce diagram for authentication

5 Implementation

5.1 Usability testing

As the enhancement of the user interface is a high priority, usability tests will be performed to ensure that users of different groups are able to comfortably navigate and understand the application. To include people of different age groups, each usability test will be performed with a minimum of 3 people, all of different age groups.

5.1.1 Pinpoints for the developer conducting the test with a tester

- During the entirety of the usability test, it is prohibited to explain the functions of the application to the tester. Solely the idea behind the application will be given at the start of the test to ensure the user has a general idea of what he should expect. This will simulate the situation of a user interacting with the application for the first time.
- During the test, notes must be taken, especially of actions that are unexpected. As an aid, the tests may also be recorded (audio & video) and reexamined after the test is complete.
- After each round of usability testing, the results must be discussed as a team to draw the most relevant improvements that should be worked on in the next cycle.
- At least two rounds of usability testing must be absolved to test the initial concept and to ensure the improvements thereafter have the expected effect.

5.1.2 Usability Testing protocol

Before letting the user begin with the testing, some preparation needs to be done:

- Set up a device with a local instance of the application running.
- Explain the importance of testing and why it is done.
- Explain the procedure of the test.
- Describe to the tester, that the application is used to keep track of recipes and that it is food related to give them a general sense of what they are dealing with.
- Ask questions, to make the tester comfortable by going through the first set of questions. Some of the questions are not directly tied to the design but serve more of a purpose to relax the user and get them into a talkative mood.
- Afterwards, the testers must go through the scenarios and complete the objectives to the best of their abilities. It is important to emphasize that it is not the user

being tested, but the application itself and that there are no wrong ways to go about completing the scenarios.

- In the end, the final questions need to be discussed, of which the developer must take notes. If at any point the developer believes that a question was already answered during the scenarios or by an elaborate answer to a previous question, then the developer should refrain from asking the question as this would break the flow of the "interview".

Questions before the scenarios

These are the questions to ask before conducting the actual testing.

Question 1	How do you keep track of your recipes at home - if at all?
Question 2	How often do you cook?
Question 3	Do you use exact measurements when following a recipe or just eyeball it?
Question 4	For how many people do you normally cook?
Question 5	Do you like exploring different cuisines and trying new recipes?
Question 6	Have you ever cooked according to a recipe you found online?

Table 6: Usability testing - Questions before the scenarios

Scenarios

These are the testing scenarios. The user must complete them without any help.

Base Scenario	You just registered to Smart Eating because you want to keep a strict regiment on your micro nutrients and would like to expand your personal recipe collection.
Switch to dark-mode	You open the website and it's a little too bright, turn on the dark mode instead.
Modify nutrients	As the micro nutrients are so important you want to start by changing the biometric settings in your account.
Analyze recipe	To kick off the experience you wouldn't want a meal plan without your world famous plain fried egg recipe. Add the recipe to your collection.
Update recipe	You forgot to add the secret ingredient "Salt" to your recipe, find and edit the recipe accordingly.
Analyze recipe with photo	You remember that you have a picture of your mothers banana bread recipe and want to add this as well.
Check meal suggestions	Thinking about all these recipes you are getting hungry and want to check your meal suggestions so that you can go shopping for the groceries.
Update meal preferences	Lastly you notice the meals all seem quite heavy and wish to change your calorie intake settings since you are on a diet.

Table 7: Usability testing - Scenarios

Questions after the scenarios

The purpose of the final questions is to gather general feedback of the application.

Question 1	Is the app appealing to you and why?
Question 2	Did the color-scheme match the topic of cooking or food?
Question 3	Are the navbar-titles intuitive or do you have other ideas?
Question 4	Is the style modern or a little outdated?
Question 5	Is there a feature you think is missing?
Question 6	Would you change anything about how you add a recipe?
Question 7	Would you want to use this application?
Question 8	On a scale from 1 to 10 (not including 7) what would you rate the overall aesthetic?

Table 8: Usability testing - Questions after the scenarios

5.1.3 Feedback

Usability Testing #1

The first round of usability testing has provided helpful feedback about the general appearance but also smaller details that were disturbing the test subjects.

1. **Add description to analyze page**
   - Problem: The analyze page was a little confusing to most people as it wasn't clear that the page was meant to add recipes and instead it was seen as a completely different feature.
   - Solution: To ensure a better understanding when visiting the page for the first time, a short description will be added under the title of the page to describe what the page is for and how it is meant to be used.

2. Change color to something more warm and fresh

- Problem: The color seemed cold and very unrelated to food. Many test subjects have suggested a green color as it represents freshness and warmth.
- Solution: The color-theme will be adjusted as well as the logo to show a warmer shade of green.

3. Make title of recipe card prominent

- Problem: The recipe title is hard to read as it is written using a small font and a light grey color. The title is quite important as the images don't always suffice to recognise what the meal is.
- Solution: The title of the recipe will be changed to use a full black color instead of the light grey.

4. Add margins to some textfields

- Problem: The text fields in the suggestion do not show any margin between each other.
- Solution: The margin from the usersettings will be used as reference for the fields in the suggestions feature.

5. Remove analyzed ingredients after save/page switch

- Problem: When one clicks around the page after analysing a recipe the ingredients and instructions are still shown from the previous analyse attempt. This also happens when the save button is clicked however the save function was not yet fully implemented.
- Solution: If it is possible the fields should be cleared either when the page is changed or when it is re-opened. Once the save function has been fully implemented the save button should also clear the entire form of ingredients and steps.

6. Improve dark-theme toggle

- Problem: The toggle was moved to the user-settings and looks out of place. It wasn't recognised as a theme switch by some test subjects and should therefore be changed.
- Solution: There will be a new tab in the user-settings for the "Appearance" under which the option Darkmode with a toggle switch should be implemented. Having the function written out next to a toggle should be more understandable.

Usability Testing #2

In the second round of usability testing, no new bugs were found. The findings regarding styling were implemented and unanimously approved by the testers. Finding 5 *"Remove analyzed ingredients after save/page switch"* still persists for time management reasons. The Overall feedback from the usability testing round two is mostly positive:

- The green colors match the theme of freshness and warmth
- It is much clearer how to toggle the dark theme
- The overall design is more uniform

5.2 Low Fidelity Mockups

To get an idea of how the redesign of the website should be implemented, rough sketches of the user interface where created. They do not contain detailed designs, but rather serve as a point of reference for the development team. The final design might vary.

5.2.1 Navigation Structure

The existing navigation structure is unordered and confusing. To make it more user friendly, it should be restructured. Figure 6 shows the mockup of how the navigation should be structured.

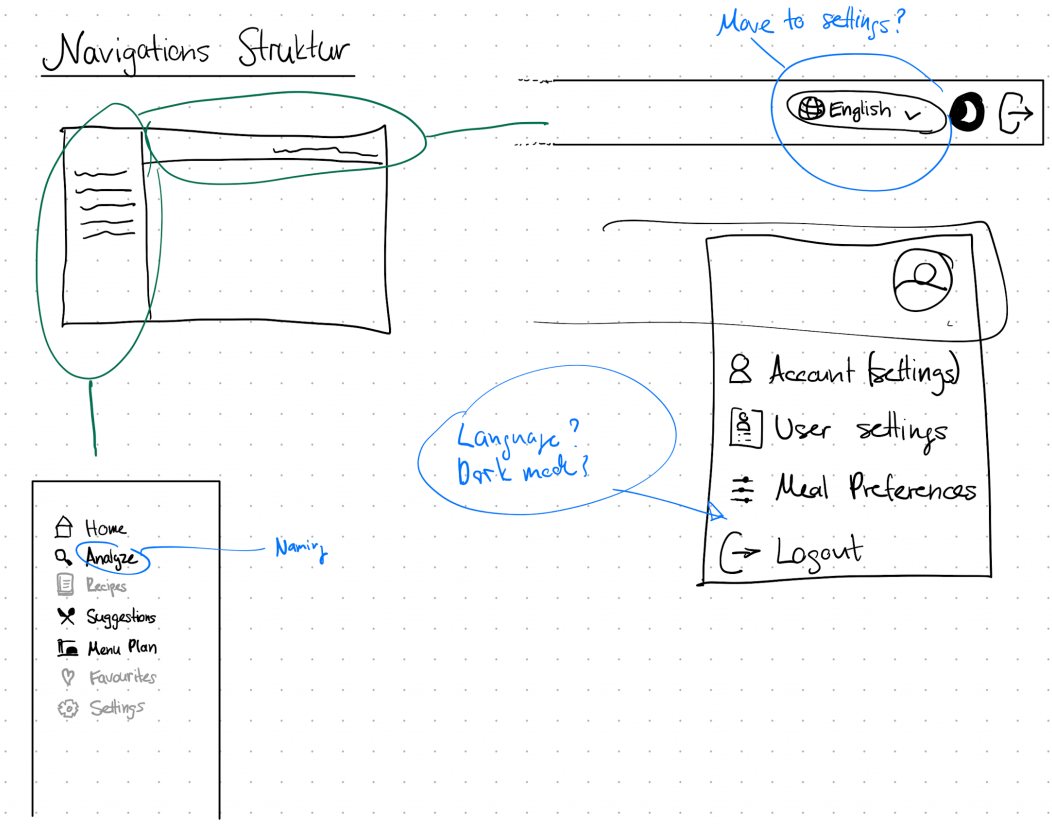


Figure 6: Mockup of navigation

In the drawer on the left side, all recipe-related pages are listed and ordered. These are the most frequently used pages of the application. The “Analyze” page is now listed at the top, directly below “Home”, as this serves as a key function of the application. Underneath “Menu Plan” is placed directly below “Suggestions” as the “Suggestions” page generates the menu plan.
All user-specific settings, which in most cases are configured by the user once, are moved to a user dropdown placed in the top-right corner of the navigation bar.
These changes should unclutter the navigation and place related items together.

5.2.2 Home and Recipe

The existing homepage only contains an image. As a homepage is the center of an application where all paths lead, it should contain (one of) the core functionalities of the app. As shown in Figure 7, the homepage should list all user-created recipes. A user should have the ability to directly edit and delete recipes. It was also discussed, that a text search for recipes could be implemented.

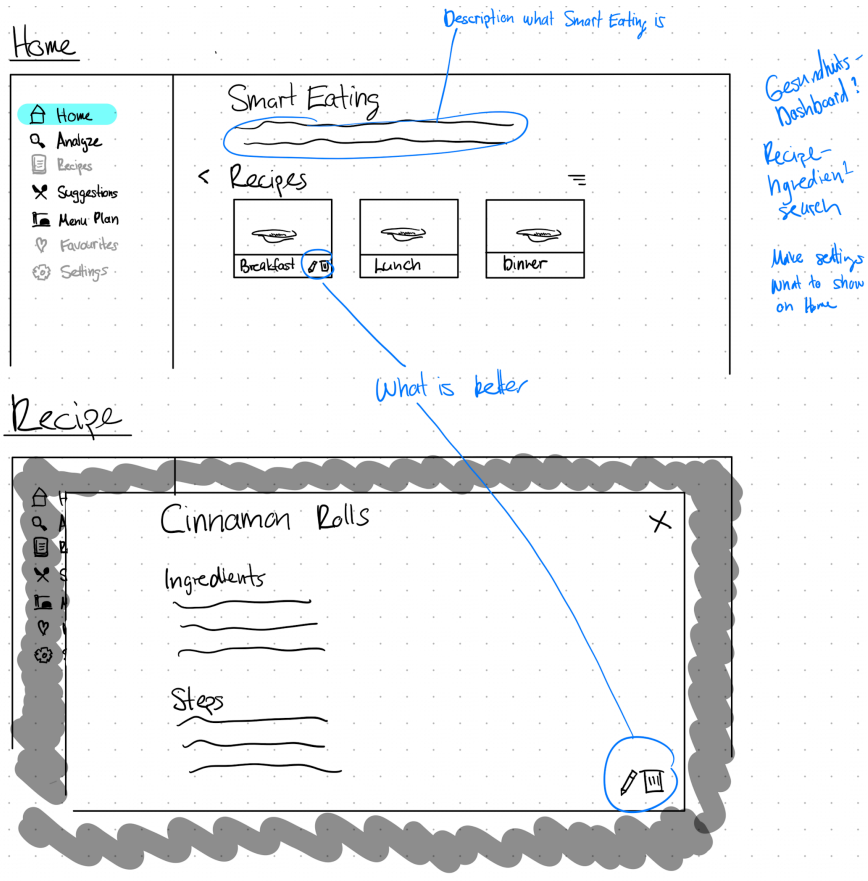


Figure 7: Mockup of home and recipe

5.2.3 Analyze

As is, the analyze page only allows a user to either directly enter text or upload an image. Once the recipe is analyzed by the AI, it only lets the user view the result. Because the AI is prone to errors, the user should be able to edit a recipe directly after generation. An optional UI feature that could be implemented, is to respectively make the “Analyze” and the “Result” part foldable as depicted in Figure 8.

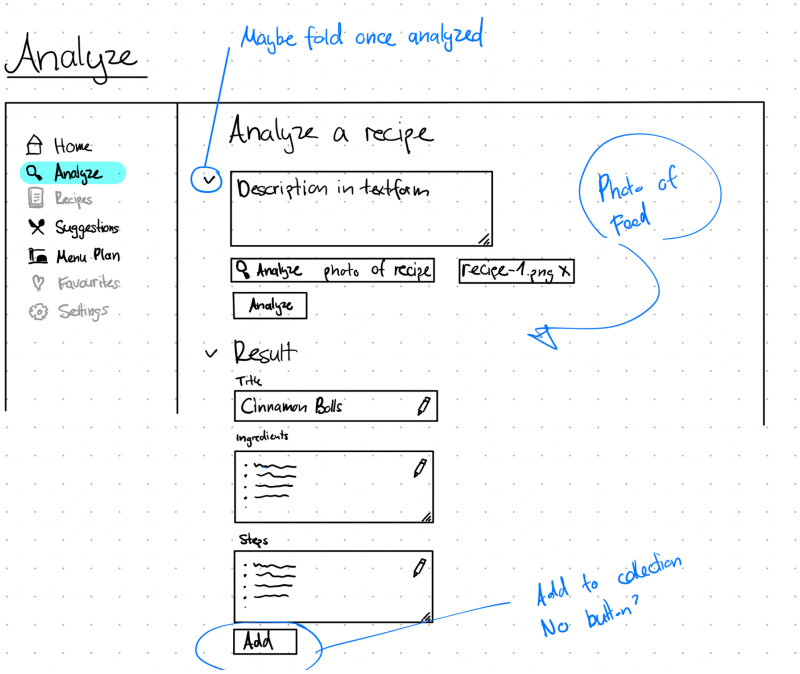


Figure 8: Mockup of analyze screen

5.2.4 Meal Preferences

The meal preferences pretty much remain as before. One minor change is, that the eating habits are ordered horizontally and the checkboxes are replaced with clickable toggle buttons. Alternatives for how the nutrition settings could be designed were discussed, but none of them were deemed more user-friendly.

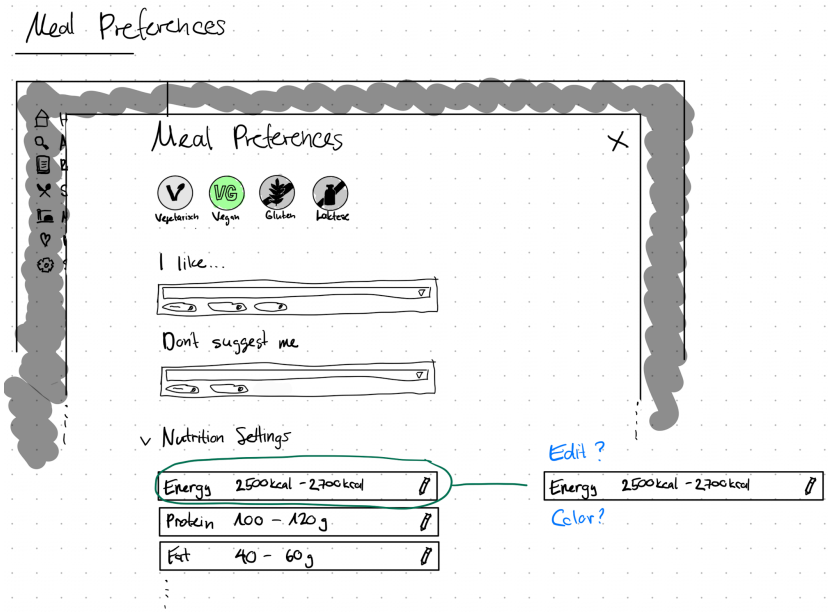


Figure 9: Mockup of meal preferences

5.2.5 Admin Panel

Figure 10 shows the entirely new admin panel page. It should allow an admin to create, read, update and delete users. By clicking the edit icon on the “Name” and “Email” fields, it converts into an editable text field and an admin can update the info. By clicking the edit icon on a “Roles” field, a drop-down appears which allows an admin to either assign or revoke the “Admin” role of a user.

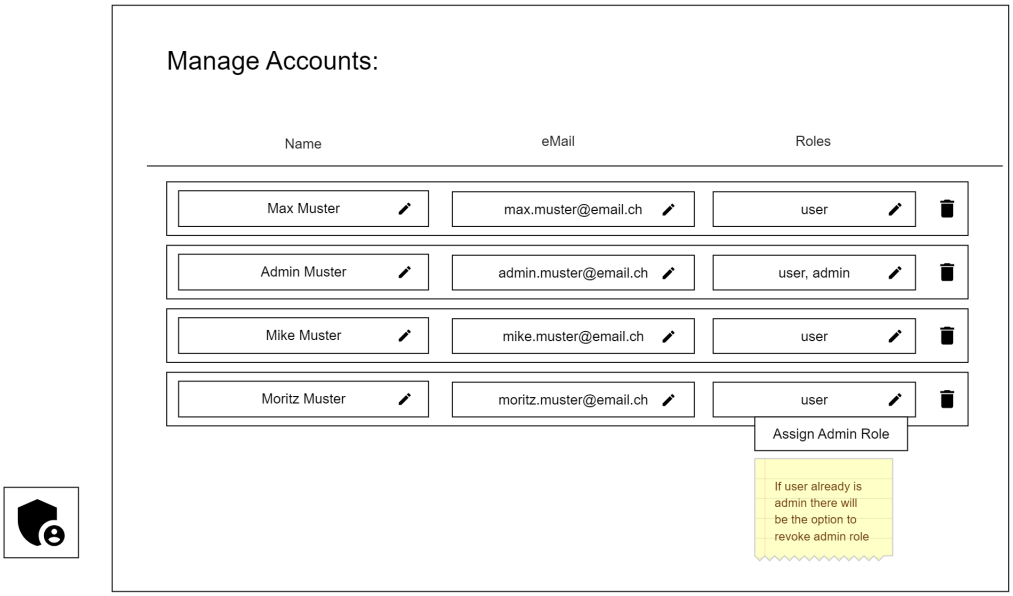


Figure 10: Mockup of admin page

5.2.6 Sharing functionality

Even though the possibility to share recipes is not yet implemented, the mockups should help implement this feature in the future. The following mockups show what the share function should look like. Figure 11 shows how the user can find the option to share any recipes in the action menu.

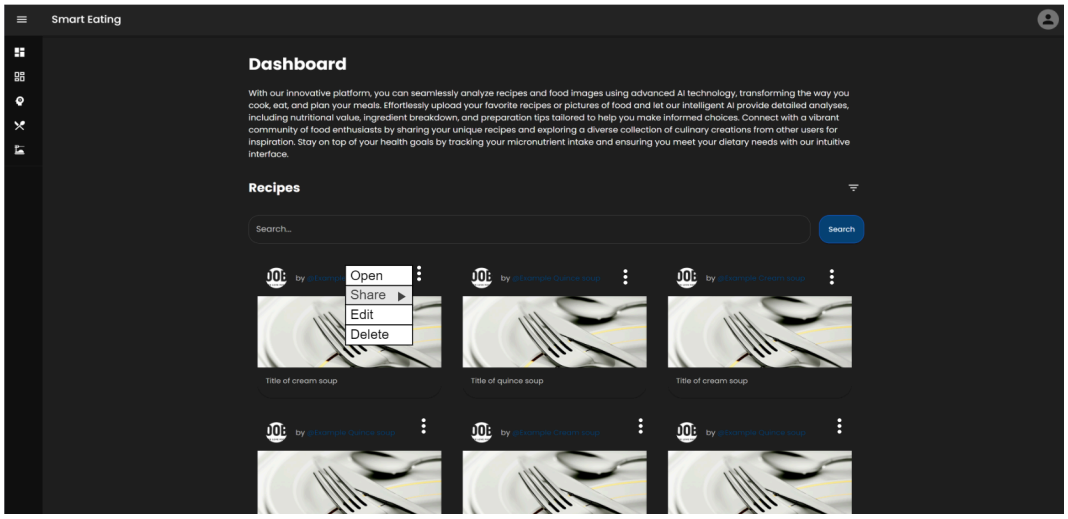


Figure 11: Mockup of share menu

Selecting this option leads the user to the popup displayed in Figure 12. There he is able to look for other users by searching for their name or e-mail address. Pressing the share button sends a share request to the selected person.

Figure 12: Mockup of share popup

Figure 13 shows how the recipient of the shared recipe is notified about incoming messages. The bell icon which is located next to the profile gets a red notification badge displaying the number of incoming recipes.

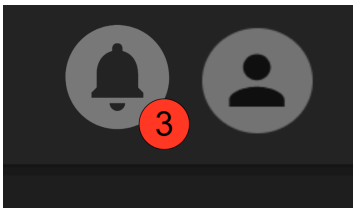


Figure 13: Mockup of inbox notification

Pressing the bell icon opens the inbox as illustrated in Figure 14. The user gets an overview of all the share requests and can save them to his own recipes with the floppy disk icon or delete the request by clicking the garbage bin icon.

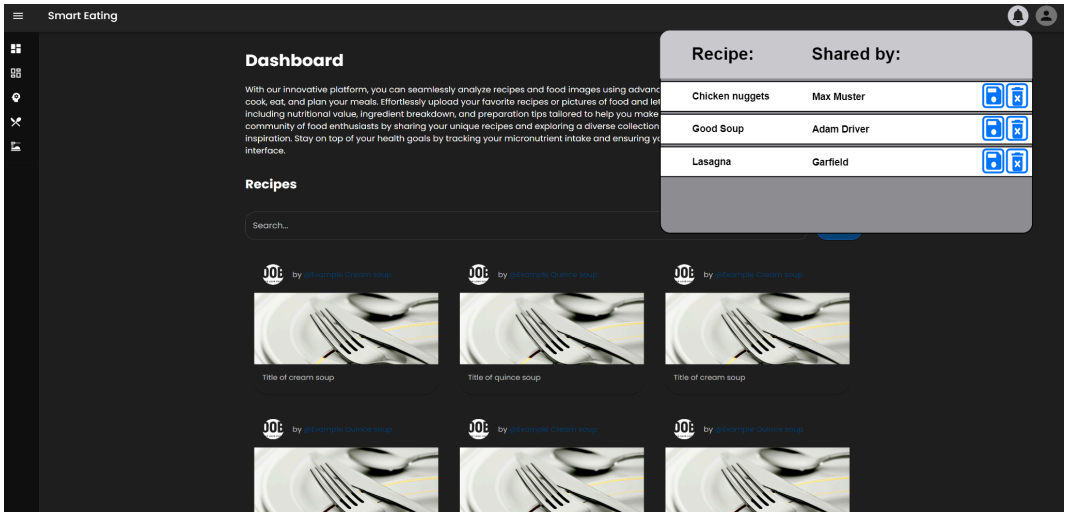


Figure 14: Mockup of inbox

5.3 Process to update DB schema

The general structure for us to interact with the database in the scope of this project looks as follows:

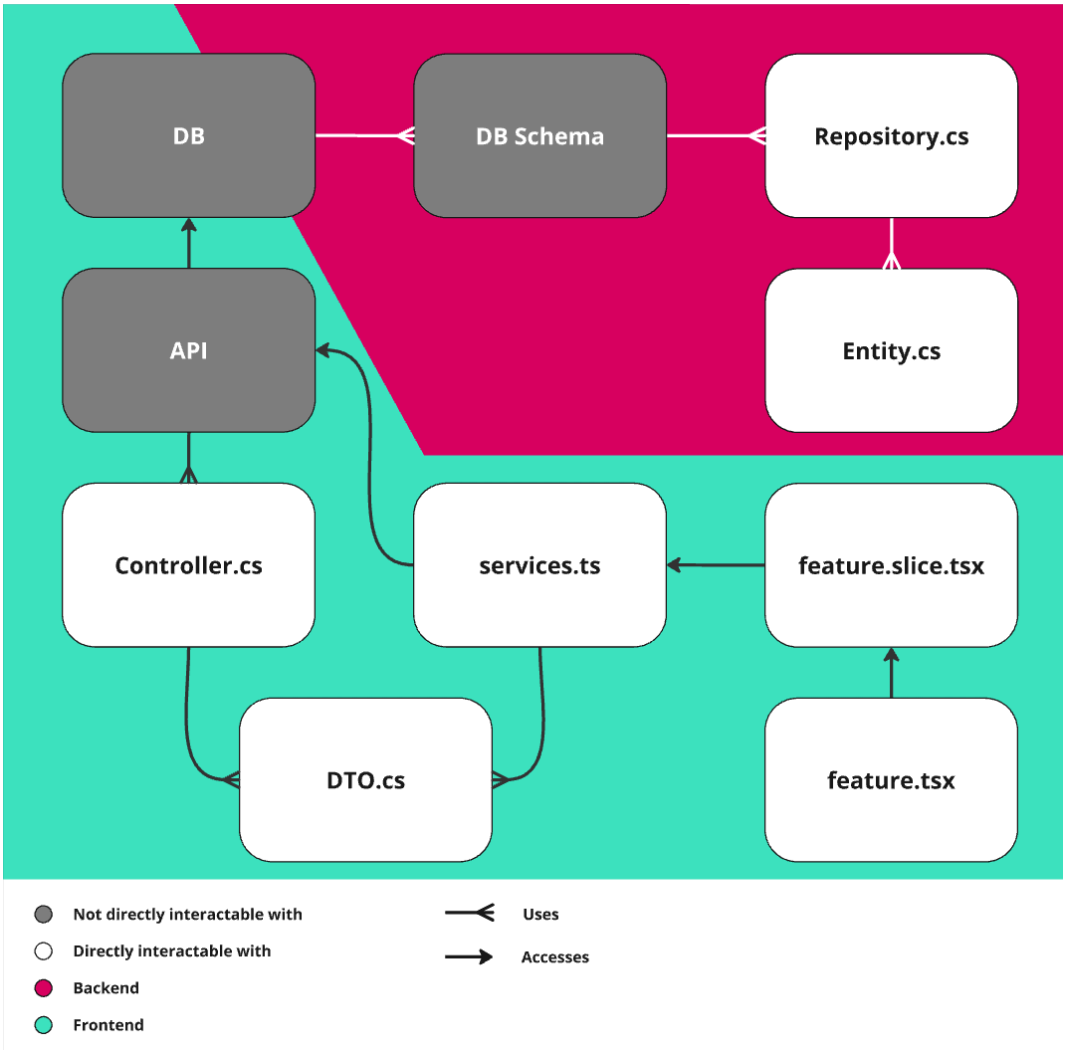


Figure 15: Database access schema

To further describe the works of the schema, the process of adding a username to the user is explained. To add the username field to the database itself, the username is added as a variable to the User.cs (Entity). The UserRepository.cs does not require adaptation as only a new variable is added. If we need to add a query for fetching users by username, we do so within the UserRepository.cs file.

Next, the username field is also added to the UserDto.cs. The UserDto, like the User.cs, describes what the user object is and has. However, the entity generates the database,

while the DTO represents the object structure in the frontend and is what is used in our service and controller functions. Indirectly, the slices also use DTOs because the service.ts returns them from its functions. Just like with the UserRepository.cs, the frontend counterpart UserController.cs also requires no changes for the same reason. Similarly, since no new functions and therefore no API changes are made, the service.ts does not need to be extended. If a new function is added to both the UserRepository.cs and the UserController.cs, the service.ts accesses this new function through the API, and this function can be accessed by the user.slice.tsx through the service.ts, completing the cycle from the backend to the frontend.

5.4 Process to update translation

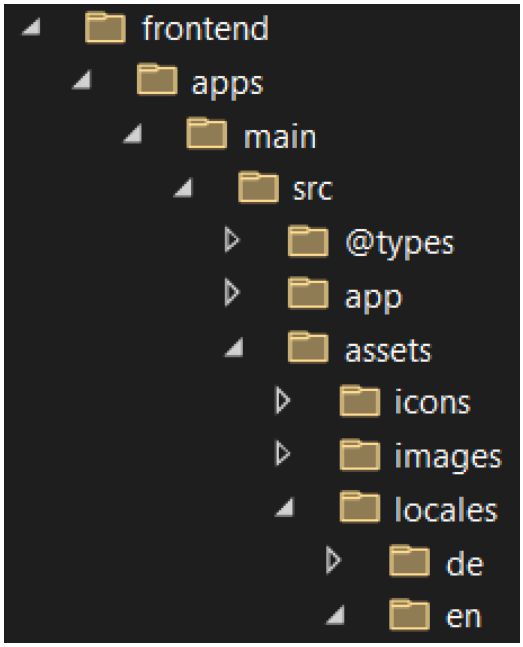


Figure 16: File structure for translations

When adding a new feature or updating any text in an existing one, the translations should always be created or updated along with it. In order to create new translations one must first create a new translation schema in the smarteating_webapp in the “locales/” folder depicted on the left. newschema.schema.json must be created according to the existing files - containing a schema reference, description, type properties and a *required* field. Next a newschema.json must be created in each subfolder which each represent a language. The translations can be written according to the existing json files (the english ones must be provided before moving on to the next step). Now the new schema must be added to the package.json located directly in the frontend folder. In the package.json under “scripts” there is an item “locales-schema” here the line “ && json-schema-generator ./apps/main/src/assets/locales/en/newschema.json -o ./apps/main/src/assets/locales/newschema.schema.json –force“. Lastly the corresponding data can be generated using the command “npm run locales” in the terminal. After that step the translations can be adjusted and used.

5.5 User Management

Initially, there is no possibility to manage users. To provide user management, an admin panel is introduced. Because not everyone should be able to manage users, a new entity “Role” is introduced. Users should have at least one role. Therefore, the database schema needs to be updated as subsequently shown in Figure 17.

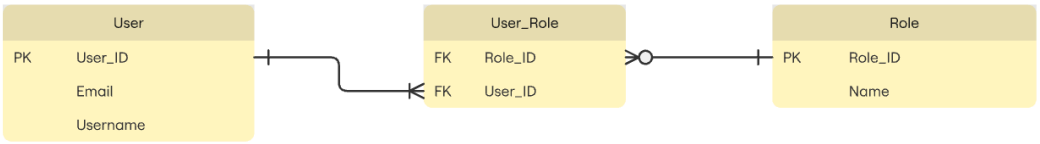


Figure 17: database schema update

5.5.1 Schema update

To update the schema in the application several modules in the web application backend are updated. Figure 18 shows the classes and interfaces requiring modification.

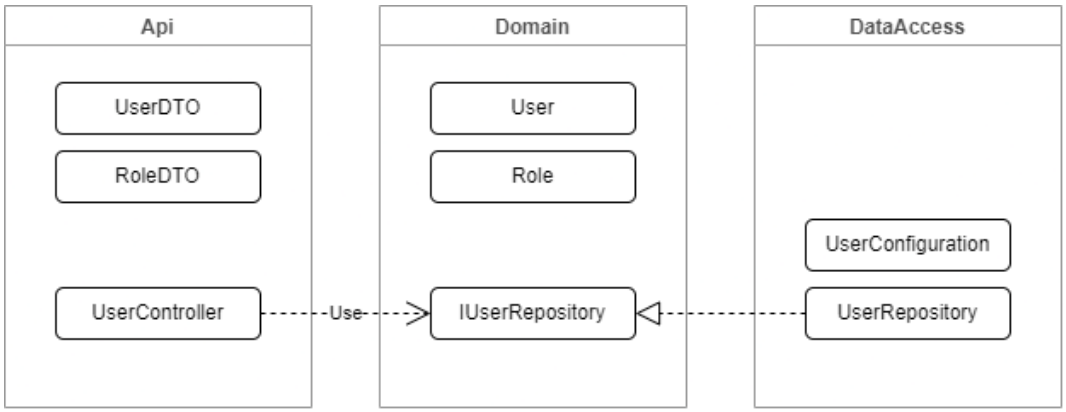


Figure 18: Backend structure for adding role entity

The API component acts as the interface between the backend and frontend. It contains DTO’s and Controllers. These are used to generate an API with NSwag, with which the frontend can access the database.

The Domain component contains data classes which are used to generate the database schema using the Entity Framework. The repository interfaces define the operations which can be performed on the schema.

The DataAccess layer serves as the interface between the backend and the database. The

repositories define how the operations on the database should be done. The Configuration allows further refinement of relations between entities. As the relation between “User” and “Role” is a many-to-many relation, an intermediate table is generated. This relation requires further refinement, which is depicted below:

```
builder.HasMany(x => x.Roles)
    .WithMany(s => s.Users)
    .UsingEntity<Dictionary<string, object>>(
        "UserRole",
        ur => ur.HasOne<Role>()
              .WithMany()
              .HasForeignKey("RoleId")
              .OnDelete(DeleteBehavior.Cascade),
        ur => ur.HasOne<User>()
              .WithMany()
              .HasForeignKey("UserId")
              .OnDelete(DeleteBehavior.Cascade));
```

The refinement defines, that when a user or role gets deleted, all corresponding rows in the intermediate table corresponding will also be deleted.

To persist the previously described schema evolution, a migration is done to update the SmartEatingDbContext. This class acts as the data access layer to interact with the database for CRUD operations.

5.5.2 Admin panel implementation

To be able to use the updated schema, the API in the frontend needs to be updated. This is done by running `npm run nswag`.

The admin panel is then implemented according to the mockups with one modification. The OAuth authentication uses the user email to check if he already exists in our database. Therefore an admin is not able to update the email.

To only allow admins to access the admin panel, guards are added to the routing in app.tsx:

```
export function App() {
  ...
  const isAdmin = useAppSelector((s) => s.user.isAdmin);
  ...
  return (
    <ErrorBoundary>
       <Routes>
```

```
          ...
          <Route>
            path="admin"
            element={
            <ErrorBoundary>
              {isLoggedIn && settings && isAdmin && (
                <Suspense>
                  <AdminPanel />
                </Suspense>
              )}
            </ErrorBoundary>
          }>
          </Route>
          ...
        </Routes>
      </ErrorBoundary>
  );
}
```

5.6 Edit recipes

Using the analyze function to get new recipes returns a recipe analyzed by AI. However the AI can make mistakes or the user could want to individualize a recipe for another reason. The analyzed recipe that is displayed needs to be editable by the user for this to be possible. To make the user experience as good as possible an approach was chosen, where any changes made to the recipe would immediately be saved in the `analyzedRecipe` object, which is later stored. To make the recipe easy to edit, the ingredients and instructions are displayed in the same format as in the data transfer object (DTO). Specifically, each ingredient is split into three fields: quantity, name, and unit of measurement (if existent). Then they are displayed in separate editable boxes. Any changes to these boxes would immediately activate a handler function. The handler works by mapping the modified ingredient with the following code:

```
analyzedRecipe?.ingredients?.map((ingredient, i) =>
```

When a user edits a specific field, the corresponding property of the ingredient is updated with the new value entered in the input box. Any updates are displayed and stored in real-time. Here the biggest challenge was ensuring that the “unit” box would not disappear when the entire content of the box was deleted in the progress of editing. Since the “unit” box doesn’t get displayed if there is no unit this would result in it vanishing and therefore

the user would be unable to edit them any more. In order to solve this the handler would replace the empty string with an placeholder:

```
name: e.target.value.trim() === "" ? "-" : e.target.value,
```

This placeholder would later be interpreted in the text box to stop it from disappearing even when it's empty:

```
value={ingredient.unit?.name === "-" ? "" : (ingredient.unit?.name ?? '')}
```

5.7 Save recipe

After analyzing, the recipes need to be saved to the database so that the user can access them in the future. This is crucial for the application to be fully usable. The recipes need to be stored in the database so they can be listed in the dashboard or shared across the application. These features are critical to delivering a user-friendly and comprehensive experience for the end user. To accomplish this, the analyzed recipe must first be converted into a format that can be stored in the database. The required structure for that already exists in the Smart Eating application. The following function that is located in own-recipes.slice.ts indicates that the save functionality expects a SaveRecipeRequestDto which is then sent to the controller. This means that the analyzed recipe must be mapped to match the structure defined by this DTO.

```
export const saveRecipe = createAsyncAppThunk('own-recipes/save', async
(request: SaveRecipeRequestDto, { extra }) => {
  console.log(request);
  return await extra.clients.userRecipeClient.save(request);
});
```

This approach, however would turn out to be a time consuming mistake. The issue emerged because the SaveRecipeRequestDto includes an AnalyseRecipeRequestDto. The AnalyseRecipeRequestDto has an Id property which is currently not being assigned. To avoid this problem a new object the SaveRecipeRequest2Dto is created, which does not have the AnalyseRecipeRequestDto property. To use a new object in the UserRecipeController some auto-generated code needs to be regenerated. The code generation is done with the Swagger API. This resulted in new problems because the usage of NSwag required the entire project to be upgraded from .Net7 to .Net9. After resolving the dependency issues and the progress of this feature already being heavily behind the time schedule even more problems arose. Because the currently saved recipes are all from Fooby, new entities have to be created in order to save the analyzed recipes. But at this point is where

the time ran out and the save recipes feature could not be implemented in the scope of this project. This has some widespread influence on the project since additional features that depend on the save function are already planned. It is for sure the feature that needs to be implemented next in order to present Smart Eating to a wider audience.

5.8 Share functionality

The sharing functionality is dependent on recipes being saved. As saving recipes is not implemented due to problems described in Section 5.7, the sharing functionality is not realized. Nevertheless planning for this functionality exists in form of mockups which can be found in Section 5.2.6.

5.9 OAuth GitLab login

The entire flow of the login-process can be observed more visually in the Figure 5 under Section 4.4

After a discussion among the team the decision was made to implement this feature as simple as possible. The conclusion is to have one button “login with GitLab” added to the already existing login page that will either log in the user if an account already exists or create and then login the user to the new account. First the existing functions and architecture regarding the login and user creation are inspected. Within the service.ts in the frontend there is an AuthClient class that already provides the functions loginDev and createUser. Furthermore these functions are fully functional and connected to the back-end. The users are stored in the local SQLite database further described in Section 4.3.2. The loginDev function throws an error if the user login request fails. We decided to utilise this functionality to our advantage as no other functionality is provided in the backend regarding authentication. For this reason a login event is performed when the “login with GitLab” button is pressed and the user is logged in if the account already exists. However should an error be thrown, the error will be caught and a createUser event is dispatched. This part is also within a try-catch sequence to ensure that the login event that follows the createUser (which is contained within the finally-block) is only dispatched after the createUser call has finished:

```
try {
  dispatch(setProfilePicture(userInfo.avatar_url));
  await dispatch(loginAsync(userInfo.email)).unwrap();
} catch (error) {
  try{
    await dispatch(createUserAsync(userInfo.email)).unwrap();
  } catch (error) {
    console.error("An error occurred while creating the user: ", error);
```

```
  } finally {
    dispatch(loginAsync(userInfo.email)).unwrap();
  }
}
```

Lastly the most important part was to connect the application with GitLabs OAuth service. It is very important that this was implemented for the forked branch only and must be adjusted for the live project (this is also mentioned in the outlook under Section 7.2). In order to register an application to be used with GitLabs OAuth service one must log in to GitLab, click on the user icon and open preferences. In the "User settings" navbar on the left hand side under Applications a table will open up where a new application can be added. When adding a new application only the name (for your own reference), the callback url and the access scopes must be chosen. The standard url for a callback link is /auth/callback. The access scopes selected for this feature are read_user, openid and profile. When the "application" is saved GitLab generates a clientId as well as a client secret which are both needed to retrieve the access token.

When clicking on the login button first a call is performed to GitLab:

```
window.location.href = 'https://gitlab.ost.ch/oauth/authorize?client_id=
CLIENT_ID&redirect_uri=REDIRECT_URL&response_type=code&scope=read_user';
```

The clientId must be provided along with the redirect_url and the scope which in this case is read_user. The user is directed to a GitLab login page where the permission for the application to read the user details is requested after the user logs in. Once the permission is granted and the user is logged in GitLab will automatically redirect to the redirect_url.

In the Callback.tsx the authorization code is received and a request for an access token is sent to GitLab. The urlParams include the client_id, client_secret, code (authorization code), grant_type (in this case 'authorization_code') as well as the redirect_url again.

```
const response = await fetch('https://gitlab.ost.ch/oauth/token', { ...
```

Lastly once the token is recieved one more request is performed to GitLab, this time accessing all the users information through a simple fetch request adding only the acces-token as an authorization header.

```
const response = await fetch('https://gitlab.ost.ch/api/v4/user', { ...
```

Utilizing the user data the login/createUser request is performed as described above.

6 Results

In the following, the functional and non-functional requirements are verified by comparing them to our results.

6.1 Releases

Two releases are planned in the milestones under Section 8.1.5, the alpha and the beta release. The alpha release planned to include all features except sharing recipes. This feature is left for the beta release. As the save feature was delayed a decision was made that the the alpha version should not yet be released without the save feature. As the project progressed and the feature was further delayed another decision was made to exclude the alpha release and instead release only the beta version as soon as the save feature was implemented. By the end of the projec,t the beta release omitted the save feature and released without it instead. Therefore the FRs and NFRs are only verified once which is after the beta release.

6.2 Verifying FRs

The functional requirements are verified by checking if the previously defined use cases can be successfully executed. The following use cases are implemented in the final product:

- **UC2:** The authentication is now possible with OAuth. The user only needs an OST GitLab account to be able to access the Smart Eating webapp.

- **UC4:** An admin page has been implemented which is only visible to users with the role admin. There they can get an overview of all the users and grant or revoke admin permissions.

Following is the use case, that is implemented partially:

- **UC1:** The user is able to edit any recipes that are analyzed in the analyze page. This is currently the only way to add any recipes. Because the recipes do not get saved it is not possible to read, update or delete them.

Below is the use case that was not implemented, as it is dependent on **UC1**:

- **UC3:** It is not possible to share recipes because they are not saved to the database. However the implementation was planned for with mockups which describe how the share feature can be implemented.

6.3 Verifying NFRs

The Non-functional requirements which have been defined in Section 3.2 are tested after every release. The testing criteria is defined separately for each NFR starting with NFR-1 at Section 3.2.0.1 up to NFR-5 at Section 3.2.0.5. As the project is progressing, it is clear that only one release will be done by the end of the 6th or 7th iteration.

- **NFR-1 - Confirmed:** During the usability tests the users have shown that they can complete the scenarios as required with 2 or less mistakes. There was some confusion around the saving of the recipe as it is not implemented and the therefore also the updating of a recipe scenario was not possible to complete.

- **NFR-2 - Confirmed:** As per the verification process description erroneous data was entered wherever possible without any errors popping up for the user to see nor did the application crash completely. Therefore this NFR is also approved.

- **NFR-3 - Declined:** This NFR has been declined as the feature to save a recipe has not been implemented.

- **NFR-4 - Confirmed:** As described in the verification of NFR-2, during testing the application did not crash and kept running even when erroneous inputs did not allow the application to perform certain functions.

- **NFR-5 - Confirmed:** During the entire project the usability tests were run twice on multiple subjects. The first time the result of the question regarding the overall appearance averaged at a rating of 6.7. The second time however the rating averaged an entire point higher at 7.7 hereby approving the NFR and leading to it's confirmed status.

6.4 Final product

This project significantly transforms the existing Smart Eating application, although not all planned features could be implemented. Before the realization of this project, the Smart Eating application was mostly functional and not much time has been spent on user experience. This is achieved by many different improvements like overworking the components and giving them a more modern design. Putting short descriptions on the

top of chosen pages like the dashboard helps new users get an overview of the application and all its features.

6.4.1 Login with OAuth

Users are now able to log into the Smart Eating application using the GitLab OAuth method. This requires the user to have an OST GitLab account. The previous login with email and password still exists and is planned to be replaced by OAuth in the future.

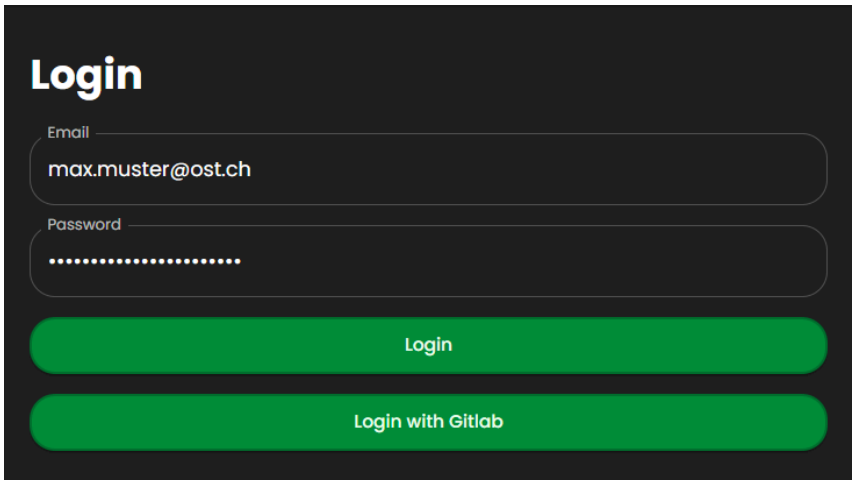


Figure 19: login with OAuth

6.4.2 Dashboard

The dashboard is an alternative to the current landing page which is currently only filled with the Smart Eating logo. In the dashboard the user is able to see his saved recipes. It can also include a search and filter function. On top of the dashboard is a short introduction to Smart Eating, which aims to introduce the user to the application.

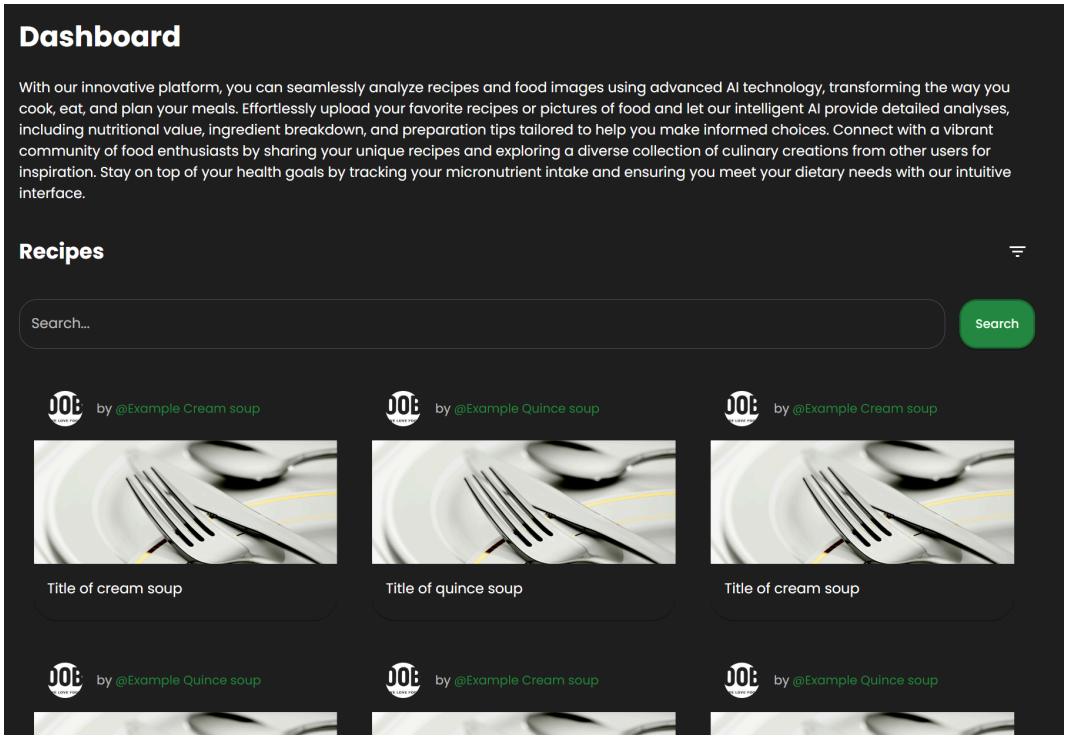


Figure 20: dashboard

6.4.3 Admin panel

The admin panel is located within the navigation structure on the left side but is only available and visible to users with the admin role. There they can see all users and edit their username, give or revoke the admin role and delete the user. Currently all users that log onto the Smart Eating application are automatically granted the admin role. This is implemented in this way for developing purposes and should be changed before the application is introduced to a public audience.

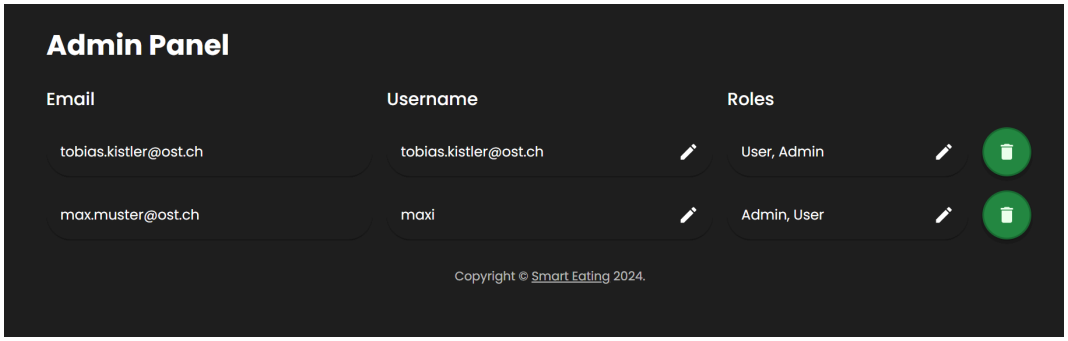


Figure 21: admin-panel

6.4.4 Analyze

The analyze page displays the ingredients, their quantity and if existent the unit as well as the instructions. Those are split up in different text fields that the user can easily edit, which are also ensuring that the page remains clear and user-friendly. All changes are immediately included so there is no need to press any save button unless the user wants to save the recipe to the database.

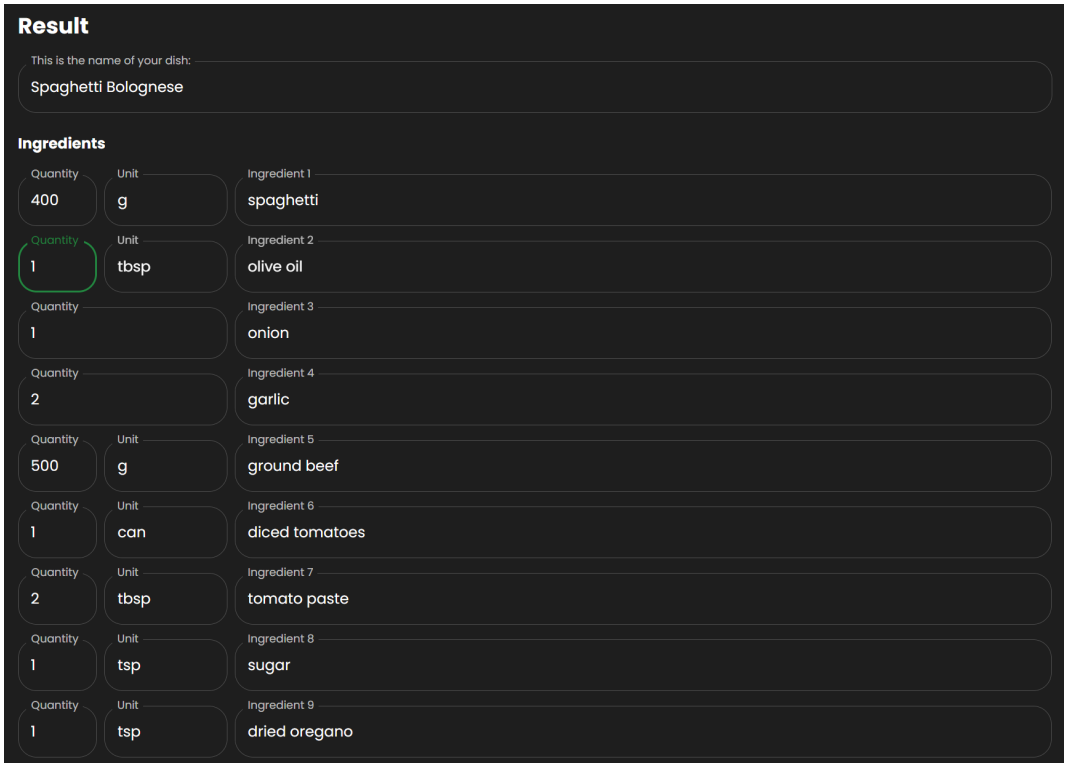


Figure 22: analyze page

6.4.5 Meal preference icons

In the meal preference section not only the overall design has been improved. The previous checkboxes for vegan, vegetarian, lactose free and gluten free have been replaced by stylish icons.

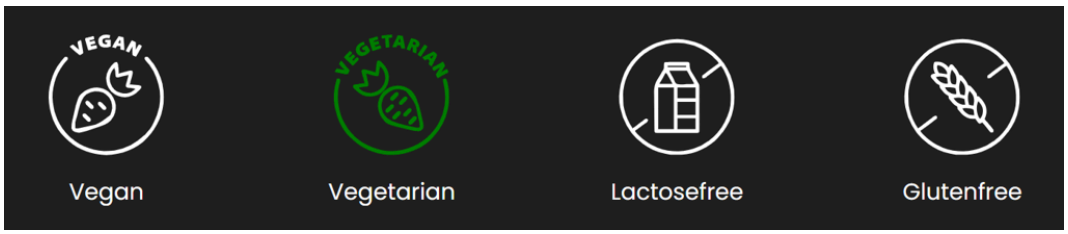


Figure 23: icons (https://www.flaticon.com/authors/revolutionizzed-1)

7 Conclusion

The conclusion evaluates the results and reflects upon them. Then possible progressions of the application are discussed.

7.1 Result reflection and achievements

The goal of this application was to extend the existing application with new features. Furthermore, the user experience should be improved by implementing a more intuitive and appealing. The general idea was to give it features, such that the application is getting closer to being used in real-world scenarios.

We were able to implement the authentication and user management as requested. Furthermore, we restructured and restyled the application, which should allow the user to navigate and use it more easily. This was verified by testers with usability testing, which confirms our opinion that the UX was improved.

While we were able to implement the editing of recipes after generation, we were not able to implement the saving process. There are two reasons for that, on one hand the architecture is utterly complex in a way that finding out how this should be implemented is difficult. Here it might be good to consider that the lack of documentation wasn't helping as well. In the end time pressure meant that we couldn't implement the saving of recipes. And as the feature "Sharing of recipes" is dependent on recipes being saved, we also weren't able to implement it.

Despite not being able to fulfill all requirements, we are confident that our implementation improves the usability with the implemented features and provides a more intuitive user experience.

7.2 Outlook

The Smart Eating project still has room for improvement. In the course of the project, we had several ideas how the application could be further improved. The following list gives an overview of the feature ideas:

- As described in Section 5.9 the GitLab-OAuth must be adjusted for the actual project. In Section 5.9 the process to make this change is described in more detail for additional convenience.
- Generate a shopping list from the recipes in the menuplan

- Search on the dashboard, which allows the user to search for fooby recipes and his own recipes. This could also include searching by recipe ingredients.
- A filter for the search that lets the user filter by recipe type (fooby, own) and favourites.
- The dashboard could be augmented with a health section, which compares the current eating habits with the personal nutritional settings.
- Implement sharing of fooby recipes between users.

8 Project- and Time management

8.1 Planning

The planning describes how the project is managed and planned.

8.1.1 Methodology

Our project is being run with the OST-made Scrum+, a combination of Rational Unified Process (RUP) and Scrum. Doing so, helps to keep the long term goals in sight, while simultaneously being able to make changes and remain agile within the sprints.

8.1.2 Roles and Responsibilities

The roles are mostly inline with Scums traditional roles with one exception. The decision was made to fuse Scrum Master and Project Manager into one general, organizational role, due to the size of the team and the project.

Note that these assignments do not mean that the persons sole job is bound to their role, but rather that the person is the one responsible for the overall work pertained to the role and is always in the know of its current status. Different members of the team will help and assist one another as needed.

- **Scrum-Master + Project Manager:** Simon Peier
  - Responsibility: Leading Scrum Meetings, Writing Meeting Minutes
- **Product Owner:** Everyone
- **Frontend:** Tobias Kistler
  - Responsibility: Overview and general responsibility over the frontend
- **Backend:** Simon Peier
  - Responsibility: Overview and general responsibility over the backend
- **User Experience (UX):** Leonardo Ravani
  - Responsibility: Overview and general responsibility of the overall User Experience

8.1.3 Meetings

Since the project is working with Scrum+ most of the meetings will be related to it.

- Sprint Review/Retro: Every second Monday 10:00 - 11:00
- Sprint Planning: Every second Monday 11:00 - 12:00
- Weekly Sync: Every Thursday 10:00 - 10:30
- Meetings with coaches: Schedule when needed

8.1.4 Long-Term Plan

The following graphic depicts the Scrum+ long-term plan. These tasks and features are estimated roughly and only serve as a general guideline, meaning they may be due to change during the course of the project. For this reason, it was decided to build in a buffer, should any delays occur.
The light blue fields in Figure 24 depict the planned time, the dark blue fields depict the time when tasks are actually worked on.

LONG-TERM PLAN

PROJECT TITLE	PhotoChef
PROJECT MANAGER	Simon Peier
DATE	19-Dec-24

		INCEPTION		ELABORATION		CONSTRUCTION								TRANSITION	
TASK ID	TASK TITLE	ITERATION 1		ITERATION 2		ITERATION 3		ITERATION 4		ITERATION 5		ITERATION 6		ITERATION 7	
		KW38	KW39	KW40	KW41	KW42	KW43	KW44	KW45	KW46	KW47	KW48	KW49	KW50	KW51
1	Documentation														
2	Project Management														
3	Project Planning														
4	Requirements Engineering														
5	Software Architecture														
6	Setup environment														
7	Feature: Add and edit recipes														
8	Feature: Authentication and User Management														
9	Feature: Share recipes														
10	Feature: Improve User Experience														
11	Testing														
12	Buffer / Additional Features														

↓ Milestone 1: Initial Project Setup

↓ Milestone 2: End of Elaboration

↓ Milestone 3: Alpha

↓ Milestone 4: Beta

↓ Milestone 5: Final Submission

Figure 24: Long-term plan

8.1.5 Milestones

The project is divided into milestones to show how and when major activities and requirements are achieved.

Milestone 1: Initial Project Setup

The goal of the first milestone is to create an initial plan for the project

Planned Deliverables

- Define and assign roles
- Plan necessary meetings
- Create long and short-term plan
- Define tooling
- Identify and assess risks

Milestone 2: End of Elaboration

This milestone focuses on the elaboration of appropriate functional and non-functional requirements of the project.

Planned Deliverables

- Define functional requirements
- Define non functional requirements
- Planning of these requirements
- Define quality measures
- Create low fidelity mockups

Milestone 3: Alpha

With this milestone, the aim is to complete all required features except the sharing of recipes, such that it is possible to conduct usability testing

Planned Deliverables

- Alpha version of Photochef extension

Milestone 4: Beta

The beta version is a refined version built upon feedback and insights from the alpha release. It also includes the sharing of recipes, thus containing all required features.

Planned Deliverables

- Usability Testing
- Beta version of Photochef extension

Milestone 5: Final Submission

With this milestone, the final version of the application is delivered. This version should not contain any bugs introduced by us. If time permits it, additional features, such as a shopping list, may be implemented.

Planned Deliverables

- Final version of the Photochef extension
- Finalized documentation

8.1.6 Short-Term Plan

Since it was have chosen to work agile, there are iterations of two weeks. No complete short-term plan will be created initially. Instead, every iteration short-term planning will conduct be conducted. This includes

- splitting up epics into smaller tasks
- creating new tasks if required (e.g. if a bug occurs)
- estimating time effort of tasks using the Planning Poker method
- distributing issues to team members

All information about how the iterations are planned and executed and the respective tasks can be found on Jira (Link).

8.2 Tooling

This section describes the used tools and how they are applied.

8.2.1 Documentation

Our documentation is built with Typst. For ease of use, it was decided to use the online editor as this allows simultaneous editing and everyone always has the newest version. The structure of the documentation is based on the SEProject template from OST.[1]

[1] https://gitlab.ost.ch/SEProj/latex-documentation-template

8.2.2 Communication

The main channel of communication is MS Teams, as it offers a wide range of features and allows to easily write with each other, share documents and have meetings all at the same place.

8.2.3 Code

The version control of the code is managed with Git. In consultation with our supervisor, our own fork of the Photochef project was created. (Link).

8.2.4 Tracking

To track our issues, sprints and working hours Jira is used (Link). Jira is a project management tool that allows us to plan, track, and manage the project.

8.2.5 Workflow

The workflow defines, what stages all tickets will go through. Initially its status is ”To Do”. Then the ticket is assigned to someone and will be moved to ”In Progress” when they start working on it. After the work is done it will be moved to ”In Review” and assigned someone else to review the work. If the review is successful, it will be marked as reviewed (and moved to status “Reviewed”). Should any manual testing be necessary, the issue will be moved to “Testing”. Should any error occur during testing, it will be moved back to “In Progress”. Otherwise the issue can be moved to “Done”.

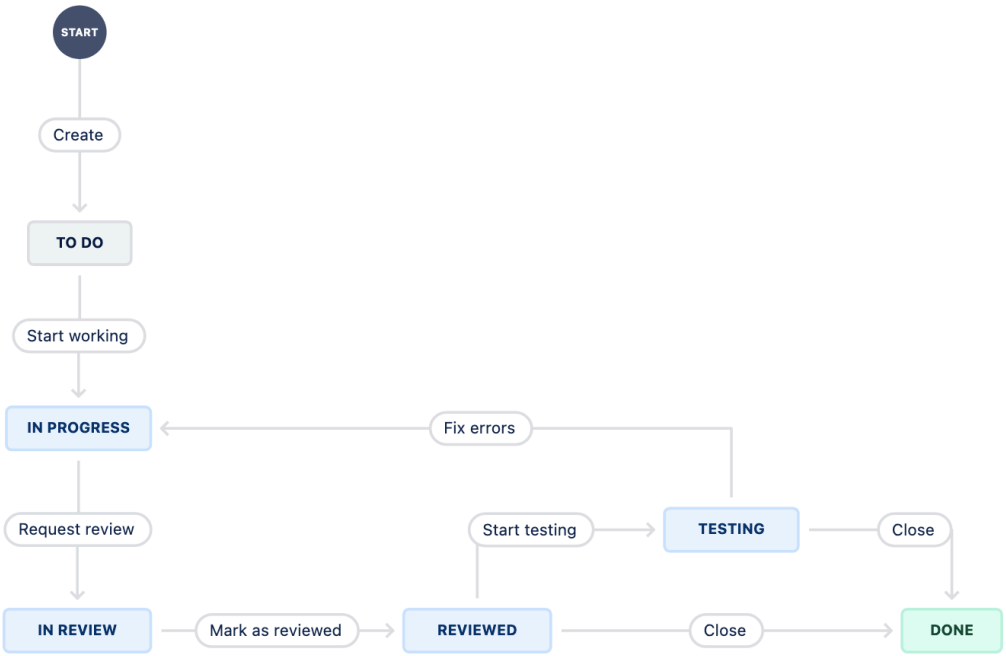


Figure 25: Jira Workflow

8.3 Quality Measures

Outlined below are the quality measures. On one hand they describe how to ensure good code quality and to keep the application as error-free as possible. On the other hand the measures also describe how to ensure a smooth development process.

8.3.1 Git Workflow

This section describes the git workflow of our project.

- After every milestone, we will merge from the develop to the main branch. This is only done, when the develop branch is stable.
- For every issue, one should work on a separate branch, which is branched from the develop branch.
- The branches should be named in the following manner:
  - Start with the type of branch such as ”feature” or ”bugfix”
  - Contain the Jira issue ID
  - The title of the Jira issue in a short summarized form
  - Examples: feature/PHO-14-add-and-edit-recipes, bugfix/PHO-69-fix-ui-error

- Before merging a branch to develop, the changes have to be reviewed by another developer from the team.
- Once the review is done, the Merge Request is approved by the reviewer and merged by the developer.

Workflow Diagram

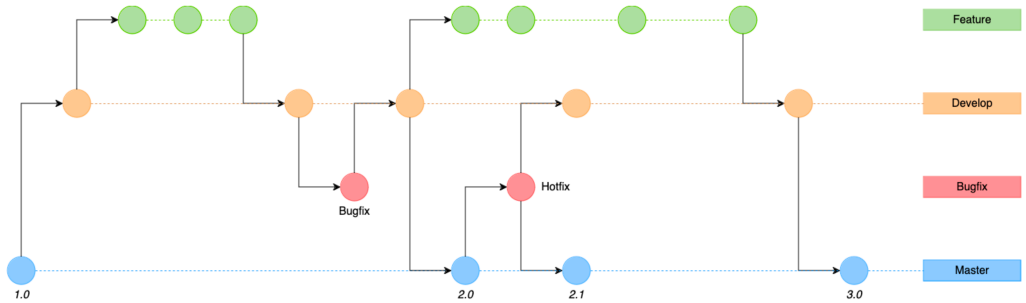


Figure 26: Git Workflow

8.3.2 Jira

- Definition of Done: An Issue is put into the ”Done” status, when
  - the functionality described is completely implemented
  - all related branches have been reviewed and merged into develop
- Definition of Ready: An Issue is ready for development, when
  - the functionality to implement is briefly described
  - dependencies have been identified and documented
  - time effort has been estimated

8.3.3 Test Strategy

This section describes the processes of testing and how it should be executed.

Functional Testing

The existing Smart Eating application is architecturally complex and the code difficult to read. No one in our team has experience with writing functional tests (unit, integration and system tests) in C# and no reference points on how to implement such tests exist in the project. Therefore no automated Functional Testing is performed.
Despite this, testing the functionality of our implemented features manually is possible. This is done during each merge request when the reviewer needs to ensure that the defined functionalities of an issue are implemented correctly.

Non-Functional Testing

It is possible to create automated UI tests in React, for example with the Cypress testing framework. Taking our limited time budget into account, a decision was made against UI Tests. The non-functional testing solely relies on Usability Testing. How the Usability Testing is conducted can be seen in Section 5.1

8.4 Time Management

The working hours of the team are tracked in Jira. Time efforts are estimated as described in Section 8.1.6. When time is spent on a task, it will be associated with the corresponding issue. Every member of the team is responsible to log their time accordingly.

8.4.1 Statistics of time tracking

The following graphics illustrate the time tracking statistics. The time spent is compared to the time that should be achieved.
In the lack of time spent in Sprint 1 is due to the Study objective only being released in week 2 as seen in Figure 27. The time spike in Sprint 7 is mainly due to implementing the documentation feedback from the final meeting. Figure 28 shows the total time spent of each member

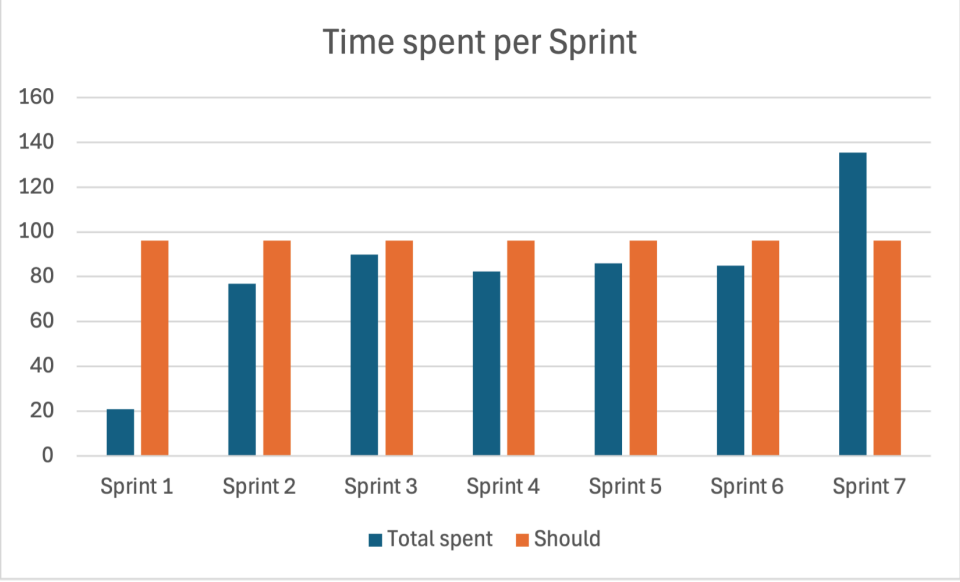


Figure 27: Time per Sprint

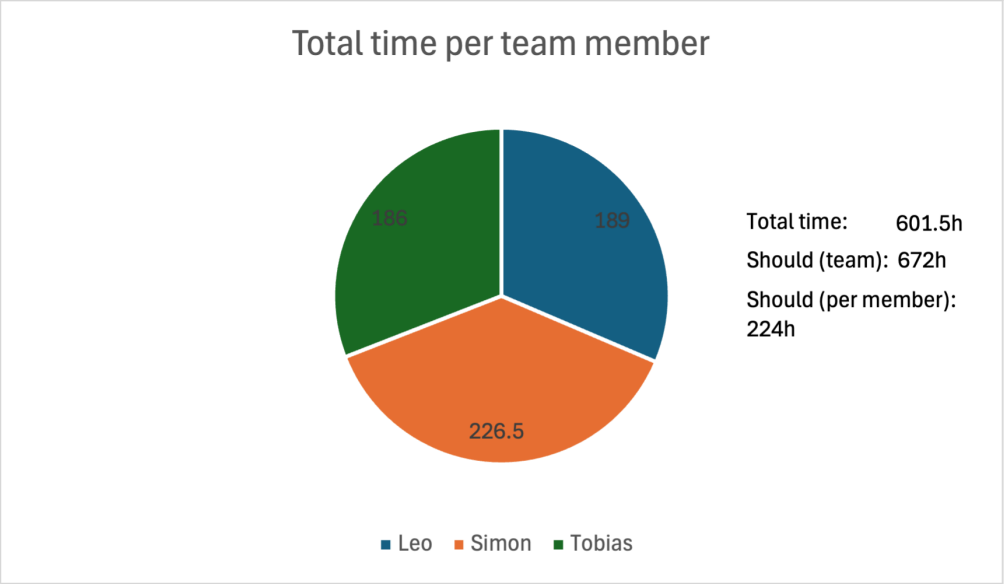


Figure 28: Time per team member

8.5 Risk Management

To manage risks a risk analysis is used to identify the key risks for our project. These risks are regularly reviewed and updated to ensure preparation for potential issues.

In order to keep an overview of the identified risks, a risk matrix was modeled during the elaboration phase. The risk matrix will be updated after every sprint. If new risks are found, they will be added to the table and risk matrix.

Risk	Description	Mitigation
Existing Risks	Since there is an already existing project which already has some unsolved problems it is possible, that those problems could have a negative impact on our contribution.	By focusing on our features and not changing much of the existing functionalities we can make sure to keep the technical risks as low as possible. Furthermore our advisors are always open to questions which will be fully utilized should we be confronted with any existing risks.
Scope Creep	Scope Creep is when a Project grows uncontrollably and gets too big. Causes include poor definition, control, and documentation of the project scope.	By clearly defining our main tasks and declaring other tasks as optional we can keep a good overview of our progress and ensure the completion of the main tasks. Should the scope increase during the project for unforeseen reasons setting priorities will make sure the most important features are developed.

Table 9: Risks part 1

Risk	Description	Mitigation
Technical Risks	Most of the existing technology for development has already been chosen. Since we are not very familiar with every technology it is possible that this can lead to problems.	Most of the problems in this category can be solved by spending time to get more familiar with the technology. Should there be a mismatch within the already used technologies an advisor must be contacted to ensure the correct solution is found
Accident/Illness	As a small project team, disruptions due to long term accidents or illness can significantly impact the project's progress.	Since accidents and illnesses are unpredictable, no preventative measures have been implemented. To reduce the impact of this risk the last iteration is a buffer and would allow to catch up on missed work. In the event of such scenarios however the scope of the project will most likely be reworked.
Time Pressure	If planned tasks take longer than anticipated, it can lead to time pressure towards the end of the project.	To prevent this, a time buffer in the form of optional tasks has been included.

Table 10: Risks part 2

8.5.1 Iteration 2

The risks are created during this Iteration and therefore count as the base of the risk matrix. The numbers represent the risks order in the table above and the colors represent changes as follows:

- white - new
- magenta - worsened
- green - improved

- grey - unchanged

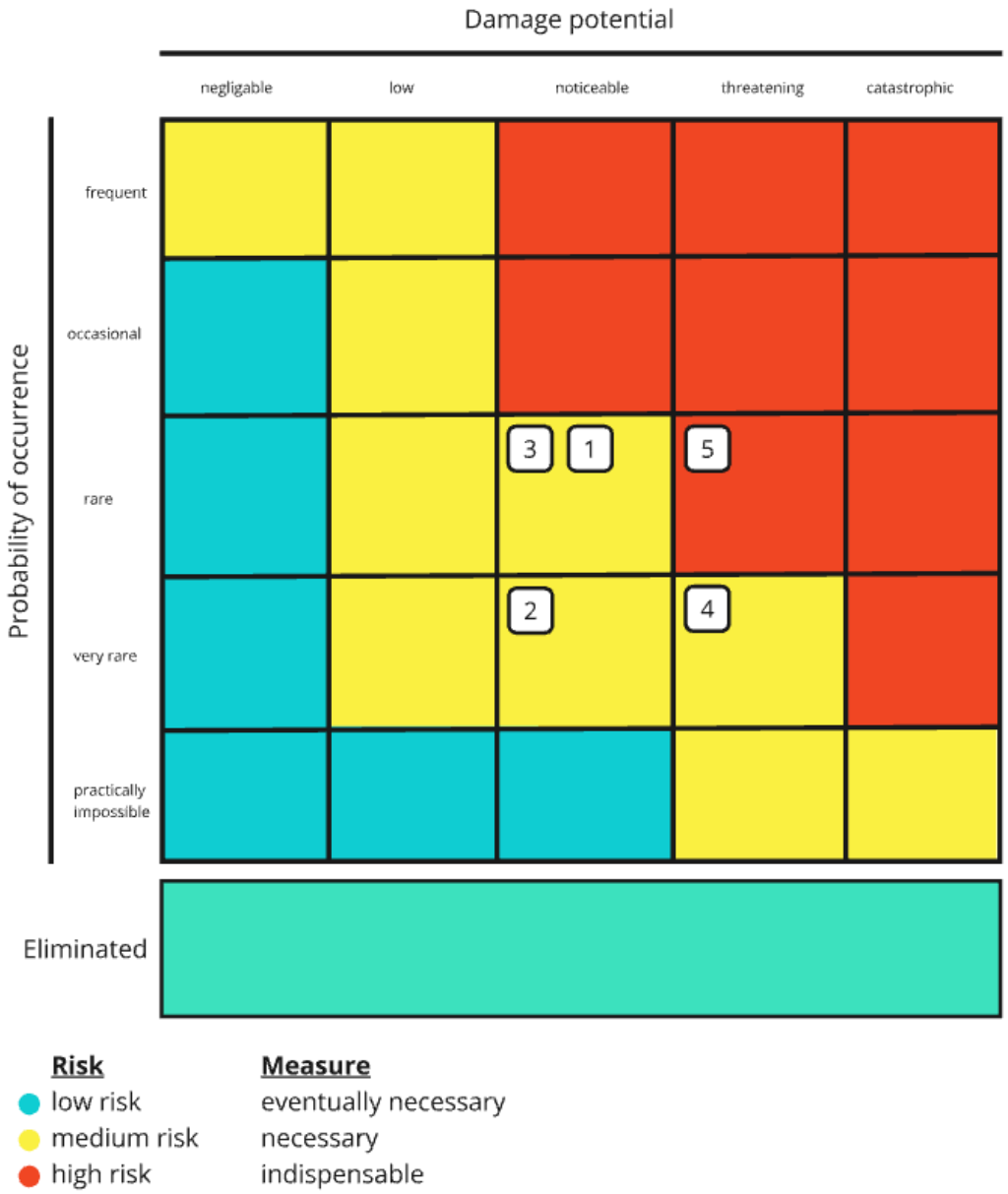


Figure 29: Risk matrix - Iteration 2

1. Existing risks

This risk is right in the center as it is unclear where the existing risks and issues lie within the project. If we do not run into any existing risks throughout the project, the probability can be lowered.

2. Scope creep

The scope creep is placed in the “very rare” field as the project was clearly defined in the beginning and won’t grow throughout the project.

3. Technical risks

This risk is right in the center as per the description in the risk table above, our team is unfamiliar with some of the used technologies and must get to know the project first.

4. Accident/illness

For this risk only more severe illnesses or accidents are accounted for as the duration of one iteration is 2 weeks if someone were to fall ill for 2 days they could still catch up to the required work. This is why the probability is set to very rare. However if someone were to be absent for an entire iteration a lot of working hours would be lost and could result in a reduction of the scope.

5. Time pressure

The damage potential of this last risk is set to the most threatening level compared to the rest of the risks, as there isn’t much that can be done once the time runs out. To mitigate the risk a time buffer is planned. In case the buffer is not enough, the project scope most likely has to be reduced.

8.5.2 Iteration 3

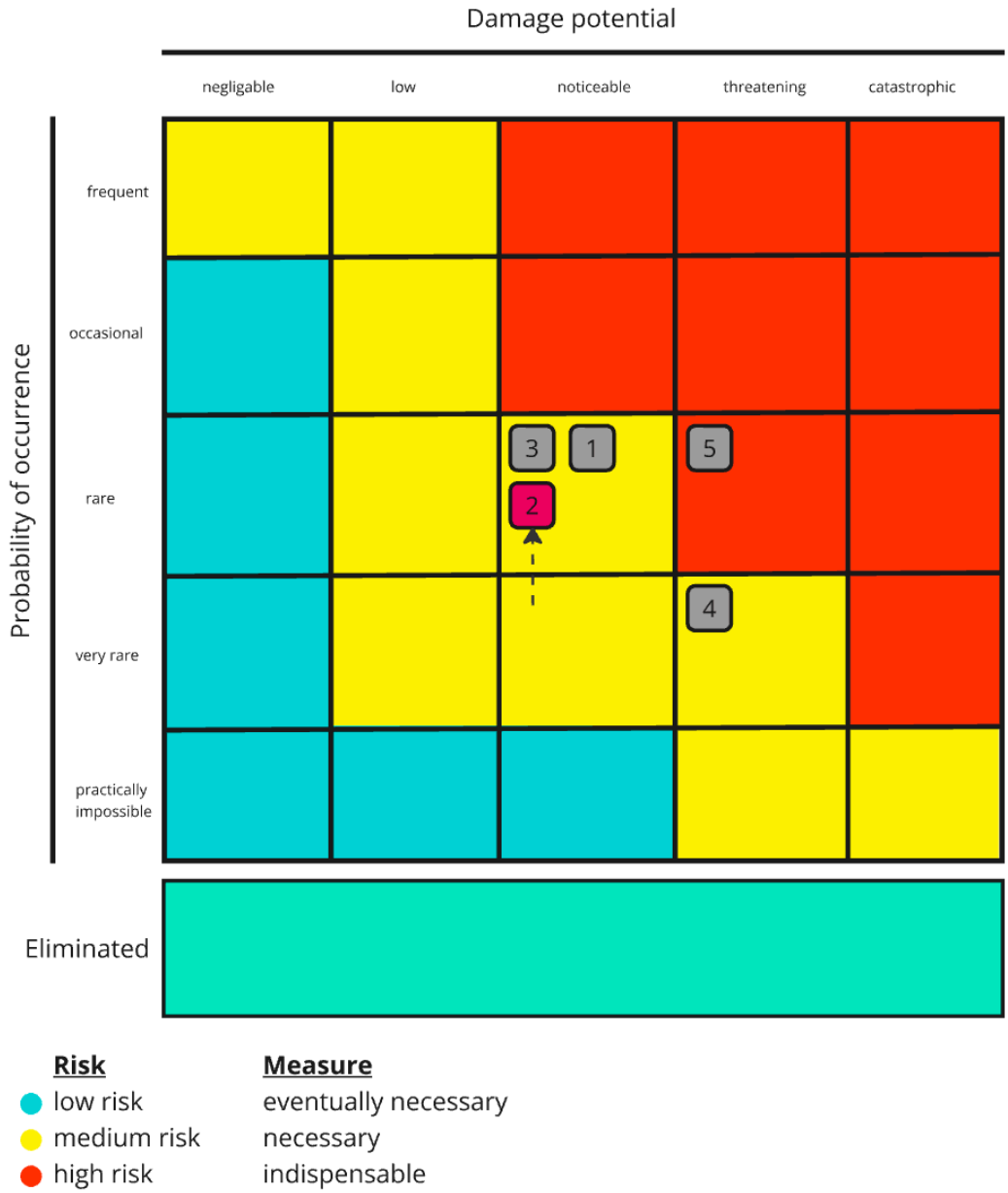


Figure 30: Risk matrix - Iteration 3

1. Existing risks

No problems have been encountered so far so the existing risks remain the same.

2. Scope creep

Incident: After discussing the mockups, additional features to be implemented, such as a dashboard, were planned for. Therefore the scope increases, which might result in scope creep. Therefore the probability of occurrence is increased for this risk.

Countermeasure: The increase in scope cannot be directly mitigated however the last iteration is planned as a buffer to allow for more features to be developed or if needed for the increased scope creep.

3. Technical risks

At this point it is not yet possible to change this risk as not enough work has been performed on the project.

4. Accident/illness

Considering that the risk occurrence is unchangeable and the resulting damage is inevitable this risk description will be omitted from future risk matrix iterations and will remain unchanged.

5. Time pressure

The time pressure description will also be omitted from future risk matrix as the risk occurrence rate will remain unchanged and the damage is also inevitable.

8.5.3 Iteration 4

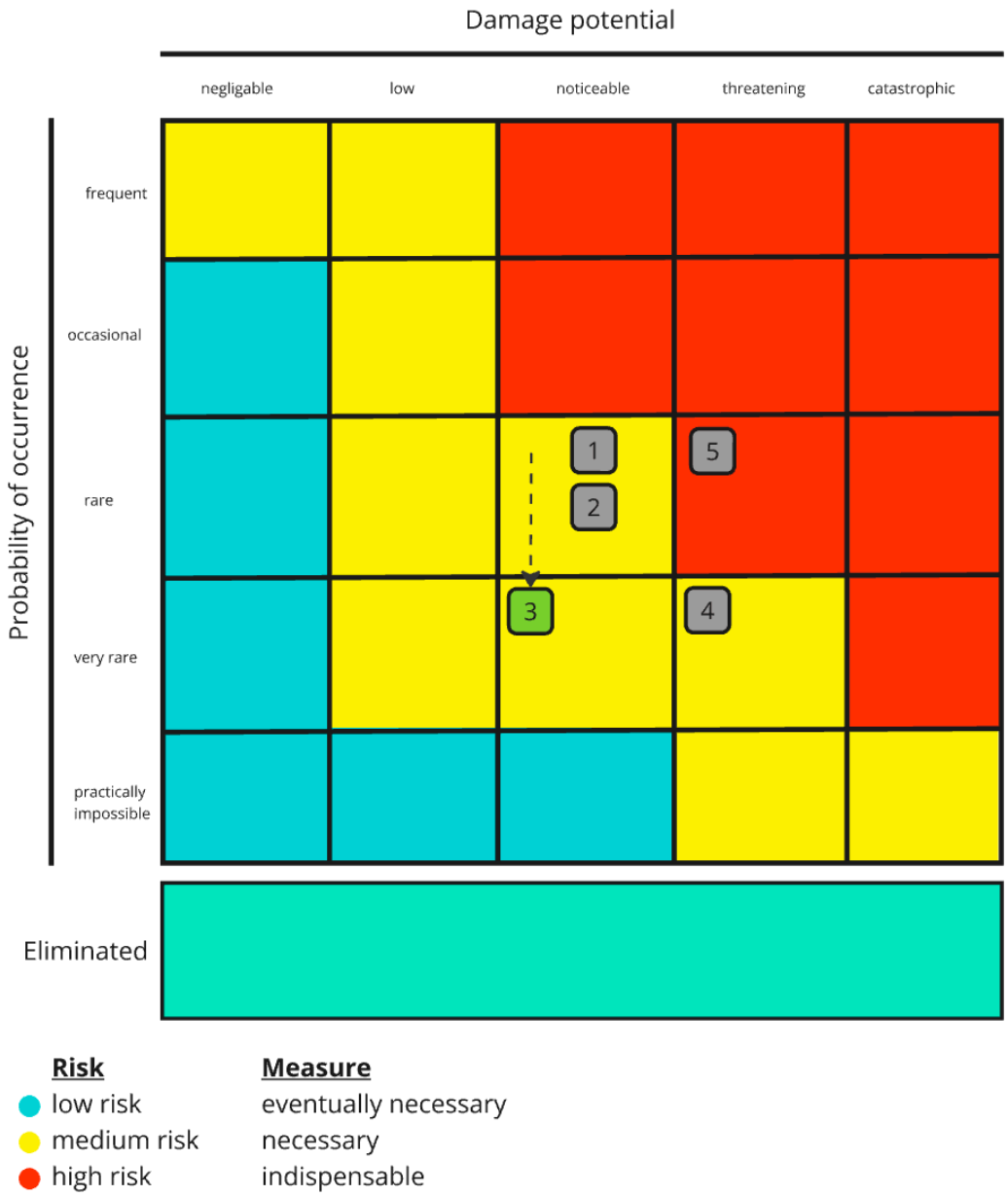


Figure 31: Risk matrix - Iteration 4

1. Existing risks

Incident: Moving forward some issues have arisen that are slowing down the process of implementing some of the features.

Countermeasure: To prevent being stuck for too long a meeting with the advisor is organized to discuss the issues at hand.

2. Scope creep

The scope creep remains unchanged and the description will be omitted from further risk matrix iterations as the scope will not be expanded anymore.

3. Technical risks

The general understanding of the structure of the project has increased across the team. Therefore the occurrence of technical risks arising has been reduced.

4. Accident/illness

Unchanged

5. Time pressure

Unchanged

8.5.4 Iteration 5

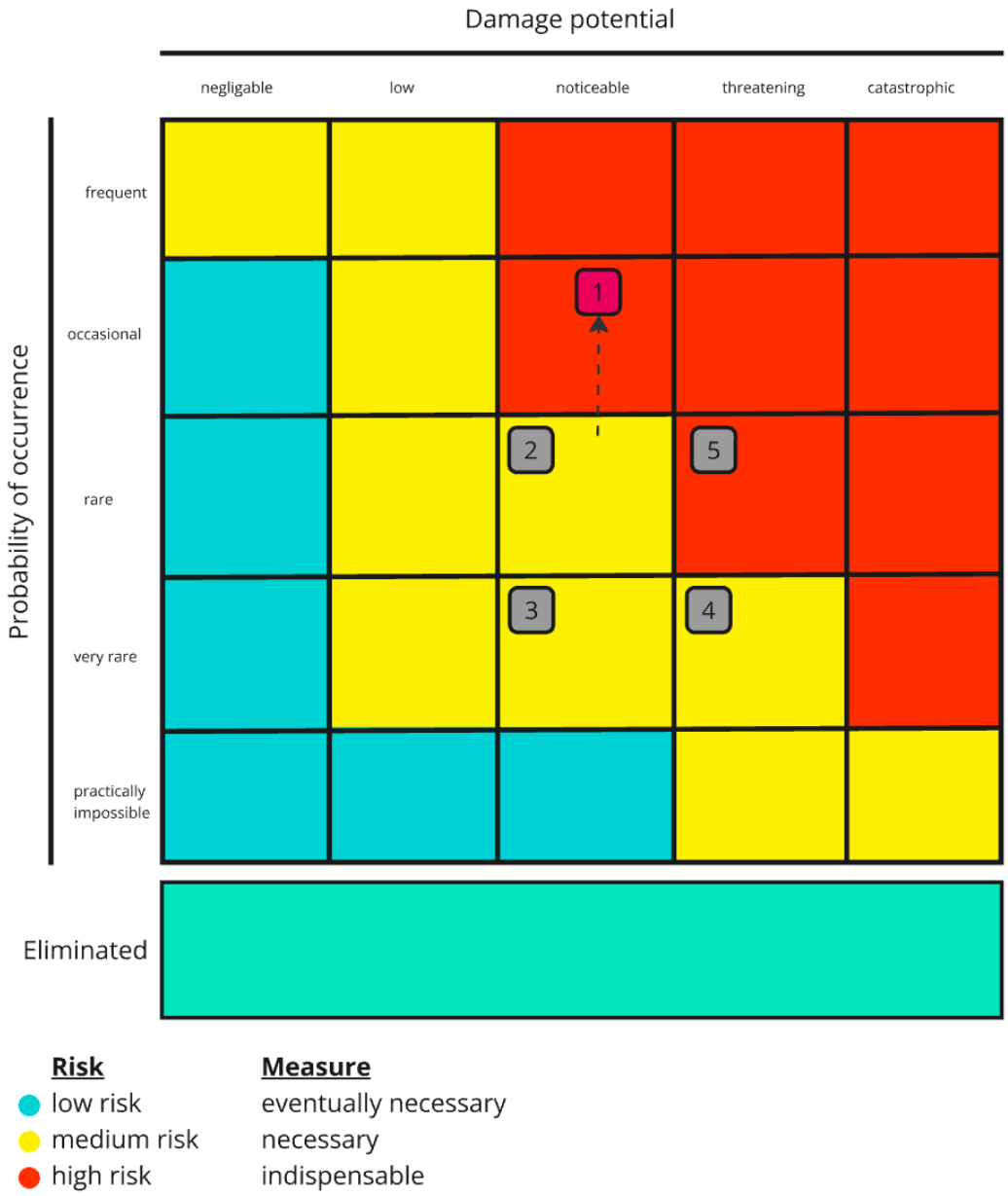


Figure 32: Risk matrix - Iteration 5

1. Existing risks

Incident: The issues from the last iteration are expanding and are being worked on. As one issue is resolved another can pop up. The occurrence probability has been increased as more issues are being uncovered more frequently.

Countermeasure: Constant contact to the advisors is being upheld through Microsoft Teams to resolve any blockages caused by issues in the backend section of the application. Additionally other unrelated issues are being resolved in parallel to minimize the damage.

2. Scope creep

Unchanged

3. Technical risks

The risk occurrence may not be further decreased as the risk itself involves both the technical risks that already exist in the project as well as our understanding of the technologies used. The occurrence rate is being balanced as the understanding of the structure is improving with every iteration while on the other hand issues such as the .NET version requiring an upgrade have caused delays.

4. Accident/illness

Unchanged

5. Time pressure

Incident: as some features are taking up much more time than anticipated the time pressure is increasing and it could lead to issues such as not being able to implement every feature in the scope of this project.

Countermeasure: The last iteration is planned as a buffer to ensure that more time is available than is planned for.

8.5.5 Iteration 6

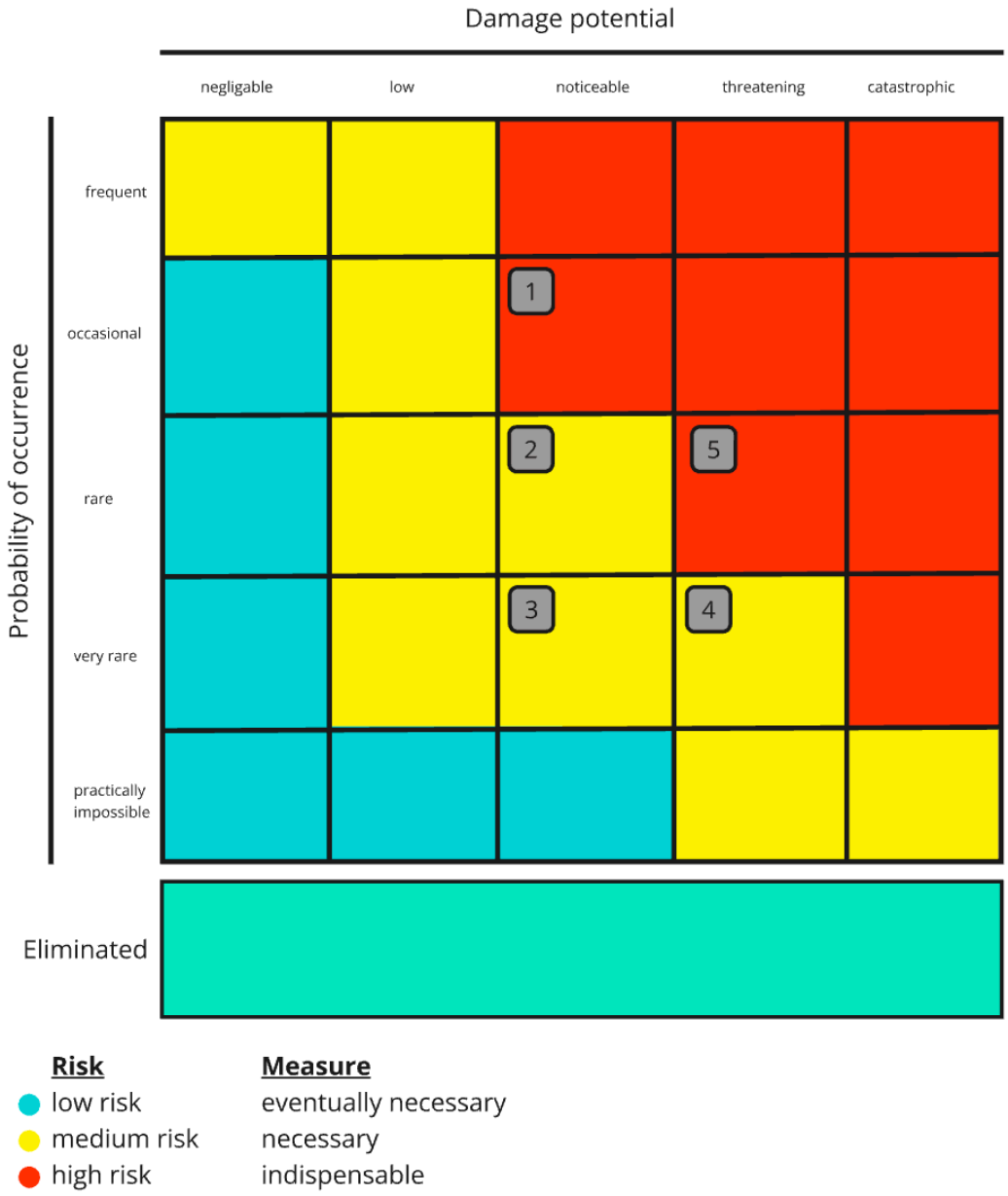


Figure 33: Risk matrix - Iteration 6

1. Existing risks

Because in the beginning it was discussed that the necessary features were already implemented that would allow us to make our changes and add our features, the issue that

some of those features were missing instead lead to a longer development and continued affecting the technical risks as well.

2. Scope creep

Unchanged

3. Technical risks

Some programming had to be performed in areas that was not anticipated in the beginning of the project and this uncharted territory required a deeper understanding which lead to a longer development time for certain features.

4. Accident/illness

Unchanged

5. Time pressure

Following this iteration the last iteration starts which was planned as a buffer in the beginning. As the existing risks and some technical risks have drawn out the development of some features it is highly likely that not all features will be implemented during this project.

There will not be another risk assessment for the last iteration as the project will be handed in during this iteration.

9 Bibliography

[1] Cypress website [“https://www.cypress.io/”] (visited on 05 November 2024)

[2] LaTeX Documentation Template [https://gitlab.ost.ch/SEProj/latex-documentation-template] (visited 26 September 2024)

[3] Smart Eating repository [https://gitlab.ost.ch/sa-smarteating/smarteating-project] (visited 30 September 2024)

[4] Jira board [https://peier.atlassian.net/jira/software/projects/PHO/boards/1] (visited 30 September 2024)

[5] Preference Icons [https://www.flaticon.com/authors/revolutionizzed-1] (visited 17 October 2024)

10 List of Figures

11 List of Tables

12 Appendix

12.1 Personal Reports

12.1.1 Tobias

What things did go well?
Our project planning went smoothly. Using Scrum+ was particularly helpful in maintaining a clear overview of our progress. Another element that went well was the team communication. Utilizing teams to share documents and to hold meetings was a good choice but thanks to our WhatsApp group chat we could guarantee a quick communication since it is our main way to communicate in our daily lives.

Which areas could we improve?
Diving into an existing project of this size was difficult. Maybe I was too confident after a good start in the project but after getting stuck while implementing the save option I should have asked for help much quicker and more often instead of spending many hours trying to fix the problem myself.

What were your personal highlights?
To me it was really nice seeing how much the Smart Eating webapp has optically changed. Giving the page a modern design and changing the colors really improves the experience. Another personal highlight was finding a solution to the problem of editing quantity and unit of ingredients separately.

12.1.2 Simon

The Studienarbeit was the first time in my studies, that I worked on an existing software project. Before, all study projects were about creating new applications.

Over the course of the Studienarbeit, I had to learn that it is way more difficult and time consuming to implement features in an existing application. You first have to understand how the current architecture is built. In the Smart Eating application, this was difficult because of a complex architecture and the lack of documentation. For example

Despite problems, we were able to give the UI a modern look and improve it's usability. I'm also happy with the quality of the features that were implemented.

The team collaboration went mostly smoothly. Already knowing each other helped maintaining good communication and solving problems. For this project we decided to not set up regular meetings with the coaches, as in the Software Engineering project, these meetings were mostly obsolete. This was due to the SE project proceeding without any problems. In future projects I would set up regular meetings with the coaches/customers no matter what, such that they stay up to date and that problems and mitigations can be discussed. However, I would have appreciated it if my teammembers put in more effort whether it was creating mockups or writing documentation.

My personal highlight is how we improved the User Experience of the application. In my eyes, it now looks very modern and is much easier to navigate. Overall I think this project is a success. I made a lot of learnings and collected valuable experience.

12.1.3 Leonardo

What things did go well?
The teamwork overall was very positive and there was a lot of mutual support. The team-meetings assured that we were all aware of the current status of the issues which helped maintain a overview.

The features that were implemented maintained a very goal fidelity. We implemented the visual components very close to the original mockups and the feedback from the usability tests was very helpful in further improving the design.

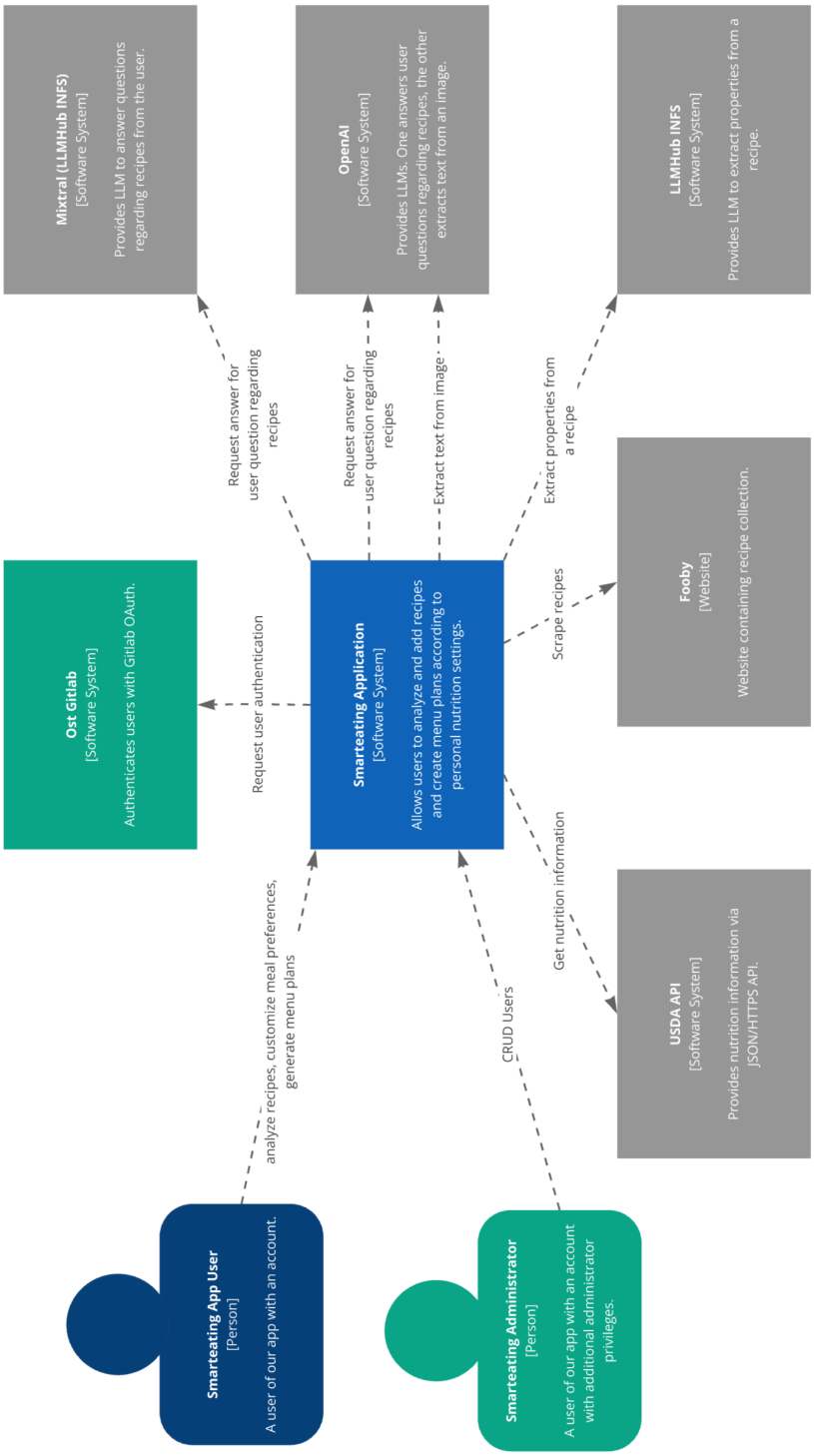
Which areas could we improve?
One big issue in my eyes was the late start of the project, missing out on almost a full sprint accounts for up to 32 hours per person. However it is imperative to remember that it is in everyones **equal** interest to arrange the first meetings before the semester starts. For future projects I will definitely push for an earlier introduction to the project to ensure that less time will be spent getting to know the project during the semester.

Furthermore in retrospective I wonder if it would've been a good idea to create diagrams of the uncertain parts (in this case the backend) of the application and hand them in over and over discussing them with the advisor to ensure everyone is able to develop a deeper understanding and would not have to deal with uncertainties during the project.

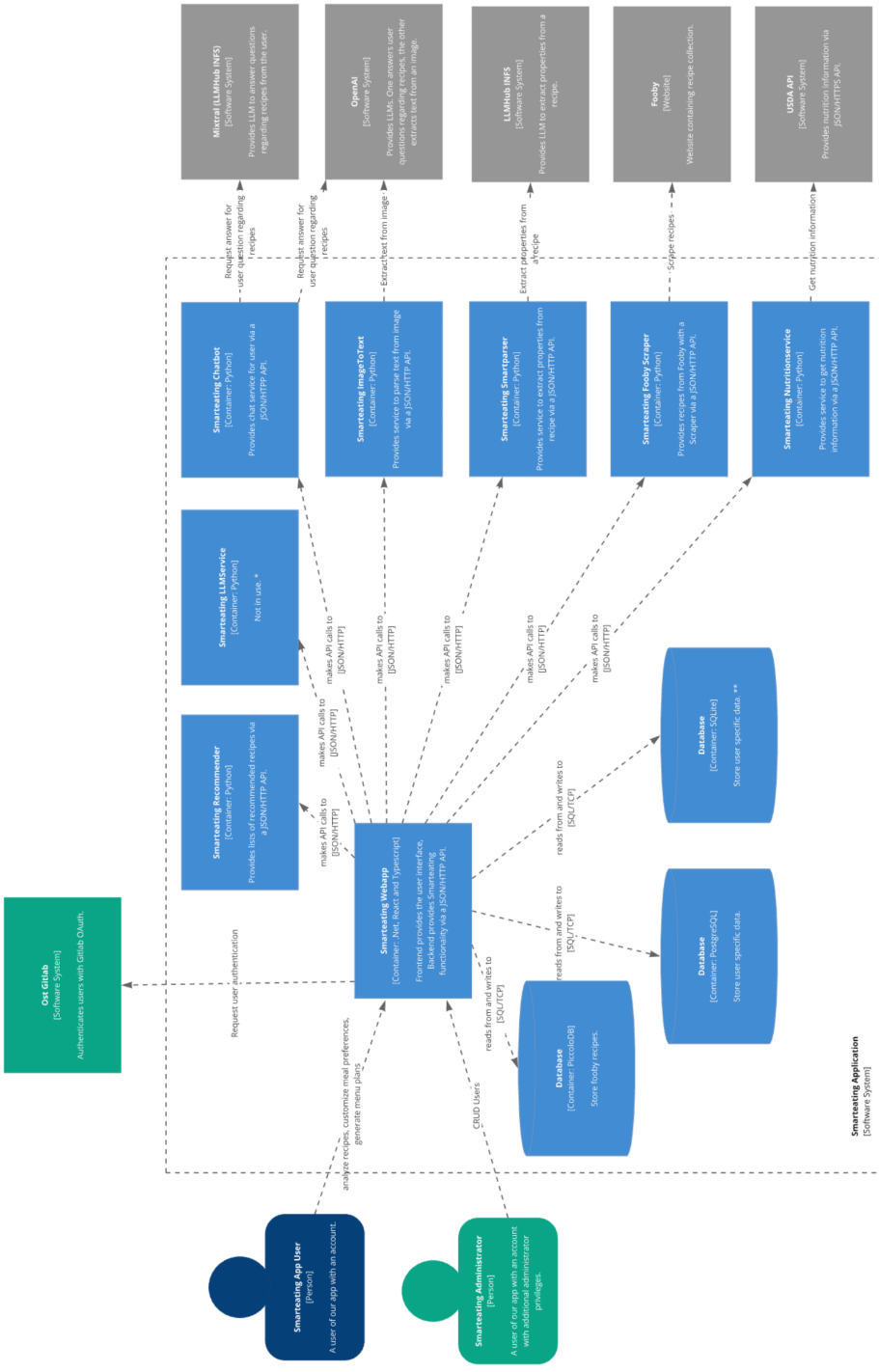
Lastly having more meetings with the advisors could lead to resolving issues faster as without a proper information flow it can become unclear where an issue or blockage is rooted.

What were your personal highlights?
I truly enjoyed the programming aspect of this project as most of it atleast in my case remained within the frontend. To be able to work on the usability test was also a highlight as I was hoping to implement the many tricks and details I previously learned in the UX module. Implementing the OAuth login through gitlab was interesting as I had never worked with OAuth before. Lastly working on an already existing project with all the ups and downs was important and a great experience as it represents the future work-life more accurately and allows for better growth in this sense.

12.2 C4 Context Diagram



12.3 C4 Container Diagram



* No team member knows what the LLMService is used for.

** The SQLite db is currently contained inside the Smarteating Webapp container. For clarity reasons and as it is planned to decouple this db in the future, it is depicted as a separate container.

12.4 Aufgabenstellung

Semesterarbeit
«Ausbau der Smart Eating Web App»

Betreuungsperson

Mitra Purandare (mitra.purandare@ost.ch)

Studierende

Leonardo Ravani (leonardo.ravani@ost.ch)
Simon Peier (simon.peier@ost.ch)
Tobias Kistler (tobias.kistler@ost.ch)

Ziel der Arbeit

Die Smart Eating App ist bisher nur soweit ausgebaut, dass spezielle AI Kernfunktionalität an Anlässen gezeigt werden kann. Die Funktionalität soll nun in dieser Studienarbeit um weitere Funktionalität erweitert werden.

Features

Wichtige Features sind:

- Authentication und Benutzerverwaltung
- Erfassen und Ändern von Rezepten (mit oder ohne KI)
- Teilen von Rezepten (innerhalb der Plattform)
- Einkaufsliste aus geplanten Rezepten generieren (optional)

Weiter können sich die Studieren auch ihre eigenen Ideen einbringen, wie man die Applikation ausbauen könnte.
Die Studierenden analysieren und planen die neuen Features, setzen sie um und integrieren sie in die bestehende Applikation.

User Experience

Die bisher wenig beachtete Userexperience soll den Anforderungen entsprechen. Damit soll auch das UI bestehender Features überarbeitet werden.

Auftrag

Die folgenden Teilaufgaben sind zu bearbeiten

- Analyse:
    - Funktionale Anforderungen (FR) ermitteln (notwendige und optionale)
    - Nicht-Funktionale Anforderungen (NFR) ermitteln
- Architektur & Design:

12.5 Meeting Minutes

12.5.1 14.10.2024

Review/Retrospective Meeting

Attendees: Leonardo Ravani, Simon Peier, Tobias Kistler

- Pro:
  - Overall progress is looking good. Milestones must be defined more precisely
  - The group collaboration is positive
- Con:
  - Communication, e.g. when and where a meeting takes place, should happen earlier (2-3 days in advance, not on the evening before)

Sprint Planning

Attendees: Leonardo Ravani, Simon Peier, Tobias Kistler

- Issues have been assigned and time estimated

12.5.2 17.10.2024

Weekly Meeting

Attendees: Leonardo Ravani, Simon Peier, Tobias Kistler

- Milestones have been refined
- Application is difficult to get to run on Mac

12.5.3 24.10.20

Weekly Meeting

Attendees:

- Application still does not run on Mac
- Everyone working on First Issue to get to know codebase

12.5.4 28.10.2024

Review/Retrospective Meeting

Attendees: Leonardo Ravani, Simon Peier, Tobias Kistler

- Everyone got the project working locally

- At the moment nothing can be saved, because the db should be initialized with ...
- Issues are almost done, we are a bit behind because the codebase is more complex than we presumed
- Recipe should not be shown in popup-window, but rather on a normal page with a back-navigation -> Update Mockups
- Pro:
  - We were able to make up the time wasted for getting the project running on our local machines
- Con:
  - The project is more complex than assumed and difficult to navigate (bad naming, code not clean, etc.)

12.5.5 31.10.2024

Coach Meeting

Attendees: Clemens Meier, Leonardo Ravani, Simon Peier, Tobias Kistler
User Management:

- Admin Panel
  - List of users
  - RUD user
  - Manage authorization
  - People with different authorizations
  - Should be able to give admin roles
  - Admin has two roles (user and admin)
- User registration
  - Have a look at OAuth (Google or MS)
  - Create user in backend and db with OAuth DB
  - Account settings obsolete when OAuth is implemented
  - Username per default email, but should be modifiable

DB questions

- DB
  - User ID is in PiccoloDB
  - SQLLite structure is generated through EntityFramework
  - execute addMigration manipulates DB schema
  - Use ...Repository classes to make query to db

    - NSwag generates client??? npm run nswag
    - Role
        - Create new entity
        - Add relations to ...Configuration
        - Add new stuff to API (Controller)

  -Save own recipes in SQLLite DB

UI

- Search only with Mockdata
- Save recipes with api call /recipe/create

12.5.6 11.11.2024

Review/Retrospective Meeting

Attendees: Leonardo Ravani, Simon Peier, Tobias Kistler

- Much clearer how User Management and Authentication should be implemented, also thanks to the Meeting with Clemens
- Mockup for User Management Page should be implemented
- Progress-wise we are slightly behind planning, as lots of things we have to implement are more complicated than they appeared to be

Sprint Planning

- Merge open merge requests, then the new sprint can be started
- Plan/Design how to implement user management and authentication
- Implement user management and authentications

12.5.7 14.11.2024

Weekly Meeting

Attendees: Leonardo Ravani, Simon Peier, Tobias Kistler

- Mockups and Diagrams for admin will be created this afternoon, such that the implementation of Login/Registration and User Management can begin right after

12.5.8 21.11.2024

Weekly Meeting

Attendees: Leonardo Ravani, Simon Peier, Tobias Kistler

- Mockups of admin panel look good
- Still issues with save recipe => regular contact with Clemens

12.5.9 25.11.2024

Review/Retrospective Meeting

Attendees: Leonardo Ravani, Simon Peier, Tobias Kistler

- Authentication/Login is implemented, but not merged yet
- User management not yet finished, as it is not clear how users can be saved (api calls...)
- Mockups for next feature (share recipe) have been created
- Milestone 3 Alpha is not yet achieved, as certain required features are not yet finished
- The goal is to finish these features until the coming Thursday.
- Pro
  - Good start into sprint (diagrams and mockups)
  - Authentication took some time to implement, but it now works
- Con
  - For implementation of User Management, it is unclear whats the best way to get data from backend into frontend
  - DB was not working correctly, despite it was set up and the application was running

Sprint Planning

- Sprint is not finished yet, thus Milestone is delayed
- Goal is to finish milestone this Thursday and start new Sprint then

12.5.10 29.11.2024

Weekly Meeting

Attendees: Leonardo Ravani, Simon Peier, Tobias Kistler

- We had to upgrade the application to .NET 9 to fix errors with NSwag
- Usability Testing is prepared. Everyone will conduct the testing until Monday.

12.5.11 02.12.2024

Biweekly Meeting

Attendees: Leonardo Ravani, Simon Peier, Tobias Kistler

- Upgrade to .NET9 resulted in new problems => talk to Clemens
- Admin panel is being implemented

12.5.12 09.12.2024

Review/Retrospective Meeting

Attendees: Leonardo Ravani, Simon Peier, Tobias Kistler

- Last feature (sharing of Recipes) most likely cannot be implemented due to time pressure
- Usability testing got useful and mostly similar and positive feedback
- Most of the feedback already implemented

Sprint Planning

- Finish all open issues and then focus on writing/improving documentation

12.5.13 12.12.2024

Weekly Meeting

Attendees: Leonardo Ravani, Simon Peier, Tobias Kistler

- Admin Panel is in review
- Saving recipes is still in progress. Errors had to be fixed and now the DB schema needs to be extended
- Documentation is being extended and improved

12.5.14 19.12.2024

Advisor Meeting

Attendees: Clemens Meier, Mitra Purandare, Leonardo Ravani, Simon Peier, Tobias Kistler

- Take share for fooby recipes into documentation outlook
- Abstract ⟶ present tense, also write about what is smart-eating application,
- Management summary - 1.5 to 2 pages.

- Glossary → Limited to a software-developer perspective.
- C4 Diagram add colours for what was before and after. - also add higher resolution to the appendix.
- Contact mirko about Eigenständniskeitserklärung.
- C4 sqlite - add asterisk to label it as “is integrated in the webapp”
- Say that no one knows about the LLM
- Sqlite is thought to have only user data in it. Postgres should be mainly read access so it can also be mainly scaled.
- PiccoloDB is a python service. Add unfortunate naming (named after utilised technology) to the outlook. ORMapper and Webservice
- No clear separation between Server and client.
- Implementation → Don’t mind having code-snippets → highlights of the most interesting → Adjust validate User in Sequence Diagram.
- Risk Matrix - Add mitigations ←
- Move Mockups to implementation.
- Outlook is from the project point of view →
- Whatever we received came with this →
- Don’t create a section just for judging the existing project but instead add what was in the project that stopped us from creating the features whenever necessary.
- If we know we should write down which feature was not working that prevented us to start the application on the mac.
- Usability testing as a seperate chapter. Right after discussion of NFR and FR.
- Bibliography → no links but previous SA BA, Blog, Papers that we read.
- Take out the links to weekly sprints time planning and leave the graphics.
- Personal report in the appendix.
- Results → also describe the final product.
- Extend Implementation section →