Bachelor Thesis

PanPal: A chef's chatbot

Semester: Spring 2024

Version: 1.0
Date: 2024-06-13 19:39:31Z
Git Version: 29cefe1

Author	Andri Joos
Advisor	Prof. Dr. Mitra Purandare
External Co-Examiner	Dr. Raphael Polig
Internal Co-Examiner	Prof. Frank Koch

Department of Computer Science
OST Eastern Switzerland University of Applied Sciences

Abstract

This report presents the comprehensive development of “PanPal: A chef’s chatbot”, aimed at enhancing the Smart Eating platform by integrating an interactive cooking assistant. The project addresses the absence of a chatbot to assist users with various cooking-related tasks, leveraging both an OpenAI GPT model and Mixtral, an open source large language model (LLM) hosted at OST. The report provides an in-depth analysis of the planning, execution, and evaluation phases of the project, highlighting the methodologies, risk management strategies, and the overall development lifecycle.

The final product is a robust, user-friendly chatbot fully integrated into the Smart Eating infrastructure. It offers a hands-free experience, allowing users to interact with the assistant without needing to touch their devices. Despite some limitations, the project successfully meets its objectives and lays the groundwork for future enhancements. Key achievements include the development of a versatile cooking assistant, the successful integration of the chatbot into the Smart Eating platform, and the implementation of effective project management strategies.

The report includes a comparative analysis of the available assistants to aid users in making informed choices. The report also highlights areas for future work, such as improving the Mixtral assistant’s capabilities.

Acknowledgements

In this chapter, I would like to express my gratitude to those who supported me throughout my bachelor thesis.

First and foremost, I would like to thank my advisor, Prof. Dr. Mitra Purandare. Her valuable feedback and helpful ideas significantly improved my work. I greatly appreciated having her as my advisor for the bachelor thesis.

I also want to thank Lars Herrmann, Abinas Kuganathan and Clemens Meier for their help throughout the project. Their inputs were greatly appreciated.

Finally, I would like to thank my friends and family for their support, not only during this thesis, but over the past three years.

Contents

Part I

Management Summary

Chapter 1

Management Summary

1.1 Problem

As described in the Initial Project Description, the Smart Eating platform lacks an interactive chatbot. This chatbot should be able to assist the user with various cooking related tasks. The chatbot should integrate a GPT model from OpenAI as well as an large language model hosted at OST. Furthermore, the chatbot should provide a hands-free experience for the user.

1.2 Techniques

As a first step, Scrum was selected as a short-term project management framework to iteratively create a working, presentable product. To ensure that the project would be completed within the given timeframe, RUP was chosen as a long-term project management framework. I have these project management frameworks because I already gathered some experience with them in previous projects.

1.3 Results

The final solution successfully addresses the problem by delivering a robust, user-friendly chatbot. The user can choose between the OpenAI assistant, which uses the OpenAI API and the Mixtral assistant, which uses the Mixtral model hosted at OST. The chatbot was developed such that a new assistant can be easily integrated.

To provide a hands-free experience, the assistants can transcribe audio and respond by speaking their answer.

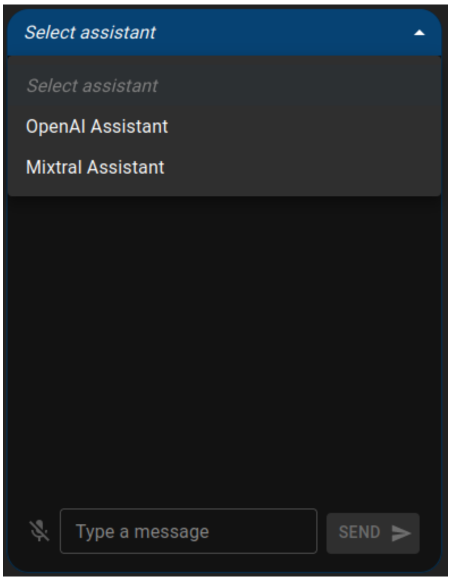
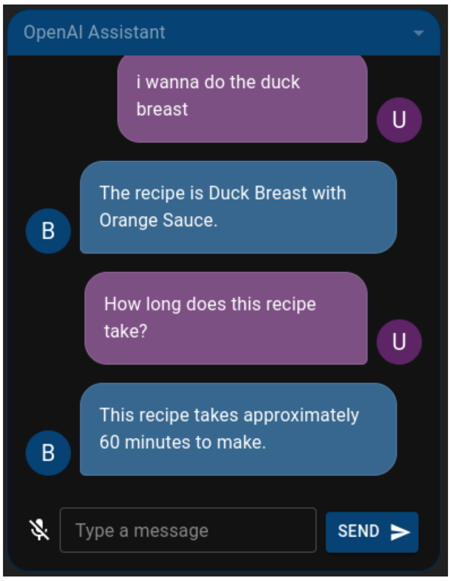


Figure 1.1: Assistant selection



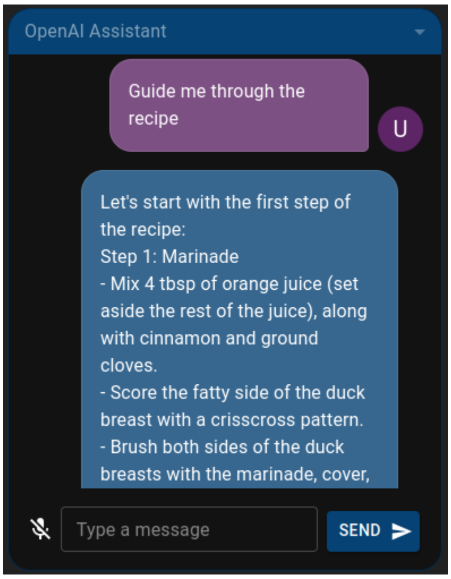
i wanna do the duck breast

The recipe is Duck Breast with Orange Sauce.

How long does this recipe take?

This recipe takes approximately 60 minutes to make.

Figure 1.2: Assistant selection

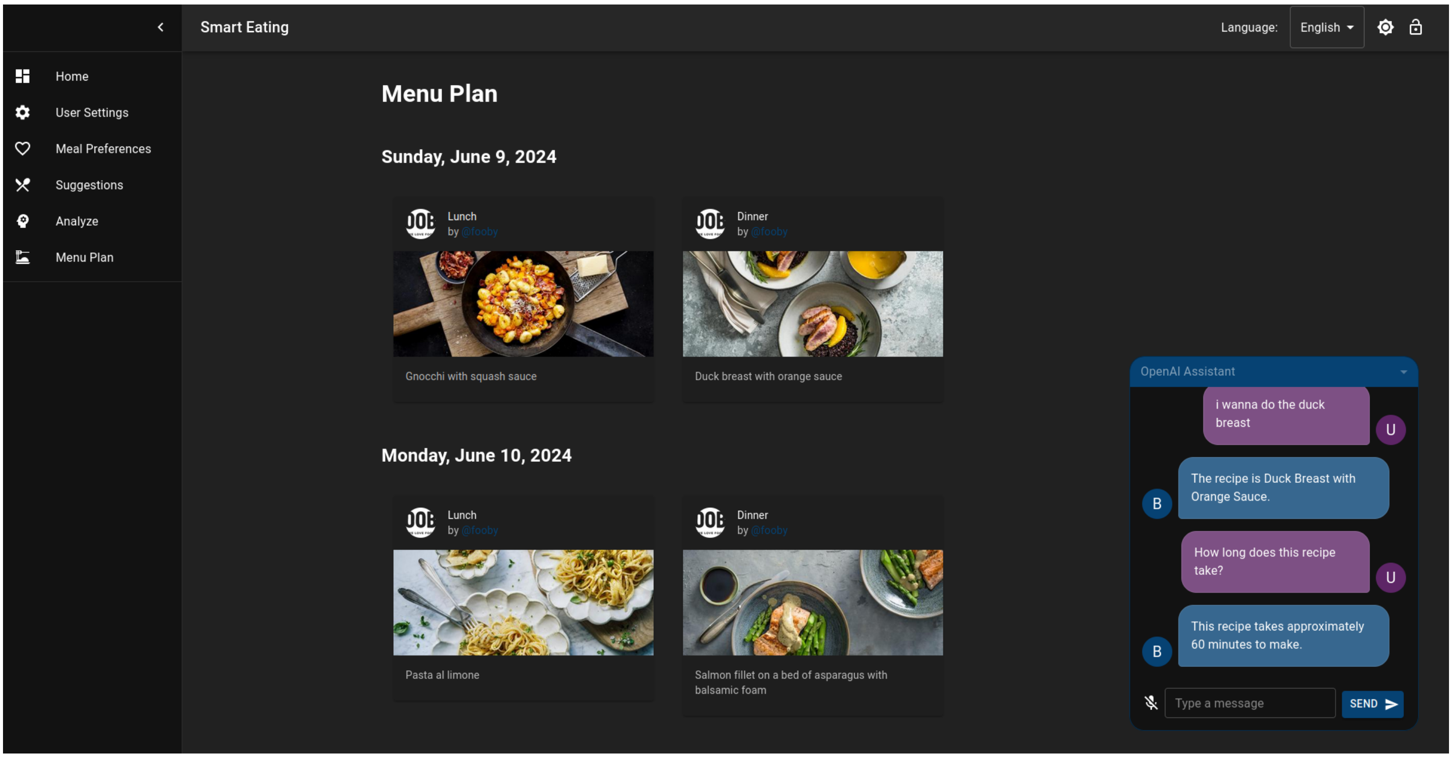


Guide me through the recipe

Let's start with the first step of the recipe:
Step 1: Marinade
- Mix 4 tbsp of orange juice (set aside the rest of the juice), along with cinnamon and ground cloves.
- Score the fatty side of the duck breast with a crisscross pattern.
- Brush both sides of the duck breasts with the marinade, cover,

Figure 1.3: Assistant selection

The chatbot is fully integrated into the Smart Eating infrastructure.



Menu Plan

Sunday, June 9, 2024

Lunch by @fooby — Gnocchi with squash sauce

Dinner by @fooby — Duck breast with orange sauce

Monday, June 10, 2024

Lunch by @fooby — Pasta al limone

Dinner by @fooby — Salmon fillet on a bed of asparagus with balsamic foam

U: i wanna do the duck breast

B: The recipe is Duck Breast with Orange Sauce.

U: How long does this recipe take?

B: This recipe takes approximately 60 minutes to make.

Type a message

Figure 1.4: Smart Eating platform with integrated chatbot

To help users make an informed decision about which assistant is best for them, the assistants are compared.

1.4 Outlook

During the project, several improvements were identified.

The Mixtral assistant could be improved with further prompt engineering and providing a tool to search for the relevant recipe information. The implementation of noise cancelling will allow users to use the chatbot even in noisy conditions. If a user wants to swap ingredients, the chatbot should respect the user's preferences, allergies and eating habits.

These improvements, as well as eliminating the current limitations, will provide a better user experience.

1.5 Conclusion

The "PanPal: A chef's chatbot" project integrates a robust and user-friendly chatbot into the existing Smart Eating platform. Despite some limitations, such as the Mixtral assistant's issues with ingredient quantities and step-by-step instructions, the project successfully meets its objectives and lays the groundwork for future enhancements.

Part II

Product Documentation

Chapter 2

Requirements

2.1 Functional Requirements

2.1.1 Actor

There is only one actor, specifically a user visiting the Smart Eating platform. This actor wants to be guided through cooking process, as well as get related information. Either, he wants to communicate with the system via text or voice.

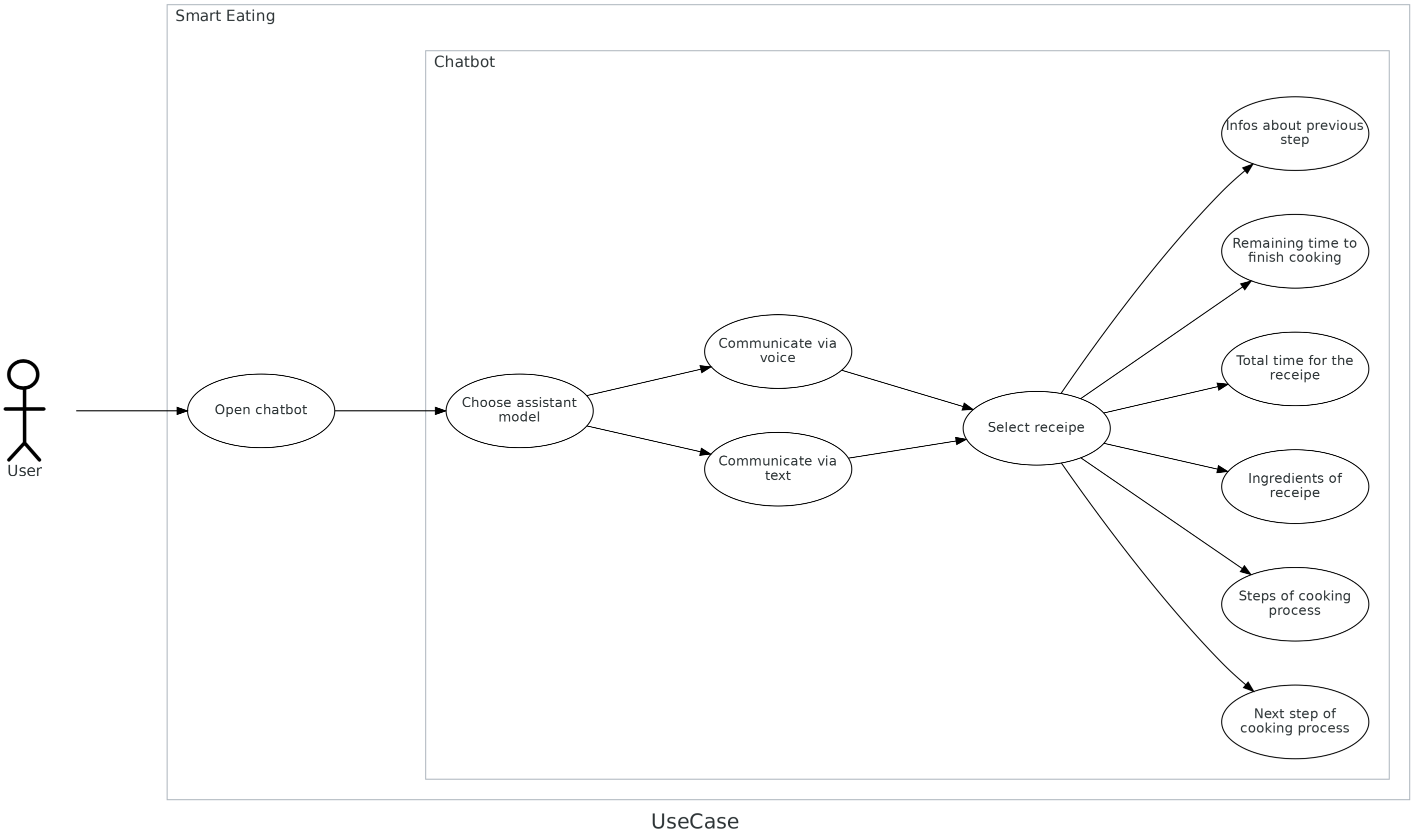


Figure 2.1: Use case diagram

Note: The user can hop from one use case to another in the last stage. However, to increase readability and decrease complexity of the diagram, this is ommitted in the image.

2.1.2 User Stories

The actions a user wants to perform are defined by the following User Stories. They are marked with US-X, where X is the number corresponding to the User Story.

All User Stories will be given a priority of development, as well as rough estimation of development effort needed. The Fibonacci scale is a common measure in the scrum process and reflects the relative effort of a task in the project (Story Points). Additionally, the MoSCoW method will be used to prioritize the User Stories.

US-1: Chatbot UI

As a user, I want the chatbot to be integrated into the Smart Eating platform. The chatbot should be easily accessible.

Estimation 1

Prioritization Must Have

US-2: Assistant model selection

As a user, I want to be able to select the assistant model so I can interact with my preferred model.

Estimation 1

Prioritization Must Have

US-3: GPTAgent

As a user, I want to communicate with a GPTAgent.

Estimation 5

Prioritization Must Have

US-4: In-house LLM

As a user, I want to communicate with an OST in-house LLM. Which one will be evaluated during the project.

Estimation 8

Prioritization Must Have

US-5: LLM comparison

As a user, I want to have a clean and simple comparison of the used LLMs.

Estimation 3

Prioritization Must Have

US-6: Communicate with text

As a user, I want to be able to communicate with the chatbot via text.

Estimation 5

Prioritization Must Have

US-7: Select receipe

As a user, I want to be able to select the receipe to be cooked.

Estimation 5

Prioritization Must Have

US-8: Ingredients of receipe

As a user, I want to know which ingredients are needed in the current receipe.

Estimation 5

Prioritization Must Have

US-9: Total time of receipe

As a user, I want to know how much time is needed to complete the current recipe.

Estimation 3

Prioritization Must Have

US-10: Step-by-step guidance

As a user, I want to be able to tell the chatbot to provide step-by-step guidance, with the chatbot stopping at each step until I tell it to move on to the next step in the cooking process.

Estimation 5

Prioritization Must Have

US-11: Communication with audio

As a user, I want to be able to communicate with the chatbot using my voice.

Estimation 5

Prioritization Should Have

US-12: Assistant takes user's eating habits into account

As a user, I want the assistant to take my eating habits, such as allergies, if I am vegetarian, vegan etc. into account.

Estimation 5

Prioritization Could Have

US-13: Further in-house LLMs

As a user, I want to be able to use all OST in-house LLMs.

Estimation 21

Prioritization Could Have

US-14: Remaining time

As a user, I want to know how much time is left until the receipe is completed.

Estimation 8

Prioritization Could Have

US-15: Steps of cooking process

As a user, I want to get on overview of the required steps to cook the meal.

Estimation 8

Prioritization Could Have

US-16: Infos about previous steps

As a user, I want to be able to get information about previous steps.

Estimation 3

Prioritization Could Have

US-17: Saving the chat

As a user, I want my chat to be saved automatically, so I can pick up the conversation at any time again, even if I closed the application in the meantime.

Estimation 5

Prioritization Won't Have

US-18: Recipe suggestion

As a user, I want to receive recipe suggestions based on my preferences.

Estimation 21

Prioritization Won't Have

2.2 Non-Functional Requirements

Each NFR has a "Fulfillment Check" row which describes who has the responsibility to check if the NFR is fulfilled and how this is checked.

2.2.1 Compatibility

ID	NFR-1
Subject	Integration into existing system
Requirement	Co-existence
Priority	High
Measures	If there is an existing technology used in the existing project for a similar problem, the same technology must be used. New technologies must be discussed with the architect and the team members of the existing system.
Fulfillment check	The architect is responsible to check the technology- and toolstack.

2.2.2 Usability

ID	NFR-2
Subject	Model response quality
Requirement	Operability
Priority	High
Measures	The models must be carefully evaluated.
Fulfillment check	A predefined testset of questions which determine the quality of the models.

ID	NFR-3
Subject	Chatbot accessibility
Requirement	Accessibility
Priority	High
Measures	The chatbot must be easily accessible.
Fulfillment check	The location of the chatbot in the existing system is determined together with the stakeholders.

2.2.3 Security

ID	NFR-4
Subject	User authentication
Requirement	Authenticity
Priority	High
Measures	The existing system already has authentication in place.
Fulfillment check	No fulfillment check needed as requirement is already fulfilled.

2.2.4 Maintainability

ID	NFR-5
Subject	Software maintainability
Requirement	Modularity
Priority	High
Measures	The software architecture must be designed using state-of-the-art principles.
Fulfillment check	The architect is responsible to check if the defined architecture is implemented accordingly.

ID	NFR-6
Subject	Unit Tests
Requirement	Testability
Priority	High
Measures	The unit tests must cover a vast part of the business logic. However, it is not foreseeable at the moment if there even is unit-testable code.
Fulfillment check	Test concept

2.2.5 Portability

ID	NFR-7
Subject	User installability
Requirement	Installability
Priority	High
Measures	As the software runs as website, it is not necessary to install anything.
Fulfillment check	No fulfillment check needed, as this is fulfilled anyways.

ID	NFR-8
Subject	Developer installability
Requirement	Installability
Priority	Medium
Measures	A docker image must be created and integrated into the current docker ecosystem.
Fulfillment check	The docker image is created and builds sucessfully. The project manager is responsible to check this requirement.

Chapter 3

Architecture

This chapter explores the main structure of the project.

3.1 Project Architecture

This section provides an overview of the Smart Eating project structure and the structure of the chatbot project that it contains.

3.1.1 Smart Eating Project Structure

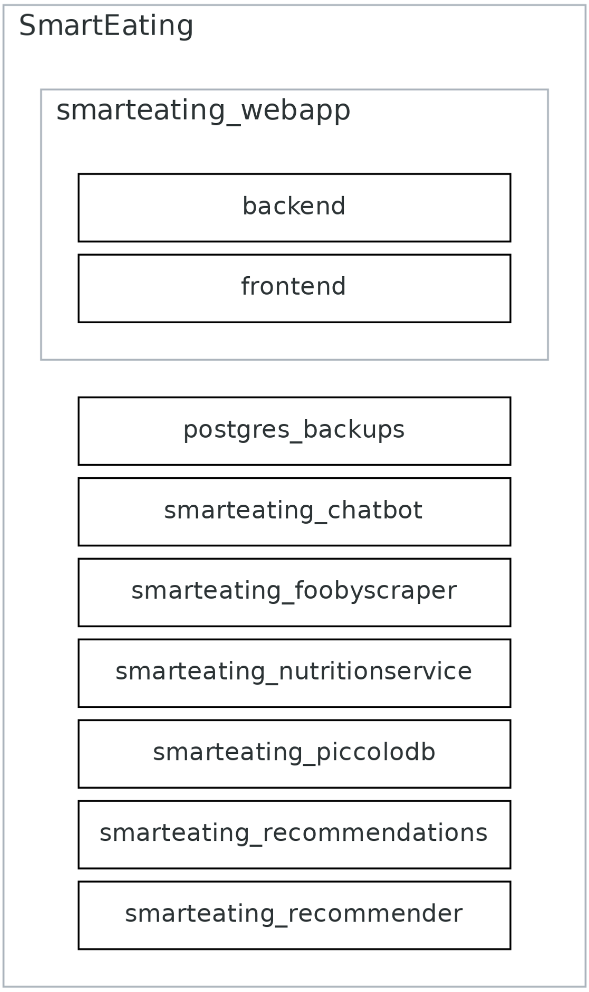


Figure 3.1: Smart Eating project structure

The Smart Eating project is a collection of multiple independent applications. Each application has its own directory and can be built independently. The diffrent applications are interconnected through the use of APIs.

All projects already exist, apart from the smarteating_chatbot. The primary logic for processing user input and generating responses for the chatbot will be implemented within this project.

3.1.2 Chatbot Project Structure

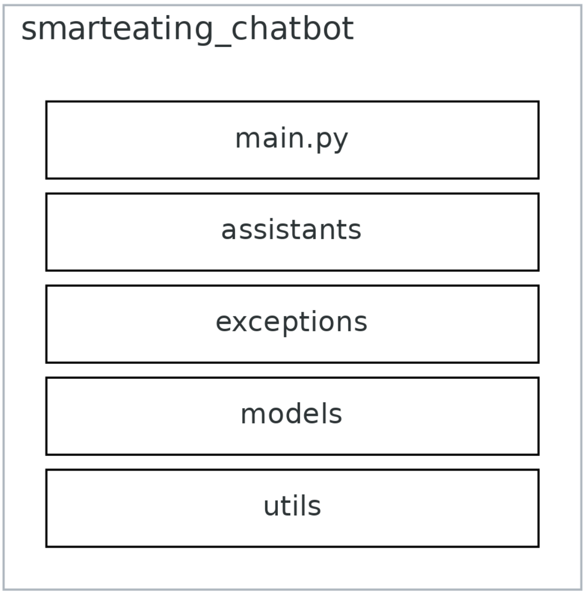


Figure 3.2: Chatbot project structure

The main.py sets up the API and handles the API calls. It mainly interacts with the assistants. The code diagrams provide an overview of these interactions.

The assistants module contains all of the logic associated with the assistants. The section on the assistants structure offers a more detailed analysis of the module's structure.

The remaining modules are sufficiently self-explanatory that further discussion is unnecessary.

Assistants Module Structure

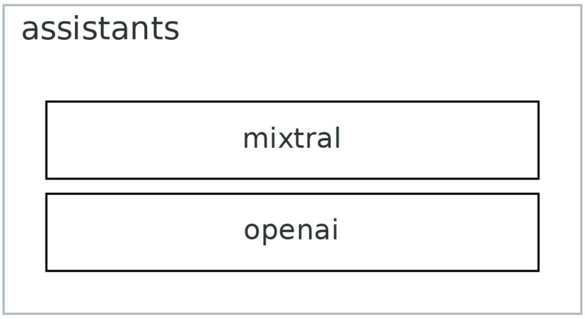


Figure 3.3: Assistants module structure

The assistants module is structured in such a way that different assistants can utilise different technologies and are completely independent of each other. It may be advisable, however, to transfer some shared logic to the assistants module in the future, thus avoiding the duplication of code in assistants with similar technologies. As the openai assistant utilises the openai package, while the mixtral assistant employs the LangChain framework, there is currently no shared code.

3.2 C4 Diagrams

C4 diagrams visualize the software architecture.

It should be noted that the diagrams are not exhaustive; in order to enhance readability and comprehensibility, details have been omitted. They are intended to facilitate understanding of the architecture of the application.

3.2.1 System Context Diagram

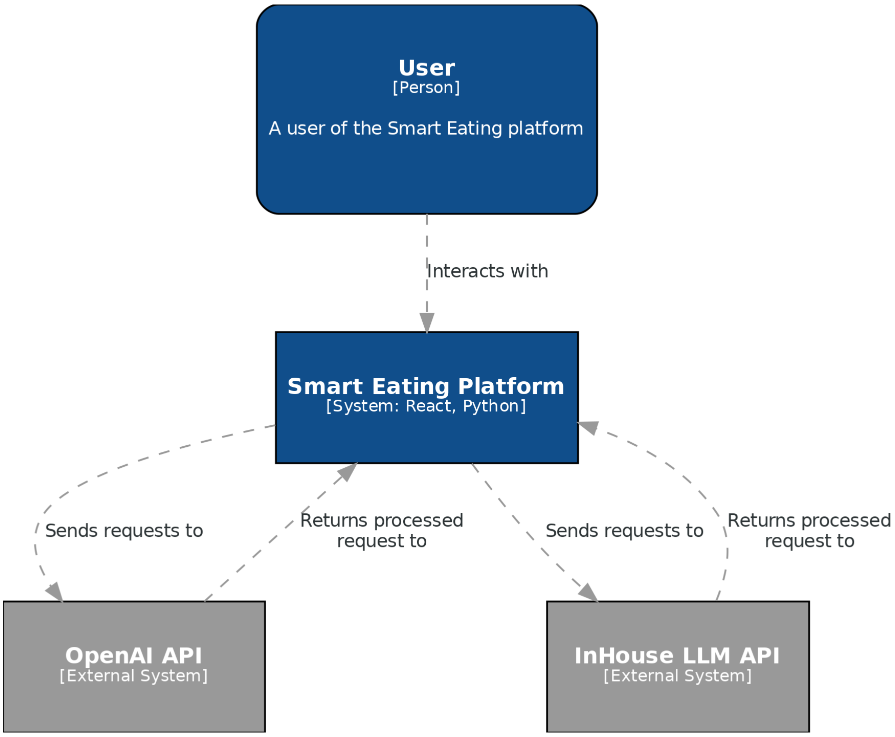


Figure 3.4: System Context Diagram

3.2.2 Container Diagram

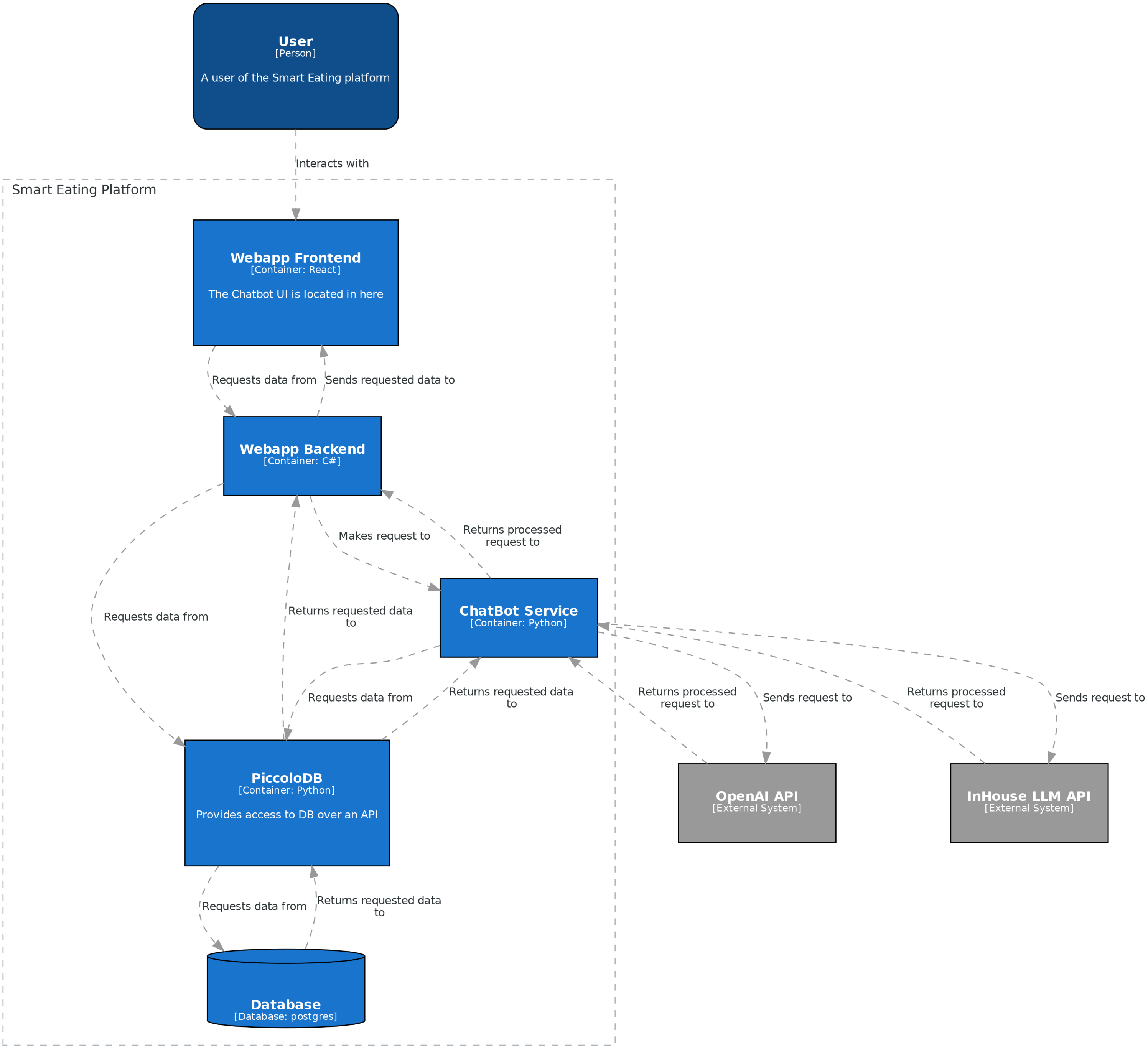


Figure 3.5: Container Diagram

3.2.3 Component Diagram

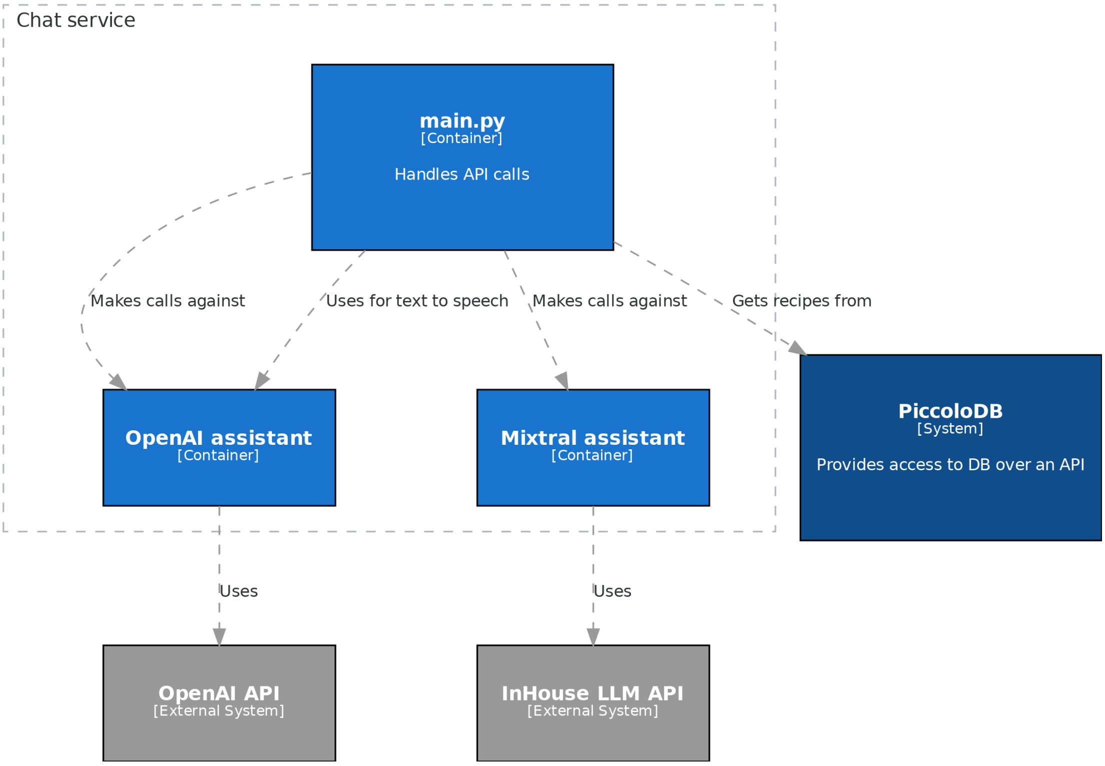


Figure 3.6: Component Diagram

Given the absence of a open-source and freely accessible alternative to the OpenAI text-to-speech model, the OpenAI model is currently employed for text-to-speech, even in the event that another assistant, such as the Mixtral assistant, is selected.

3.2.4 Code diagram

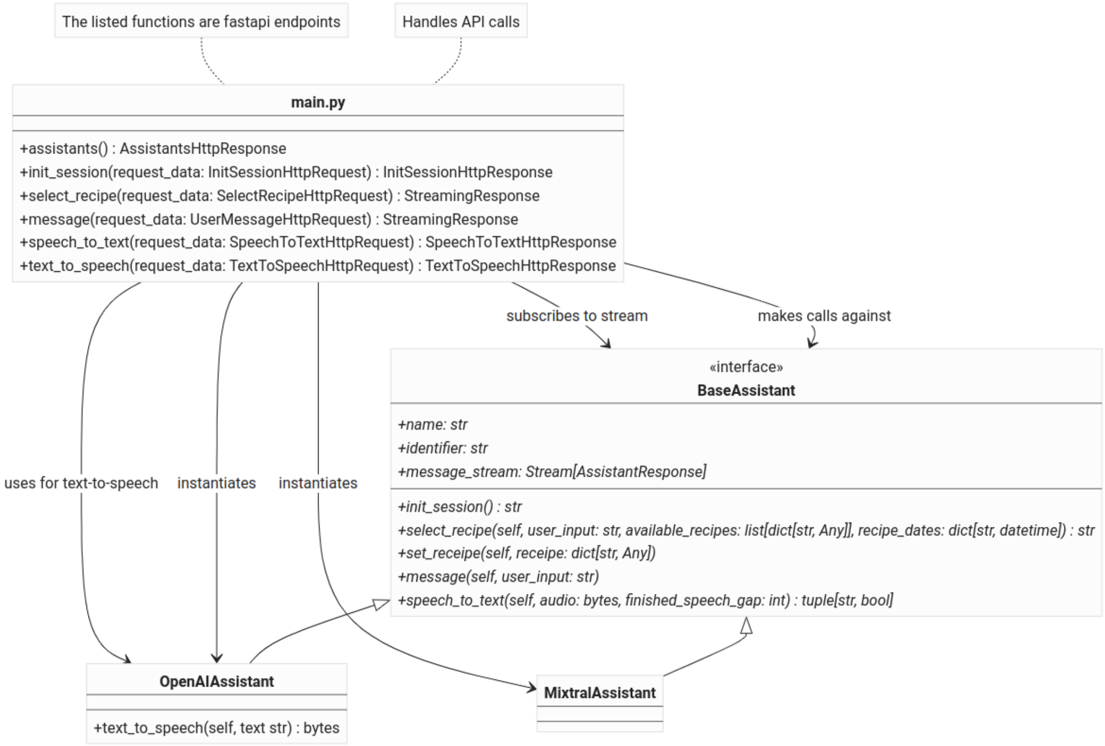

The listed functions are fastapi endpoints
Handles API calls
main.py
+assistants() : AssistantsHttpResponse
+init_session(request_data: InitSessionHttpRequest) : InitSessionHttpResponse
+select_recipe(request_data: SelectRecipeHttpRequest) : StreamingResponse
+message(request_data: UserMessageHttpRequest) : StreamingResponse
+speech_to_text(request_data: SpeechToTextHttpRequest) : SpeechToTextHttpResponse
+text_to_speech(request_data: TextToSpeechHttpRequest) : TextToSpeechHttpResponse
subscribes to stream
makes calls against
«interface»
BaseAssistant
+name: str
+identifier: str
+message_stream: Stream[AssistantResponse]
+init_session() : str
+select_recipe(self, user_input: str, available_recipes: list[dict[str, Any]], recipe_dates: dict[str, datetime]) : str
+set_receipe(self, receipe: dict[str, Any])
+message(self, user_input: str)
+speech_to_text(self, audio: bytes, finished_speech_gap: int) : tuple[str, bool]
uses for text-to-speech
instantiates
instantiates
OpenAIAssistant
+text_to_speech(self, text str) : bytes
MixtralAssistant

Figure 3.7: High-level Code Diagram

Now let's have a closer look at the assistant implementations.

OpenAI Assistant Code Diagram

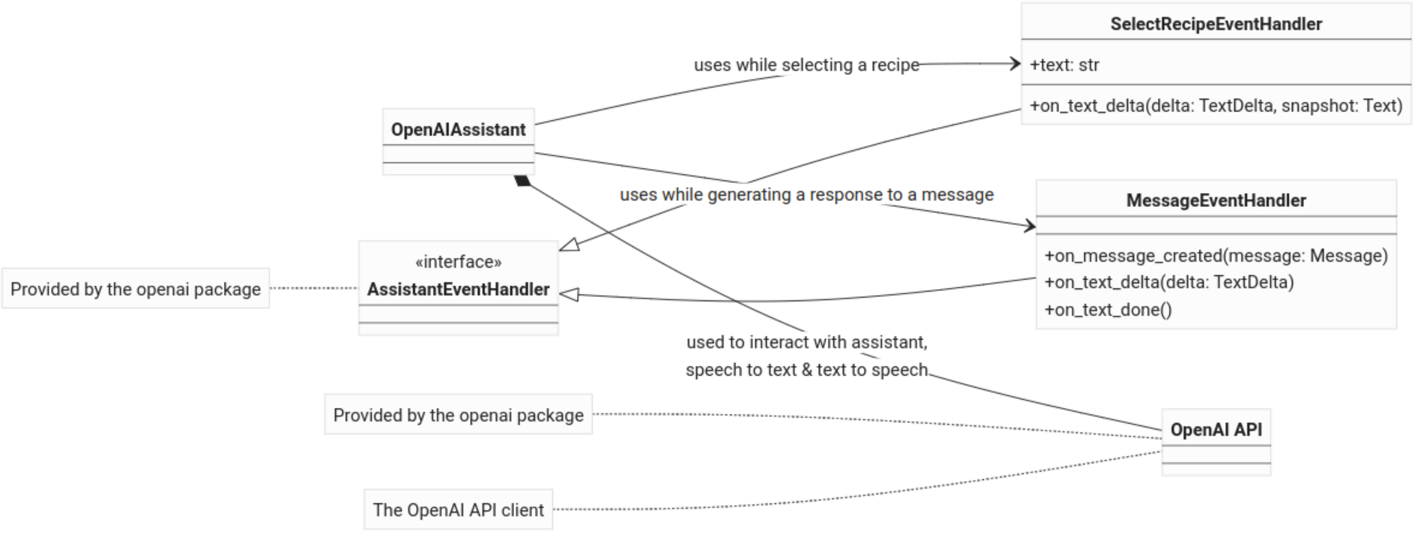

SelectRecipeEventHandler
+text: str
+on_text_delta(delta: TextDelta, snapshot: Text)
uses while selecting a recipe
OpenAIAssistant
uses while generating a response to a message
MessageEventHandler
+on_message_created(message: Message)
+on_text_delta(delta: TextDelta)
+on_text_done()
«interface»
AssistantEventHandler
Provided by the openai package
used to interact with assistant,
speech to text & text to speech
Provided by the openai package
OpenAI API
The OpenAI API client

Figure 3.8: OpenAI Assistant Code Diagram

The OpenAI assistant uses GPT-3.5 via the the OpenAI API.

Mixtral Assistant Code Diagram

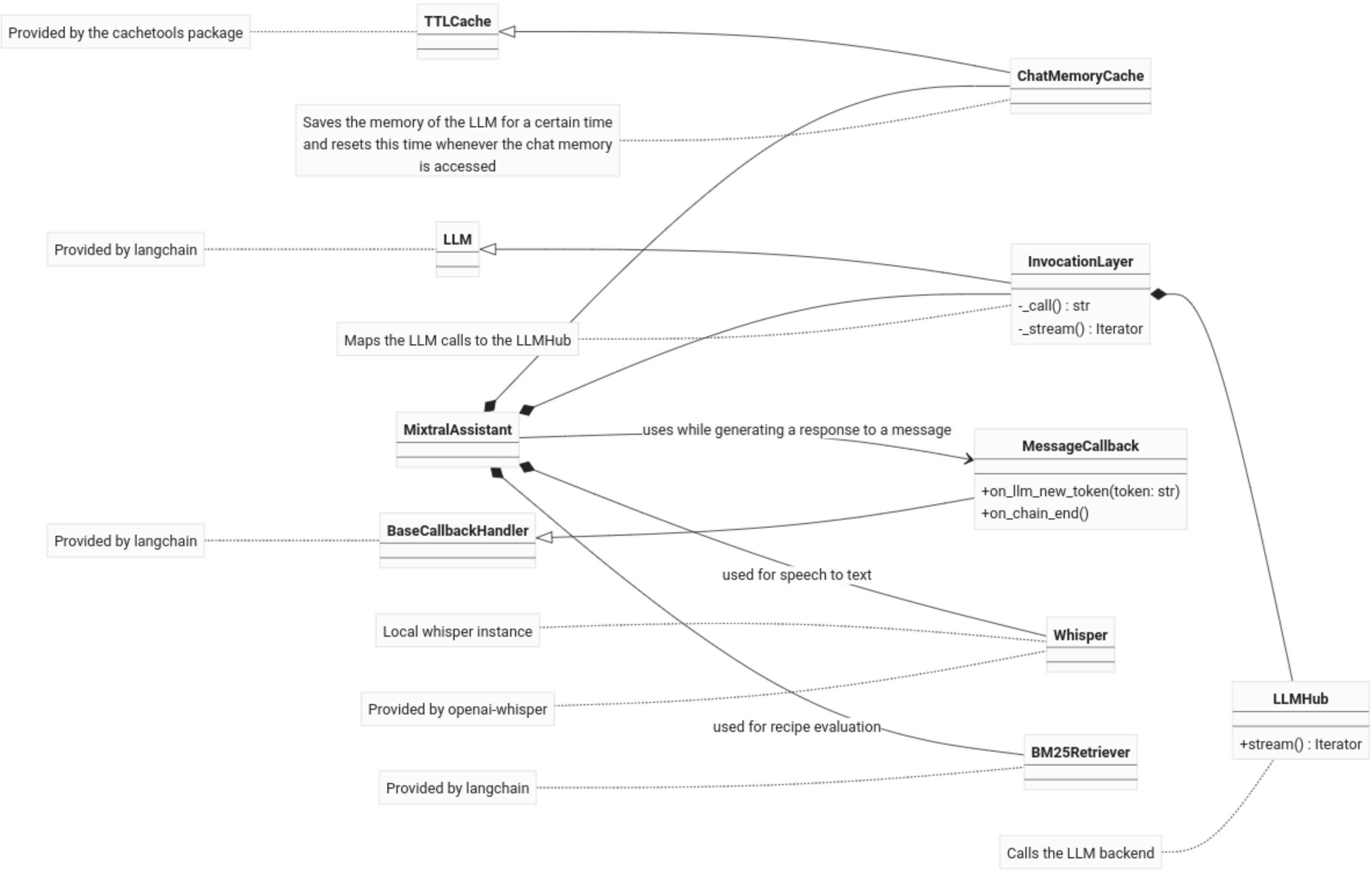


Figure 3.9: Mixtral Assistant Code Diagram

The Mixtral assistant uses the Mixtral model hosted at OST.

3.3 Data Flow

This section describes the data flow through the different containers using the different input methods.

3.3.1 Text Data Flow

The following diagram illustrates the data flow when the user communicates with text.

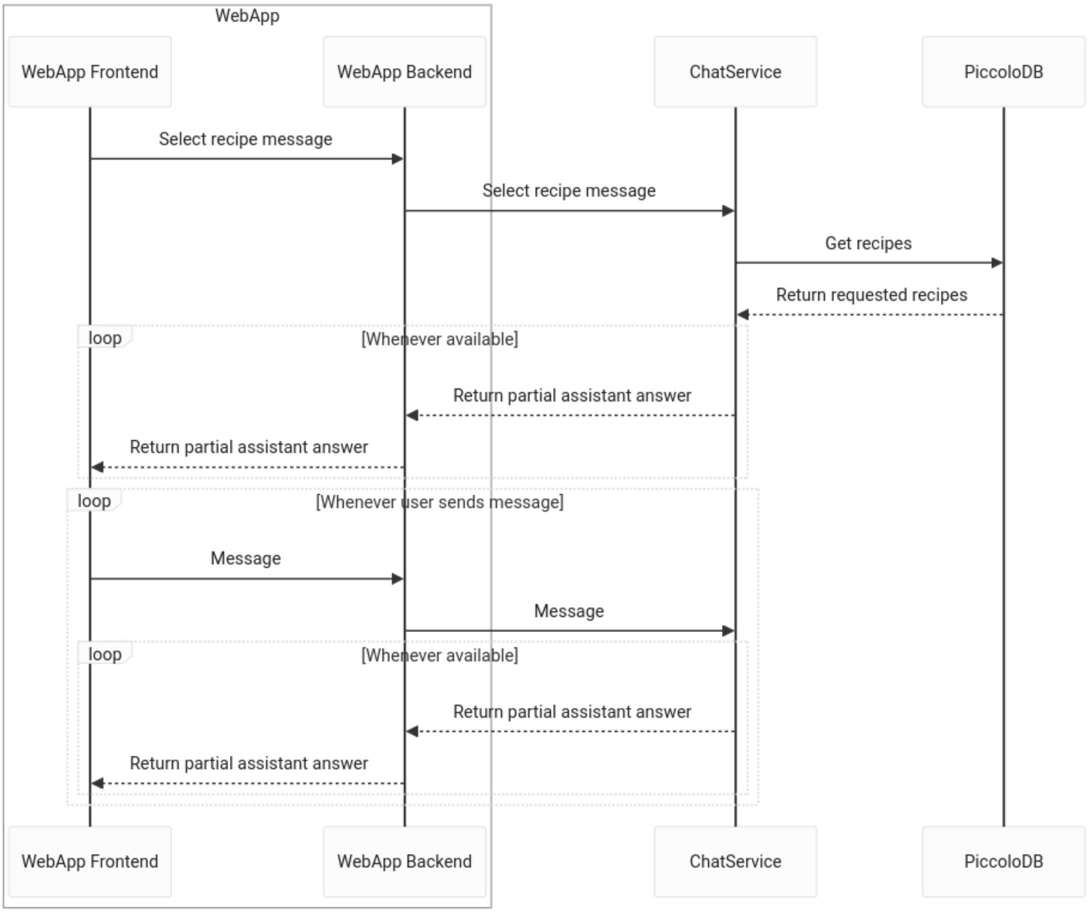
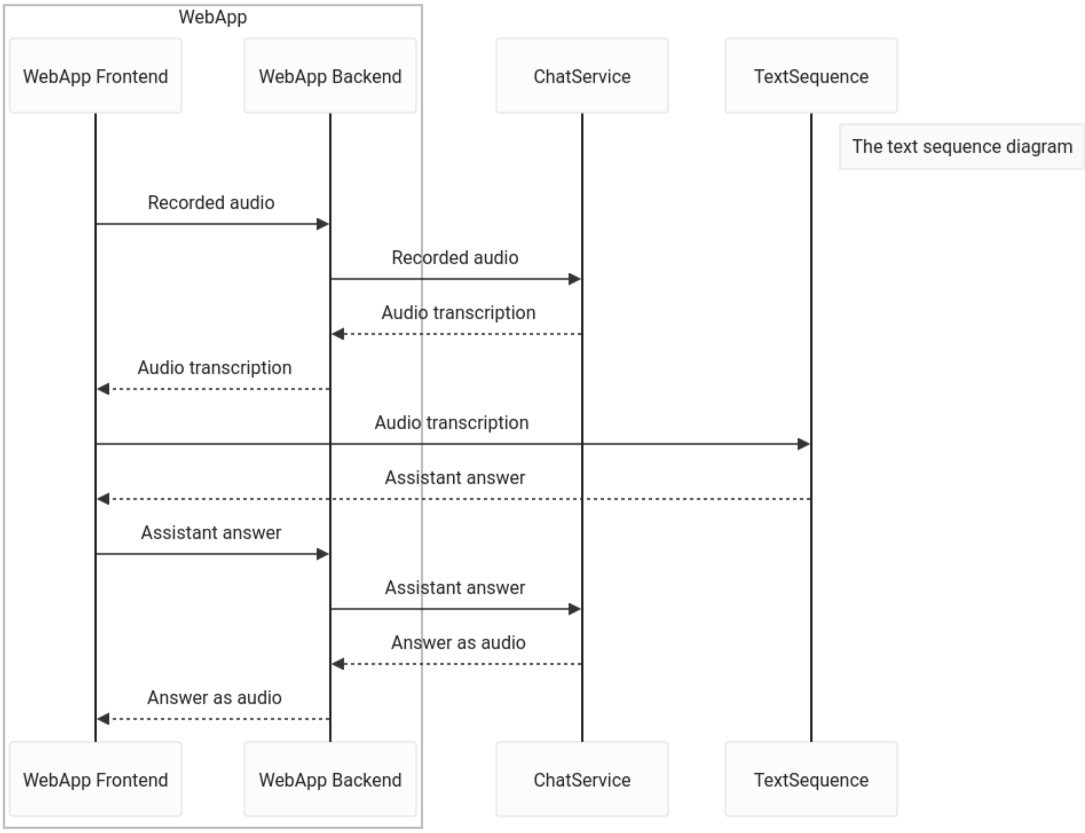


Figure 3.10: Text data flow

3.3.2 Audio Data Flow

The following diagram illustrates the data flow when the user communicates with audio, that is, when the user speaks the message instead of typing it.



At first glance, it may seem counterintuitive to transcribe the user's audio and send the transcription back to the Web application frontend. However, this approach is chosen to be able to display the transcription in the message history of the chatbot. The text-based response from the assistant is also returned to the Web application frontend for the same reason.

The effect of using the audio input method instead of the text input method can be found in the Assistant Comparison.

3.4 Toolstack

The following tools are employed and implemented in the software development process.

3.4.1 Development Process

The source code of the project is stored on the official OST Gitlab server. The documentation source code on the other hand is stored on a selfhosted instance of Gitlab. The integrated issue management system and merge request capabilities of the selfhosted instance provide a streamlined and agile workflow.

The existing project was already present on the OST Gitlab. For the documentation however, I chose Gitlab, because I am very familiar with the Gitlab ecosystem, since the used Gitlab server is a instance hosted by myself.

3.4.2 IDE

Visual Studio Code is used for React, Python and Docker development. Additionally, it is used to write the documentation.

As there are some C# components in the project, JetBrains Rider is also utilized as an IDE.

3.4.3 Frameworks & Libraries

React

The web-application's frontend is built using React.

.NET

The web-application's backend is built using .NET 7.

Python Frameworks & Libraries

openai The openai package simplifies the interaction with the OpenAI API.

LangChain LangChain is a python framework to interact with an LLM backend, such as llama.cpp. It provides built-in context management support and chains, which facilitates the usage of custom LLM's. For a justification of why I chose this tool and not haystack, another popular framework to interact with LLM backends, have a look at the Tooling evaluation.

fastapi The fastapi framework is used to easily set up an API.

uvicorn As the code must run as a web service, uvicorn is utilized.

openai-whisper Openai-whisper is a speech-to-text model and is used as an alternative to whisper accessible via the OpenAI API. This, because there may be people, which are not willing to share their voice with OpenAI. With the locally hosted whisper, this can be guaranteed.

pydub Pydub is a package to handle and analyze audio data. It is used for the speech-to-text part of the application.

cachetools The cachetools package is used to implement the ChatMemoryCache.

3.4.4 Programming Languages

JavaScript

The web-application's frontend is written in JavaScript.

C#

The web-application's backend is written in C#.

Python

Several services, including the chatbot service, are written in Python.

3.4.5 Build Tools

Docker

Docker is used to containerize all services of the application.

Gitlab CI

The Smart Eating Docker images are built using Gitlab CI. In addition, Gitlab CI is used to compile the documentation into a pdf.

3.4.6 Documentation

The documentation is written in $\LaTeX$, which makes collaboration and source control easy. Additionally, it allows to generate a pdf, which is the preferred format for reports.

Diagrams

Most of the diagrams are created as "diagrams as code" in python using the diagrams package and in markdown using mermaid. The long-term Gantt-Chart, however, is created using Microsoft Excel.

Chapter 4

Quality Measures

Quality measures are critical indicators used to evaluate the effectiveness and efficiency of software development, encompassing various metrics that assess the software's functionality, performance, security, maintainability, and other key factors.

4.1 Guidelines & Tools

The guidelines ensure the readability of the code as well as clean code.

4.1.1 LaTeX formatting

Most of this section is adopted from my Studienarbeit [1] and only slightly modified for this project.

I did not use any official LaTeX formatting rules. However, I used my own latex formatting rules to ensure consistent formatting throughout the document, as well as an enhanced readability of the LaTeX content. These rules are:

- Each sentence must be on a new line.
- The content belonging to the title must be one intendation more than the title itself.
- For intendation, spaces are used, not tabs.
- One intendation is 4 spaces.
- Before a title, there must be an empty line.
- Each file must end with an empty line.
- Subitems are created with an additional itemize.
- Between the end of an itemize for subitems and the next item, an empty line must be present. Excluded are the meeting notes.

- The itemize for the subitems must use one intendation more than the corresponding item
- If a file gets too big ($\geq$ 50 lines), it may be worth to outsource and break down the file content into multiple files.

4.1.2 Code

React

The React guidelines of PillarStudios are used. However, if there are deviations in the already used coding guidelines, they will be documented here.

ESLint The de-facto standart Linter for JavaScript code is ESLint. It is already used in Smart Eating project.

C#

The C# code must comply with the common C# code conventions. However, if there are deviations in the already used coding guidelines, they will be documented here.

ReSharper ReSharper is a well-known productivity extension, integrated into JetBrains Rider. It provides various features like quick fixes and code refactoring following defined coding conventions.

Python

The official Python coding guideline is used.

4.1.3 Definition of Done

The definition of done defines, when an issue can be considered as resolved.

- Acceptance criteria
    - All tasks belonging to the issue have been completed.
    - The feature is ready to demonstrate.
    - The feature complies with functional and non-functional requirements.
    - In the code review, no major issues were found and the recommendations from the reviewer(s) have been implemented.
- Quality
    - The code complies with the defined code guidelines
- Documentation
    - All necessary information has been documented.

4.1.4 Git

To allow a clean and fast workflow, some guidelines regarding git are defined.

Branching

Each task must have its own branch, allowing for later traceability of the changes. The branch names of the report must follow this pattern: `${issue_nr}-${description_of_task}`. As an example, `28-initial_architecture` is used for the branch addressing the initial architecture.

As there are diffrent Gitlab instances used for the issue tracking and code repository, branches in the latter must follow this pattern: `${description_of_task}`.

Additionally, merging without a merge request and pushing the master branch directly is not allowed. This is enforced by policies in both, the project repository and documentation repository.

Merges

To minimize errors be merged into the master, a merge request must be opened and the code must be carefully examined by a reviewer. For the project, another project member must review the changes, whereas in the report, I review my changes, as I do not have another team member.

Additionally, the code must be checked for compliance with the code guidelines.

4.1.5 Sprint Retrospective

To reevaluate the quality of the project, each sprint is ended with a review meeting, which also contains the retrospective.

4.2 Build Tools

4.2.1 Smart Eating Project

Each push to the repository executes a CI pipeline, which in turn builds docker images for all project components. These docker images are then published to the Gitlab internal registry.

4.2.2 Documentation

Each push to the documentation repository triggers a build of the documentation. As many diagrams are diagrams-as-code, these diagrams are built first and then consumed when building the documentation. This CI pipeline compiles LaTeX into a pdf file.

4.3 Test Strategy

Various testing types are utilized to ensure a high level of software quality in diffrent aspects.

4.3.1 Unit Tests

Testing of individual units of code, such as methods, to ensure they are working as expected and that changes do not introduce bugs. However, it is not foreseeable at the moment if there even is unit-testable code within this project. If there is, this section provides information how to write the unit tests.

- Execution: Automated
- Time of Execution: With every push to the repository and locally as desired

F.I.R.S.T principles

Each unit test has to conform with these principles.

- **Fast**: Unit tests must be fast, meaning they must execute quickly so that they can be run frequently during development.
- **Independent**: Unit tests must be independent of one another, meaning that the outcome of one test must not affect the outcome of another. This ensures that each test is testing a specific and isolated unit of code.
- **Repeatable**: Unit tests must be repeatable, meaning that they must always produce the same result every time they are run with the same preconditions.
- **Self-validating**: Unit tests must be self-validating, meaning that they must be able to automatically determine if they have passed or failed without human intervention. This ensures that tests can be run as part of the continuous integration process.
- **Timely**: Unit tests must be written in a timely manner, meaning that they must be written before or directly after the code they are testing is implemented, depending on the testing technique used. This ensures that the code is written with testability in mind and helps finding and mitigating bugs early in the development process.

Arrange-Act-Assert Pattern

Most of this section is adopted from my Studienarbeit [1] and only slightly modified for this project.

To structure the tests, the commonly used AAA pattern is utilized.

- **Arrange**: During the arrange phase, the necessary prerequisites for the test are set up. This involves creating any required objects, initializing variables, and configuring any dependencies that the code under test relies on.

- **Act**: During the act phase, the test executes the action being tested. This could involve calling a specific method or interacting with an object.
- **Assert**: In this phase, the test verifies that the action performed in the Act phase has produced the expected result. This involves comparing the actual result of the action with the expected result and failing the test if these do not match.

4.3.2 Acceptance Tests

The acceptance tests ensure, that the designed workflow generates the expected behavior and outputs. These tests, alongside with their results, can be found in the testing protocols.

- Execution: Manual
- Time of Execution: Before the release, during the Testing & Bugfixing period

4.3.3 UI Tests

The UI tests ensure, that the UI behaves as expected. These tests, alongside with their results can be found in the testing protocols.

- Execution: Manual
- Time of Execution: Before the release, during the Testing & Bugfixing period

4.3.4 LLM Behavior Tests

Although these tests could be considered acceptance tests, it makes sense to separate them into their own test section, since LLM's always have some uncertainty in their responses. Therefore, the acceptance tests test the coded side of the product, while the LLM behavior tests test the LLM's responses. These tests, alongside with their results can be found in the testing protocols

- Execution: Manual
- Time of Execution: Before the release, during the Testing & Bugfixing period

4.3.5 Testing Frameworks

React

While it is possible to write unit tests for React components, we decided against it in this project, as there are currently no tests available for the React components and therefore, a test project must be set up from scratch. Additionally, in this project I will implement only a very small portion of code in React and testing these components doesn't make sense in terms of the cost-benefit ratio.

.NET

The web-application's backend currently uses MSTest as testing framework. Therefore, this will be used if there is unit-testable code in the web-application's backend.

Python

In python, the de-facto standard framework for testing pytest is used if there is unit-testable code.

4.4 Assistant Comparison

The implemented assistants are compared against each other based on test results. This comparison can be found in the Assistant Comparison.

Chapter 5

Limitations

This chapter describes the limitations of the Smart Eating chatbot.

5.1 Audio limitations

For optimal performance, users must speak loudly and clearly when interacting with the chatbot. This ensures accurate audio transcription and reduces the likelihood of misinterpretations.

The assistants may struggle to transcribe the audio recorded in noisy environments effectively. For the best experience, users should use the application in a quiet setting to avoid background noise interference.

A quiet environment is also required for the assistants to pick up when the user stops speaking.

5.2 Recipe selection limitation

Once a recipe is selected within the chatbot, it cannot be changed.

5.3 Language limitations

The chatbot currently supports only English for both text messages and audio messages. Therefore, users should only interact with the chatbot in English.

5.4 Assistant limitations

While the OpenAI assistant completes all tasks without any problems, the Mixtral assistant struggles with some tasks. Specifically, the Mixtral assistant hallucinates ingredient quantities and does not provide step-by-step instructions, even when asked. More information about this can be found in the LLM behavior tests.

Part III

Project Documentation

Chapter 6

Initial Project Description

6.1 Supervisor

Mitra Purandare (mitra.purandare@ost.ch)

6.2 Student(s)

Andri Joos

6.3 Goal

This project builds on the Smart Eating platform under development at the Institute For Software. Smart Eating is a web service for creating meal plans satisfying user's nutritional requirements, allergies, as well as likes and dislikes. It also takes user's lifestyle constraints into account to come up with a meal plan.

In a next step, we would like to extend the application by helping the user via a cooking assistant that can guide the user during the cooking process. The user can ask for next steps, list of ingredients, and many other cooking related questions. Additionally, the user can select the receipe via the chatbot. The end goal is to enable the user voice chat with the assistant freeing the hands of the user from typing in text.

6.4 Assignment

The following subtasks must be completed

- Analysis
  - Establish functional requirements (necessary und optional)
  - Non-functional requirements (NFR)

- Architecture & Design
  - Architecture of the chat service (Saving the chat, API interface)
  - Analysis Libraries und Frameworks (Langchain etc.)
  - Option to choose chat assistant models
    - GPTAgent from Open AI
    - Opensource LLM service (Llama, Mixtral, Phi)
  - Comparsion of the response of the models
  - Data base choice
- Implementation & Test
  - Implementation of the functional requirements
  - Documentation
  - Test concept
  - Possibilities of extension

6.5 Deliverables

- A functional chat bot
- Support for OpenAI assistant as well as open source LLM models
- (Optional but desired) Voice Assistant

6.6 Dates

- 19.02.2024: Start
- 10.06.2024: Submission Abstract
- 14.06.2024, 17:00: Submission of all required documents
- 14.06.2024, 16:00: Start presentations

6.7 Grading

The supervisor is responsible for the assessment. The weighting of the assessment follows the guidelines for SA/BAs.

6.8 Support

If necessary, support will be provided in the areas of Smart Eating onboarding and LLM servers.

Chapter 7

Project Plan

The project plan outlines the resources, roles and processes required for the successful completion. It goes into detail which risks I expect and how I have laid out the long term plan.

7.1 Resources

Most of this section is adopted from my Studienarbeit [1] and only slightly modified for this project.

7.1.1 People

- Andri Joos

I am currently studying Computer Science with a specialization in Data Science & Machine Intelligence at the OST University of Applied Sciences. This project is part of my bachelor's thesis.

While I have a foundational understanding of Artificial Intelligence, my knowledge is limited to the courses AI Foundations and AI Applications, as well as my Studienarbeit in the field of reinforcement learning. However, I am not very familiar with Large Language Models (LLMs).

I also have a good understanding of Git and source control in general. I have gained experience in DevOps both in a business environment and in several private projects.

Additionally, I have been a part of a Scrum team in both a business environment and in the Software Engineering Project. Throughout the course, I gained the necessary understanding of project management, project documentation, and project execution.

I also have a lot of knowledge in the area of Docker, as I have built a homeserver infrastructure on Kubernetes and Docker. For this and also for Gitlab CI pipelines I had to create several docker images myself.

I am also proficient in Python and C#, as I have used them in several private projects, school projects and business projects. However, I have not yet used React.

7.1.2 Time

The project started with the kickoff meeting on February 19th 2024, 15:00 and will end on June 14th 2024, 17:00 with the final submission of the report. This, alongside with the later described milestones, are deadlines which have to be met. For the whole project, investing 360 hours per person is required. Since the project must be finished by June 14th 2024, these hours must be met in 15 weeks, where one week is holidays. Accordingly, the required time is divided into 14 weeks which results in 25 hours per week per person.

7.1.3 Costs

As this is a bachelor thesis, I do not have a budget which can be expressed in money. However, the available budget can be expressed as the earlier mentioned 360 hours, that I can use for this project.

The costs for services, such as an OpenAI API subscription, will be covered by the university OST.

7.1.4 Tooling

The main IDE for the project and the documentation is Visual Studio Code, as it's free, open source and covers all needs. For C# code, JetBrains Rider is used as it provides many useful features.

For version control, Git is used. As UI for Git, gitextensions is used. To store the source code, the official OST Gitlab instance is used as the project already lives there. The Continuous Integration of this server is used to automatically build and test the application on pushes. To store the source code for the documentation a repository on a selfhosted instance of Gitlab is used. The issue tracking system of the before mentioned server will be used to keep track of the work and the time. To collect and evaluate the time spent, gtt is used.

The documentation is written in LaTeX.

Microsoft Teams is used for the communication with the Advisor and the Smart Eating team.

7.2 Roles

Most of this section is adopted from my Studienarbeit [1] and only slightly modified for this project.

Since there is only one team member, all of the following roles will be executed by Andri Joos. He is responsible to carry out the project successfully.

- Project Manager
- Developer
- Product Owner (in agreement with the project advisor)
- DevOps-Engineer
- Software Architect

7.3 Processes & Meetings

Most of this section is adopted from my Studienarbeit [1] and only slightly modified for this project.

To achieve an agile workflow, I use Scrum to define processes in this project.

7.3.1 Backlogs

Product Backlog

The Product Backlog is realized in Excel.

Sprint Backlog

The Sprint Backlog is realized in GitLab. Product Backlog items will be refined and transformed into GitLab Issues during Planning.

7.3.2 Sprint

Sprint information:

- Sprint duration: 2 weeks
- Sprint start: Monday
- Total amount of Sprints: 7

Planning

Each Sprint is started by a planning meeting.

Planning procedure:

1. Specify what should be achieved during this Sprint
2. Evaluate which Product Backlog items should be put into the Sprint Backlog

Review

Each Sprint end is initiated by a Review. Since it doesn't make sense for such a small team to also do a separate Retrospective meeting, I will do both, the Review and the Retrospective, in one single Meeting.

Review procedure:

1. Progress evaluation of the project
2. Adjust long term plan

Retrospective procedure:

1. Documentation Quality
    - Check if the documentation if properly formatted
    - Check for grammatical errors
    - Check if anything is missing or should be improved
2. Time Tracking Quality
    - Check if the estimations for the tasks are good enough
3. Risk Reevaluation
    - Reevaluation of all current risks
    - Check for any new risks
4. Code Quality
    - Check for bad code
5. Git Quality
    - Check for leftover branches
    - Check for open merge requests

Stakeholder meeting

This meeting is about the current state of the project and to discuss any problems. The meeting is attended by the project advisor and myself.

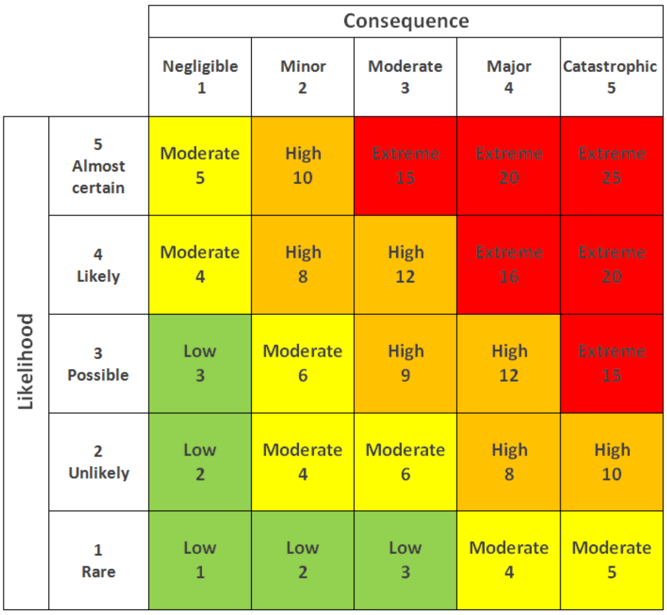
During this meeting, the current progress is presented.

7.4 Risk Management

Risk management is a critical process that involves identifying potential risks and developing strategies to mitigate or eliminate them.

7.4.1 Risk categorization method

To rate the risks, they are categorized and assigned a value using the risk matrix below. The respective values can be found right next to the title of the risk, likelihood will be referred to as "L" and consequence as "C".



		Consequence				
		Negligible 1	Minor 2	Moderate 3	Major 4	Catastrophic 5
Likelihood	5 Almost certain	Moderate 5	High 10	Extreme 15	Extreme 20	Extreme 25
	4 Likely	Moderate 4	High 8	High 12	Extreme 16	Extreme 20
	3 Possible	Low 3	Moderate 6	High 9	High 12	Extreme 15
	2 Unlikely	Low 2	Moderate 4	Moderate 6	High 8	High 10
	1 Rare	Low 1	Low 2	Low 3	Moderate 4	Moderate 5

Figure 7.1: Risk Matrix

7.4.2 Familiarization with existing project | 3 (L: 1, C: 3)

Description

The familiarization with the existing project could take some time.

Mitigation

For support, I can ask people, which have built the system.

7.4.3 Lack of experience with OpenAI assistants | 8 (L: 2, C: 4)

Description

As I have never used OpenAI assistants, getting started with them could be time consuming.

Mitigation

I have already planned to familiarize myself with OpenAI assistants by building a prototype.

7.4.4 Lack of experience with OpenAI API | 4 (L: 1, C: 4)

Description

I have never used the OpenAI API. Therefore it could be time consuming to understand how to use it effectively.

Mitigation

I have already planned to familiarize myself with the API by building a prototype.

7.4.5 OpenAI model training takes too much time | 1 (L: 1, C: 1)

Description

The GPTAgent probably needs some fine-tuning. This task will take some time, but the amount of time is hard to predict.

Mitigation

During planning, this must be taken into account.

7.4.6 In-house models not working | 2 (L: 2, C: 1)

Description

I have never worked with the OST in-house models, but as my advisor informed me, they are not fine-tuned to handle receipes.

Mitigation

If the in-house models are not working, that's a great pity. However, fine-tuning the models is not in the scope of this project.

7.4.7 Lack of experience in the in-house models API | 1 (L: 1, C: 1)

Description

I have never used the API to interact with the models.

Mitigation

I can get support from people, which are familiar with the API.

7.4.8 Gitlab failure | 2 (L: 1, C: 2)

Description

Since a selfhosted Gitlab is used to store the code for the documentation, there is a chance that the server is unavailable for some reason.

Mitigation

The server runs on a Kubernetes cluster and therefore should be restarted as soon as a crash happens. Additionally, accessing the network of the server remotely is possible in case the Gitlab instance needs to be restarted manually.

7.4.9 Specification of requirements/features is inaccurate | 3 (L: 1, C: 3)

Description

At the start of the project it's hard to perfectly define all requirements as it's unclear how the application will look like in the end.

Mitigation

The agile methodology Scrum is used to assure that requirements will be adjusted to the current situation during development.

7.4.10 Time management | 6 (L: 2, C: 3)

Description

Due to unexpected events, such as illness, I may be unable to invest the required time each sprint. There may be also some expected events, where I am unable to invest the required time.

Mitigation

The time lost must be made up for. Expected time loss must be noted during the sprint planning. The planning is done accordingly.

7.4.11 Changes History

Risk	Value change	Reason for change	Date
Familiarization with existing project	9 → 6	Architecture was thoroughly investigated	18.03.2024

Lack of experience with OpenAI assistants	20 → 8	Prototype	18.03.2024
Lack of experience with OpenAI API	20 → 8	Prototype	18.03.2024
Lack of experience with OpenAI API	8 → 4	OpenAI API is now known	03.04.2024
OpenAI model training takes too much time	12 → 1	The OpenAI assistant is good enough with finetuning the instructions, no specifically trained model needed	24.04.2024
Lack of experience in the in-house models API	12 → 4	There is someone working in the Smart Eating project, that can help me in case any clarification or help is needed	24.04.2024
Familiarization with existing project	6 → 3	I'm pretty familiar with the project	06.05.2024
In-house models not working	4 → 2	Mixtral model works very well	06.05.2024
Lack of experience in the in-house models API	4 → 3	Gained experience with the API for the in-house models	06.05.2024
Lack of experience in the in-house models API	4 → 1	I'm now quite familiar with this API	21.05.2024
Specification of requirements/features is inaccurate	9 → 6	As the project is almost finished, there are not many features left to do	21.05.2024
Specification of requirements/features is inaccurate	6 → 3	No new requirements/features	04.06.2024

7.5 Long-term Plan

Most of this section is adopted from my Studienarbeit [1] and only slightly modified for this project.

All the listed items are presented in a Gantt chart. This is attached at the end of this section.

7.5.1 Phases

I will be working according to the Rational Unified Process (RUP). This includes following phases:

- Inception (19.02.2024 - 04.03.2024)
- Elaboration (04.03.2024 - 18.03.2024)
- Construction (18.03.2024 - 03.06.2024)
- Transition (03.06.2024 - 14.06.2024)

7.5.2 Features

Primary Features

Features which are guaranteed to be implemented in the available time of the project.

- Chatbot UI
- Communication with chatbot via text
- Communication with chatbot via audio
- Ask chatbot questions about receipe, chatbot answers
- Select a receipe through chatbot
- Chatbot guides through cooking process
- Assistant model GPTAgent
- Assistant model in-house
- Model quality comparison

Secondary Features

Bonus features which will be implemented if there is some time left.

- All in-house models as assistants
- Saving chat over sessions
- Receipe suggestions
- Noise cancelling for audio

7.5.3 Epics

- Information about cooking process
- Guidance through cooking process
- Selection of assistants
- Assistant interaction

7.5.4 Milestones

- M1 - Initial Project setup (04.03.2024)
    - Creation of project documentation

    - Setup of build pipelines for project documentation
    - Setup of time tracking
    - Setup of Issue tracking
    - Defining collaboration, roles, responsibilities and meetings
    - Creation of a long-term plan
    - Risk-Manangement
    - Requirement analysis
        - Functional requirements
        - Non-functional requirements

- M2 - End of elaboration (18.03.2024)
    - Phase of Elaboration is completed and documented
    - Refined requirements are sorted according to priority
    - Architecture is defined
    - Creation of prototype is done
    - All preparations for implementation are completed
    - Quality measures are defined
- M3 - Alpha version (20.05.2024)
    - All primary features are implemented rudimentary
    - Documentation is up-to-date with the technical solution
- M4 - Final version (03.05.2024)
    - Severe bugs are mitigated
- M5 - Final Submission (14.06.2024)
    - Finished documentation
    - Finished application

Timeplan

		Sprint	Sprint 1		Sprint 2		Sprint 3		Sprint 4		Sprint 5		Sprint 6		Sprint 7		Sprint 8	
		SW	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Inception	Setup of Issue tracking / Time tracking	*Estimate*	🟨															
		Actual	🟩															
	Setup of Build pipelines	*Estimate*	🟨															
		Actual	🟩															
	Setup report	*Estimate*																
		Actual																
	Risk-Management	*Estimate*		🟨														
		Actual		🟩														
	Defining collaboration, roles, processes and meetings	*Estimate*		🟨														
		Actual		🟩														
	Creation of long-term plan	*Estimate*		🟨														
		Actual		🟩														
	Requirement Analysis	*Estimate*	🟨	🟨														
		Actual	🟩															
Elaboration	Refine Requirements	*Estimate*			🟨													
		Actual		🟩														
	Define initial architecture	*Estimate*			🟨													
		Actual			🟩	🟩												
	Test concept & quality measures	*Estimate*			🟨	🟨												
		Actual			🟩	🟩												
	POC / Prototype (cli to talk to GPTAgent)	*Estimate*			🟨	🟨												
		Actual			🟩													
Construction	Chatbot UI	*Estimate*					🟨											
		Actual					🟩											
	Assistant model selection	*Estimate*					🟨											
		Actual					🟩											
	OpenAI assistant	*Estimate*						🟨	🟨									
		Actual						🟩	🟩									
	Communicate via text	*Estimate*								🟨	🟨							
		Actual							🟩	🟩	🟩							
	In-house LLM	*Estimate*										🟨						
		Actual									🟩	🟩						
	Select receipe	*Estimate*										🟨	🟨					
		Actual						🟩			🟩		🟩					
	Ingredients of receipe	*Estimate*										🟨	🟨					
		Actual											🟩					
	Total time of receipe	*Estimate*											🟨					
		Actual												🟩				
	Next step	*Estimate*											🟨	🟨				
		Actual											🟩	🟩				
	Communication type voice	*Estimate*											🟨	🟨				
		Actual											🟩	🟩				
	Testing & Bugfixing	*Estimate*													🟨	🟨		
		Actual												🟩	🟩			
	LLM comparison	*Estimate*													🟨	🟨		
		Actual													🟩	🟩		
	Further in-house LLMs	*Estimate*																
		Actual																
	Remaining time	*Estimate*																
		Actual																
	Steps of cooking process	*Estimate*																
		Actual																
	Infos about previous step	*Estimate*																
		Actual																
	Saving the chat	*Estimate*																
		Actual																
	Receipe suggestions	*Estimate*																
		Actual																
Transition	Finalize Product	*Estimate*															🟨	
		Actual															🟩	
	Finalize Documentation	*Estimate*															🟨	🟨
		Actual															🟩	🟩
		Milestones		M1		M2								M3		M4		M5
		SW	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
		Sprint	Sprint 1		Sprint 2		Sprint 3		Sprint 4		Sprint 5		Sprint 6		Sprint 7		Sprint 8	

Legend: 🟨 Estimated time 🟩 Actual time 🟧 Milestones

Part IV

Bibliography

Bibliography

[1] Andri Joos. *Training a simulated drone with deep reinforcement learning.* `https://eprints.ost.ch/id/eprint/1177/`. Accessed: 21.02.2024. 2024.

Part V

Appendix

Chapter 8

Testing Protocols

This chapter includes protocols of the executed tests. For each test, the preconditions must be satisfied in order to ensure a clean and expected environment.

8.1 UI tests

Precondition: Empty chatbot is open, input type is text

ID	Description	Steps	Expected result	Actual result	Status	Issue nr.
1	Audio input can be started	Start audio recording	Chatbot should indicate, that the recording is running	As expected	succeeded	
2	Audio input can be stopped	Stop audio recording	Chatbot should indicate, that the recording is not running	As expected	succeeded	
3	No assistant is selected by default		No assistant is selected by default	As expected	succeeded	
4	Assistant change is visible	Open assistant selection, select an assistant	Selected assistant should change to the one chosen	As expected	succeeded	

8.2 Acceptance Tests

8.2.1 OpenAI assistant

Precondition: Empty chatbot is open, input type is text, OpenAI assistant is selected

ID	Description	Steps	Expected result	Actual result	Status	Issue nr.
1	Sending a text message adds a new message to message history	Create message, send message	Sent message shows up in message history	As expected	succeeded	
2	Sending a text message generates response	Create & send message, wait for response	The response from the assistant is displayed in the message history	As expected	succeeded	
3	Messages are not lost when switching input modes	Create & send message, turn on audio recording, create & send message, end audio recording	The first message & response show up in message history	As expected	succeeded	
4	Audio recording automatically stops in good conditions after a few seconds of silence	Start audio recording speak something, be silent for a few seconds	The audio recording is automatically stopped, a response is generated	As expected	succeeded	
5	Audio recording can be manually stopped	Start audio recording, speak something, stop voice input	The audio recording stops, a response is generated	As expected	succeeded	
6	Response from audio recorded message is spoken	Start audio recording, speak something, stop audio recording, wait for generated response	The generated response is spoken	As expected	succeeded	
7	Audio recording restarts automatically if stopped automatically	Start audio recording, speak something, be silent for a few seconds, wait until response is generated	Audio recording is restarted automatically	As expected	succeeded	
8	Audio recording does not restart automatically if stopped manually	Start audio recording, speak something, stop audio recording manually, wait until response is generated	Audio recording not restarted automatically	As expected	succeeded	

8.2.2 Mixtral assistant

Precondition: Empty chatbot is open, input type is text, Mixtral assistant is selected

ID	Description	Steps	Expected result	Actual result	Status	Issue nr.
1	Sending a text message adds a new message to message history	Create message, send message	Sent message shows up in message history	As expected	succeeded	
2	Sending a text message generates response	Create & send message, wait for response	The response from the assistant is displayed in the message history	As expected	succeeded	
3	Messages are not lost when switching input modes	Create & send message, turn on audio recording, create & send message, end audio recording	The first message & response show up in message history	As expected	succeeded	
4	Audio recording automatically stops in good conditions after a few seconds of silence	Start audio recording speak something, be silent for a few seconds	The audio recording is automatically stopped, a response is generated	As expected	succeeded	
5	Audio recording can be manually stopped	Start audio recording, speak something, stop voice input	The audio recording stops, a response is generated	As expected	succeeded	
6	Response from audio recorded message is spoken	Start audio recording, speak something, stop audio recording, wait for generated response	The generated response is spoken	As expected	succeeded	
7	Audio recording restarts automatically if stopped automatically	Start audio recording, speak something, be silent for a few seconds, wait until response is generated	Audio recording is restarted automatically	As expected	succeeded	
8	Audio recording does not restart automatically if stopped manually	Start audio recording, speak something, stop audio recording manually, wait until response is generated	Audio recording not restarted automatically	As expected	succeeded	

8.3 LLM Behavior Tests

8.3.1 OpenAI assistant

Precondition: Empty chatbot is open, input type is text, OpenAI assistant is selected

ID	Description	Steps	Expected result	Actual result	Status	Issue nr.
1	Correct recipe is evaluated and selected	Send a message like “I wanna to the [recipe name]”, wait for the response	The response indicates, that the correct recipe has been selected	As expected	succeeded	
2	Ingredients are listed	Select a recipe, wait for the response, send a message like “Which ingredients do I need?”, wait for the response	The last response contains all the necessary ingredients	As expected	succeeded	
3	Ingredients are listed with the amount needed	Select a recipe, wait for the response, send a message like “How much of the ingredients do I need?”, wait for the response	The last response contains all the necessary ingredients with the amount needed	As expected	succeeded	
4	Total time of recipe	Select a recipe, wait for the response, send a message like “How long does this recipe take?”, wait for the response	The last response contains the total time of the recipe	As expected	succeeded	
5	Step-by-step guidance	Select a recipe, wait for the response, send a message like “Guide me through the recipe”, when the response for the first step is generated, send a message like “Done, continue”, repeat the last step until the response states, that the recipe is finished	The assistant successfully guides through the recipe step-by-step	As expected	succeeded	
6	Infos about previous step	Select a recipe, wait for the response, send a message like “Guide me through the recipe”, when the response for the first step is generated, send a message like “Done, continue”, wait until the response is generated, send a message, which refers to the previous response, like “How much salt do I need in the previous step?”, wait for the response	The assistant answers correctly	As expected	succeeded	
7	Remaining time	Select a recipe, wait for the response, send a message like “Guide me through the recipe”, when the response for the first step is generated, send a message like “How much time do I need from now on?”	The stated time in the last response is lower than the total time of the recipe	The total time of the recipe was 100 minutes. The first step took 60 minutes. The chatbot said, that the remaining time after the first step is 80 minutes. The expected result is met, but the answer is not entirely correct.	succeeded	

Be aware, that these tests cover only a small subset of the assistant’s capabilities. However, the tests are intended to cover the most likely use cases.

8.3.2 Mixtral assistant

Precondition: Empty chatbot is open, input type is text, OpenAI assistant is selected

ID	Description	Steps	Expected result	Actual result	Status	Issue nr.
1	Correct recipe is evaluated and selected	Send a message like "I wanna to the [recipe name]", wait for the response	The response indicates, that the correct recipe has been selected	As expected	succeeded	
2	Ingredients are listed	Select a recipe, wait for the response, send a message like "Which ingredients do I need?", wait for the response	The last response contains all the necessary ingredients	As expected	succeeded	
3	Ingredients are listed with the amount needed	Select a recipe, wait for the response, send a message like "How much of the ingredients do I need?", wait for the response	The last response contains all the necessary ingredients with the amount needed	Tested with Panzerotti. Makes up some ingredient quantities (e.g. recipe: flour: 300g, chatbot: flour: 500g)	failed	Not resolvable at the moment
4	Total time of recipe	Select a recipe, wait for the response, send a message like "How long does this recipe take?", wait for the response	The last response contains the total time of the recipe	As expected	succeeded	
5	Step-by-step guidance	Select a recipe, wait for the response, send a message like "Guide me through the recipe", when the response for the first step is generated, send a message like "Done, continue", repeat the last step until the response states, that the recipe is finished	The assistant successfully guides through the recipe step-by-step	Works sometimes, but most of the times the assistant gives all steps at once.	failed	Not resolvable at the moment
6	Infos about previous step	Select a recipe, wait for the response, send a message like "Guide me through the recipe", when the response for the first step is generated, send a message like "Done, continue", wait until the response is generated, send a message, which refers to the previous response, like "How much salt do I need in the previous step?", wait for the response	The assistant answers correctly		Not executable, because step-by-step does not work reliably	
7	Remaining time	Select a recipe, wait for the response, send a message like "Which steps do i need for this recipe?", when the response is generated, send a message like "From step 2 how much time do i need until the dish is finished?", wait for the response	The stated time in the last response is lower than the total time of the recipe	As expected	succeeded	

Be aware, that these tests cover only a small subset of the assistant's capabilities. However, the tests are intended to cover the most likely use cases.

Chapter 9

Tooling Evaluations

This chapter contains the discussion about the advantages and disadvantages of the available tools, as well as a justification why a tool is selected in the end.

9.1 LangChain vs. Haystack

Both, LangChain and Haystack, are frameworks to simplify the interaction with an LLM backend, in our case llama.cpp.

9.1.1 LangChain

LangChain is a relatively new framework, released in October 2022 and is currently in version 0.2.

LangChain has built-in support for a variety of NLP tasks. Additionally, it has built-in support of a broad range of LLM providers, such as the OpenAI API.

A key concept of LangChain are chains, which is a sequence of components to invoke. As a basic example, we have a retriever which passes its result to an LLM. The retriever retrieves the interesting documents from a set of documents, based on retriever-specific criterias. The retrieved documents then get passed to the LLM, which can draw information from them. Particularly interesting for this project is the ConversationChain, which simplifies the usage of the LLM in a chatbot scenario. The ConversationChain also handles the memory, so the LLM can answer context-based requests.

Because of its flexibility, LangChain has a steeper learning curve.

9.1.2 Haystack

Haystack is another quite popular framework to interact with LLM backends.

It also has built-in support for a variety of NLP tasks.

The key feature Pipelines is the counterpart to the LangChain chains. Another key feature are the Agents. Particularly interesting is the Conversational Agent, which simplifies the usage of the LLM in a chatbot scenario, similar to the LangChain ConversationChain. The Conversational Agent also handles the memory, so the LLM can answer context-based requests. Unfortunately, the Agents are no longer a part of Haystack in version 2.0 and above.

As it is less flexible than LangChain, Haystack is easier to get started with.

9.1.3 Conclusion & Decision

Of course, in this evaluation, only a small subset of the available concepts and tools were discussed. However, most of the concepts can be found in both frameworks. That's why I focused on evaluating the concepts, that are important to this project.

LangChain has built-in support for a broader range of NLP tasks. Also the ConversationChain makes it a lot easier to implement chatbots. Another important factor is that LangChain is already the de facto standard for NLP tasks at OST. My personal opinion is also that LangChain has more of a future, as it already has a greater range of functions than Haystack, even though it has not been on the market for long. However, it has a steeper learning curve.

Haystack also supports many NLP tasks, although not quite as many as LangChain does. Since Agents were removed in version 2.0, it takes a little bit more effort to implement the ConversationChain counterpart with Haystack. However, it is easier to get started with.

After careful evaluation of these advantages and disadvantages, LangChain was selected as framework.

Chapter 10

Assistant Comparison

This chapter compares the available assistants.

10.1 Behavior Tests

The behavior test results can be found in the LLM Behavior Tests.

10.1.1 Interim Conclusion

The LLM Behavior Tests clearly show, that the Mixtral assistant currently has some significant deficites. In particular, the hallucination of ingredient quantities and the inability to provide step-by-step guidance make the assistant very unreliable.

The OpenAI assistant on the other hand provides assistance in all common use cases.

10.2 Processing Speed

The processing speed is tested for both text input and audio input. For text messages the time is measured from when the message is sent to when the full response is received. For audio messages the time is measured from when speaking stopps until the audio of the response starts.

Note that the application runs locally, which affects the used time to process a message.

10.2.1 OpenAI Assistant

Message type	Message	Used time
Text	Which ingredients do i need?	6s
Audio	Which ingredients do i need?	25s
Text	How long does this recipe take?	5s
Audio	How long does this recipe take?	17s

10.2.2 Mixtral Assistant

Message type	Message	Used time
Text	Which ingredients do i need?	21s
Audio	Which ingredients do i need?	57s
Text	How long does this recipe take?	19s
Audio	How long does this recipe take?	46s

10.2.3 Interim Conclusion

Although these times are not entirely representative, they clearly show that the OpenAI assistant is much faster than the Mixtral assistant in all aspects.

10.3 Privacy

The Mixtral assistant offers a high level of privacy because the used Mixtral LLM is hosted at OST and therefore the user's input is not sent to an external service. The conversation is also only stored for one day, after which it is deleted and cannot be recovered.

The OpenAI assistant on the other hand uses the OpenAI API. OpenAI stores the conversation history forever. However, OpenAI claims, that they do not use API data for training.

10.3.1 Interim Conclusion

The mixtral assistant provides a significantly higher level of privacy than the OpenAI assistant, as the user's input gets processed internally without the need for an external service.

10.4 Conclusion

The previous comparisons clearly show, that the OpenAI assistant performs better than the Mixtral assistant in both response quality and correctness and speed. What is better about the Mixtral assistant is its privacy.