Bachelor Project

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Computer Science

Mobile platform for charitable assistance to seniors using AI

Margarita Lupenko

Supervisor: Ing. Kyrylo Bulat
Field of study: Software engineering and technologies
Subfield: Enterprise systems
May 2024

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Lupenko Margarita**	Personal ID number:	**507353**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Computer Science**		
Study program:	**Software Engineering and Technology**		

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Mobile platform for charitable assistance to seniors using artificial intelligence

Bachelor’s thesis title in Czech:

Mobilní platforma pro charitativní pomoc seniorům s využitím umělé inteligence

Guidelines:

The bachelor thesis aims to develop and implement a mobile platform for providing charitable assistance to the elderly using artificial intelligence.
The application will allow users to create requests for assistance, store information about volunteers who have previously handled their requests, and ask for theoretical help through a chat interface with artificial intelligence.
The work will involve the following tasks:
- Analyzing and comparing existing mobile applications designed to help elderly individuals.
- Defining use cases for the application.
- Designing the application's architecture and user interface.
- Analyzing existing artificial intelligence technologies.
- Choosing suitable technologies for implementation.
- Developing the mobile application.
- Conducting usability testing and testing of individual application modules.

Bibliography / sources:

Phillip A. Laplante, Mohamad Kassab: “Requirements Engineering for Software and Systems” [7 June 2022]
Benjamin Bähr: “Prototyping of User Interfaces for Mobile Applications” [2017]
Karthikeyan NG, Arun Padmanabhan, Matt R. Cole: "Mobile Artificial Intelligence Projects" [2019]

Name and workplace of bachelor’s thesis supervisor:

Ing. Kyrylo Bulat   System Testing IntelLigent Lab FEE

Name and workplace of second bachelor’s thesis supervisor or consultant:

Date of bachelor’s thesis assignment: **15.02.2024**   Deadline for bachelor thesis submission: **24.05.2024**

Assignment valid until: **21.09.2025**

Ing. Kyrylo Bulat Supervisor’s signature	Head of department’s signature	prof. Mgr. Petr Páta, Ph.D. Dean’s signature

III. Assignment receipt

The student acknowledges that the bachelor’s thesis is an individual work. The student must produce her thesis without the assistance of others, with the exception of provided consultations. Within the bachelor’s thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student’s signature

Acknowledgements

From the bottom of my heart, I would like to thank my supervisor Ing. Kyrylo Bulat for his valuable advice, help, and great mentoring. He was always willing to answer all my questions and point me in the right direction.

I would also like to express my deep gratitude to my family for their endless support throughout my studies.

Declaration

Prohlašuji, že jsem předloženou práci vypracovala samostatně, a že jsem uvedla veškerou použitou literaturu.

V Praze, 24. května 2024

I declare that this work is all my own work and I have cited all sources I have used in the bibliography.

In Prague, 24. May, 2024

Abstract

This bachelor's thesis focuses on analyzing, designing, and implementing a mobile platform for charitable assistance to seniors using artificial intelligence. The main goal is to integrate artificial intelligence technology into the application to enhance the user experience for seniors. An essential part of my thesis will be developing a mobile application with an intuitive interface for seniors.

Keywords: charitable assistance, seniors, social support, AI, React Native, mobile application

Supervisor: Ing. Kyrylo Bulat

Abstrakt

Tato bakalářská práce se zaměřuje na analýzu, návrh a implementaci mobilní platformy pro charitativní pomoc seniorům s využitím umělé inteligence. Hlavním cílem je integrovat technologii umělé inteligence do aplikace pro zlepšení uživatelského zážitku seniorů. Podstatnou součástí mé bakalářské práce bude vývoj mobilní aplikace s intuitivním rozhraním pro seniory.

Klíčová slova: charitativní pomoc, senioři, sociální podpora, AI, React Native, mobilní aplikace

Překlad názvu: Mobilní platforma pro charitativní pomoc seniorům s využitím AI

Contents

Listings

Figures

Tables

Chapter 1

Introduction

Older people have difficulty adapting to new technologies and realities in a rapidly changing world. They often encounter difficulties that seem simple at first sight, such as writing a message or taking a photo. In addition, many pensioners also need social support.

In the Czech Republic, approximately 2.35 million pensioners receive an old-age pension. Of these, about 30,000 receive less than 8,000 CZK, while the average pension is 20,216 CZK. [1] However, inflation in the Czech Republic reached 15.1% in 2022, the highest rate in the last 20 years. [2] This inflationary pressure has significantly affected the country's economy and the living conditions of its inhabitants.

According to official figures, there are approximately 414,000 people with disabilities in the Czech Republic, two-thirds of whom are pensioners. [3]

Given these statistics and social experience, it is obvious that many people need social assistance. That is why my app is needed, to help those who need it.

1.1 Application description

The mobile application "My Support" will be designed to provide charitable assistance to seniors in various aspects of life using Artificial Intelligence (AI).

AI will be used in the chat to simplify the interaction of seniors with the application.

"My Support" will be designed for two roles: seniors and volunteers. The application for seniors will have a simplified interface and a set of features adapted to the needs of seniors, such as large buttons and fonts, and a chat with AI that will answer theoretical questions. The version for volunteers will have features such as monitoring tasks from seniors on a map, the ability to connect with a senior by phone, and many other features.

1.2 Description of the target user group

The primary target group is volunteers and seniors.

Volunteers are people who assist seniors. They can be members of charities or ordinary people who are interested in helping. Their ages start at 18 and up.

Seniors are people aged 60 years and older who need help in different areas of life.

The application has the potential to reach many people as many seniors need social support. The project helps to address this problem by providing seniors with easy access to help and volunteers the opportunity to get involved.

1.3 Bachelor thesis objectives

The project aims to design and implement a mobile platform for charitable assistance to seniors using AI.

The thesis will include the following points:

- analysis and comparison of existing charity platforms for the elderly;
- definition of application usage scenarios;
- design of the application architecture and its user interface;
- testing of the application.

The application modules will allow users to create, edit, or delete a task and ask a theoretical question in a chat with AI. The outcome of the project will be a starting point for the final implementation and testing, in addition to the implementation of the basic modules of the solution.

Chapter 2

Analysis of existing solutions

This chapter will analyze existing mobile applications aimed at assisting older people. The focus will be on their functionality, usability, and reliability. The key advantages of each application will be discussed and their disadvantages will be identified.

2.1 Medisafe

Medisafe is a reminder app that helps people take their medication on time and as directed by their doctor. The app is available for iOS and Android devices.

Medisafe can manage medication use and send reminders. Just enter prescriptions and times of use, and Medisafe will create a visual schedule with pictures of each tablet and a list of potentially harmful interactions. Then, throughout the day, it will remind the user when it's time to take the medication, inform them when a prescription is about to expire, and even warn a friend or family member about a missed dose. [4]

Advantages of the application:

- helps to take medication on time and as directed by the doctor;
- allows you to monitor the use of medication;
- provides information about medicines;
- can be adapted to the individual needs of the user.

Disadvantages of the application:

- it can be challenging for some seniors to use due to the complexity of the interface;
- there's no integration with other healthcare services.

2.2 Carely

Carely is a caregiving app designed to improve communication and coordination between family members and professional caregivers. [5]

It aims to provide a centralized platform for caregiving activities and information sharing. In addition, it allows family members and caregivers to form a caregiving team, facilitating effective communication, task assignment, and coordination of caregiving responsibilities. This includes features such as:

- shared calendar: this allows family members and caregivers to schedule appointments, keep track of progress, and maintain order;
- task list: this allows family members and caregivers to assign and track tasks so that everyone is on the same page;
- diary: this allows family members and caregivers to document important information such as changes in the care recipient's condition or medication adjustments.

Advantages of the application:

- centralized care management platform: Carely provides a central platform that facilitates planning, monitoring, and coordination;
- improved communication and collaboration: Carely makes it easier for family members and caregivers to communicate and collaborate, which can help to prevent misunderstandings and ensure that everyone is informed of the latest developments;
- support for professional caregivers: Carely also provides support to professional carers, which can help them provide better quality care.

Disadvantages of the application:

- the app doesn't offer detailed activity reports or summaries, making monitoring more challenging.;
- limited features in the free version.

2.3 Life360

Life360 is a mobile app that can help seniors stay safe and in touch with their loved ones.[6] The app offers several features that are specifically designed for seniors, including:

- real-time location tracking: this feature allows family and friends to track the senior's location in real-time. This can help ensure the senior's safety, especially if they live alone or are traveling in an unfamiliar environment;
- safety alerts: the app offers a variety of safety alerts that can help seniors in an emergency. These alerts include crash detection, SOS alerts, and geofence alerts;
- communication tools: the app makes it easy for seniors to communicate with their loved ones through messages, group chats, and location-sharing notifications. This can help stay in touch with family and friends and coordinate activities.

Advantages of the application:

- providing a sense of security: the app offers the possibility to provide seniors with a sense of security, knowing that their location can be tracked and loved ones will be warned in case of an emergency. This feature is especially useful for seniors living alone or moving in an unfamiliar environment;
- maintaining connections: the app helps seniors maintain connections with their loved ones, which is key to maintaining their social and emotional well-being. In this way, it allows seniors to feel less lonely and isolated;
- preventing injuries: the crash detection feature can help seniors in an emergency and prevent serious injuries or even death. This feature is especially useful for seniors who may have difficulty walking or moving.

Disadvantages of the application:

- certain features, like driving monitoring and emergency alerts, require paid subscriptions;
- constant background operation can significantly drain the battery;
- privacy can be an issue because of the location tracking feature.

2.4 eCare21

The eCare21 app monitors the health of seniors 24/7 in real-time. Through the app and a wearable fitness tracker like Fitbit or Apple Watch, it collects, compares, and analyzes seniors' heart rate, physical activity, weight, calorie intake, medication adherence, and glucose levels. This helps caregivers and healthcare providers proactively create and improve care plans for seniors. [7]

Advantages of the application:

- 24/7 real-time health monitoring: the eCare21 app constantly monitors health, allowing for timely notification of any changes in physical condition;
- data collection and analysis: the app collects and analyzes a wide range of health data, including heart rate, physical activity, weight, calorie intake, medication adherence, and blood glucose levels;
- proactive care planning: data collected by the eCare21 app can be used to proactively create and improve care plans for seniors;
- ease of use: the eCare21 app is easy to use and can be customized to the user's needs.

Disadvantages of the application:

- expensive paid subscription;
- the interface is not intuitive and is difficult to navigate for elderly users.

2.5 Senior safety app

The Senior Safety app provides real-time location tracking, geo-location alerts, and emergency assistance. It also provides peace of mind to seniors and their caregivers and is available on both iOS and Android platforms. In addition, it enables fall detection technology that detects falls and automatically sends alerts to designated contacts or emergency services. [8]

Advantages of the application:

- increases safety and well-being: the app provides a variety of features that can significantly increase the safety and well-being of seniors, especially those who live alone or have limited mobility;

- peace of mind for seniors and caregivers: the app provides peace of mind for seniors and their caregivers by providing real-time information and ensuring immediate assistance in the event of an emergency;
- user-friendly interface: the app's interface is designed with the needs of seniors in mind, so it is easy to use and navigate.

Disadvantages of the application:

- limited features in the free version;
- lacks customizable options for individual needs.

#	Name	Focus Area	Usability	User rating	Price	Android	iOS
1	Medisafe	Medicine	For older users, navigating the app might be challenging due to an abundance of features and small text on buttons. However, there's a short guide at the beginning of app usage that helps users orient themselves quickly within the app.	4.7	Free/Medisafe Premium $4.99/month or $39.99/year	Yes	Yes
2	Carely	Coordination, health tracking, and documentation	For older users, navigating the app might be challenging due to an abundance of features and small text on buttons.	3.4	Free/Carely Pro $14.99/month or $149.99/year	Yes	No
3	Life360	Safety and communication	For seniors with poor eyesight, using the app might be difficult due to an excessive number of features and small text; some features may be unnecessary for older users.	4.8	Free/Life360 Pro $4.99/month or $49.99/year	Yes	Yes
4	eCare21	Healthcare	The app has comprehensive features, and it may take a while for users to learn how to use it.	4.5	Free/Premium plan: $7.95/month	Yes	Yes
6	Senior safety app	Safety and comfort	Designed with seniors' needs in mind. The app interface is simple and easily understandable, even for seniors with no experience using phones. The app also offers a range of customization options that allow seniors to adjust the app to suit their individual needs.	4.5	Free/$4.50/month or $45.00/year	Yes	No

Table 2.1: Overview of applications

Analysis of existing solutions shows that there is currently no app that provides help for seniors in all areas of life using AI. The idea is to introduce an AI chat that will be ready to answer seniors' questions on any topic at any time, and also to create a platform where seniors can create tasks and volunteers can fulfill them.

My mobile platform will not only ensure the solution of everyday problems for seniors, but it will also provide volunteers with the opportunity to actively help those in need.

Chapter 3

Analysis

This chapter will present the functional and non-functional requirements for the mobile application, along with the key use cases. The use cases will be described in detail to clearly illustrate how users interact with the application.

3.1 Requirements analysis

Requirements analysis is an important phase in software development that helps define the features and behavior of the system. Product requirements represent specific characteristics that are necessary to satisfy user and business needs. There are two basic types of system requirements: functional and non-functional. [9]

Functional requirements describe the services the system should provide and how the system will react to its inputs.

Non-functional requirements, on the other hand, focus on qualitative attributes like security, reliability, performance, usability, and testability. In the software marketplace where functionally similar software products compete for attention, these non-functional aspects often play a crucial role in differentiating between competing options.

Analysis will enable the creation of a use case diagram, the development of a prototype application, and the determination of its structure. [10]

3.1.1 Functional requirements

Functional requirements will be divided into three groups depending on the user's role: specific for volunteers, specific for seniors, and common for all registered users. Additionally, each functionality is categorized by priority:

- Red color - high priority;
- Orange color - medium priority;
- Teal color - low priority.

Registered user

All users, regardless of their role, will have the following functions:

- **FR1 Registration**
  All users will be able to create a new account in the application.
- **FR2 Login**
  Users will be able to log in to their account.
- **FR3 Changing personal information**
  Each user will be able to edit their personal information in their account settings, including address, and email.
- **FR4 Deleting account**
  Users will be able to permanently delete their accounts from the application.
- **FR5 Logout**
  Each user will be able to safely log out of the application.

Volunteer

- **FR6 Monitoring tasks from seniors on a map**
  Volunteers will be able to view available tasks from seniors on the map.
- **FR7 Filter search tasks**
  Volunteers will be able to use filters in the search tool to find tasks that are relevant to them, including filters for date range, status of task, and category of help.

- **FR8 Recording task**

  Volunteers will be able to record a task from a senior for themselves, to assist.

  FR9 Rating seniors

  Volunteers will be able to rate seniors and their satisfaction with the provided assistance after completing the task.

- **FR10 Withdrawal of task**

  Volunteers will have the option to withdraw the task if they can no longer participate or provide the requested help.

- **FR11 Notifications of new tasks**

  Volunteers will be able to receive notifications of new tasks.

Senior

"My Support" will have a simpler interface for the elderly so they don't have trouble using it because of visual noise. When registering, seniors must provide personal information including phone number, first name, last name, and date of birth. Seniors will be able to:

- **FR12 Creating task**

  Seniors will be able to create tasks.

- **FR13 Deleting own task**

  Seniors will be able to delete a task if it is no longer current or if they decide that they do not need the given help.

- **FR14 Editing task**

  Seniors will be able to edit their task if it has not yet been accepted by any volunteer, so they can specify their needs.

- **FR15 Rating volunteers**

  Seniors will be able to rate the volunteers and their satisfaction with the help provided once the task is complete.

- **FR16 Contact list of volunteers**

  Seniors will have the opportunity to add volunteers to their contact list so they can contact them for future tasks.

- **FR17 Ask a question in chat with AI**

  Seniors will have the opportunity to ask a question to AI through the chat interface of the application.

3.1.2 Non-functional requirements

- **NFR1 Multiplatform**

  The application must be developed with full compatibility with various operating systems, including iOS and Android. This will ensure that it effectively serves a diverse audience of users using different devices and platforms.

- **NFR2 Intuitive interface**

  The design of the application must be intuitive and easy to understand for the user. Users should be able to easily navigate and use the application without the need for lengthy training, which will increase user satisfaction.

- **NFR3 Adaptive design**

  The application must be optimized for use on mobile devices and different screen sizes. Adaptive design will ensure the app runs smoothly and efficiently on different devices, which is important for users who use the app on different devices.

- **NFR4 Profile protection**

  The application must include reliable measures to protect user personal data. This includes proper authentication of user permissions to their data and protection against unauthorized access to this data.

- **NFR5 Scalability**

  The development of an application must take into account the possibility of future expansion with new features. The application should be designed with scalability in mind, meaning the ability to easily add new features and functionality without major intervention in existing code.

- **NFR6 Testability**

  To ensure the stability of the application and to facilitate bug management, it must be designed for easy testability. This will allow systematic testing and the rapid detection and elimination of potential problems.

- **NFR7 Rapid response**

  The application should respond quickly to user actions, including page load speed and processing of user requests. Page load times should not exceed 2 seconds, while processing user requests, such as form submissions or data retrieval, should complete within 500 milliseconds. Fast response is a key element of a positive user experience.

- **NFR8 Efficient use of resources**

  The application should make efficient use of device resources such as processor, memory, and battery to achieve optimal performance and extend device life.

- **NFR9 Up-to-date documentation**

  Up-to-date documentation and user guides should be available to facilitate the use and integration of the application.

- **NFR10 Availability and accessibility**

  The application should be designed to improve accessibility for users with different types of limitations and needs.

3.2 Use cases

The use case is a methodology used in system analysis to identify, clarify, and organize system requirements. Use cases contain detailed information about the system, the system's users, relationships between the system and the users, and the required behavior of the system. This method creates a document describing all the steps that a user will take to complete a certain activity.

Use cases describe the functional requirements of the system from the perspective of the end user and create a sequence of events focused on achieving goals that are easy to follow for both users and developers. [11]

3.2.1 Actors

Actors in application use cases include:

- Unauthorized user

  This actor has not yet entered the app, so most of the features are not available for it.

- Authorized user

  This category is defined for two actors who have successfully logged into the application:

  - Senior

    This actor is a user who uses the application to request help or services from other users.

  - Volunteer

    This actor is a user who offers their volunteer help or services to other users who request help.

- System

  The system is the "My Support" mobile app itself.

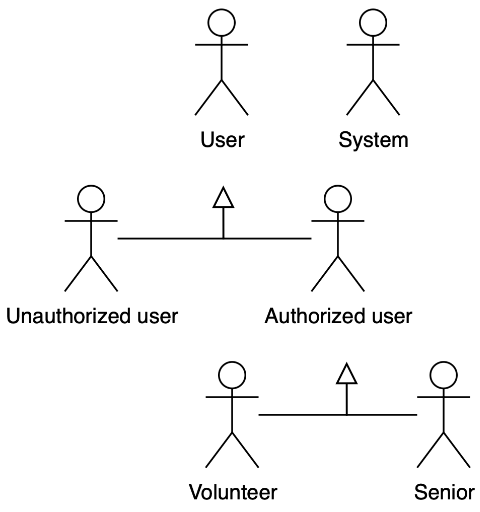


Figure 3.1: Actors.

3.2.2 Use cases

- **UC1 – Register**
  - Description:

    - New users want to create an account and register in the mobile app.
  - Pre-conditions:

    - The user has launched the “My Support” application.
  - Scenario:

    1. The user clicks the “Get started” button on the start screen.
    2. The system redirects the user to the registration screen.
    3. The user fills in the required information, such as name, last name, email address, password, contact number, and role (volunteer or senior).
    4. After entering the initial data, the user clicks the “Next” button.
    5. The system redirects the user to a role-specific page to provide additional information:
       - Volunteer: they will need to provide information about their working hours.
       - Senior: they will need to provide their date of birth and address.
    6. Once the additional information is filled out, the user clicks the “Register” button.
    7. The system verifies the entered data and creates a new account for the user, registering them as either a volunteer or a senior.
    8. After successful registration, the user is redirected to the login screen.
  - Actors:

    - Unauthorized user
    - System

- **UC2 – Log in**
  - Description:

    - Users want to log into the “My Support” mobile application with their existing accounts.
  - Pre-conditions:

    - The user has launched the “My Support” application.
    - The user has registered with the system.

- Scenario:

  1. On the start screen, click the “Already have an account? Sign in” button.
  2. The system redirects the user to the login screen.
  3. The user enters their email address and the password they used to register.
  4. He presses the “Login” button.
  5. The system verifies the entered login details and redirects the user to the main application screen.

- Actors:

  - Authorized user
  - System

- **UC3 – Log out**

  - Description:

    - Users want to securely log out of their accounts on the mobile app.

  - Pre-conditions:

    - The user has launched the “My Support” application.
    - The user has logged into the system.

  - Scenario:

    1. The user clicks the “Account” button, where the account options are located.
    2. The system will display user information and various options.
    3. Among the options, the user will select the “Log out” option and click the button.
    4. The system will ask the user to confirm the logout to prevent accidental logout.
    5. The user confirms this choice by clicking on the “Yes” button.
    6. The system confirms the logout action and terminates the current user session.
    7. The user will be redirected to the login screen where they will need to enter their login details to log back in.

  - Actors:

    - Authorized user
    - System

- **UC4 – View your personal information**

- Description:

  - Users want to view their data stored in the “My Support” mobile application.

- Pre-conditions:

  - The user has launched the “My Support” application.
  - The user has logged into the system.

- Scenario:

  1. The user clicks the “Account” button, where the account options are located.
  2. The system displays the user’s personal information, including first name, last name, date of birth, email address, address, and contact number.
  3. The user can view their data and verify that it is correct.

- Actors:

  - Authorized user
  - System

- **UC5 – Change your personal details**

  - Description:

    - Users want to update their personal information in the “My Support” mobile app in case of changes or updates.

  - Pre-conditions:

    - The user has launched the “My Support” application.
    - The user has logged into the system.

  - Scenario:

    1. The user clicks the “Account” button, where the account options are located.
    2. The system will display user information and various options.
    3. Among the options, the user will select the “Change data” option and click the button.
    4. The system redirects the user to a screen where they can edit their details.
    5. The user makes the necessary changes to their personal information.
    6. When the editing is finished, the user presses the “Save” button.
    7. The system verifies the entered data and updates the account information.

8. The user receives confirmation of the successful update of their personal information.

- Actors:

- Authorized user
- System

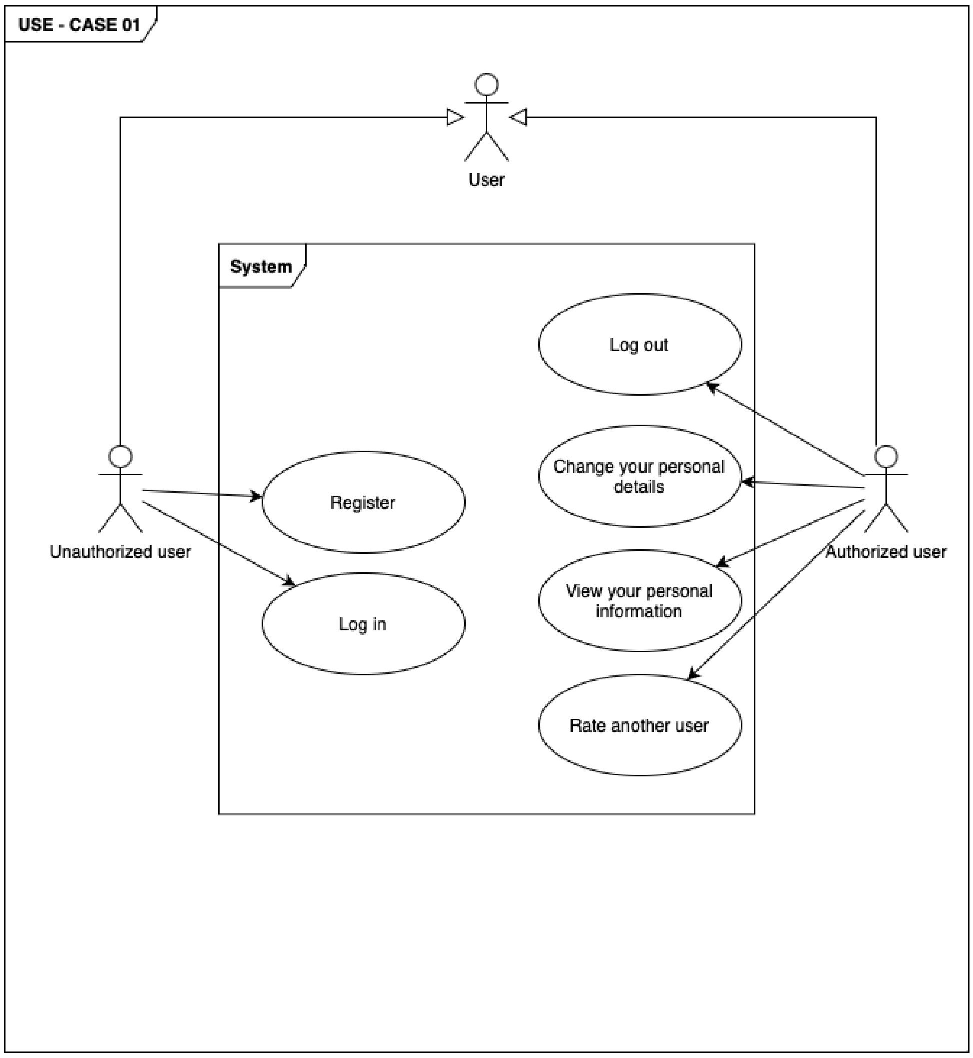


Figure 3.2: Use case - Diagram.

- **UC6 – Create a task**
  - Description:

    - Seniors want to create a new task in the mobile application "My Support".
  - Pre-conditions:

    - The senior has launched the "My Support" app.
    - The senior has logged into the system.
  - Scenario:

    1. Senior will go to the "Create task" page.
    2. The system displays a form for creating a task, which the senior must fill in:
       - The title of the task.
       - Description of the task.
       - Specify the address where help is needed.
       - Select a task category from the list.
    3. The senior fills in the required information.
    4. The senior click on the "Create" button.
    5. The system accepts the created task and includes it in the system. The task will be available to volunteers who offer help.
  - Actors:

    - Senior
    - System
- **UC7 – Delete a task**
  - Description:

    - Seniors want to delete their task for help if they no longer need it.
  - Pre-conditions:

    - The senior has launched the "My Support" app.
    - The senior has logged into the system.
  - Scenario:

    1. The senior navigates to the "Home" page.
    2. The system will display the main page to the senior with a list "My tasks".
    3. The senior from the list of statuses selects tasks with the status "Created".

4. The system displays a page with a list of created help tasks.
5. The senior finds the task they want to delete and clicks on it.
6. The system displays the details of this task, including its title and description.
7. On this page, the senior finds the option “Delete task” and clicks on it.
8. The system asks the senior to confirm the deletion of the task to prevent unintended removal.
9. The senior confirms their choice to delete the task.
10. The system deletes the task from the system displays a confirmation of successful deletion, and also notifies the volunteer, who might have taken on this task, that their help is no longer needed.

- Actors:

  - Senior
  - System

- **UC8 – Edit a task**

  - Description:

    - Seniors want to modify their existing task for assistance if they need to update information.

  - Pre-conditions:

    - The senior has launched the “My Support” app.
    - The senior has logged into the system.

  - Scenario:

1. The senior navigates to the "Home" page.
2. The system will display the main page to the senior with a list “My tasks”.
3. The senior from the list of statuses selects tasks with the status “Created”.
4. The system displays a page with a list of created help tasks.
5. The senior finds the specific task they want to edit and clicks on it.
6. The system displays the details of that task, including the title, description, and other information.
7. On this page, the senior finds the “Edit task” option and clicks on it.
8. The system will redirect seniors to a page with a form to edit the task, where they can edit the title, description, category and address.

9. After making the desired edits, the senior clicks on the “Save” button.
10. The system verifies the information entered and updates the task for assistance, then displays a confirmation that the task was successfully updated.

- Actors:

  - Senior
  - System

- **UC9 – View a task**

  - Description:

    - Users want to view the details of a specific help task in the “My Support” mobile app.

  - Pre-conditions:

    - The user has launched the “My Support” app.
    - The user has logged into the system.

  - Scenario:

    1. The user navigates to the "Home" page.
    2. The user finds the specific task they want to view and clicks on it.
    3. The system displays the details of that task, including the title, description, date, category.
    4. The user can view all available information about the task and find out more details about how they can help.

  - Actors:

    - Authorized user
    - System

- **UC10 – Record a task**

  - Description:

    - Volunteers want to take on a specific task for help on the “My Support” mobile app.

  - Pre-conditions:

    - The volunteer has launched the “My Support” app.
    - The volunteer has logged into the system.

- Scenario:

  1. The volunteer goes to the “Home” page.
  2. The system will display a page with a list of all available help tasks.
  3. The volunteer finds the specific task they want to accept and clicks on it.
  4. The system displays the details of this task, including the title, description, date and category.
  5. The volunteer clicks on the “Accept task” button.
  6. After accepting the task, the system returns to the main screen and sends a notification to the seniors that their task has been accepted (UC19).

- Actors:

  - Volunteer
  - System

- **UC11 – Unsubscribe a task**

  - Description:

    - Volunteers want to withdraw an accepted task for help in the mobile application “My Support” if they can no longer or do not wish to fulfill the given task.

  - Pre-conditions:

    - The volunteer has launched the “My Support” app.
    - The volunteer has logged into the system.
    - The volunteer has accepted a task for help that he wants to remove.

  - Scenario:

    1. The volunteer goes to the “My tasks” page.
    2. The system displays a page showing the tasks that the volunteer has accepted.
    3. The volunteer finds the specific task they want to remove and clicks on it.
    4. The system displays the details of this accepted task, including the title and description.
    5. The volunteer clicks on the “Leave task” button.
    6. When a task is leaved, the system is updated and the task is removed from the list of accepted tasks.

- Actors:

  - Volunteer
  - System

- **UC12 – View history of fulfilled tasks**

  - Description:

    - Volunteers want to view the history of tasks for help they have already fulfilled in the “My Support” mobile app.

  - Pre-conditions:

    - The volunteer has launched the “My Support” app.
    - The volunteer has logged into the system.

  - Scenario:

    1. The volunteer is taken to the “My tasks” page, which shows the tasks that the volunteer has already completed.
    2. The system will display a page where the volunteer can select the "Completed" or "In processing" tab.
    3. The volunteer will click on "Completed".
    4. The system will display a list of tasks for assistance that the volunteer has completed in the past.

  - Actors:

    - Volunteer
    - System

- **UC13 – Ask a question to the AI chat**

  - Description:

    - Seniors want to ask questions or seek advice from the smart chat in the “My Support” mobile app.

  - Pre-conditions:

    - The senior has launched the “My Support” app.
    - The senior has logged into the system.

  - Scenario:

    1. The senior goes to the “Chat bot” page where the smart chat system is available.
    2. The system will display a chat window where the seniors can start typing their questions or requesting advice.

3. The senior writes their question or request and sends it to the chat.
4. The system processes the question or request and starts providing an answer or advice to the senior.
5. The senior can continue the conversation with the AI chat if they need more information or have further questions.

- Actors:

  - Senior
  - System

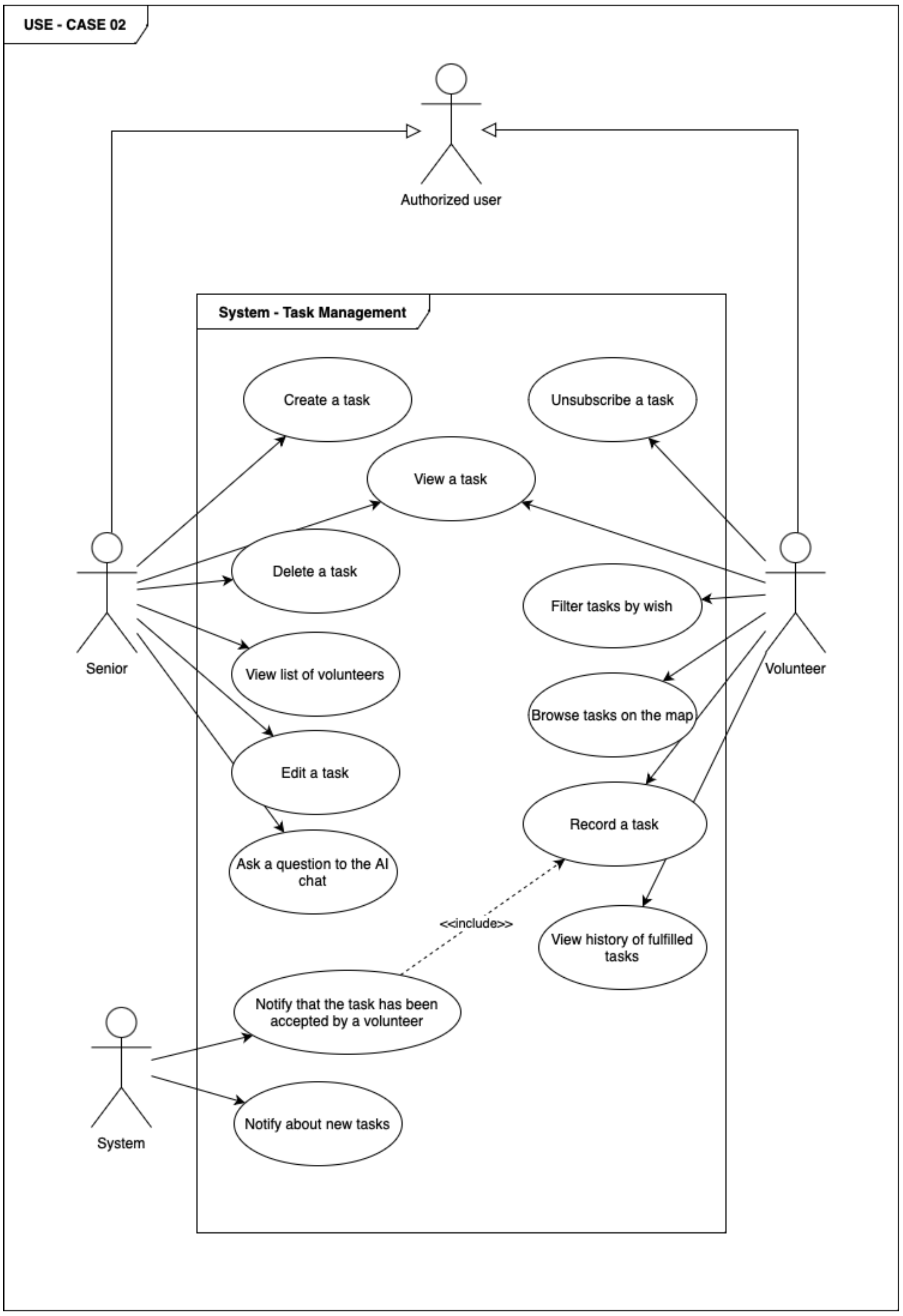


Figure 3.3: Use case - Diagram.

Chapter 4

Design

This chapter will cover class and sequence diagrams, applications, and the rationale for their use in implementation. It will also comparatively analyze existing AI APIs, assessing their key features, advantages, and disadvantages.

4.1 Class diagram

Class diagrams, a core component of Unified Modeling Language (UML), graphically represent class relationships and code dependencies in a system. They aid system design by clarifying problem domain requirements and identifying components. In object-oriented development, early class diagrams often reflect actual classes and objects in the code, serving as a blueprint for development. [12][13]

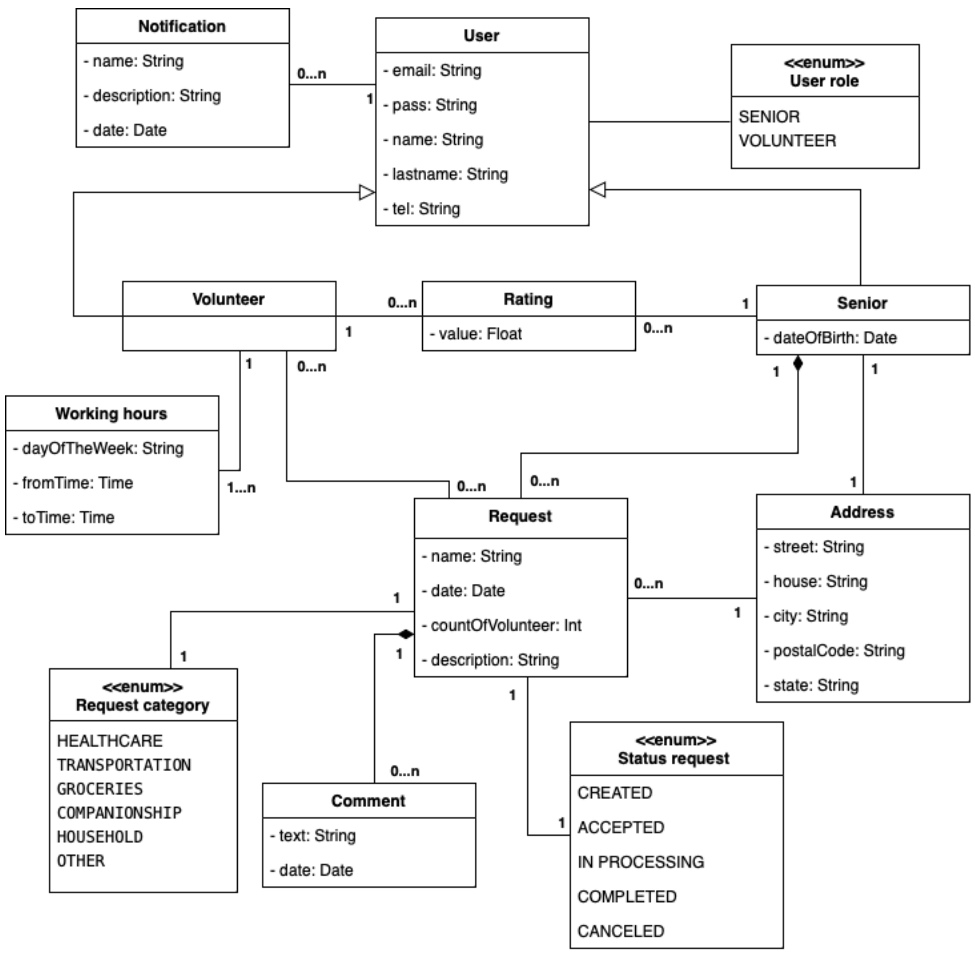


Figure 4.1: Class diagram.

The class diagram I created reflects the structure of a system designed to manage interactions between volunteers and older people, their tasks, ratings, and comments. The main entities represented in the diagram are:

- User: this is the primary class that contains information about all users. Fields include email, password, first and last names, and telephone numbers.
- Volunteer: this subclass inherits from the User class and is specialized for volunteers. It's connected to the "Working hours" class to indicate when a volunteer is available to provide services. The separation of "Working hours" into its own entity allows for a flexible representation of a volunteer's availability, as one volunteer might be available at multiple non-continuous times throughout the week.
- Senior: another subclass of "User", designed for elderly people. It has an additional field for the date of birth and a connection to the "Address" class, indicating the location of the elderly person.
- Task: a class that represents tasks that seniors can make. It holds the name of the task, the date it was created, the count of volunteers needed, a description, and a rating field containing information on who gave ratings to whom and the values (from 1 to 5).
- Address: a class that contains address details such as street, house, city, postal code, and state.
- Comment: this class is designated for remarks left by users about tasks. It includes the comment text, the author of the comment, and the date it was posted.
- Working hours: a class associated with the "Volunteer" class, which shows the days of the week and time intervals when the volunteer is available.
- Task category: this enumeration defines categories of tasks, making it easier to filter and manage them.
- Task status: an enumeration that describes the possible statuses of a task with values like created, accepted, in processing, completed, and canceled. Enums are used here to standardize the statuses across the system and to simplify the process of status management.

4.2 Sequence diagram

A sequence diagram, a type of interaction diagram, presents objects as lifelines extending vertically on the page. The interactions between these objects over time are depicted as arrows representing messages, flowing from the originating lifeline to the destination lifeline. These diagrams excel in illustrating the communication between different objects and the specific messages that initiate these interactions. [19]

The provided sequence diagram (see Figure 4.2) details the process of initiating a new task by elderly users. This diagram shows the interaction between the user, system controllers, and databases during the creation of a task. The procedure involves the following steps:

1. The user selects the option to create a new task on the task addition screen.
2. The system, through the task controller, verifies the accuracy of the data entered by the user.
3. If errors are detected in the data:
    a. The system displays validation error messages.
    b. The user has the opportunity to correct the entered data.
4. If there is an authorization error:
    a. The system displays an authorization error.
5. Upon successful validation:
    a. The system transfers the data to the task service, which creates a new task and stores it in the task collection.
    b. The user receives information about the successful creation of the task.

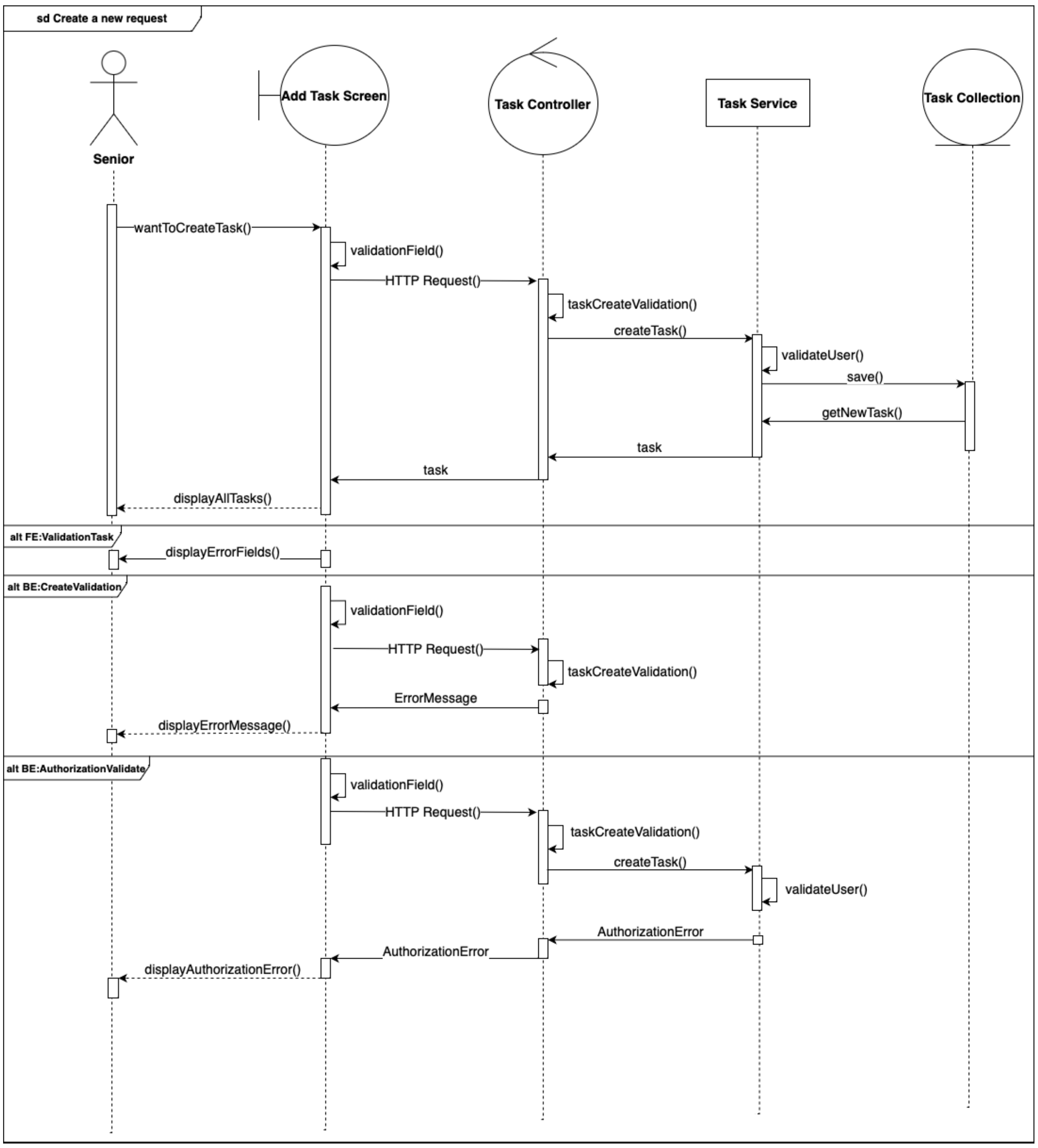


Figure 4.2: Sequence diagram "Create a new task".

The sequence diagram presented (see Figure 4.3) illustrates the process of an interactive dialogue between a user and an AI system through a chat interface. The interaction procedure is described by the following steps:

1. The user inputs and sends their message through the chat interface.
2. The entered message is passed to the message processing module where it undergoes initial analysis.
3. The message is then forwarded to the chatbot Application Programming Interface (API), developed using GPT technology (OpenAI API), to extract relevant information and generate a response.
4. If an error occurs during the API call:
   a. the system receives information about the error.
   b. The message handler receives and processes the error, determining its type.
   c. If the error is related to access issues, the system displays an error message.
5. If no error occurs:
   a. The response generated by the ChatGPT API is sent back to the message handler.
   b. The message handler forwards the response to the Response Generator module, where it is formatted for further delivery to the user.
6. The formatted response is sent back to the message handler.
7. The message handling module sends the final response back to the chat interface.
8. The chat interface displays the response or error message on the user's screen.

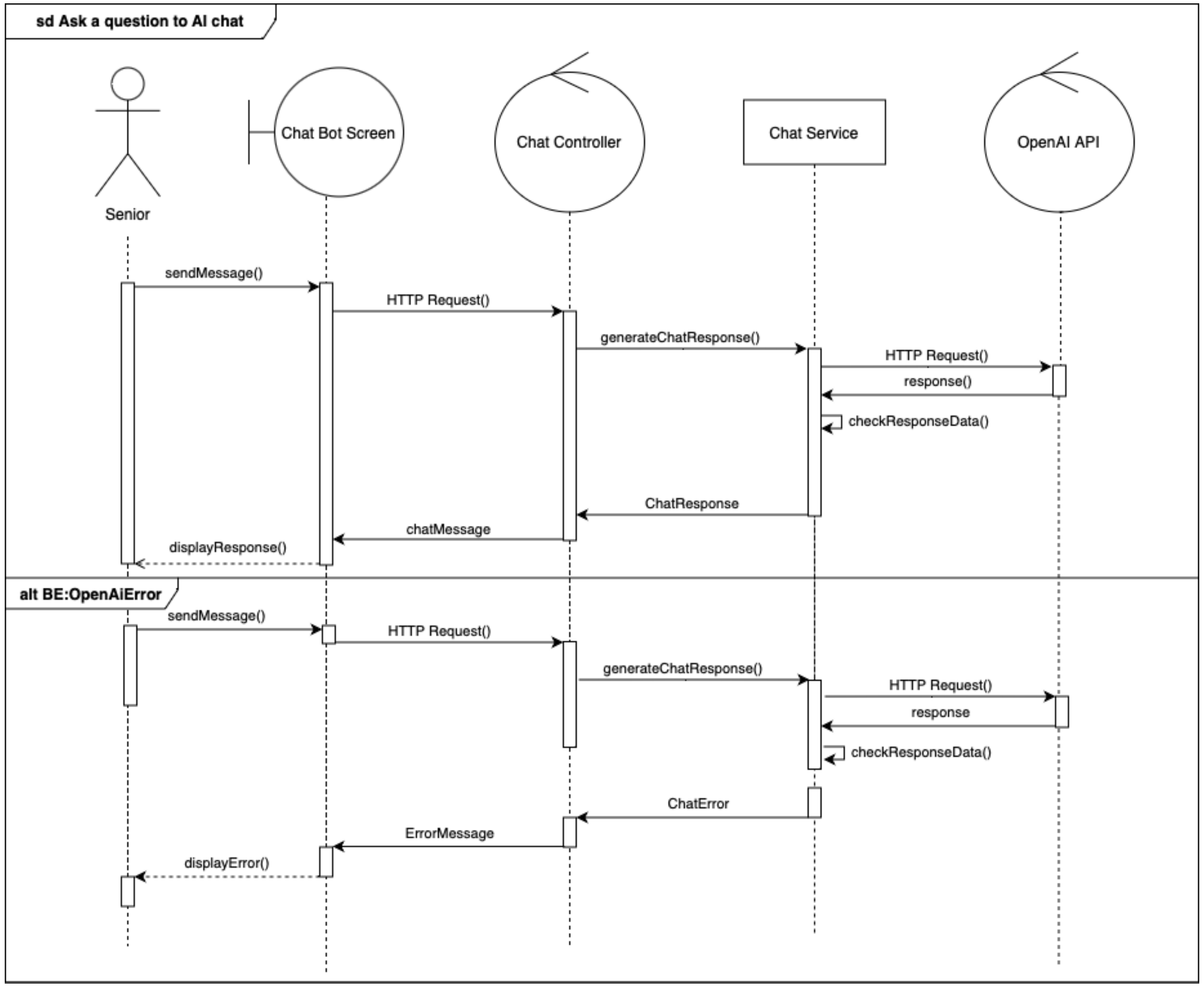


Figure 4.3: Sequence diagram "Ask a question to AI chat".

4.3 Analysis of available AI APIs

AI API

AI is increasingly becoming a tool for solving problems in various fields. AI libraries are often used in application development to provide the necessary set of functions and options to efficiently realize specific tasks.

This chapter aims to explore and compare different AI libraries in depth. The main objective will be to identify the key characteristics and to distinguish the advantages and disadvantages of each of them. Such a review will serve as a basis for an informed choice when developing applications.

4.3.1 Dialogflow Google AI

Dialogflow is a tool developed by Google to create chatbots and Natural Language Processing (NLP) applications. It is designed to facilitate interaction with users through conversations and understanding input text. [14]

API's advantages:

- Easy to use: Dialogflow is relatively easy to learn how to use. It offers an intuitive interface and extensive documentation;
- Extensive documentation and active developer community: there is comprehensive documentation that facilitates development and a developer community that provides support and shares experiences;
- Supports many languages: Dialogflow supports over 100 languages, making it easier for developers to create chatbots and applications for a global audience.

API's disadvantages:

- Cost: rates increase rapidly with high query volume, making it expensive for large projects;
- Limited flexibility: the concept of intents simplifies development but is limited in complex scenarios.

4.3.2 Wit.ai by Facebook

Wit.ai is an open platform for developing NLP applications by Facebook. It's a cloud service that uses AI to understand and generate human language. [15]

API's advantages:

- Easy to use: Wit.ai is relatively easy to learn how to use. It offers an intuitive interface and extensive documentation;
- Integration: Wit.ai can be easily integrated with other platforms and services;
- Learning from large datasets: Wit.ai learns from large datasets, which allows it to provide more accurate results.

API's disadvantages:

- Privacy: as part of Facebook, raises data privacy concerns for some users;
- Limited analytics: does not offer as rich analytics and monitoring tools as other APIs;
- Flexibility: suitable for basic scenarios, but complex cases may require extra effort;
- Dependence on Facebook's infrastructure: integration with Facebook's infrastructure can be seen as a disadvantage, especially for those who prefer independent platforms.

4.3.3 Rasa

Rasa is an open-source framework for creating chatbots and NLP systems. It is a flexible and powerful platform that allows developers to create custom chatbots and applications. [16]

API's advantages:

- Complete flexibility and customization: Rasa provides developers with full control and flexibility over the creation of chatbots and NLP systems;
- Works both in the cloud and locally: Rasa can be operated either in the cloud or locally, offering flexibility to developers;
- Large and active developer community: Rasa has a large and active developer community, and provides support and resources.

API's disadvantages:

- Complexity: requires technical expertise in machine learning and development;

- Resources: requires large computational resources to deploy and train models;
- Support: the free version is under-supported at the enterprise level.

4.3.4 OpenAI API

The OpenAI API provides access to powerful language models such as GPT. These models can be used for generating text, translating languages, writing various kinds of creative content, and other tasks. [17]

API's advantages:

- Strong language models with high-quality text generation: the OpenAI API employs advanced language models capable of generating natural text;
- High quality of text generation: the OpenAI API offers high-quality text generation that can be used for various purposes.

API's disadvantages:

- Cost: usage costs are significant when processing large volumes of requests;
- Data security: restrictions on processing sensitive data due to sending to a third-party server;
- Does not contain inbuilt functions for creating chatbots: unlike specialized tools such as Dialogflow, the OpenAI API does not contain inbuilt functions for creating chatbots.

4.3.5 LaMDA by Google AI

LaMDA is a language model from Google, designed for more natural and free-flowing communication. It is a generative model capable of generating text, translating languages, writing various types of creative content, and answering questions in an informative manner. [18]

API's advantages:

- Focusing on a variety of conversations: LaMDA can host conversations on a variety of topics, including open-ended, challenging, or unusual ones;

- High quality of generated text: LaMDA is capable of generating text that is grammatically correct, coherent, and engaging;
- Customizability: LaMDA can be customized to meet the specific needs of an application.

API's disadvantages:

- Security issues: LaMDA may inadvertently generate sensitive information or unsafe content if such material is present in the training data;
- Contextual limitations: sometimes the model does not preserve long context in a conversation, resulting in incoherent or repetitive responses in long conversations;
- High difficulty in training: LaMDA is a generative model, meaning it can generate new text even if it has never seen it before. This makes LaMDA more complex to train than other models that are limited to the dataset on which they were trained.

4.3.6 Bard API

Bard API is an API for creating conversational interfaces from Google AI. It provides access to a powerful language model capable of processing and generating text, translating languages, writing various types of creative content, and answering your questions in an informative manner.

API's advantages:

- High accuracy and performance of the language model: Bard API is based on the LaMDA language model, which is one of the most powerful language models in the world. LaMDA is capable of generating text that is grammatically correct, coherent, and informative;
- Supports many languages: Bard API supports more than 100 languages, enabling developers to create conversational interfaces for a global audience;
- Simple API for use: Bard API provides a simple and intuitive API that is easy to use for developers of all levels of experience;
- Scalable architecture: Bard API is based on a flexible architecture that allows developers to expand and customize their conversational interfaces.

API's disadvantages:

- Limited access: in the early stages, the service may not be available to all users, limiting the practical applicability and testing of the model in different areas;
- Integration: does not support integration with systems outside the Google ecosystem;
- New product: as a relatively new tool, it is not as robust compared to more mature competitor products.

Criterion	Dialogflow	Wit.ai	Rasa	OpenAI API	LaMDA	Bard API
Ease of use	Easy	Easy	Demanding	Very Demanding	Easy	Easy
Flexibility	Low	Low	High	High	Medium	High
Customization	Limited	Limited	High	Limited	Limited	High
Price	Free	Free	Free	Paid	Free	Paid
Integration	Extensive	Limited	Medium	Limited	Extensive	Limited
Accuracy	High	High	High	Low	High	High
Performance	High	High	High	Low	High	High
Developer community	Large	Large	Large	Small	Large	Small
Availability in czech	Yes	Yes	Yes	Yes	Yes	Yes

Table 4.1: AI API comparison

Based on analyzing all the AI APIs, I decided to choose OpenAI API because it does not require additional model training and provides students with a free rate, which is perfect for my project.

4.4 Selected technologies

In this section, we will take a detailed look at the structure of the application and familiarise ourselves with the technologies used at each level of that structure.

The model describes the use of technologies at each level and their interaction within the system's architecture.

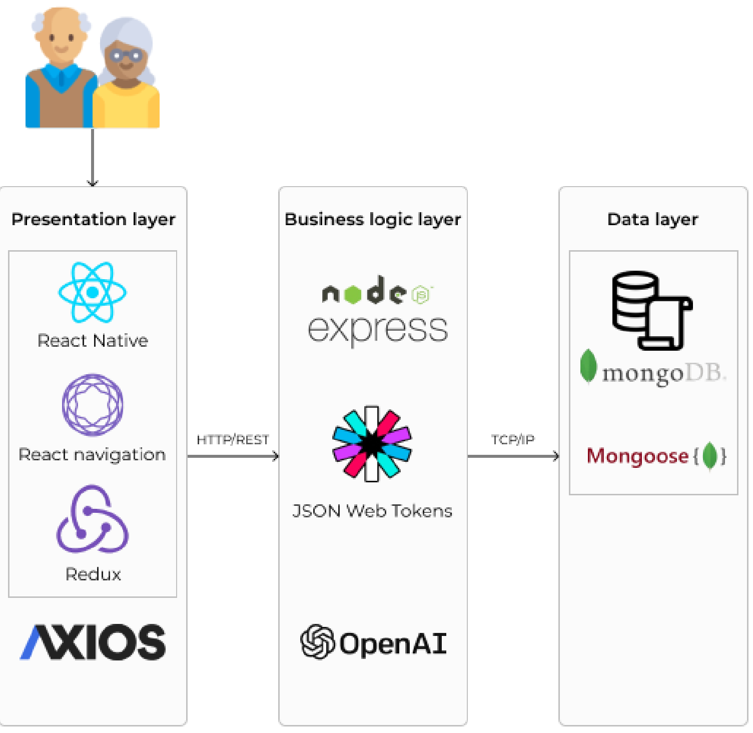


Figure 4.4: Deployment model.

4.4.1 Presentation layer

React Native

React Native is a JavaScript framework for writing real mobile apps with native rendering for iOS and Android. It is based on React, Facebook's JavaScript library for creating user interfaces, but is mobile platform-centric instead of browser-centric. Using JavaScript interfaces to the platform's APIs, React Native apps can directly access features such as camera and geolocation. It maintains high performance by separating threads from the main UI and allows simultaneous development of cross-platform applications using shared code. [26]

React Navigation

React Navigation is a popular navigation library for React Native that provides tools for implementing different types of navigation structures in mobile applications.

Redux

Redux is a state management library for JavaScript applications, commonly used with React. It centralizes the application's global state in a single store, allowing consistent and predictable data flow throughout the app.

Axios

Axios are used to make HyperText Transfer Protocol (HTTP) requests to the server side of the application. It simplifies asynchronous communication with APIs by providing features like automatic JSON conversion, request/response interceptors, and error handling.

4.4.2 Application layer

Node.js

Node.js is a server-side platform, designed to efficiently handle client requests and manage data processing. In addition to managing client-server communication, Node.js can interact seamlessly with databases, perform business logic, and handle asynchronous tasks.

JSON Web Tokens

JSON Web Tokens are a compact, self-contained way to transmit information between client and server securely. They are often used for user authentication and authorization, ensuring secure data exchange between the client and server.

OpenAI

Integration of artificial intelligence APIs to enhance the application's functionality, used for working with an AI-based chatbot.

4.4.3 Data layer

MongoDB

MongoDB is a non-relational database that stores structured data in a flexible format called Binary JSON. Unlike traditional relational databases, MongoDB does not use tables or apply a strict schema, allowing you to store different types of documents in a single collection.

Mongoose

Mongoose is an object modeling tool for MongoDB and Node.js. Its peculiarity is that programmers do not need to create a schema directly in the database, link it to Object Relational Mapping (ORM), or bind it to project objects and classes. It is enough to define the data structure in JSON format in the project code. [27]

Communications

- The presentation layer interacts with the business logic through an HTTP/REpresentational State Transfer (REST) API, using tools such as Axios to send requests.
- The business logic processes requests, authenticates users with JSON Web Token (JWT) and interacts with the data layer through the TCP/IP protocol.
- The data layer provides and stores information necessary for the application's operation, ensuring CRUD operations (create, read, update, delete data).

Each of these components plays a specific role in the application's architecture, ensuring its efficiency, scalability, and security.

MERN stack

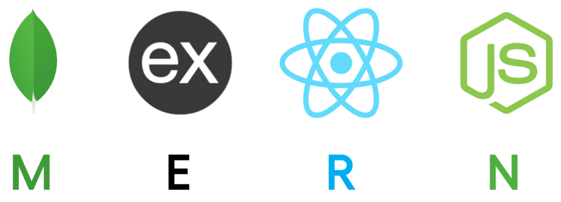


Figure 4.5: MERN stack.

As a result, it was decided that the most efficient development solution would be to use a variation of the MERN stack. In the classic version, this stack includes MongoDB, Express.js, React.js and Node.js. The main difference in our variation is that the user interface will be built on React Native, focused on creating mobile applications.

A significant advantage of the MERN stack is the use of a single programming language, JavaScript, for both client-side and server-side development. This consistency simplifies the coding process.

The choice of the MERN stack variation in this project provides flexibility, scalability, and unification of the technology stack. This allows for efficient and fast development while maintaining a high-quality application. [28]

4.5 User interface design

In the context of mobile app development, prototyping is the creation of an initial version of an app to demonstrate an idea, evaluate design decisions, and understand potential problems and solutions. The prototyping process is flexible as it does not follow strict rules. The prototype does not have to be perfect, its main function is to facilitate testing of the idea and reduce risks in further development. [20]

Before implementing the application, you need to decide how it will look and for this purpose, you will need to create a prototype of the application. To create a prototype I used such a tool as Figma [21], as it is intuitive and suitable even for those without previous experience in prototyping websites or applications.

When designing this app, the main priority was to ensure that it was easy to use for pensioners and that the interface elements were clear and visible. I chose the "Poppins" font for its good readability and soft, rounded letterforms that provide a comfortable perception of the text. The main rule of thumb when choosing a font for people with low vision is to avoid serif fonts and use large font sizes.

The color palette of the app was chosen to reflect the color trends of 2023 and to provide a calming and contrasting visual experience. Shades of blue are considered the most calming in the colour palette and they were key in my selection.

In addition, special attention was paid to the layout of buttons and forms on the page to maximize the ease of interaction between pensioners and the app.

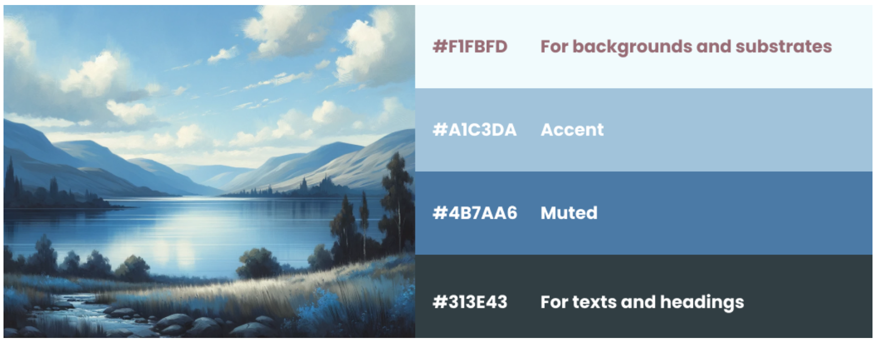

#F1FBFD For backgrounds and substrates
#A1C3DA Accent
#4B7AA6 Muted
#313E43 For texts and headings

Figure 4.6: Color palette.

Chapter 5

Implementation

This section focuses on the implementation of the application. The first part presents a model illustrating the architecture of the application. The second part discusses the project structure, and its implementation and describes the technologies used to create the application “MySupport”.

5.1 Diagram component

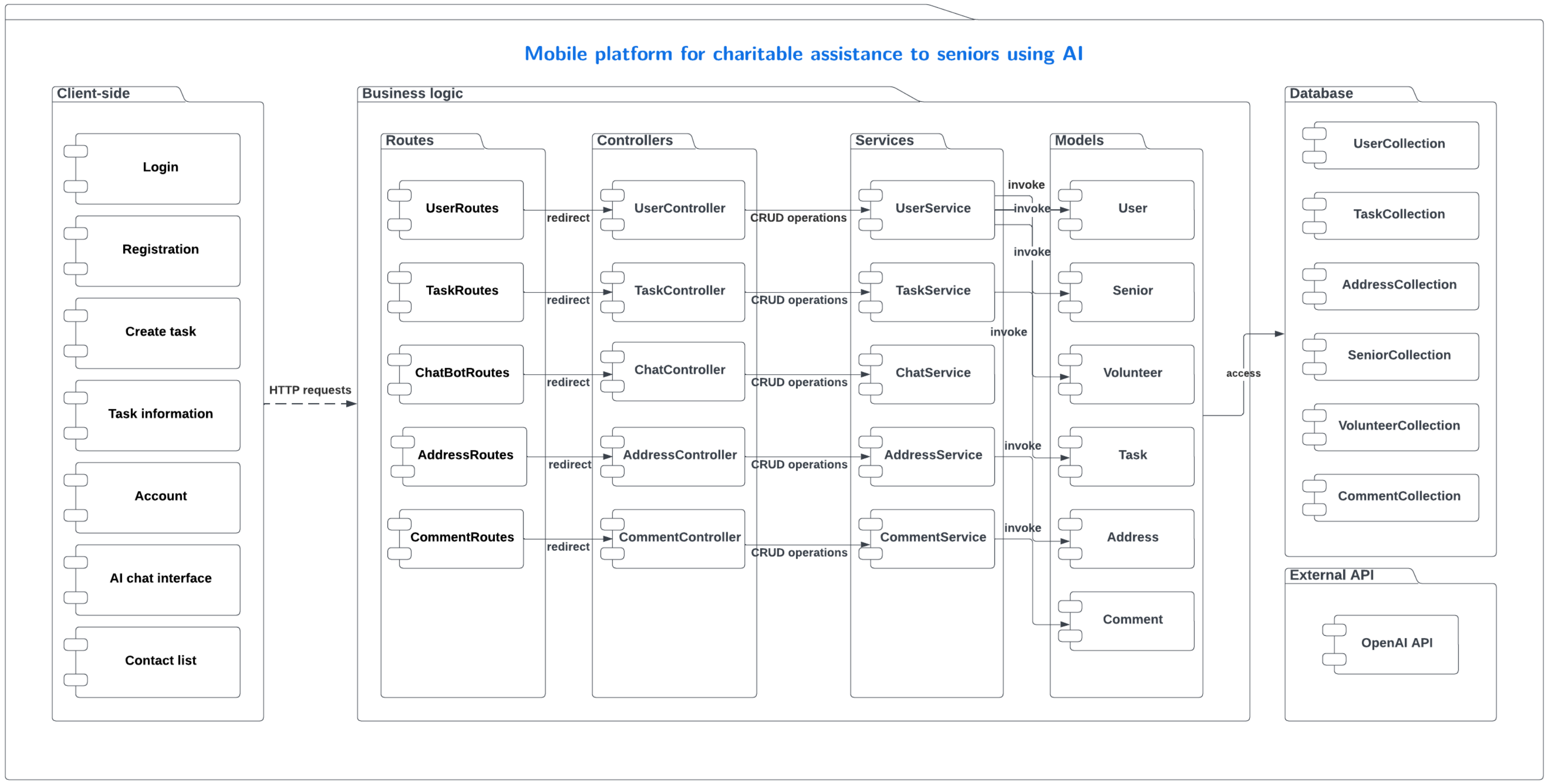


Figure 5.1: Diagram component.

The image shows a diagram of the mobile platform architecture. It consists of three main parts:

- Client-side: includes login, registration, task creation, task overview, account management, AI-based chat interface, and contact list. This is what users interact with directly.
- Business logic: consists of controllers that accept HTTP requests from the client side and delegate them to the appropriate services. These services perform the basic processing logic.
  - Routes: define the HTTP endpoints for various functions and correspond to controllers. They facilitate routing requests from the client side to controllers.
  - Controllers: includes UserController, TaskController, ChatController, AddressController, and CommentController. They are the entry points for processing requests related to their functions.
  - Services: UserService, TaskService, ChatService, AddressService, and CommentService handle more complex logic and interact with external APIs and models.
  - Models: are used to interact with the database, and store and retrieve data from collections.
  - External API: used to integrate with external services such as the OpenAI API, which can enrich the platform with advanced AI capabilities.
- Database: contains collections such as UserCollection, TaskCollection, AddressCollection, SeniorCollection, VolunteerCollection, and CommentCollection where the data used in the application is stored.

This structure reflects a three-tier architecture and illustrates a clear separation of client-side, business logic, and database, thus providing easy scalability and support for the mobile platform.

5.2 Application architecture

My project will use a three-tier architecture, a client-server model with distinct layers for user interface, business logic, and data storage. Each layer is maintained independently on different platforms. [22] It includes the following tiers:

- Presentation tier: manages the user interface and user interactions.
- Application tier: handles business logic and data processing.

- Data tier: responsible for data storage and ensuring data integrity and security.

The benefits of this architecture include scalability, increased reliability, enhanced security, and improved performance, allowing each tier to be scaled and updated independently. [23]

5.3 Presentation tier

This section describes the key components of the presentation tier of the application, as well as the overall code structure.

5.3.1 Code structure

In the project, the `frontend/MySupport` folder includes the following subfolders and key files:

- `assets/` — contains static resources such as images, fonts, and other media files used in the project.
- `components/` — includes reusable interface components, such as modal windows, app footer, task, and other UI elements.
- `reducers/` — contains reducer functions for managing the application state through Redux.
- `screens/` — includes components, each corresponding to an individual screen of the mobile application.
- `styles/` — contains style files that define the appearance of the screens in the project.
- `App.js` — the main JavaScript file that initializes key resources.
- `AppRoutes.js` — defines the routing of the application.

5.3.2 Routing

React-navigation library is used for in-app navigation, allowing to creation of navigation stacks. Each screen is associated with a unique route and component:

```
const Stack = createNativeStackNavigator();

const AppRoutes = () => (
  <Stack.Navigator initialRouteName="StartScreen">
    <Stack.Screen name="StartScreen" component={StartScreen} />
    <Stack.Screen name="RegisterScreen" component={
   RegisterScreen} />
     {/* Other screens */}
  </Stack.Navigator>
);
```

Listing 5.1: Routing in "My Support"

StartScreen is set as the initial screen, ensuring it loads at application startup. Using the `navigation.navigate` method, transitions are managed and parameters are passed:

```
navigation.navigate('UserInfoScreen', { userId: 123 });
```

Listing 5.2: `navigation.navigate`

On the target screen, parameters are retrieved as follows:

```
const { userId } = route.params;
```

Listing 5.3: `route.params`

The use of React navigation ensures smooth and efficient screen transitions.

5.3.3 Redux Toolkit

In the project, state management is handled using Redux with the Redux Toolkit. The management of the user authentication state illustrates how Redux works. The `authSlice` contains the user Identifier (ID) and access token, along with reducers to update these values:

```
const authSlice = createSlice({
  name: "auth",
  initialState: { id: 0, token: null },
  reducers: {
    saveUser(state, action) { state.id = action.payload.id;
   state.token = action.payload.token; },
    logoutUser(state) { state.id = 0; state.token = null; },
  },
});
```

Listing 5.4: `authSlice`

Components modify the state using the dispatch function, for example, `dispatch(logoutUser())`. The `authSlice` is integrated into the main store through `configureStore`, ensuring centralized management and access to data:

```
import authReducer from "./authSlice";

export default configureStore({
  reducer: { user: authReducer },
});
```

Listing 5.5: `configureStore`

5.3.4 Communication with application tier

Communication between client-side and business logic is done through HTTP requests using the `axios` library.

Example of using `axios` :

```
const loginUser = async (email, password) => {
  try {
    const response = await axios.post("http://172.20.10.9:4444/
   api/auth/login", { email, password });
    dispatch(saveUser({ id: response.data._id, token: response.
   data.token }));
  } catch (error) {
    setErrorMessage(error.response.data.error);
    setModalVisible(true);
  }
};
```

Listing 5.6: `Axios example`

In this example, we send a POST request to the `/auth/login` endpoint, passing the user data. If the response is successful, we save the user data to the Redux repository. If errors occur in the process, they are handled and displayed to the user.

5.4 Application tier

This section will look at how the application tier is organized. It includes the code structure and describes the interaction between the main components: routes, controllers, services, and models.

5.4.1 Code structure

In the project, the `backend/` folder includes the following subfolders and key files:

- `config/` — contains configuration settings and environment variables for the application, including database connection settings.
- `controllers/` — responsible for handling incoming requests and sending responses to the client.
- `middleware/` — middleware functions for processing requests before reaching the controllers.
- `models/` — MongoDB database schemas that define the structure of the data.
- `routes/` — defines routes to handle requests from the client.
- `services/` — provides logic to interact with the database and perform data operations, servicing the controllers.
- `test/` — includes tests for various application components.
- `utils/` — utility functions and helpers that are used across the application.
- `validations/` — contains middleware for validating data sent to the server.
- `index.js` — the main entry point of the application which sets up and starts the server.

5.4.2 Routes

In the application, routing on the back-end side is managed using Express Router, which is used to define various routes that handle HTTP requests.

An example of using Express Router can be seen in the route for creating a new task. Here's how it's implemented:

```
import express from "express";
import { taskCreateValidation } from "../validations/taskCreateValidation.js";
import { TaskController } from "../controllers/index.js";
import { validate } from "../middleware/validationMiddleware.js";
import authMiddleware from "../middleware/authMiddleware.js";

```

```
const router = express.Router();

router.post(
  "/tasks",
  authMiddleware,
  validate(taskCreateValidation),
  TaskController.createTask
);
```

Listing 5.7: taskRoutes – create task

- `authMiddleware` is used to ensure that the user is authenticated before allowing operations related to creating a task.
- `validate(taskCreateValidation)` is applied to check the correctness of the data submitted by the user. This helps to prevent errors and enhances the reliability of data processing.
- `TaskController.createTask` is a function of the controller that handles incoming requests and sends responses to the client.

5.4.3 Controller layer

After defining routes in Express Router, the next key component of the application architecture is the controllers. Controllers serve as a link between routing and business logic, implemented at the service level. They handle incoming HTTP requests, call the appropriate services to perform business operations, and send responses to the client.

An example of a controller operation in Express is illustrated by the `createTask` function:

```
import taskService from "../services/TaskService.js";

export const createTask = async (req, res, next) => {
  try {
    const userId = req.userId; // Extracting the user ID from
    the request
    const taskData = req.body; // Receiving the task data from
    the request body
    const task = await taskService.create(taskData, userId); //
    Calling the service to create a task

    res.status(201).json(task); // Sending a response with the
    created task and status 201
  } catch (err) {
    next(err); // Passing the error to the error handler
  }
};
```

Listing 5.8: TaskController - createTask

5.4.4 Service layer

After controllers, the next key component of the application architecture is the service layer. Services are responsible for executing business logic and interacting with data models. This abstraction helps to separate HTTP request processing in controllers from operations directly related to business logic and the database.

An example of how the service layer is implemented can be illustrated with the `create` function from the task service:

```
import Task from "../models/Task.js";
import User from "../models/User.js";

async create(taskData, userId) {
    const user = await User.findById(userId);
    if (!user || user.role !== "SENIOR") {
      throw new Error("Only SENIOR users are allowed to create
   tasks");
    }

    const doc = new Task({
      ...taskData,
      senior: userId,
    });
    return await doc.save();
  }
```

Listing 5.9: TaskService – create

In the example provided, the `create` function from the task service demonstrates how services can be effectively used to encapsulate the business logic of the application. This is particularly important for maintaining order and cleanliness of the code, as well as ensuring its scalability and ease of testing.

5.4.5 Model layer

Models in Mongoose serve as the schema definition and the interface to the underlying database for creating, querying, updating, and deleting records.

Here is an example of how models are structured, using Task as an example:

```
import mongoose from "mongoose";
import { TaskCategory } from "./TaskCategory.js";
const Schema = mongoose.Schema;

const TaskSchema = new Schema(
  {
    title: { type: String, required: true }, // Task title, a
   required field
```

```
    countOfVolunteer: { type: Number, default: 1 }, // Number of volunteers
    description: { type: String, required: true }, // Task description, a required field
    status: {
      type: String,
      enum: ["CREATED", "IN PROCESSING", "COMPLETED", "CANCELED"], // Enumeration of possible task statuses
      default: "CREATED", // Default status when a task is created
    },
    comments: [{ type: Schema.Types.ObjectId, ref: "Comment" }], // References to comments on the task
    ratings: [
      {
        ratedBy: { type: mongoose.Schema.Types.ObjectId, ref: "User" }, // User who provided the rating
        ratedUser: { type: mongoose.Schema.Types.ObjectId, ref: "User" }, // User who is rated
        rating: Number, // The rating value
        createdAt: Date, // Date when the rating was created
      },
    ],
    senior: {
      type: mongoose.Schema.Types.ObjectId,
      ref: "User",
      required: true, // The user responsible for the task, a required field
    },
    volunteers: [
      {
        type: mongoose.Schema.Types.ObjectId,
        ref: "Volunteer", // References to volunteers participating in the task
      },
    ],
    category: {
      type: String,
      enum: Object.values(TaskCategory), // Task category from a predefined list
    },
    address: { type: mongoose.Schema.Types.ObjectId, ref: "Address" }, // Address associated with the task
  },
  {
    timestamps: true, // Automatically adds createdAt and updatedAt fields to the record
  }
);

// Exporting the model for use in other parts of the application
export default mongoose.model("Task", TaskSchema);
```

Listing 5.10: Task – model

5.4.6 Connection to database

The code snippet below demonstrates how to set up a connection to MongoDB using Mongoose:

```
import mongoose from "mongoose";

const connectDatabase = async () => {
  mongoose
    .connect(
      "mongodb+srv://{username}:{password}@cluster0.dmp2pra.
    mongodb.net/mySupport?retryWrites=true&w=majority&appName=
    Cluster0"
    )
    .then(() => console.log("MongoDB connected successfully"))
    .catch((error) =>
      console.error("MongoDB connection failed:", error.message)
    );
};
export { connectDatabase };
```

Listing 5.11: Connection to database

5.5 Data tier

The MongoDB database was used in the development of the application.

MongoDB is a Not only SQL (NoSQL) database of the “document-oriented” type, which does not require a fixed data schema. It allows storing documents in JSON format with a flexible structure that can be changed without the hard constraints typical of Structured Query Language (SQL) databases. This approach speeds up application development and makes it easier to scale and maintain. [29]

Mongoose is used to facilitate communication between the application tier and the database. It is an Object Data Modeling (ODM) library designed for use with MongoDB and Node.js. It handles data relationships, offers schema validation, and makes it easy to convert objects in code to their representation in MongoDB. [29]

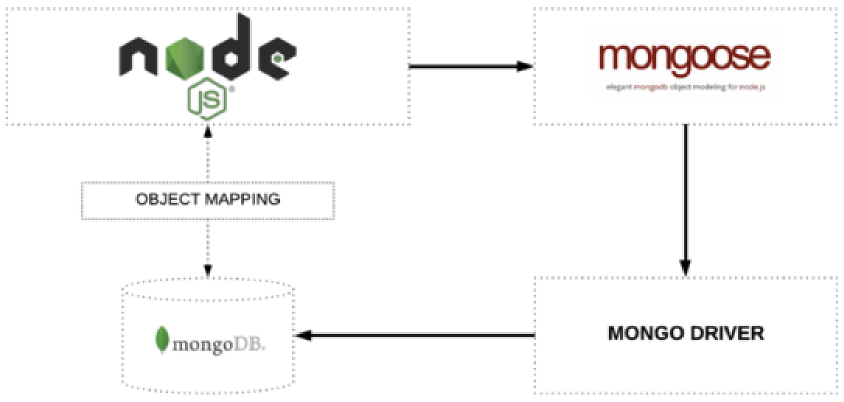


Figure 5.2: Object mapping between Node and MongoDB is handled by Mongoose.

List of database collections:

- `addresses` - contains address data for seniors, including street, house number, city, and postal code.
- `comments` - stores user comments on tasks, including the author, text, and date of creation.
- `seniors` - information about elderly users, including their address and date of birth.
- `tasks` - a list of tasks assigned to volunteers, with descriptions, status of completion, and task category.
- `users` - general data about system users, including login, password, role in the system, and contact details.
- `volunteers` - data about volunteers, including their working hours.
- `workinghours` - records of volunteers' working hours, detailing start and end times.

5.6 Tools used

While developing the server-side of the application, a range of tools and libraries were utilized to ensure high performance, security, and ease of data management:

- Mongoose: this is an ODM library for MongoDB and Node.js. Mongoose allows for managing data and interacting with MongoDB through convenient models.

- DotENV: a tool for handling environment variables stored in .env files. It allows for the secure management of sensitive data.
- Express-validator: a library for string validation that is used to check the correctness of input data.
- bCrypt: a module for hashing passwords.
- JWT: a technology for the secure exchange of information between a client and a server. JWTs are used for authentication and authorization in applications, where tokens contain all necessary data to verify a user, avoiding repeated database queries.
- Cross-Origin Resource Sharing (CORS): a mechanism that allows or restricts requested resources on a web page depending on the domain or port from which the request was made.
- Nodemon: a tool that automatically restarts the server whenever project files are changed.

5.7 OpenAI API integration

ChatGPT, developed by OpenAI, is an advanced language model that has significantly popularized NLP by making it more accessible. Previously, engaging with NLP required extensive datasets and specialized expertise. Now, ChatGPT enables even beginners to generate human-like text from simple prompts, thus bridging the gap between advanced technology and the general public. [30]

5.7.1 Implementation

Chat service

The `generateChatResponse` method in the ChatService class makes requests to the OpenAI API to generate text responses using the Axios library. This method is key for processing requests from the client application and interacting with artificial intelligence. The main steps of the method are described below.

```
async generateChatResponse(text) {
  // Sending a POST request to the OpenAI API using the "gpt-3.5-turbo" model to generate responses
  const response = await axios.post("https://api.openai.com/v1/chat/completions", {
    model: "gpt-3.5-turbo",
    messages: [{ role: "user", content: text }],  // Sending user's text
  }, {
    headers: {
      Authorization: `Bearer ${process.env.OPENAI_API_KEY}`,  // Authentication using the token from environment variables
      "Content-Type": "application/json",  // Data format - JSON
    },
  });

  // Response analysis: check for the necessary data in the response
  if (response.data && response.data.choices && response.data.choices[0].message) {
    return response.data.choices[0].message.content.trim();  // Return the message text, trimmed of spaces
  } else {
    throw new Error("Invalid or unexpected response structure");  // Error if the response structure is incorrect
  }
}
```

Listing 5.12: ChatService – generateChatResponse

Chat controller

An instance of ChatService is exported and used in the API controller. This controller handles POST requests from the client, where the request body contains the user's message text:

```
export const postChat = async (req, res, next) => {
  try {
    const message = await chatService.generateChatResponse(req.body.text);
    res.send({ message });
  } catch (err) {
    next(err);
  }
};
```

Listing 5.13: ChatController - postChat

Here, the `generateChatResponse` method is called, passing in the text from the request. The server's response is then sent back to the client, allowing the mobile application to receive responses from OpenAI.

Integration into a React Native mobile application

On the client side, the ChatBotScreen component plays a key role in facilitating interactive communication between the user and AI through chat. This component implements the `onSend` function, which is responsible for sending user text messages to the server and handling responses from the bot. The `onSend` function works as follows:

- Sending messages

  The `onSend` function initiates a POST request to the server API at `http://localhost/api/chat`, including the user's message text in the request body:

```
axios.post("http://172.20.10.9:4444/api/chat", { text })
.then((response) => {
  // Handling successful server response
  const botResponse = response.data.message;
  setMessages((previousMessages) =>
    GiftedChat.append(previousMessages, [{
      _id: Math.round(Math.random() * 1000000),
      text: botResponse,
      createdAt: new Date(),
      user: {
        _id: 2,
        name: "ChatBot",
        avatar: "https://freelogopng.com/images/all_img
/1681038242chatgpt-logo-png.png",
      },
    }])
  );
})
.catch((error) => {
  // Logging errors when accessing the server
  console.error("Error when accessing the server:", error)
;
});
```

Listing 5.14: ChatBotScreen - onSend

- Handling responses

  After successfully receiving a response from the server, the response text is integrated into the chat interface. This is accomplished using the `setMessages` method, which adds new messages to the existing ones in the chat.

- Exception handling

  In case of errors during the request transmission or response reception from the server, the function actively logs these errors.

Chapter 6

Testing

Software testing is essential for identifying and addressing bugs in software development, aiming not only to find defects but also to evaluate the software's quality against specified goals. Due to software complexity, achieving complete testing is challenging, but it remains a crucial indicator of quality, including aspects like correctness and reliability. [24]

In this chapter, we will look at usability testing and unit testing.

6.1 Unit testing

A unit test is a segment of code that checks the precision of a specific, isolated section of application code, usually a function or method. It is crafted to ensure that this code segment functions as anticipated, in line with the developer's intended logic. The unit test interacts with the code segment solely through inputs and evaluates the outcomes as either true or false. [31]

As part of the bachelor's thesis, unit tests were conducted for controllers and services. In this section, we will examine the tools used during testing and demonstrate the implementation of tests.

6.1.1 Tools used

- `Mocha`: a framework for organizing and executing unit tests. It is identified by the keywords `describe` and `it`, which structure the tests and define test cases, respectively.
- `Chai`: a library for formulating assertions that check the correctness of code implementation.
- `Sinon`: a tool for creating stubs and mocks, allowing for the testing of modules without external dependencies.

- `supertest`: simulates HTTP requests to the server for testing controllers.

6.1.2 Testing controllers

Controllers are tested for the correct handling of HTTP requests and generating responses, using `supertest` to simulate requests and check responses, ensuring the API functions correctly in isolation.

```
describe("register", () => {
    it("should register a new user successfully", async () => {
        // Create a mock user object and a mock token to simulate database response
        const mockUser = { _id: "user1", email: "test@example.com", name: "Test User" };
        const mockToken = "jwtToken";

        // Stub the registerUser method of UserService to resolve with the mock data
        sinon.stub(UserService, "registerUser").resolves({ user: mockUser, token: mockToken });

        // Define the request body to simulate the incoming request data
        const requestBody = { email: "test@example.com", password: "password", name: "Test User" };

        // Send a POST request to the register endpoint with the request body
        const response = await request(app).post("/register").send(requestBody);

        // Check if the response status code is 201 (created)
        expect(response.status).to.equal(201);
        // Verify that the response body matches expected data structure and content
        expect(response.body).to.deep.equal({
            data: mockUser,
            token: mockToken,
            message: "User successfully registered",
        });
    });
});
```

Listing 6.1: UserControllerTest - register

6.1.3 Testing services

Services are tested for their ability to perform user data operations, including CRUD - operations and authentication. Stubs are used to simulate interactions with the database and authentication processes, allowing for precise testing of business logic.

```
describe("Login user test", () => {
    let bcryptCompareStub, jwtSignStub;

    beforeEach(() => {
        sinon.restore();
        bcryptCompareStub = sinon.stub(bcrypt, "compare").
    resolves(true);
        jwtSignStub = sinon.stub(jwt, "sign").returns("fake_jwt_
    token");
    });

    it("should authenticate the user and return token", async ()
     => {
        const email = "V@example.com";
        const password = "12345";
        const result = await UserService.loginUser(email,
    password);

        expect(result).to.have.property("token");
        expect(result.token).to.equal("fake_jwt_token");
        expect(bcryptCompareStub.calledOnceWith(password, sinon.
    match.string)).to.be.true;
    });
});
```

Listing 6.2: UserServiceTest - login

6.2 Usability testing

Usability testing in software testing is a type of testing that is performed from the end user's perspective to determine the ease of use of a system. Usability testing is generally the practice of testing how easy a design is for a group of representative users. [25]

Usability testing was conducted with a group of three elderly people. This demographic segment was chosen because the application is primarily focused on helping the elderly. The main objective was to evaluate the intuitiveness and accessibility of the interface for users who may not be very familiar with modern technology.

Each test participant was given scenarios covering key functions of the app. In addition to completing the assigned tasks, participants were given the freedom to use the application to gain a deeper understanding of the user experience.

6.2.1 Testing goals

The main purpose of testing was to identify aspects of the application that may cause difficulties for the average user. An important objective was to make the prototype more intuitive and user-friendly. Special attention was paid to the usability of the interface, readability of text, and ease of interaction with the application.

6.2.2 Testing scenarios

Scenario 1

1. Sign up.
2. Login.
3. Create a task:
   a. Enter a task title.
   b. Enter a description of the task.
   c. Select the category corresponding to the description.
   d. Set an address of the task.
   e. Find and click on the "Create" button.
4. On the home page, review your tasks.

Scenario 2

1. Login.
2. Ask AI chat a question:
   a. Enter your question in the text box.
   b. Send your question.

Scenario 3

1. Login.
2. Modify account information:
   a. Modify any account information settings.
   b. Save the changes.

3. Check on the account page to make sure the data is displayed correctly.
4. Delete the account by clicking on the “Delete account” button:
    a. In the pop-up window, confirm your wish to delete the account.

6.2.3 Feedback from testers

Tester 1

“I found the registration process quite complicated due to the numerous steps. It would be more convenient if the number of mandatory items to fill out was reduced. I liked the ability to ask AI chat questions in the app - it will make my life much easier and save me from having to ask my son for help.”

Tester 2

“The color palette of the app was a pleasant surprise - my eyes didn’t get tired looking at the screen. The large buttons make the app easy to use, unlike other apps where I find it difficult to even hit a button. However, the sign-up form proved to be complicated for me. It would have been better to split it into several steps, and at the end provide a confirmation of the information entered, so I could make sure the app recognized it correctly and that I had spelled everything correctly.”

Tester 3

“I was very pleased that the app warns of possible erroneous actions when changing account details or deleting an account. This helps to avoid unwanted actions. I did not encounter any problems during testing: everything was clear and well visible.”

6.2.4 Test results

Usability testing of the app with older adults provided valuable feedback and unique insights into the functionality and design of the product. Based on feedback from testers, a few key takeaways can be drawn.

- Simplifying the registration process: many participants struggled with the long and complicated registration form. Reducing the number of required fields and breaking the process into several steps with clear instructions can greatly improve the user experience.

- Confirmation of user actions: alerts for important actions, such as changing account details or deleting an account, were highly appreciated by users. This emphasizes the need to integrate additional warning and confirmation messages to prevent erroneous actions.
- Interface optimization: positive feedback on the color palette and large buttons highlights the importance of creating a visually comfortable and intuitive interface. At the same time, attention should be paid to a clearer presentation of key controls.

Chapter 7

Conclusion

The aim of this bachelor's thesis was to analyze, develop, and implement the mobile application “MySupport”, which allows elderly people to send requests for assistance and volunteers to provide the necessary support. The project involved a thorough analysis of competitors, identifying their strengths and weaknesses, which formed the basis for defining the key requirements for the application. Special attention was given to comparing existing APIs for integrating AI into the mobile application. During this analysis, a user interface prototype was also developed, specifically tailored to the needs of senior users. Thanks to an in-depth analysis, the optimal technological solution for implementation was selected. In the process, a lot of literature and other data sources were studied, contributing to the creation of an efficiently working application. After the implementation of the application, unit testing and usability testing were conducted with the elderly. These tests helped identify and correct minor flaws and identify areas for future product improvement.

In my opinion, the goals of the bachelor's work were successfully achieved: all planned functionalities were implemented, and the application is ready for use.

7.1 Future development plans

As part of the app improvement plan, the following areas for further development have been identified:

- adding functionality to attach photos to tasks.
- Creating a chat to facilitate communication between volunteers and seniors.
- Simplifying the registration process for seniors.

- Introducing new user roles: administrator and moderator.
- Implementing a feature to save chat conversations with the chatbot for seniors.

Appendix A

Bibliography

[1] Č. j.: MPSV-2023/45298 *Žádost o poskytnutí informace dle zákona č. 106/1999 Sb., o svobodném přístupu k informacím – statistické info - důchodci* [online] [22 February 2023] Available from: `https://www.mpsv.cz/documents/20142/4837218/ÄŊ.j.+MPSV+2023-45298+StatistickĂľ+info+dĹŕchodci.pdf/7c135e5a-b71a-9697-8887-c13226d33358`

[2] Č. j.: Český statistický úřad *Průměrná roční míra inflace v ČR v roce 2022 byla 15,1%* [online] [11 December 2023] Available from: `https://www.czso.cz/csu/xe/prumerna-rocni-mira-inflace-v-cr-v-roce-2022-byla-151-`

[3] Jan Cieslar *Výdaje na invalidní důchody loni přesáhly 50 miliard korun* [online] Available from: `https://www.czso.cz/csu/czso/vydaje-na-invalidni-duchody-loni-presahly-50-miliard-korun`

[4] *Mediasafe* [online] Available from: `https://www.medisafe.com`

[5] *Carely* [online] Available from: `https://www.care.ly`

[6] *Life360* [online] Available from: `https://www.life360.com/intl/`

[7] *eCare21* [online] Available from: AppStore Google Play

[8] *Senior safety app* [online] Available from: `https://www.seniorsafetyapp.com`

[9] Rahul Awati *What is requirements analysis (requirements engineering)?* [online] [June 2023] `https://www.techtarget.com/searchsoftwarequality/definition/requirements-analysis`

[10] By Phillip A. Laplante, Mohamad Kassab: *Requirements Engineering for Software and Systems* [online] [7 June 2022]Available from: `https://www.taylorfrancis.com/books/mono/10.1201/9781003129509/requirements-engineering-software-systems-phillip-laplante-mohamad-kassab`

[11] IBM *Use cases* [online] [5 March 2021] Available from: https://www.ibm.com/docs/en/rsar/9.5?topic=diagrams-use-cases

[12] IBM: *Class diagrams* [online] [5 March 2021]Available from: https://www.ibm.com/docs/en/rsm/7.5.0?topic=structure-class-diagrams

[13] TechTarget Contributor: *Class diagram definition* [online] [May 2019] Available from: https://www.techtarget.com/searchapparchitecture/definition/class-diagram

[14] *Dialogflow Google AI* [online] Available from: https://cloud.google.com/dialogflow?hl=ru

[15] *Wit.ai by Facebook* [online] Available from: https://wit.ai

[16] *RASA* [online] Available from: https://rasa.com/docs/

[17] *OpenAI API* [online] Available from: https://openai.com/blog/openai-api

[18] *LaMDA by Google AI* [online] Available from: https://blog.google/technology/ai/lamda/

[19] SPARX SYSTEMS Enterprise Architect *UML 2 Tutorial - Sequence Diagram* [online] Available from: https://sparxsystems.com/resources/tutorials/uml2/sequence-diagram.html

[20] Benjamin Bähr *Prototyping of User Interfaces for Mobile Applications* [online] [2017] Available from: https://link.springer.com/content/pdf/10.1007/978-3-319-53210-3.pdf

[21] *Figma* [online] Available from: https://www.figma.com/community

[22] Margaret Rouse *Three-Tier Architecture* [online] [26 June 2023] Available from: https://www.techopedia.com/definition/24649/three-tier-architecture

[23] IBM *What is three-tier architecture?* [online] Available from: https://www.figma.com/community

[24] Jiantao Pan *Software Testing* [online] [Spring 1999] Available from: http://www.sci.brooklyn.cuny.edu/~sklar/teaching/s08/cis20.2/papers/software-testing.pdf

[25] *Usability Testing* [online] [15 Dec, 2023] Available from: https://www.geeksforgeeks.org/usability-testing/

[26] Bonnie Eisenman *Learning React Native: building native mobile apps with JavaScript* [online] [2015] Available from: https://books.google.cz/books?hl=ru&lr=&id=274fCwAAQBAJ&oi=fnd&pg=PR2&dq=Learning+React+Native:+building+native+mobile+apps+with+JavaScript&ots=tHpnbBj4o3&sig=9K7EUzJmMZG0HYwksFbdR09K7fs&redir_esc=y#v=onepage&q=Learning%20React%20Native%3A%20building%20native%20mobile%20apps%20with%20JavaScript&f=false

[27] Simon Holmes *Mongoose for Application Development* [online] [2013] Available from: https://www.google.cz/books/edition/Mongoose_for_Application_Development/DxTFX0771tIC?hl=ru&gbpv=1&dq=Mongoose+for+Application+Development&pg=PT19&printsec=frontcover

[28] Chris Blakely *MERN Stack Roadmap – How to Learn MERN and Become a Full-Stack Developer* [online] [4 Jan, 2024] Available from: https://www.freecodecamp.org/news/mern-stack-roadmap-what-you-need-to-know-to-build-full-stack-apps/#what-is-the-mern-stack

[29] Nick Karnik *Introduction to Mongoose for MongoDB* [online] [11 February 2018] Available from: https://www.freecodecamp.org/news/introduction-to-mongoose-for-mongodb-d2a7aa593c57/

[30] Henry Habib *OpenAI API Cookbook* [online] [March 2024] Available from: https://www.google.cz/books/edition/OpenAI_API_Cookbook/pKP6EAAAQBAJ?hl=ru&gbpv=1&dq=OpenAI+API+Cookbook:+Build+intelligent+applications+including+chatbots,+virtual+assistants,+and+content+generators&printsec=frontcover

[31] *What is Unit Testing?* [online] Available from: https://aws.amazon.com/what-is/unit-testing/?nc1=h_ls

Appendix B

Acronyms

AI Artificial Intelligence. 1, 2, 9, 14, 26, 29, 34, 36, 40, 49, 61, 66, 67, 69

API Application Programming Interface. 34, 39, 43, 60, 61, 64, 69

CORS Cross-Origin Resource Sharing. 59

HTTP HyperText Transfer Protocol. 42, 43, 49, 52–55, 64

ID Identifier. 51

JWT JSON Web Token. 43, 59

NLP Natural Language Processing. 36, 37, 59

NoSQL Not only SQL. 57

ODM Object Data Modeling. 57, 58

ORM Object Relational Mapping. 43

REST REpresentational State Transfer. 43

SQL Structured Query Language. 57

UML Unified Modeling Language. 29

Appendix C

Use cases - extra

- **UC1 – Rate another user**

  - Description:

    - Users want to rate other users in the “My Support” mobile app based on their experience of giving or receiving help.

  - Pre-conditions:

    - The user has launched the “My Support” application.
    - The user has logged into the system.

  - Scenario:

    1. The user filters tasks by selecting the "Completed" status on the main page.
    2. The system displays all tasks that have been marked as completed, showing information about who requested help and who provided it.
    3. The user selects a specific completed task from the list.
    4. The system displays a page with detailed information about the selected task.
    5. At the bottom of the task details, a list of all users involved in the task appears with their ratings.
    6. The user can select a rating from 1 to 5 stars for each individual without needing to confirm their rating.
    7. The system saves the rating and updates the user’s average rating, which becomes visible in their profile to other users.

  - Actors:

    - Authorized user
    - System

- **UC2 – View contact list of volunteers**
  - Description:

    - Seniors want to view a list of volunteers who have assisted seniors in past tasks.
  - Pre-conditions:

    - The senior has launched the “My Support” app.
    - The senior has logged into the system.
  - Scenario:

    1. Senior will go to the “Contacts” page.
    2. The system will display a list of volunteers who have already assisted seniors in past tasks.
    3. The seniors can view information about each volunteer, including their name, lastname and contact number.
    4. The seniors can click on a specific volunteer to make a phone call.
  - Actors:

    - Senior
    - System
- **UC3 – Filter tasks**
  - Description:

    - Volunteers want to filter tasks for help according to their skills or abilities.
  - Pre-conditions:

    - The volunteer has launched the “My Support” app.
    - The volunteer has logged into the system.
  - Scenario:

    1. The volunteer goes to the “Home” page.
    2. The system will display a page showing all available tasks for help.
    3. The volunteer clicks on the “Filter” button.
    4. The system displays the filter options that the volunteer can use to search for tasks for help. Filter options can include date, category, and status.
    5. The volunteer selects specific filters based on their preferences, such as choosing “Health care” for qualifications and “Created” for status.

6. The system will display a list of tasks for help that match the selected filters.
7. The volunteer can view and select the task that suits him/her best and accept it.

- Actors:

  - Volunteer
  - System

- **UC4 – Browse tasks on the map**

  - Description:

    - Volunteers want to view tasks for help on a map in the "My Support" mobile app so they can easily see where help is needed.

  - Pre-conditions:

    - The volunteer has launched the "My Support" app.
    - The volunteer has logged into the system.

  - Scenario:

    1. The volunteer goes to the "Home" page.
    2. The system displays a map with icons representing the different tasks for help in different locations.
    3. The volunteer can scroll on the map and zoom in on the task icons to get more information.
    4. The volunteer taps on a specific task icon on the map.
    5. The system displays the details of that task, including the title, description, date, and category.

  - Actors:

    - Volunteer
    - System

- **UC5 – Notify that the task has been accepted by a volunteer**

  - Description:

    - The system should notify the seniors that their task has been accepted by volunteers.

  - Pre-conditions:

    - The system has the senior's consent to receive notifications.

  - Scenario:

1. A volunteer records a new task.
2. After successfully recording the task, the system notifies the senior that a volunteer has accepted their task for help. The notification will include a message about the task, including its title, description, and other details.

- Actors:

  - Volunteer
  - System

- **UC6 – Notify about new tasks**

  - Description:

    - The system should notify volunteers about new tasks for help from seniors.

  - Pre-conditions:

    - The system has consent from the volunteer to receive notifications.

  - Scenario:

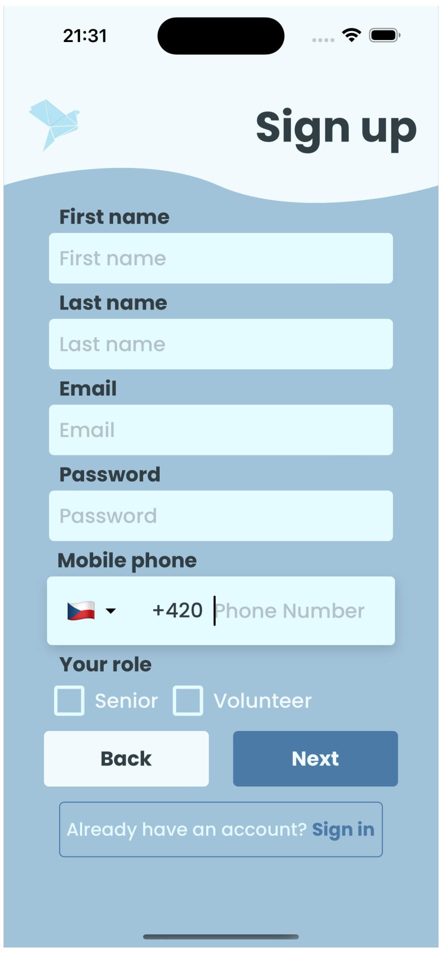
    1. The system regularly monitors and tracks new help tasks that have been created by seniors in the mobile application.
    2. If a new task appears that has not yet been accepted by volunteers, the system generates a notification and sends it to all registered volunteers who are interested in helping seniors. The notification will contain information about the new task, including its title, description, and date.

  - Actors:

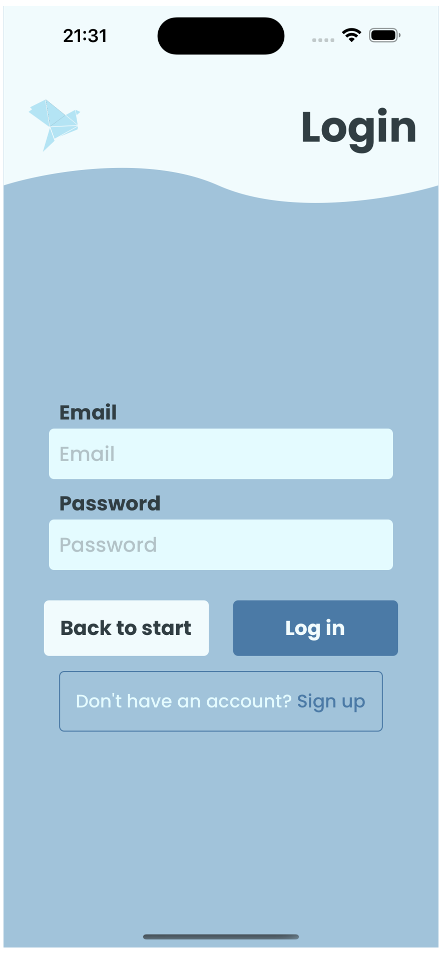
    - Senior
    - System

Appendix D

Final GUI

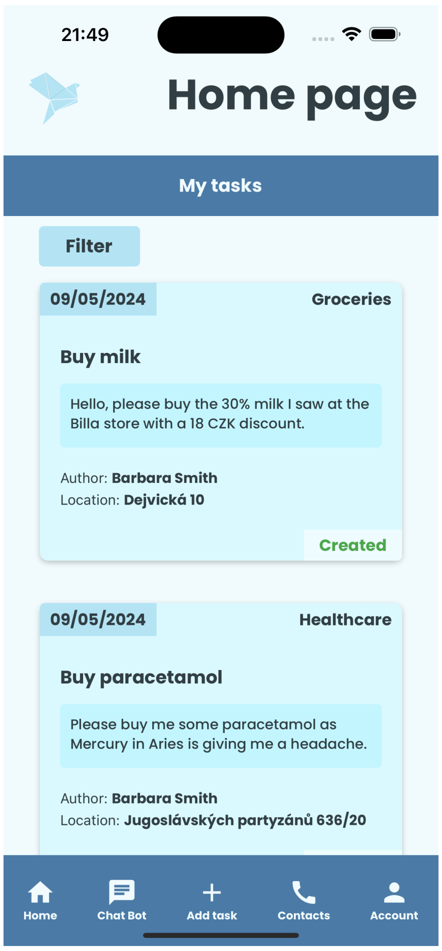


(a) : Register page.



(b) : Login page.

Figure D.1: User interface of the register page and login.


21:49

Home page

My tasks

Filter

09/05/2024 Groceries

Buy milk

Hello, please buy the 30% milk I saw at the Billa store with a 18 CZK discount.

Author: **Barbara Smith**
Location: **Dejvická 10**

Created

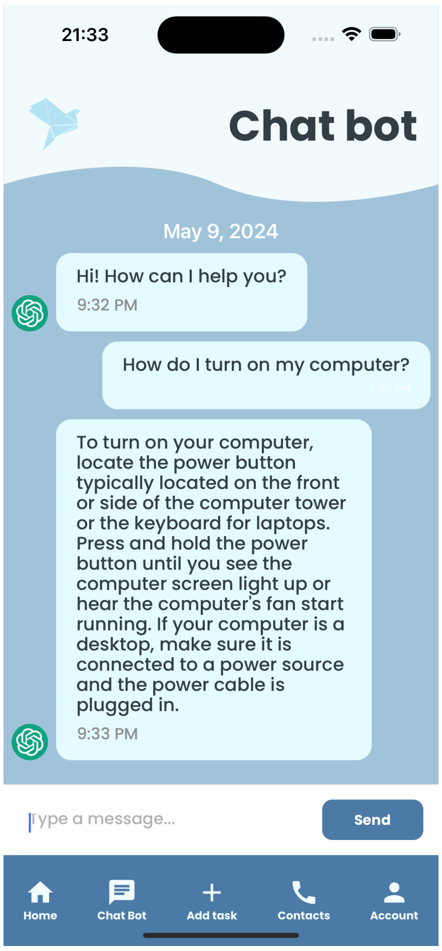
09/05/2024 Healthcare

Buy paracetamol

Please buy me some paracetamol as Mercury in Aries is giving me a headache.

Author: **Barbara Smith**
Location: **Jugoslávských partyzánů 636/20**

(a) : Home page for seniors.


21:33

Chat bot

May 9, 2024

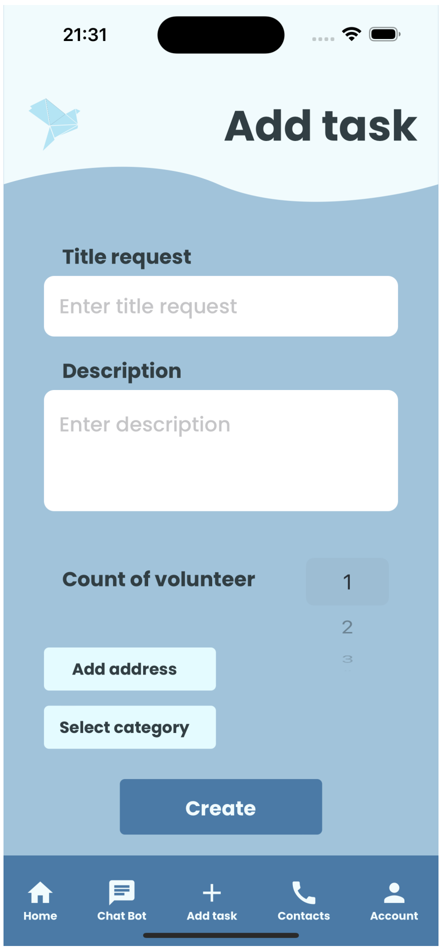
Hi! How can I help you?
9:32 PM

How do I turn on my computer?

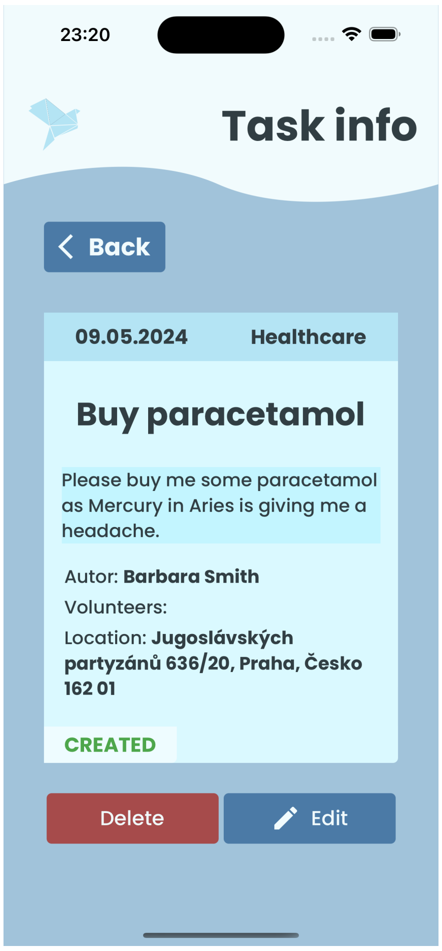
To turn on your computer, locate the power button typically located on the front or side of the computer tower or the keyboard for laptops. Press and hold the power button until you see the computer screen light up or hear the computer's fan start running. If your computer is a desktop, make sure it is connected to a power source and the power cable is plugged in.
9:33 PM

(b) : AI chat page.

Figure D.2: User interface of the home page and chat AI.



(a) : Add task page.



09.05.2024 Healthcare

Buy paracetamol

Please buy me some paracetamol as Mercury in Aries is giving me a headache.

Autor: **Barbara Smith**

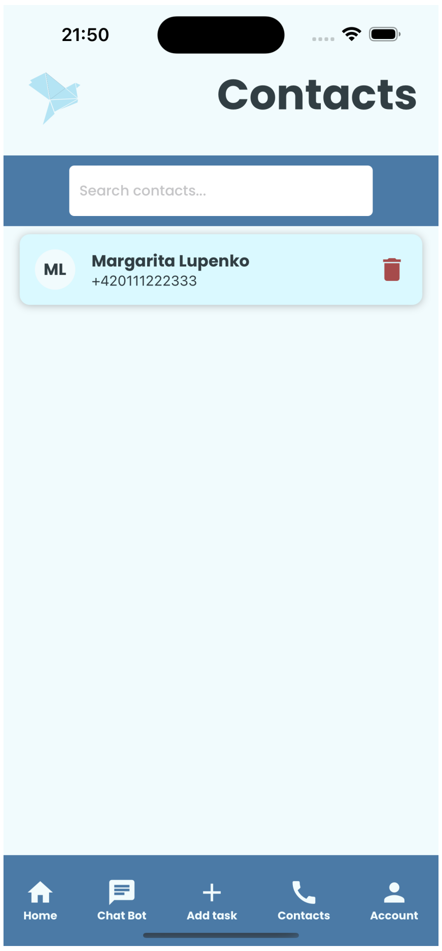
Volunteers:

Location: **Jugoslávských partyzánů 636/20, Praha, Česko 162 01**

CREATED

(b) : Task page

Figure D.3: User interface of the task and creation of the task.



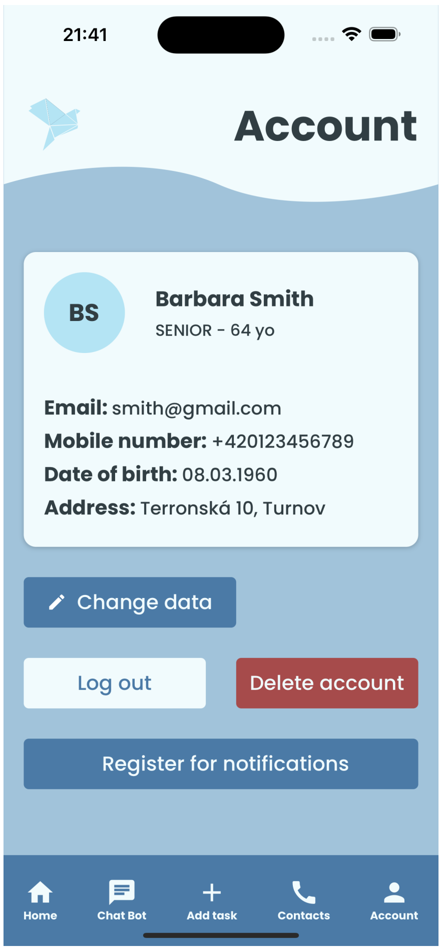
21:50

Contacts

Search contacts...

ML Margarita Lupenko
+420111222333

Home Chat Bot Add task Contacts Account



21:41

Account

BS Barbara Smith
SENIOR - 64 yo

Email: smith@gmail.com
Mobile number: +420123456789
Date of birth: 08.03.1960
Address: Terronská 10, Turnov

Change data

Log out

Delete account

Register for notifications

Home Chat Bot Add task Contacts Account

(a) : Contacts page. **(b) :** Account page.

Figure D.4: User interface of contacts and account.

Appendix E

Run application

To correctly install and launch the back-end and front-end parts of the project, the following steps must be taken:

Back-end installation and launch

- The back-end directory can be accessed with the command: `cd backend/`.
- Dependencies are installed using the command: `npm install`.
- The development server can be started with the command: `npm run start:dev`.

Front-end installation and launch

- The Expo app should be installed on the mobile device.
- Access to the front-end directory is gained by using the command: `cd frontend/MySupport`.
- Dependencies are installed with the command: `npm install`.
- The front-end is initiated with the command: `npm start`.

Upon execution of the last command, a QR code will be displayed in the terminal. The QR code should be scanned with the camera of the mobile device, and the option to open the application in Expo should be selected.