A blue and white sign with white text

Description automatically generated

Software Engineering Department  
ORT Braude College

Capstone Project Phase B – 61999

**ElderEase- An application that connects between elder people and volunteers**

**Authors:**

Natalie Nanikashvily – Natalie.Nanikashvily@e.braude.ac.il

Dina Kakiashvili – dina.kakiashvili@e.braude.ac.il

**Supervisor:**

Dr. Zakharia Frenkel

**24-1-D-23**

Table of Contents:

[Abstract 4](#_Toc177722768)

[Introduction 5](#_Toc177722769)

[The Problem 6](#_Toc177722770)

[Our Solution - "ElderEase" 7](#_Toc177722771)

[**General Description** 7](#_Toc177722772)

[**Elderease Goals and Their Implementation** 8](#_Toc177722773)

[**Target Audience Identification** 9](#_Toc177722774)

[**Methodology and Development Process** 10](#_Toc177722775)

[**Tools Used and Client Interaction During Development** 12](#_Toc177722776)

[Solution Description 14](#_Toc177722777)

[**The ElderEase system** 14](#_Toc177722778)

[**Diagrams Describing the Project** 16](#_Toc177722779)

[Technologies used 17](#_Toc177722780)

[**Client-Side (Front-End)**: 17](#_Toc177722781)

[**Server-Side (Back-End)** 17](#_Toc177722782)

[**Database**: 18](#_Toc177722783)

[**Deployment:** 18](#_Toc177722784)

[Challenges Faced and Solutions Found 19](#_Toc177722785)

[Results and Conclusions 21](#_Toc177722786)

[**Achievements:** 21](#_Toc177722787)

[**Challenges and Unmet Goals:** 22](#_Toc177722788)

[**Decision-Making and Adaptation:** 23](#_Toc177722789)

[Lessons Learned 24](#_Toc177722790)

[**What Worked Well:** 24](#_Toc177722791)

[**What Could Be Improved:** 24](#_Toc177722792)

[User Guide 26](#_Toc177722793)

[**Both Elder’s & Volunteer side:** 26](#_Toc177722794)

[**Login Screen:** 26](#_Toc177722795)

[**Elder’s side:** 27](#_Toc177722796)

[**Registration Screen for seniors:** 27](#_Toc177722797)

[**Elderly Dashboard (on initial entry):** 28](#_Toc177722798)

[**Elderly Dashboard:** 29](#_Toc177722799)

[**Create New Request:** 30](#_Toc177722800)

[**Ongoing Tasks**: 31](#_Toc177722801)

[**Completed Tasks**: 32](#_Toc177722802)

[**Archived Tasks:** 33](#_Toc177722803)

[**Volunteer's side** 34](#_Toc177722804)

[**Registration Screen for volunteers:** 34](#_Toc177722805)

[**Volunteer Dashboard (on initial entry):** 35](#_Toc177722806)

[**Volunteer Dashboard:** 36](#_Toc177722807)

[**New Tasks:** 37](#_Toc177722808)

[**Ongoing Tasks:** 38](#_Toc177722809)

[**Awaiting Confirmation:** 39](#_Toc177722810)

[**Archive:** 40](#_Toc177722811)

[**Volunteer & Elder side** 41](#_Toc177722812)

[Maintenance Guide 43](#_Toc177722813)

[**Purpose** 43](#_Toc177722814)

[**1. System Environment** 43](#_Toc177722815)

[**2. Custom Software Components** 43](#_Toc177722816)

[**Backend - server.js:** 43](#_Toc177722817)

[**FrontEnd Components:** 46](#_Toc177722818)

# **Abstract**

Older individuals, often referred to as seniors or elderly, constitute a diverse demographic with unique needs and challenges. As people age, they may encounter physical, cognitive, and emotional changes that can impact their ability to perform daily activities independently. Mobility issues, chronic health conditions, memory loss, and social isolation are common challenges faced by many older adults. These factors can make tasks such as grocery shopping, meal preparation, medication management, and household chores more challenging. Additionally, the need for companionship and social interaction becomes increasingly vital to combat feelings of loneliness and maintain mental well-being. Recognizing and addressing these needs is crucial to ensuring a high quality of life for the elderly, and volunteer assistance can play a significant role in providing the support and companionship necessary for them to age with dignity and fulfillment. Up until now, there hasn't been a convenient solution for the problem described above.

We propose the development of an application that serves as a bridge between older adults and volunteers eager to assist them in their daily routines. This application leverages modern technology to facilitate meaningful connections and support networks. Users, both older adults, and volunteers, can create personalized profiles with information on interests, skills, and availability. Volunteers can help with daily routines such as grocery shopping, meal preparation, medication reminders, or companionship through planned activities. Feedback and ratings mechanisms are implemented to ensure accountability and maintain a high standard of service, with safety features such as background checks and emergency contact integration.

By providing a user-friendly and secure platform, this application aims to empower volunteers, make a positive impact on the lives of older individuals, and contribute to the creation of a more interconnected and supportive community.

# **Introduction**

In the realm of aging, older individuals often find themselves grappling with a myriad of challenges that significantly impact their daily routines. Physical limitations, cognitive changes, and the complexities of managing healthcare needs can collectively pose formidable obstacles, hindering their ability to navigate even the simplest tasks.

Delving deeper into the daily struggles faced by the elderly, these challenges extend beyond the physical domain. Many older individuals experience social isolation, exacerbated by mobility issues or the absence of a robust support system. The cumulative effect of these difficulties often leads to a diminished quality of life and heightened healthcare costs. Moreover, the lack of accessible and comprehensive support exacerbates the situation, leaving many older individuals underserved and their needs unmet. As our global population continues to age, the demand for tailored solutions to address these challenges becomes increasingly urgent.

Due to the circumstances described above, we propose an innovative solution in the form of an application. This digital platform aims to bridge the gap between older individuals and compassionate volunteers, offering a lifeline of support. Through the application, older users can articulate their daily routines, challenges, and preferences, providing a comprehensive overview of their unique needs. The application's algorithm then leverages this information to seamlessly match them with qualified volunteers who possess the skills and availability to assist. Be it grocery shopping, transportation, or companionship, the application cultivates symbiotic relationships that transcend mere assistance. Integrating features such as volunteer background checks, ratings, and reviews ensures a secure environment, fostering trust between the older individuals seeking support and the volunteers eager to make a positive impact. By harnessing technology to facilitate these connections, the application aims to enhance the overall well-being of older individuals, foster a sense of community, and address the challenges of aging in a more interconnected and supportive society.

# **The Problem**

The Elderease application is designed to address a critical challenge faced by older adults—managing daily tasks and maintaining independence while grappling with the limitations that come with aging. Many elderly individuals experience physical or cognitive decline, which can make basic activities such as shopping, household maintenance, attending medical appointments, or even walking their pets difficult. These challenges are often compounded by social isolation, with many seniors lacking a reliable support system to assist them in their daily lives. This situation can lead to increased dependency, reduced quality of life, and emotional distress.  
  
Furthermore, the aging population often feels disconnected from their communities, and the traditional means of receiving help, such as relying on family or paid services, may not always be available, affordable, or suited to their specific needs. On the other side, there are numerous people who are willing to volunteer their time to help the elderly but may not have an easy or structured way to connect with those in need.  
  
Elderease provides a streamlined solution by creating a platform where older adults can easily post requests for help with specific tasks, and volunteers can browse and accept tasks that fit their schedule and skills. The app not only helps seniors remain independent but also fosters a sense of community and intergenerational connection, providing both practical support and social engagement. Through features like real-time communication, task tracking, and notifications, Elderease ensures that the elderly receive the assistance they need in a timely and efficient manner, all while maintaining their dignity and autonomy.

# **Our Solution - "ElderEase"**

## **Deployment to our project:** [**https://incomparable-taiyaki-956d9f.netlify.app/**](https://incomparable-taiyaki-956d9f.netlify.app/)

## **General Description**

Our envisioned project aims to revolutionize support for older individuals by introducing a comprehensive application that seamlessly connects them with compassionate volunteers, addressing the intricate challenges encountered in their daily lives. The primary issue revolves around the multifaceted nature of aging, encompassing physical limitations, cognitive changes, and the intricacies of managing healthcare needs. These challenges often lead to social isolation, compounding the overall impact on the well-being of older individuals. Beyond the physical realm, the lack of a robust support system exacerbates these difficulties, leaving many older individuals underserved and their unique needs unaddressed.

In response, our proposed application serves as a dynamic solution to bridge this gap and enhance the lives of older individuals. By allowing users to articulate their daily routines, challenges, and preferences, the application facilitates algorithmic matching with qualified volunteers. This personalized matching ensures that volunteers can provide targeted assistance, whether it's in grocery shopping, transportation, or companionship. The integration of features such as volunteer background checks, ratings, and reviews enhances the reliability and trustworthiness of the assistance provided, creating a secure environment for both parties involved. By harnessing technology to foster meaningful connections, the application aspires to mitigate the challenges of aging, elevate the quality of life for seniors, and instill a sense of community support in our interconnected world.

## **Elderease Goals and Their Implementation**

• Improve the quality of life for seniors:  
The app allows seniors to request help with daily tasks like grocery shopping, transportation, and companionship, ensuring they receive support for their essential needs.  
• Provide an intuitive and user-friendly interface:  
Elderease is designed with simplicity in mind, offering seniors an accessible platform to create requests for assistance with minimal effort.  
• Ensure flexible and fulfilling volunteering opportunities:  
Volunteers can choose tasks that fit their schedules, providing a rewarding way to give back, whether through short-term or long-term support.  
• Promote intergenerational relationships:  
The platform encourages connections between seniors and younger generations, helping to reduce social isolation and foster meaningful interactions.  
• Implement a rating system for reliability:  
A built-in rating and review system allows seniors to rate volunteers and provide feedback, ensuring quality and trustworthiness in the services offered.  
• Offer in-app notifications for updates:  
Elderease sends real-time notifications to keep both seniors and volunteers informed about task updates, new requests, and task completions, ensuring a smooth and timely flow of communication.  
• Track ongoing and completed tasks:  
The app maintains a clear record of tasks, both in progress and completed, giving seniors and volunteers an organized overview of their activities.  
• Facilitate easy communication between volunteers and seniors:   
The app provides a seamless communication system, allowing volunteers and seniors to easily exchange messages through an in-app chat. Additionally, users have the option to communicate via phone calls, ensuring flexible and efficient task coordination.

## **Target Audience Identification**

The target audience for Elderease is divided into two primary groups: seniors and volunteers, each with distinct needs and motivations.

Seniors:   
This group consists of older individuals who may require help with everyday tasks due to physical limitations, cognitive changes, or a need for social support. Many seniors face challenges in maintaining their independence and managing daily activities such as grocery shopping, transportation, and household chores. Additionally, some experience social isolation, which can impact their overall well-being. Elderease provides a platform where these seniors can request both practical assistance and companionship, helping them stay connected to their community and improving their quality of life.

Volunteers:   
Volunteers are typically younger individuals or those with flexible schedules who are motivated by the desire to help others and give back to their community. They value the opportunity to provide meaningful support, whether through assisting with physical tasks or offering social engagement for seniors. Elderease offers volunteers flexibility in selecting tasks that fit their availability and interests. The platform enables volunteers to make a positive impact, fostering intergenerational connections and creating a sense of fulfillment through their contributions.

By addressing the needs of both seniors and volunteers, Elderease builds a supportive, compassionate community where seniors receive the help they need, and volunteers can engage in meaningful and rewarding service.

## **Methodology and Development Process**

To build the Elderease system, we undertook a detailed research and development process that focused on understanding the needs of both seniors and volunteers, ensuring that the platform would effectively address the challenges they face.  
  
Research Phase:  
• User Research:   
We began by conducting surveys and interviews with seniors and volunteers to identify their specific needs. For seniors, the research focused on daily challenges like managing household tasks, transportation, and combating social isolation. For volunteers, we explored their availability, motivations, and preferences in terms of task selection and time commitment.  
• Competitive Analysis: We analyzed similar platforms to understand the strengths and weaknesses of existing solutions. This gave us insights into how we could differentiate Elderease, particularly by creating a system that emphasizes personalized task matching and fosters strong community connections between volunteers and seniors.  
• Technology Exploration: Based on the needs identified in our research, we selected appropriate technologies for the project. React was chosen for its flexibility and ability to create a user-friendly interface, while Express.js and Sequelize were selected to provide a robust and efficient back-end solution. PostgreSQL was chosen as the database to ensure reliable and secure data storage, and the entire back-end, including the database, was deployed on Render to ensure seamless scalability and integration.  
  
Development Phase:  
• Prototyping: We created wireframes and interactive prototypes to map out the user experience. The interface was designed to be simple and accessible, especially for seniors, ensuring that they could easily request help with tasks and communicate with volunteers.

• System Architecture: The system was designed with a clear separation between the client (React) and the server (Express.js and JSON Server). This modular approach allowed for efficient task management, notifications, and real-time chat capabilities.  
• Agile Development: We employed an iterative development process, building features in small increments and continuously refining the system. Initial features included task creation, notification systems, and chat functionalities. Each development cycle was followed by user testing, allowing us to gather feedback and make adjustments.  
• User Testing and Feedback: Throughout the development, we involved seniors and volunteers in usability testing. Their feedback was instrumental in refining both the user interface and overall functionality, ensuring the system was intuitive and easy to use.  
  
In the development of our application aimed at facilitating support for older individuals, we have opted for an Agile methodology, which aligns seamlessly with the dynamic nature of our use case. Embracing an iterative approach, this methodology allows us to break down feature delivery into small, manageable components, providing maximum flexibility for adaptability to evolving requirements. Our development process unfolds through several key stages:

• User-Friendly Interface Design: Commencing with the creation of an intuitive and user-friendly interface, our development kicks off by building the mobile application using a well-established platform suitable for older users.

• Functionalities and Features Implementation: We proceed by incorporating essential tracking and movement functionalities within the application, ensuring a seamless user experience for both older individuals and volunteers.  
• Data Storage and Retrieval Mechanism: We integrate features allowing the secure storage and retrieval of essential data, enhancing the reliability and efficiency of the application's support services.  
• User Feedback Integration: A key element of our iterative development is the incorporation of user feedback. We actively seek input from older individuals making adjustments based on their experiences to enhance the application's effectiveness.  
• Continuous Improvement: The development process includes ongoing assessments and adaptations. We proactively address any challenges or inefficiencies identified during user testing, applying changes to ensure continuous improvement.  
• Web-Based Interface Development: As a final step, we create a web-based interface accessible to screening initiators. This interface will showcase the results, allowing for a comprehensive overview of the support activities and analysis.  
  
Throughout the entire development cycle, each iteration concludes with an evaluation phase, where feedback from users is carefully considered. This feedback-driven approach ensures that the application evolves in line with the needs and preferences of older individuals and volunteers, ultimately delivering a comprehensive, user-centric solution to enhance the support system for seniors.

## **Tools Used and Client Interaction During Development**

During the development of the Elderease platform, maintaining clear and consistent communication with the users and stakeholders was a key focus. This ensured that the platform met their needs, stayed aligned with their expectations, and was user-friendly, particularly for seniors who may have varying levels of familiarity with technology.  
  
Why We Maintained Communication:  
User-Centric Development: The main reason for ongoing communication was to ensure that the platform addressed real-world needs. By involving the target audience (seniors and volunteers) early and often, we could gather insights on their preferences, challenges, and expectations.  
Feedback Integration: Regular feedback allowed us to refine the platform’s features and user interface, ensuring that it was intuitive and functional. This was particularly important given the diverse user base of seniors and volunteers, each with different technical abilities and expectations.  
  
How Communication Was Managed:  
• Regular Meetings: We held frequent virtual meetings with stakeholders and potential users using tools like Zoom. These meetings were essential for discussing the progress of the project, gathering feedback on new features, and demonstrating updates. It also provided an opportunity to address any concerns or adjustments required based on real-world use.  
• Surveys and User Testing: To gather more structured feedback, we conducted surveys and usability tests. These allowed us to directly understand how seniors and volunteers interacted with the platform. User testing sessions helped identify pain points in the user experience, such as task creation or messaging, and allowed us to make necessary improvements.  
• Collaborative Tools: Communication and collaboration with both the internal team and external stakeholders were facilitated through Slack, where discussions about progress, issues, and feedback were ongoing. Email was also used for formal updates and feedback collection, particularly for stakeholders who preferred written updates.

How the Process Worked:  
• Initial Requirements Gathering: Early in the project, we held meetings with stakeholders to understand their vision and goals for the platform. This ensured that the core features, like task management, notifications, and communication between seniors and volunteers, were prioritized from the start.  
• Feedback Loops: After each development sprint, new features or updates were presented to users and stakeholders for feedback. These sessions were usually conducted over Zoom, with users testing the features in real-time while the development team observed and noted any areas for improvement.  
• Adjustments and Iterations: Based on the feedback collected from users, adjustments were made to the platform. This iterative process of building, testing, and refining helped ensure the final product was aligned with user expectations.  
• Continuous Engagement: Throughout the development, we made sure to keep users informed about the progress and upcoming changes. This helped foster a sense of involvement, ensuring that both seniors and volunteers felt heard and valued.

In summary, consistent user engagement during the development process helped us create a platform that was tailored to the needs of the end users.

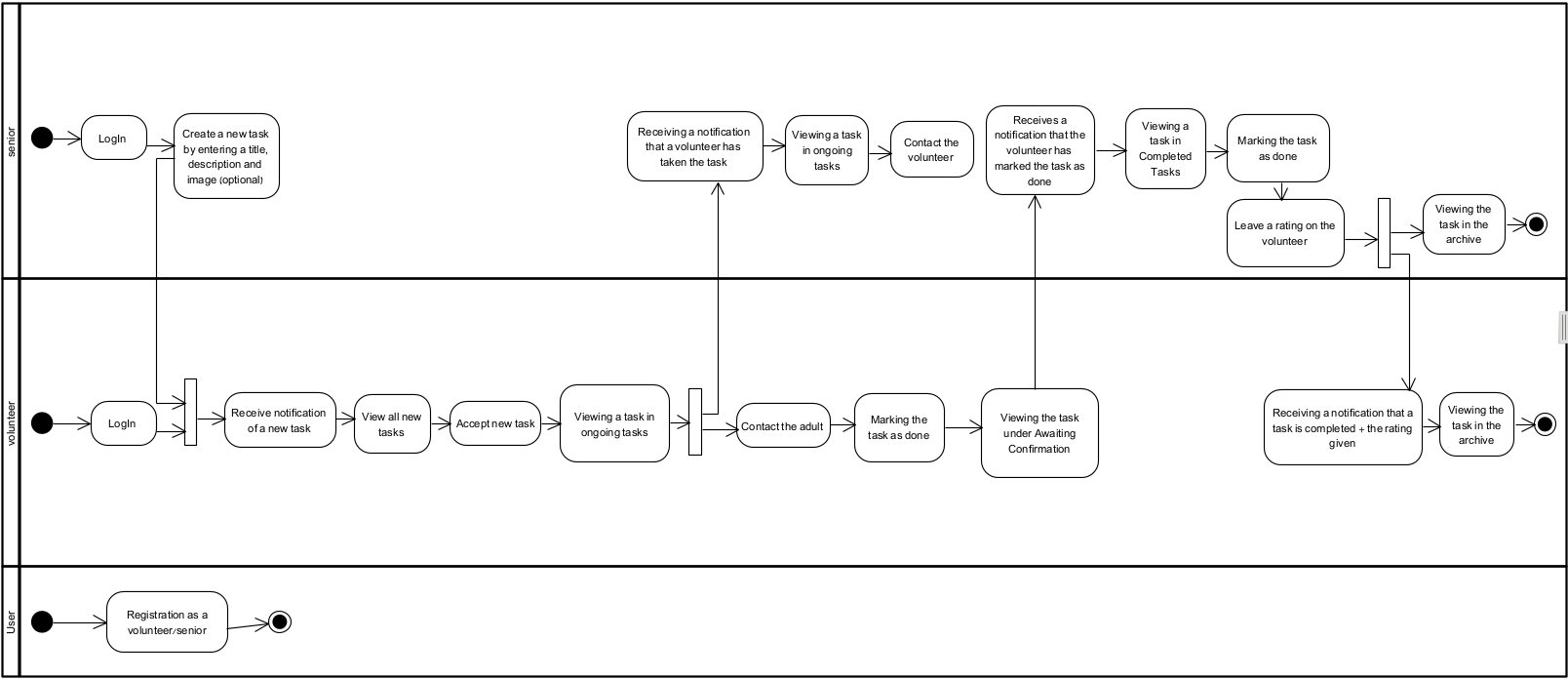
# **Solution Description**

## **The ElderEase system**

The Elderease system is structured to serve two distinct user groups: seniors and volunteers, each with its own tailored experience within the platform.  
  
 Seniors:  
Seniors have access to a simple and intuitive interface where they can create profiles, detailing their daily needs, preferences, and any specific assistance required. They can submit requests for help with tasks such as shopping, transportation, or companionship, which are then posted for volunteers to view. Seniors also have access to the communication tools within the app, including a built-in chat and the ability to reach out to volunteers by phone. Additionally, they can track the progress of their requested tasks and view the history of completed tasks, ensuring they stay informed and organized. The system also includes a rating feature, allowing seniors to rate the volunteers after each task is completed, fostering trust and accountability.  
Senior User Capabilities:  
• Create and manage a personal profile, detailing daily needs and preferences.  
• Submit requests for help with tasks such as shopping, transportation, or companionship.  
• Communicate with volunteers via in-app chat or phone.  
• Track ongoing and completed tasks.  
• View the history of past tasks.  
• Rate and provide feedback for volunteers after task completion.  
  
Volunteers:  
Volunteers, on the other hand, can create their own profiles, outlining their availability, skills, and areas where they are willing to offer support. They can browse through the task requests posted by seniors and choose the ones that fit their schedule and skillset. The app provides volunteers with real-time notifications about new tasks and updates on the ones they have accepted. Volunteers can communicate with seniors directly through the in-app chat or by phone, ensuring smooth coordination. After completing tasks, volunteers can receive feedback and ratings from seniors, which helps build a trustworthy reputation within the community.  
Volunteer User Capabilities:  
• Create a personal profile, specifying availability and skills.  
• Browse task requests posted by seniors.  
• Receive real-time notifications about new task requests and updates on accepted tasks.  
• Communicate with seniors via in-app chat or phone for smooth coordination.  
• Complete tasks and receive feedback and ratings from seniors.  
  
The Elderease project is built on a dual-user structure, catering to both seniors and volunteers, with each having specific roles and functionalities within the system. Seniors are empowered to request assistance with daily tasks, communicate with volunteers, and manage their task history, all through an easy-to-use interface. Volunteers, on the other hand, have the flexibility to choose tasks that suit their schedule and skillset, while maintaining communication with seniors and receiving real-time updates. Both user types benefit from a reliable rating system that fosters trust and accountability. This well-defined structure ensures a seamless and efficient interaction between seniors and volunteers, creating a supportive community that improves the quality of life for seniors and offers meaningful volunteering opportunities.

## **Diagrams Describing the Project**

A diagram of a computer program

Description automatically generated with medium confidenceActivity Diagram:Package Diagram:

# ‪Your first steps with Express.js - DEV Community‬‏A Comprehensive Guide to React for Complete Beginners | by Eric Flynn | MediumJavascript (JS)" Icon - Download for free – Iconduck**Technologies used**

## **Client-Side (Front-End)**:

Languages:  
• JavaScript - JavaScript is the primary language used in the React-based front-end for building dynamic and interactive user interfaces. It is used to handle component logic, manage state, and handle events.  
• HTML - HTML is used within React components (in JSX format) to define the structure of the user interface.  
• CSS - CSS is used to style the components, ensuring the user interface is visually appealing and responsive.  
Technologies:  
• React.js - React is a JavaScript library used for building user interfaces, particularly single-page applications (SPAs). It allows for the creation of reusable components that manage their own state.  
Why Chosen: React was chosen because of its flexibility, modular architecture, and ability to create dynamic, responsive UI elements.  
• React Router - A library used to handle routing within the React app, allowing users to navigate between different pages without reloading the entire app.  
Why Chosen: React Router provides seamless page transitions, maintaining the app's SPA architecture.

## **Server-Side (Back-End)**

Languages:• JavaScript (Node.js) - JavaScript is also used on the server-side within the Node.js runtime, allowing developers to use the same language on both the client and server.Technologies:• Node.js - Node.js is a JavaScript runtime environment built on Chrome’s V8 engine. It allows JavaScript to run server-side, enabling the development of scalable network applications.   
• Express.js - Express is a minimalist framework built on top of Node.js for handling the server’s routing and middleware. It defines API endpoints and manages requests to the server.

**•** Sequelize (ORM) - Sequelize is an Object-Relational Mapping (ORM) tool that provides a way to interact with the database using JavaScript objects instead of raw SQL queries. It manages database models, relationships, and migrations.

## **Database**:

Languages:  
• SQL - Importance: SQL is used by PostgreSQL to interact with and manage the data. It is the language used for querying the database, inserting data, and managing relationships between tables.  
Technologies:  
• PostgreSQL (Hosted on Render) - PostgreSQL is the relational database management system (RDBMS) used in the project to store all the core data, including user profiles, tasks, messages, and notifications. It is known for its scalability, performance, and compliance with SQL standards.  
Hosted on: The PostgreSQL database is hosted on Render, where it securely stores all of the application’s data.

## **Deployment:**

Technologies:  
• Netlify (for Front-End Deployment) : Netlify is used to host and deploy the front-end React application. It automates builds and deploys from the GitHub repository, ensuring the latest version of the code is always live.  
• Render (for Back-End Deployment and Database Hosting) - Render is the platform used to deploy both the back-end (Node.js and Express.js) and the PostgreSQL database. Render automates scaling, monitoring, and security management, providing an all-in-one solution for hosting the back-end and database.

# **Challenges Faced and Solutions Found**

During the development of the Elderease platform, we encountered a range of technical and analytical challenges.  
• Analytical Problems: One of the main challenges was ensuring the platform was user-friendly for seniors, many of whom might not be tech-savvy. To solve this, we conducted user testing with potential users from the target demographic. Based on their feedback, we simplified the interface, reduced the number of actions required to complete tasks, and added features like larger buttons and clear navigation instructions.  
• Data Structures and Algorithms: Another challenge was managing the dynamic nature of task assignments. We used efficient data structures like hash maps to store user tasks and their statuses, allowing quick retrieval and updates. For efficient matching between seniors and volunteers, we implemented an algorithm that prioritized task urgency and volunteer availability, ensuring optimal task assignment while maintaining a balanced workload for volunteer.  
• Technology Selection: During the early stages of the project, another significant challenge was choosing the right technologies to support both the scalability and ease of development. While initially, we considered using more basic or lightweight technologies, we realized the need for a robust system that could support real-time communication, reliable API calls, and efficient database management. After evaluating different frameworks and libraries, we opted for React for the front-end due to its flexibility and component-based structure, and Node.js/Express.js for the back-end to handle API requests efficiently. The decision to use Sequelize as the ORM (Object-Relational Mapping) to manage the database simplified the interaction with PostgreSQL, ensuring secure and efficient data handling.  
• Deployment Issues: One major challenge that we didn't fully anticipate during the initial planning phase was the deployment process. Initially, we focused on the development of the application without considering the complexities of deploying a full-stack system. As we neared the project's completion, we realized we needed to find suitable deployment solutions for both the front-end and back-end. To address this, we chose Netlify for the front-end deployment (React), which allowed for continuous deployment and easy management of static files. For the back-end, we opted for Render, which provided auto-scaling, database hosting (PostgreSQL), and seamless integration with our Node.js application. This solution allowed us to deploy the application without requiring significant changes to the codebase while ensuring that both the client-side and server-side could be scaled based on user demand.

In retrospect, incorporating deployment planning earlier in the project would have prevented some of the later hurdles we faced, but the final solution was effective and allowed the application to run smoothly in a live environment.

# **Results and Conclusions**

Overall, we successfully met most of the key objectives set at the beginning of the Elderease project. The main goal of creating a platform that connects seniors with volunteers to assist with daily tasks was fully achieved, and we delivered an application that is both functional and user-friendly.

## **Achievements:**

• User-Friendly Interface: One of the top priorities of the project was to ensure that the interface was intuitive and accessible, particularly for elderly users who may not be familiar with complex technology. We received positive feedback from test users, many of whom found the interface simple to navigate with clear instructions and large, easy-to-read buttons. This helped seniors easily create tasks and communicate with volunteers without encountering difficulties. The minimal steps required to perform actions, such as posting a task or viewing task updates, made the platform highly effective for its target audience.  
• Efficient Task Management: The platform's task management system worked smoothly, allowing seniors to create tasks and volunteers to browse and accept them with ease. The notifications system kept both seniors and volunteers updated about task progress, while the chat feature enabled real-time communication. This was crucial in ensuring that tasks were well-coordinated and completed on time.  
• Volunteer-Friendly Design: The platform was designed not just with seniors in mind, but also volunteers. The volunteer dashboard provided a clear overview of tasks, with filters allowing them to choose tasks that best fit their schedules and skills. The platform also helped volunteers build a sense of community by giving them meaningful ways to contribute, enhancing the overall success of the system.

• Scalable and Reliable Infrastructure: We succeeded in creating a scalable and reliable infrastructure using technologies like React, Node.js, and PostgreSQL. The platform was able to handle concurrent users and ensure smooth performance, thanks to optimizations such as load balancing and efficient data handling via Sequelize ORM.  
• Learning New Technologies and Platforms: One of the significant achievements of this project was the opportunity to learn and master the technologies and platforms we used from scratch. Throughout the development process, we became proficient in React for the front-end, Node.js and Express.js for the back end, and Sequelize ORM for managing interactions with PostgreSQL. We also gained valuable experience deploying the application using Netlify and Render, which allowed us to understand the full process of developing, deploying, and maintaining a scalable web application. This hands-on experience has significantly expanded our technical skills and knowledge, positioning us well for future development tasks.

## **Challenges and Unmet Goals:**

Although we accomplished most of our objectives, there were several areas where we didn't fully achieve what we had initially planned: **•** Admin Side: One key goal we were unable to implement in this phase was the admin side of the platform. This feature was intended to give administrators control over the system, allowing them to manage user accounts, oversee task assignments, and monitor overall activity. Due to time constraints, we had to prioritize the core functionality of the app for seniors and volunteers, and the admin functionality was postponed for future development. **•** Volunteer Training Platform: We also envisioned creating a built-in volunteer training platform to ensure that users received proper guidance before becoming volunteers. This platform was meant to offer tutorials or onboarding sessions to help volunteers understand their roles and responsibilities in assisting seniors. While we were able to create a seamless experience for volunteers once they signed up, the formal training aspect remains an important feature to develop in future iterations. **•** Reward System for Volunteers: Another feature we aimed to implement was a reward system where volunteers could receive recognition or benefits for their assistance. This system was designed to encourage volunteer participation by offering tangible rewards or recognition for their efforts. Unfortunately, this feature was not completed in the current phase, but it remains a key component for future development to boost volunteer engagement.

## **Decision-Making and Adaptation:**

Throughout the project, we made key decisions to prioritize certain features over others based on the project's timeline and resource constraints.

In conclusion, while we met most of our core goals, such as creating a user-friendly platform and building a robust infrastructure, there are several important features—like the admin side, volunteer training, and reward system—that we plan to implement in future phases of the project. Our decisions were primarily driven by time constraints and the need to prioritize essential features, but the platform is in a strong position for continued development and improvement.

**Lessons Learned**

Looking back at the project, while our workflow was efficient, there are a few things we would adjust for future development:

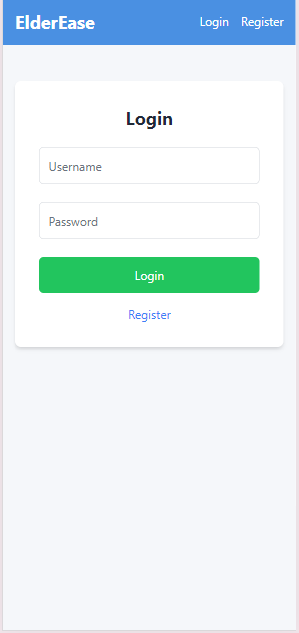
**What Worked Well:**

• Agile Methodology: The Agile approach and frequent testing cycles allowed us to stay on track and implement user-driven features. Regular feedback from real users helped us adjust the interface and add necessary features like task filtering, clearer navigation, and better notification systems. This iterative approach was instrumental in ensuring that the platform remained aligned with user needs throughout the development process.  
• Scalable Infrastructure: Another achievement was building a scalable and reliable infrastructure using technologies like React, Node.js, and PostgreSQL. The platform handled multiple concurrent users with ease, and optimizations like load balancing ensured that the system could handle increased traffic efficiently.  
• Learning New Technologies: The project gave us an excellent opportunity to learn new technologies and platforms from scratch. We successfully mastered React for front-end development, Node.js and Express.js for back-end logic, and Sequelize ORM for database management. The experience with deployment tools like Netlify and Render also added to our skillset, helping us gain confidence in deploying full-stack applications.

**What Could Be Improved:**

• System Architecture Planning: In hindsight, we would have dedicated more time to the early planning stages, particularly in terms of system architecture. Some components, such as the chat system, required significant rewrites due to scalability issues we did not foresee initially. Conducting better foresight and thorough load testing early on could have mitigated these challenges and saved development time.  
• Deployment Planning: One of the key lessons we learned was the importance of considering deployment from the very beginning of the project. We didn't anticipate the complexity of deploying a full-stack application until later in the project, which led to some last-minute adjustments in how we structured the code and managed the environment. If we had integrated deployment planning into the early stages of development, we could have streamlined the process and avoided some inefficiencies. For future projects, we plan to incorporate deployment strategies into the initial system design to ensure a smoother transition from development to production.  
• Admin Features: We initially intended to implement an admin panel that would allow administrators to manage users and tasks, but we didn't have time to complete this in the current phase of the project. This feature would have provided important oversight for platform management and is something we plan to include in future development.  
• Volunteer Rewards and Training: Another area we did not have time to implement was a reward system for volunteers and a training platform for new volunteers. Both features were designed to enhance volunteer engagement and ensure that volunteers were properly onboarded and motivated. While the core functionality of matching volunteers with tasks was successful, these additional features would have added further value to the platform.  
  
By recognizing these areas of improvement, we are better prepared for future iterations and projects, and we will apply these lessons to make future developments smoother and more efficient.

# **User Guide**

Welcome to the Elderease User Guide. This guide is designed to help you navigate the Elderease platform with ease, whether you're a senior seeking assistance with daily tasks or a volunteer looking to offer support. In this guide, you will find step-by-step instructions on how to create an account, manage tasks, communicate with others, and utilize all the key features of the app. Our goal is to make your experience as smooth and intuitive as possible, ensuring that you can quickly access the help you need.

## **Both Elder’s & Volunteer side:**

### **Login Screen:**

This is the Login Screen for the Elderease app. features:  
1. Login Form:  
• Username Field: Users, whether seniors or volunteers, are prompted to enter their unique username, created during registration.  
• Password Field: password to access the app.  
• Login Button: After entering the correct username and password, users can press the green "Login" button to access their personal dashboard, where seniors can manage tasks and volunteers can view and accept tasks.  
2. Register Option: If the user is new to the platform, they can click on the Register link below the login button to create a new account.  
The login screen is designed to be simple and user-friendly, ensuring both seniors and volunteers can easily log in or register. The use of clear fields and buttons ensures accessibility for all users.

## **Elder’s side:**

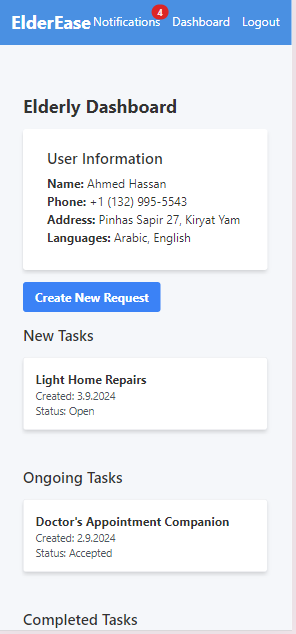
### **Registration Screen for seniors:**

This is the Registration Screen for seniors (Elderly) in the Elderease app Here's an overview of the key features:  
1. Username and Password Fields: The user enters their unique username and password, which will be used for future logins.  
2. Role Selection: The user can choose between the Elderly or Volunteer role by selecting the appropriate button. In this case, the "Elderly" role is selected, indicating the user is registering as a senior.  
3. Personal Details:  
• First Name and Last Name: The user fills in their personal information, including first and last name.  
• Phone Number: The user provides their phone number for contact purposes.  
• Address: The user's address is entered, allowing volunteers to know the location where assistance is needed.  
4. Additional Languages: The user can select additional languages they are comfortable with, ensuring that communication is easy and accessible. Available language options include English, Arabic, French, and Russian.  
5. Register Button: After filling in all the details, the user can click the green "Register" button to complete the sign-up process.

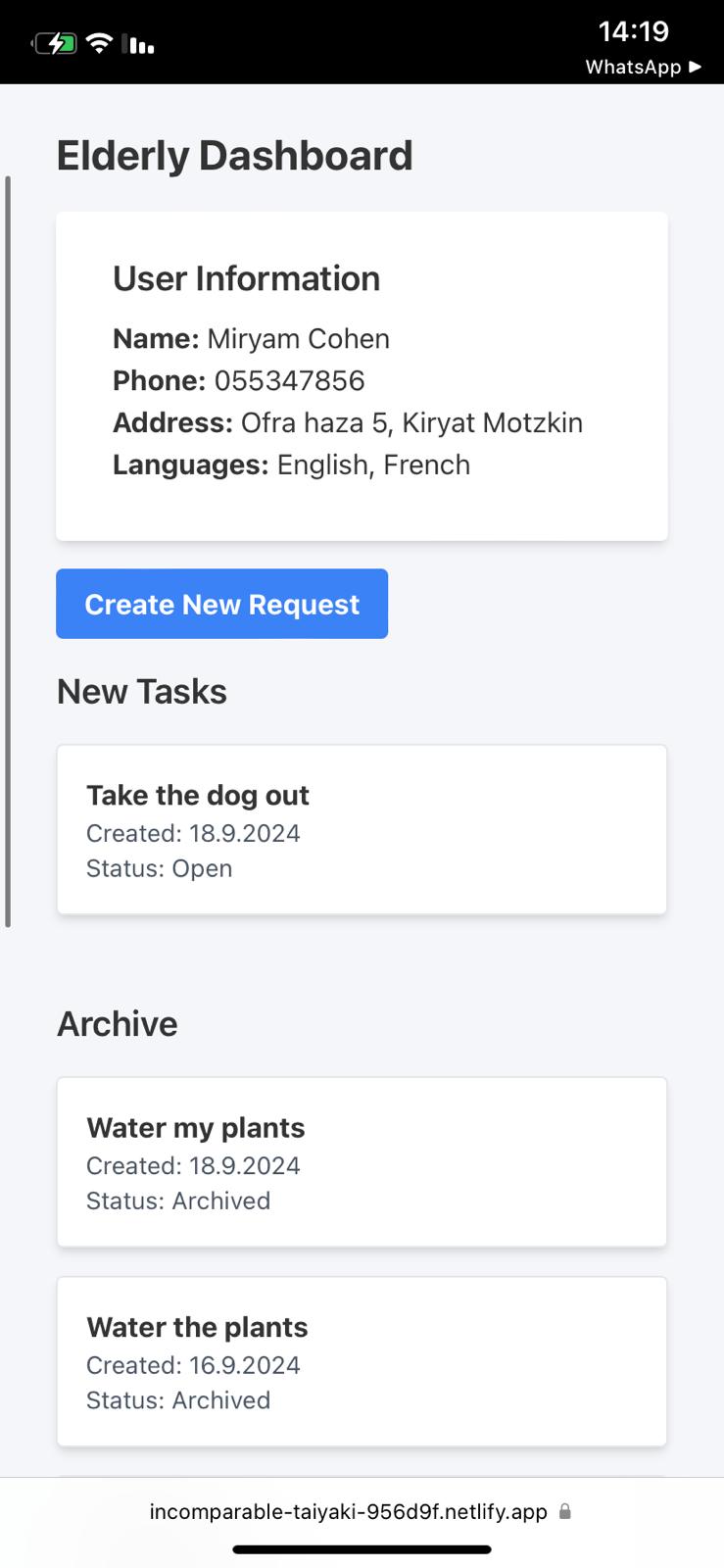
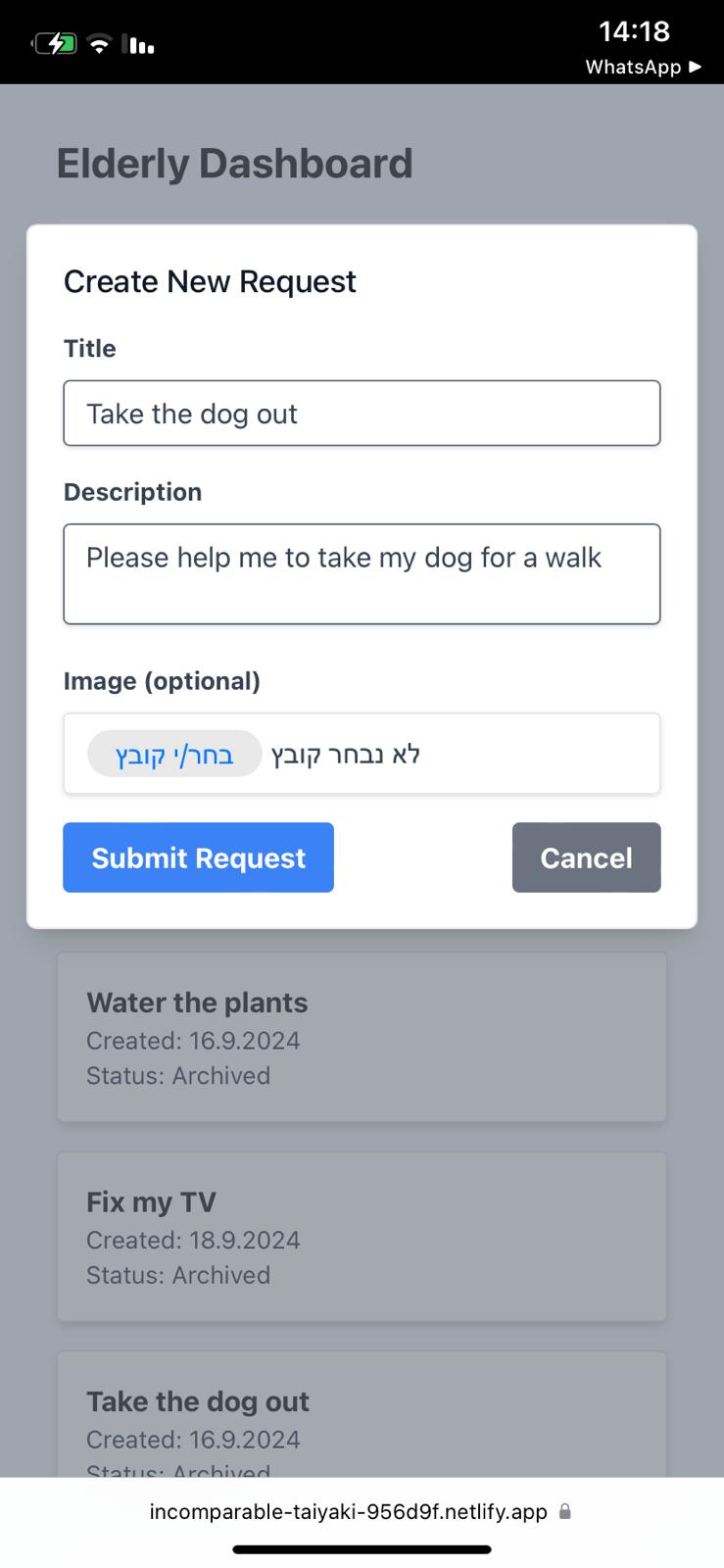
### Uploaded image**Elderly Dashboard (on initial entry):**

After completing the registration process, this is the Elderly Dashboard screen that the senior user will see:  
• User Information: Displays the user's name, phone number, address, and selected languages.  
• Create New Request Button: The senior can create a new request for assistance by clicking the "Create New Request" button.  
• Navigation Bar: At the top, the user can access Notifications, return to the Dashboard, or Logout.

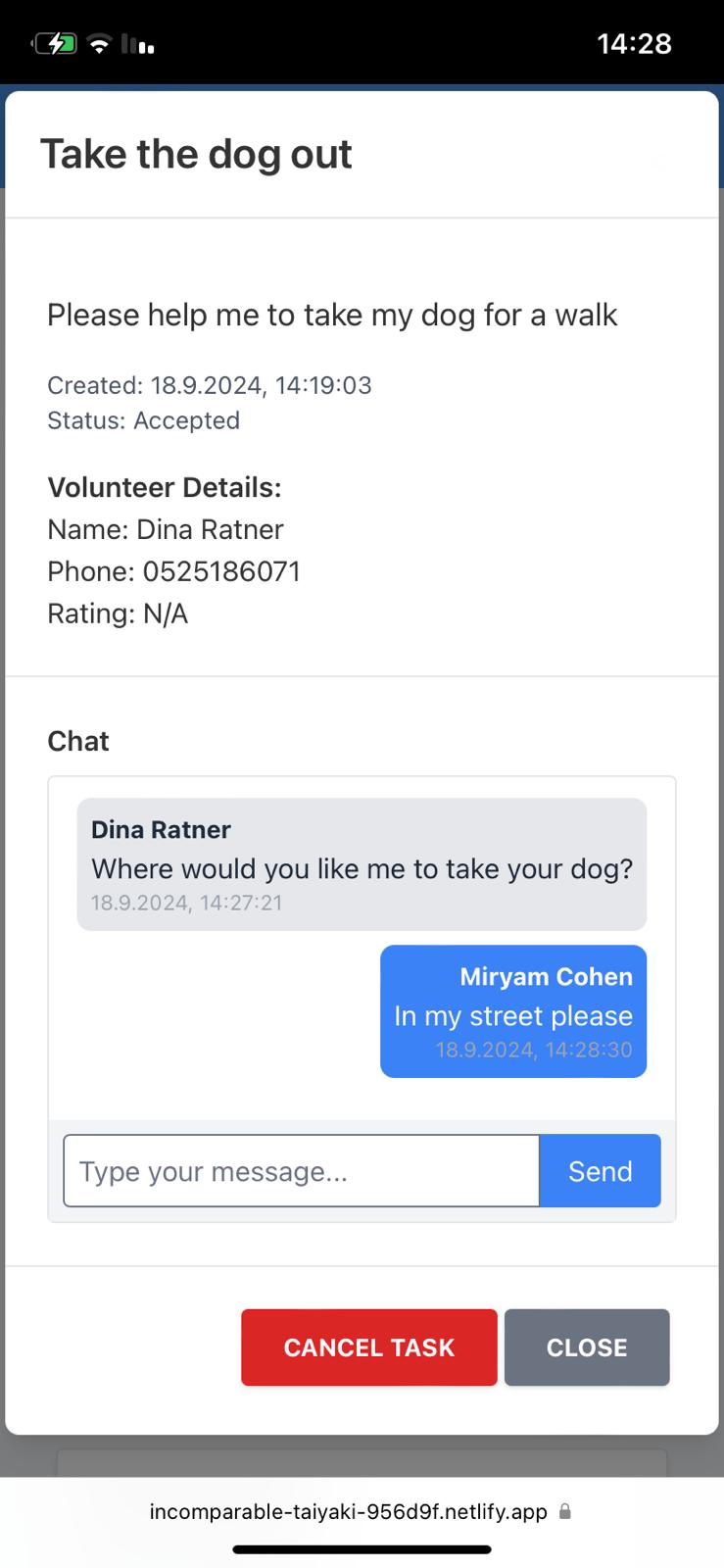
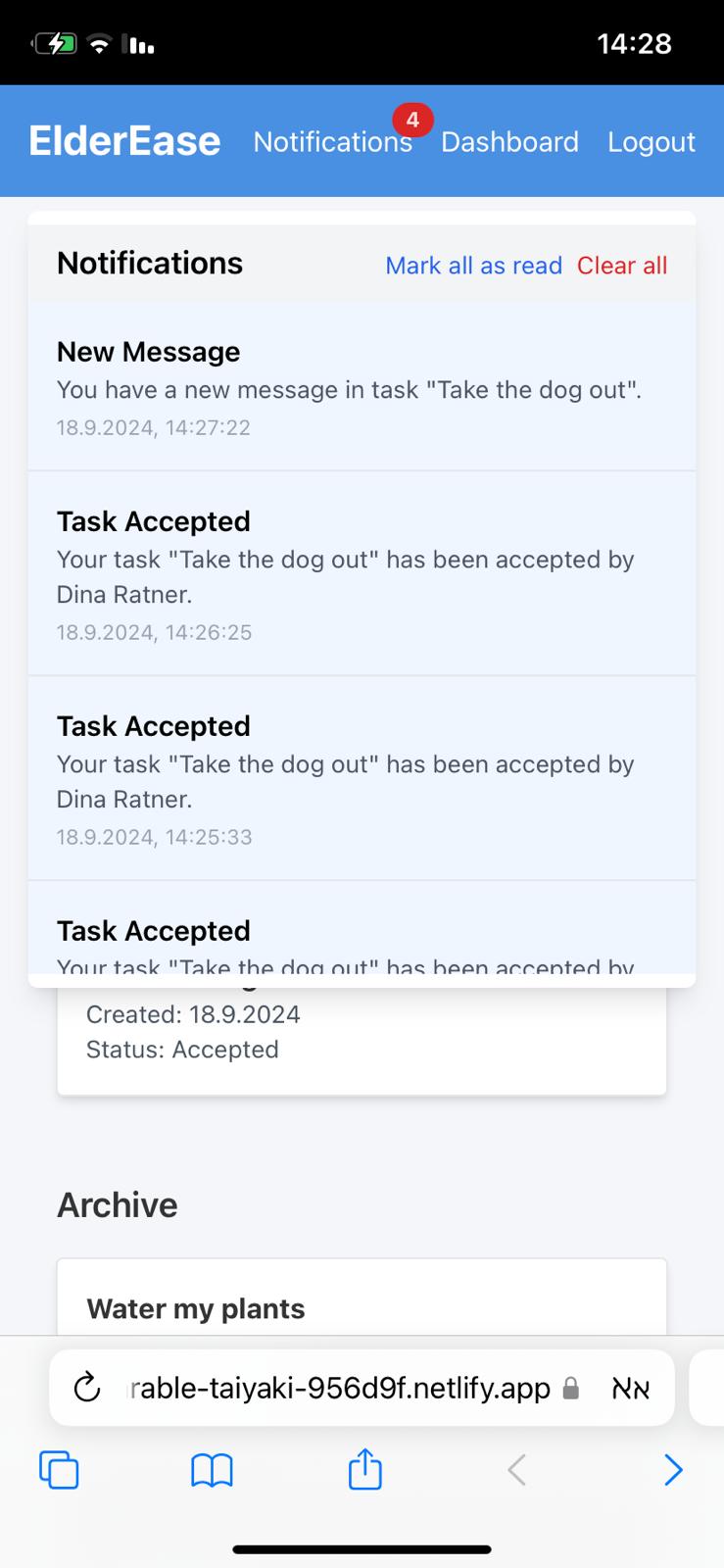
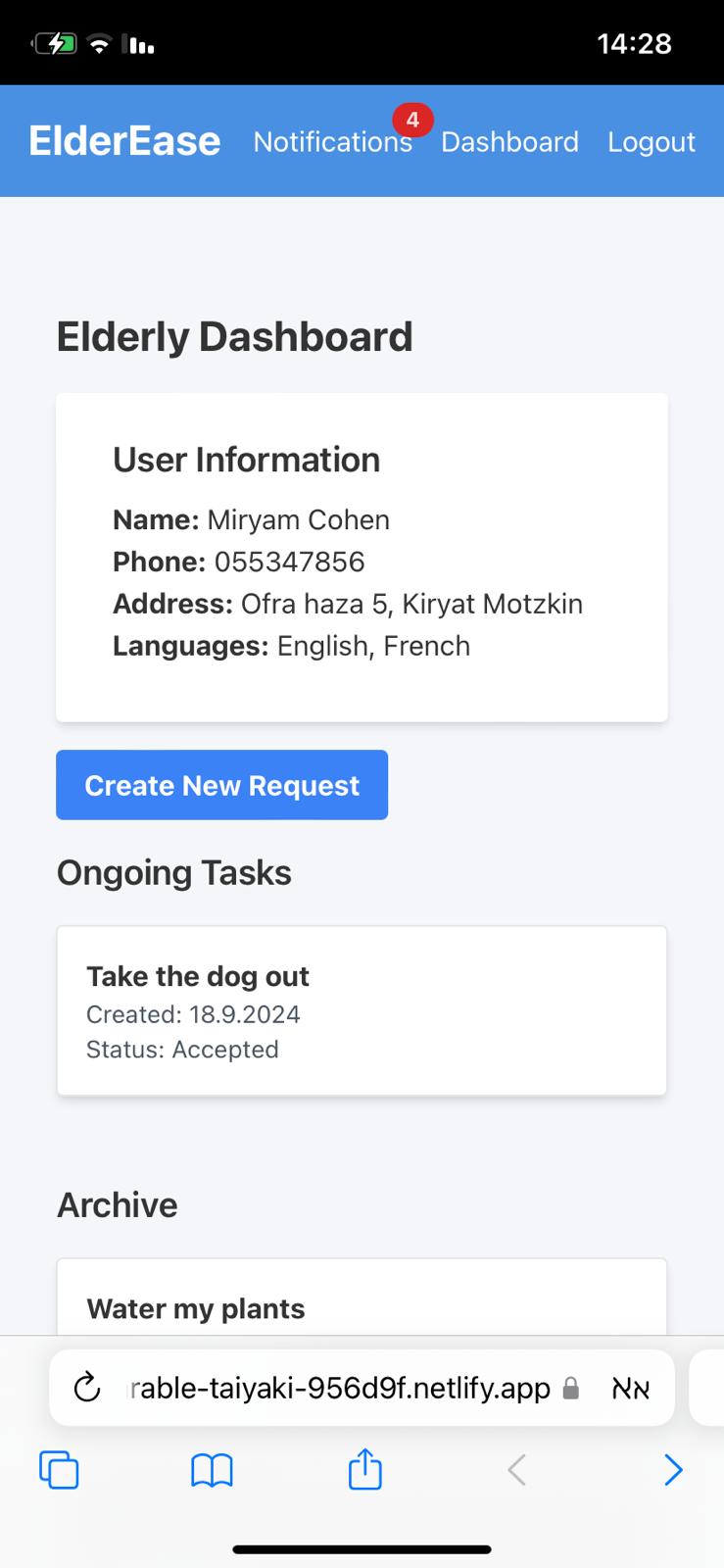
A screenshot of a computer

Description automatically generated**Elderly Dashboard:**

In these screens, a senior user can:  
1. View Personal Information: See their name, phone, address, and languages.  
2. Create New Requests: Easily request help for tasks using the "Create New Request" button.  
3. Track Tasks: View and manage tasks in different statuses:  
• New Tasks: Tasks waiting for volunteers.  
• Ongoing Tasks: Tasks that volunteers have accepted and are in progress.  
• Completed Tasks: These are tasks that the volunteer has marked as finished. They are now awaiting confirmation from the senior to verify that the task has been fully completed to their satisfaction.  
• Archived Tasks: Tasks moved to the archive.  
4. Check Notifications: Receive and view task updates and alerts via the notification system at the top.

**Create New Request:**

This screen shows the Create New Request form on the Elderly Dashboard in Elderease. It allows the senior user to subm it a new task request for assistance. Here are the key elements:  
1. Title Field: The senior enters a short title for the task (e.g., "Take the dog out").  
2. Description Field: A more detailed explanation of the request (e.g., "Please help me take my dog for a walk").  
3. Image Upload (Optional): The user can upload an image related to the task, though it is not required.  
4. Submit Request Button: After filling out the form, the senior clicks this button to post the request for volunteers to view and accept.When finished, the elder will be able to see his help request among the list of new tasks.

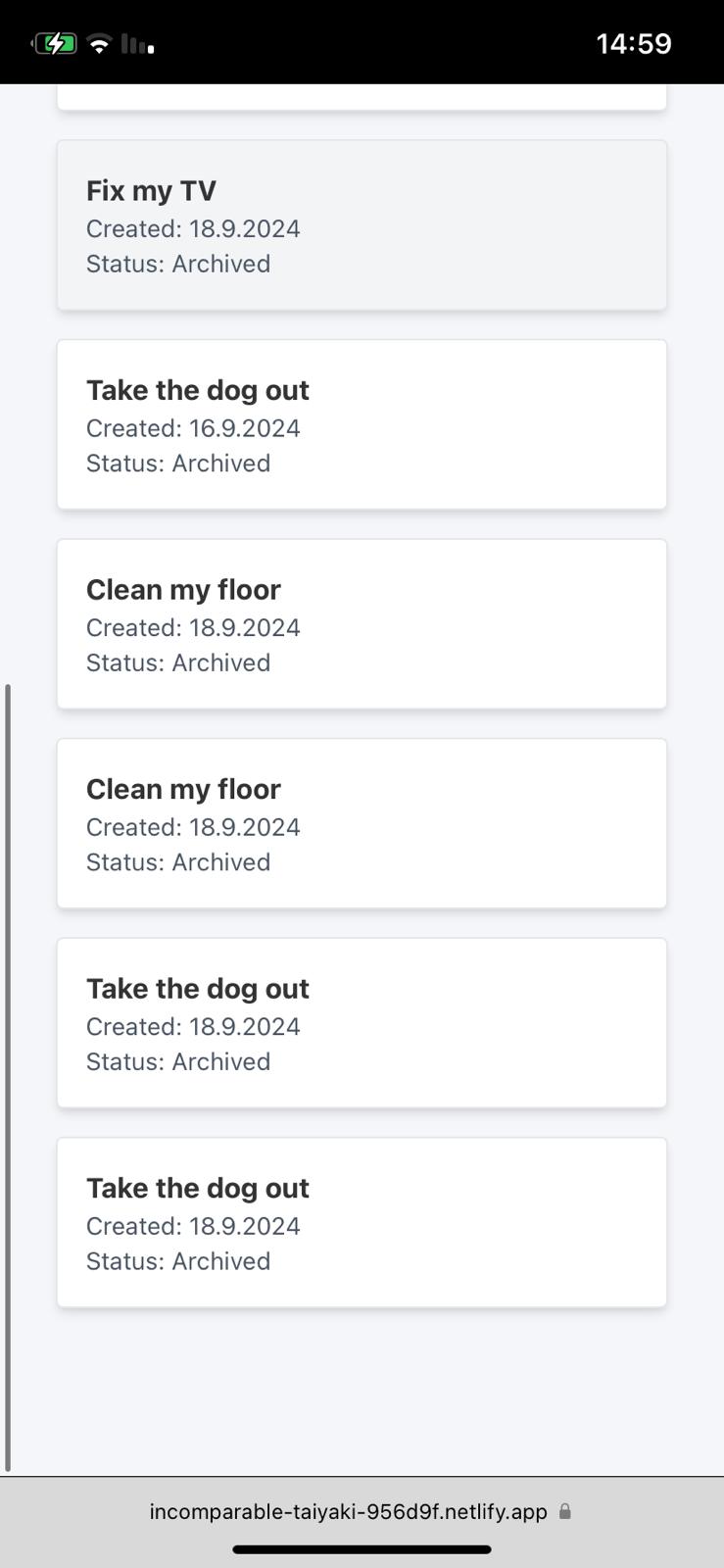
**Ongoing Tasks**:

Once a volunteer accepts a task, the elderly person receives a notification informing them that the task has been accepted. Additionally, the elderly person can communicate with the volunteer through the built-in chat system for coordination. The task then moves from "New Tasks" to "Ongoing Tasks," and its status changes to "Accepted," indicating that the task is currently in progress and actively being handled by the volunteer.

### **Completed Tasks**:

Once a volunteer marks a task as completed, it moves to the **Completed Tasks** section in the senior's dashboard. After that, the senior will need to confirm that the task was fully completed. In the task details view, the senior can:  
• **Chat with the volunteer**: Exchange messages with the volunteer before confirming the completion.  
• **Confirm Completion**: By clicking the **Confirm Completion** button, the senior officially approves that the task is done.  
• **Rate the Volunteer**: The senior has the option to rate the volunteer's performance using a star rating system.  
• **Submit Rating**: Once the senior provides a rating, they can submit it to finalize the task.

### **Archived Tasks:**



Once the elderly person confirms that the task has been completed, it moves from the "Completed Tasks" section to the "Archive." This indicates that the task is fully closed, allowing the elderly person to keep a record of their past tasks in the archive for future reference.

## **Volunteer's side**

### **Registration Screen for volunteers:**

This is the volunteer registration page for individuals who do not yet have a profile. Here, new volunteers can create an account by providing their personal details, language preferences, and transportation options. Once registered, volunteers will be able to take on tasks and assist others in their community.

• Enter Login Details:  
In the top fields, enter your Username, Password, and confirm your Password.  
• Select Volunteer Role:  
Choose the "Volunteer" option by clicking on the corresponding button.  
• Enter Personal Information:  
Fill in your First Name, Last Name, Phone Number, and Address.  
• Select Additional Languages:  
Choose any additional languages you speak (English, Arabic, French, Russian) by clicking on the respective buttons.  
• Enter Date of Birth:  
Input your Date of Birth in the provided field.  
• Select Transportation Method:  
Choose your preferred mode of transportation by selecting from the options: Car, Bike, Scooter, or Public Transportation.  
• Complete Registration:  
Once all information is filled out, click the green "Register" button to complete your registration.

### **Volunteer Dashboard (on initial entry):**

This is the volunteer dashboard, which the volunteer sees immediately after registering. It displays the volunteer's personal information, including their name, phone number, address, languages spoken, transportation method, and their average rating (if applicable). This dashboard serves as a central place where volunteers can quickly access and review their information.

### **Volunteer Dashboard:**

This is the volunteer dashboard for Gavriel Levi, a volunteer with personal information displayed, including name, phone number, address, languages spoken (English and Russian), transportation method, and average rating (currently N/A).

• "New Tasks" section, open tasks such as "Grocery Shopping Assistance," "Light Home Repairs," and "Take the dog out" are listed, all awaiting acceptance.   
• "Ongoing Tasks" section shows a task titled "Doctor's Appointment Companion," which has already been accepted and is in progress.  
• "Awaiting Confirmation" section, the task labeled "test" is completed and is waiting for confirmation.  
• "Archive" section contains tasks that Gavriel has completed in the past, such as "Pick up Medications," "Buy Medications," and "Take the dog out," all marked as archived for reference.

### **New Tasks:**

When an elderly person adds a new task, the volunteer immediately receives a notification about the available task. The task is then listed under the volunteer's "New Tasks" section. The volunteer has the option to view details about the task, including information about the elderly person and the task description. After reviewing the details, the volunteer can decide whether to accept the task or decline it.

### **Ongoing Tasks:**

Once the volunteer accepts the task, it moves to the "Ongoing Tasks" section. The volunteer can click on the task to view the elderly person's details, such as name, phone number, and address, and communicate with them via a built-in chat. At any point, the volunteer has the option to cancel the task by clicking the "Cancel Task" button. When the task is completed, the volunteer needs to update the status by clicking the "Complete Task" button to mark it as finished.

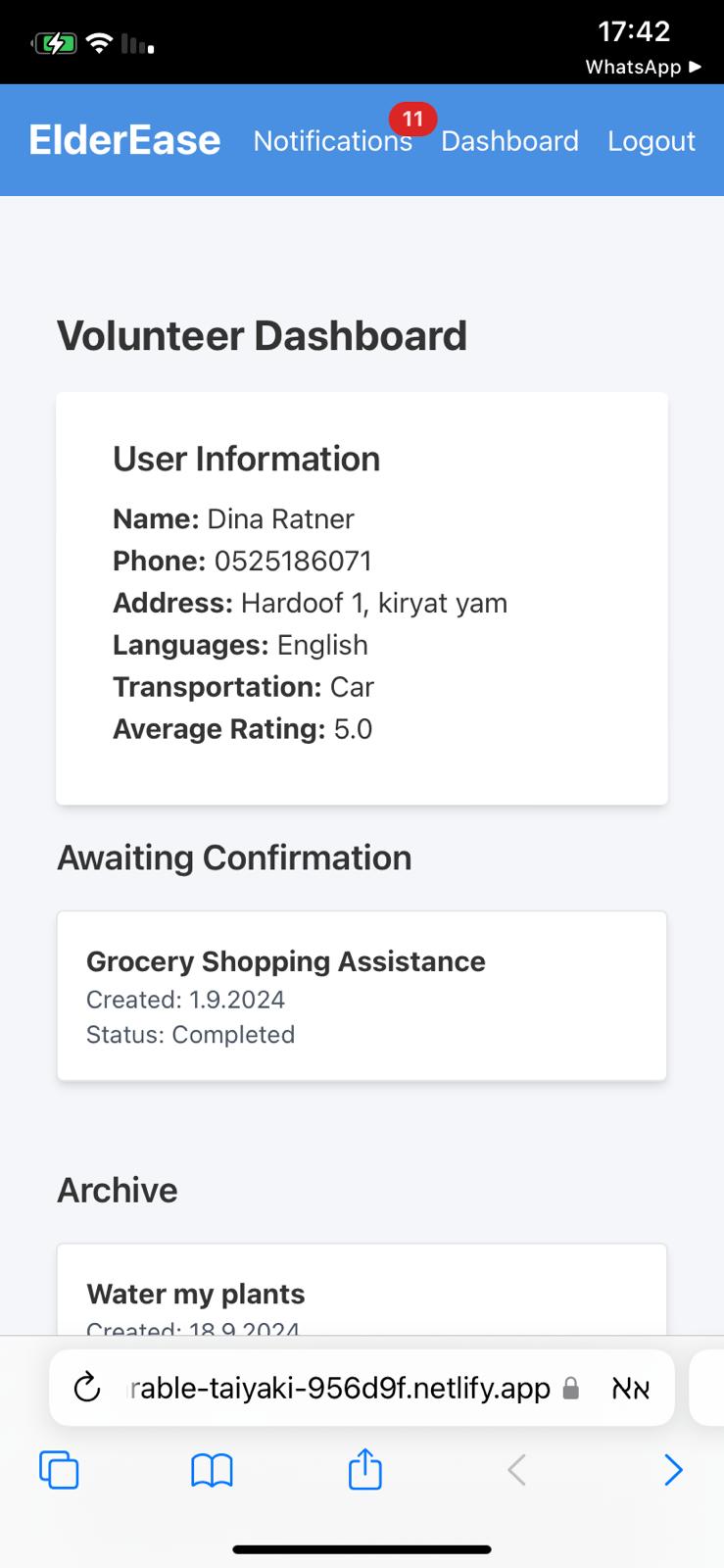
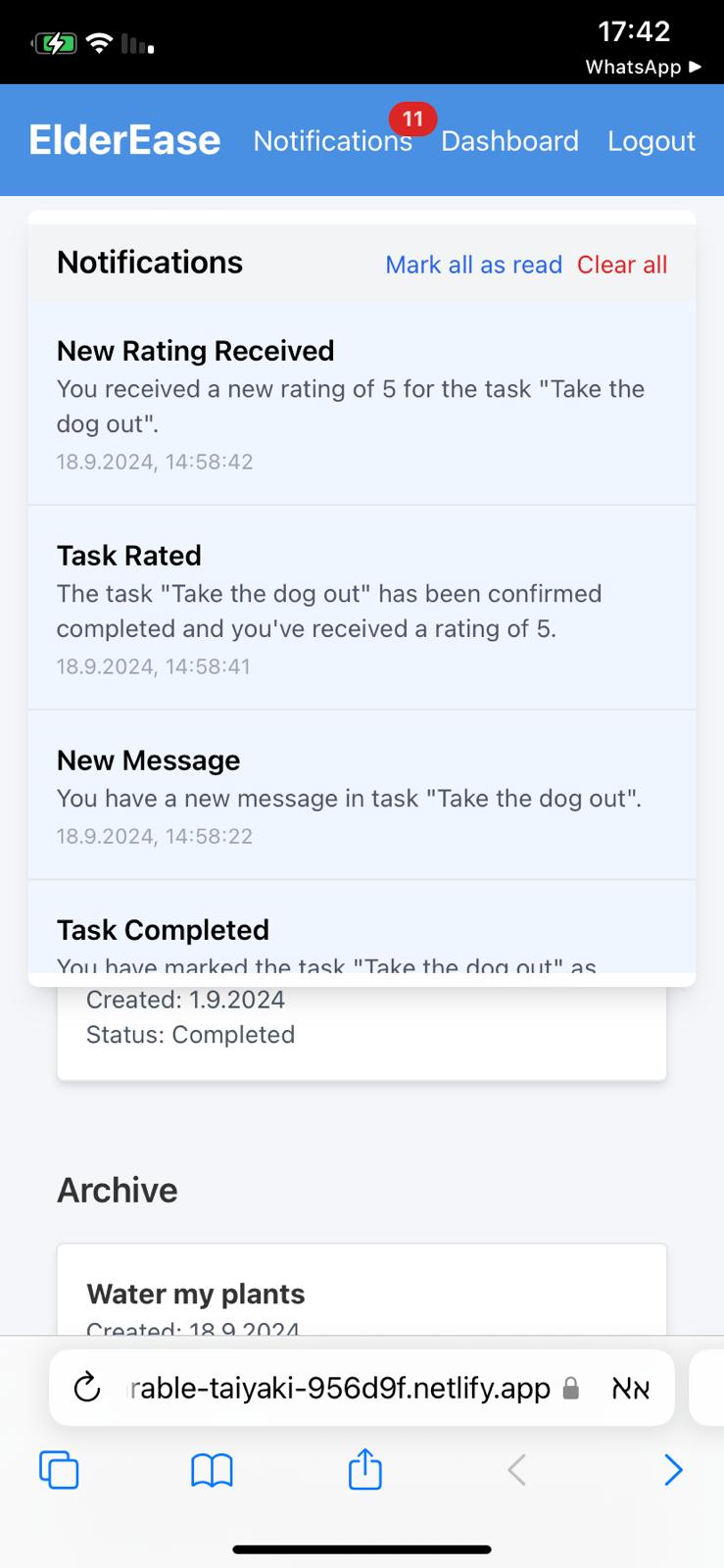
### **Awaiting Confirmation:**

Once the volunteer completes a task by clicking "Complete Task" in the "Ongoing Tasks" section, the task moves to the "Awaiting Confirmation" section. This indicates that the task is completed from the volunteer's side, but it is now waiting for confirmation from the elderly person to officially mark the task as finished.

### A screenshot of a phone Description automatically generated**Archive:**

Once the elderly person confirms that the task is completed and provides a rating for the volunteer, the volunteer receives a notification about the rating. At this point, the task is moved to the volunteer's "Archive" section, marking it as fully completed and stored for future reference.

## **Volunteer & Elder side**



Both the Volunteer Dashboard and the Notifications screen in the Elderease application share a consistent and accessible toolbar for both seniors and volunteers. This toolbar, located at the top of the screen, includes the following options:  
• Notifications: Users can view their notifications by clicking on the notifications icon. The number of unread notifications is displayed prominently, and users have the option to mark all notifications as read or clear all notifications using the buttons provided.  
• Dashboard: Users can quickly navigate back to their personal dashboard (whether it is the volunteer or elderly dashboard) by clicking on the "Dashboard" link.  
• Logout: Users have the option to securely log out of their account by selecting the "Logout" link.

The Notifications screen provides additional options for managing alerts, including:  
• Mark All as Read: This option allows users to mark all notifications as read in one click.  
• Clear All: This option enables users to clear all notifications from their list.  
This consistent design ensures ease of use and accessibility, allowing both seniors and volunteers to manage their tasks and notifications seamlessly.

# **Maintenance Guide**

## **Purpose**

This guide ensures continued usage, modification, and improvement of the system after its initial development, allowing for updates, changes, and longevity.

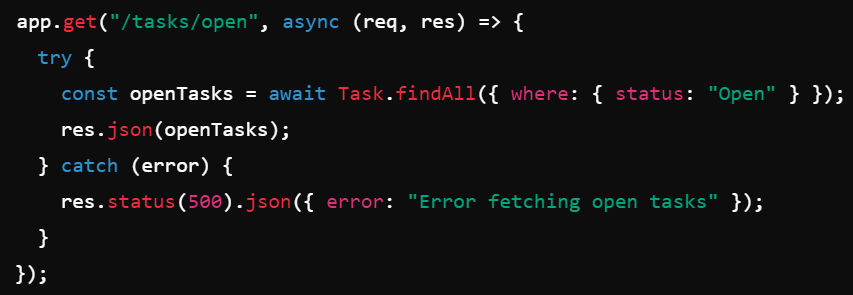
## **1. System Environment**

Software Requirement:  
• Node.js (v14.17 or higher): Required for backend API and server.  
• Express.js: Framework for backend development.  
• React.js: Frontend framework for building user interfaces.  
• PostgreSQL: Database used for storing data such as users, tasks, and messages.  
• Sequelize ORM: Tool for interacting with the PostgreSQL database.  
Hardware Requirements:  
• Server: Minimum 2GB RAM, 50GB storage.  
• Client: Modern web browser like Chrome, Firefox, or Safari.  
Installation Steps:  
• Backend Setup:  
- Clone repository, install dependencies, set up .env with database credentials and run: npm install  
npx sequelize-cli db:migrate  
npm start  
• Frontend Setup:  
- Navigate to the client directory, install dependencies, set up .env, and run:  
npm install  
npm start

## **2. Custom Software Components**

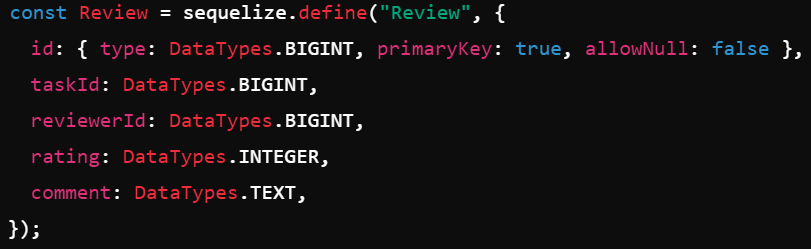
### **Backend - server.js:**

The server.js file is the core of the backend API for the application. It is built using Node.js with Express.js as the framework, allowing for the creation of RESTful API endpoints. The backend communicates with a PostgreSQL database using Sequelize ORM for database operations, such as creating, reading, updating, and deleting records. **Key Components of server.js:**1. Express.js Setup:  
• Express is used to define and handle routes for different API endpoints, manage middleware, and handle incoming HTTP requests and outgoing responses.  
• The server listens on a port for incoming requests and responds accordingly.  
2. Database Connection (Sequelize):  
• The server connects to a PostgreSQL database using Sequelize, which is an ORM that helps manage database models and perform queries.  
3. Models:  
• Sequelize models define the structure of the database tables. Each model corresponds to a table in the database, and you can use Sequelize methods (like create, findAll, etc.) to interact with these tables.  
4. Routes (API Endpoints):  
• The routes handle specific HTTP requests (GET, POST, PATCH, DELETE) that the frontend makes to the backend. Each route is connected to a function that performs certain operations like creating or retrieving data.  
5. Middleware:  
• Middleware is used to handle tasks like parsing request bodies, adding headers for security, handling CORS (Cross-Origin Resource Sharing), etc.  
6. Error Handling:  
• The server has global error handling to catch unhandled errors and provide a proper response to the frontend.  
7. Syncing with the Database:  
• Sequelize syncs the models with the database tables, ensuring that any changes to models reflect in the database. **How to Add Features to the Server:**You can add new functionality to the server by following these steps:1. Add a New API Route

To allow the frontend to request additional data or perform new actions, you can add new API routes.

Example: Adding a route to get tasks by status

2. Add a New Model  
If you need to store new types of data in the database, you can add a new Sequelize model. This model will represent a new table in the PostgreSQL database.



Example: Adding a Review model

3. Add Helper Functions  
Helper functions can be used to perform repetitive or specific tasks, such as sending notifications, logging, or interacting with external APIs.

  
  
  
  
  
  
This function sends a notification to a user when called within a route (e.g., when a task is created or updated).

Example: Sending a notification

An example from Server.js:

**Key Points in the Example:**

1. **Receiving Data**: Data is received from the frontend via req.body.
2. **Database Interaction**: A new task is created in the database using the Task.create() method.
3. **Notifications**: After the task is created, volunteers are notified using the createNotification helper function.
4. **Sending Response**: The newly created task is sent back to the frontend as a JSON response using res.json().

Summary:  
• Express.js handles the API routes, request parsing, and responses.  
• Sequelize manages database interactions, ensuring data is saved or retrieved from the PostgreSQL database.  
• New functionality can be added by defining new routes, models, or helper functions, depending on the feature you want to implement.

### **FrontEnd Components:**

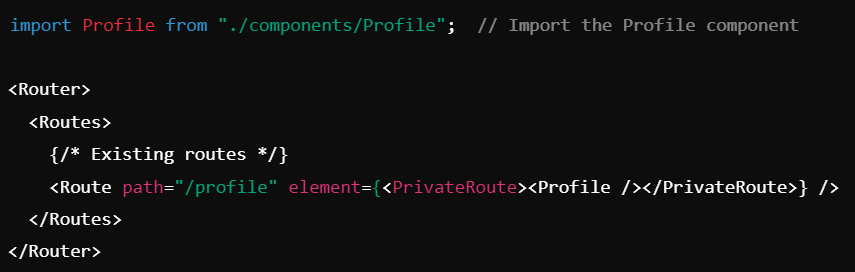
**1. App.js:**The main entry point of the React frontend that handles routing between different views (e.g., login, register, dashboards). Uses the React Router for navigation and state management for notifications and user data.Key Components of app.js:  
1. React and React Router:  
• React is used to manage the components of the frontend and handle the user interface.  
• React Router is used to manage navigation between different pages (routes) in the application. It enables users to navigate between login, registration, dashboards, etc., without reloading the page.

2.Authentication Context (AuthContext):  
• Authentication is managed globally using Context API with an AuthContext that stores information about the currently logged-in user and provides authentication methods like login, registration, and logout.  
• The app checks whether the user is logged in and controls access to certain routes based on that.  
3. Private and Public Routes:  
• PrivateRoute: Ensures that only authenticated users can access certain pages like the dashboard. If a user is not logged in, they are redirected to the login page.  
• PublicRoute: Ensures that unauthenticated users access pages like login and registration. If the user is already logged in, they are redirected to the dashboard.  
4. Global UI Components:  
• The app includes components that are globally available, such as:  
Notifications: Displays user notifications fetched from the backend.  
Alerts: Shows alert messages (e.g., success or error) in response to user actions like logging in or registering.  
5. State Management:  
• useState and useEffect are used for managing local component state and performing actions like fetching data from the backend or controlling notifications.

How to Add Features to the app.js File

You can extend the functionality of the app.js file by following these steps:

1. Add a New Route

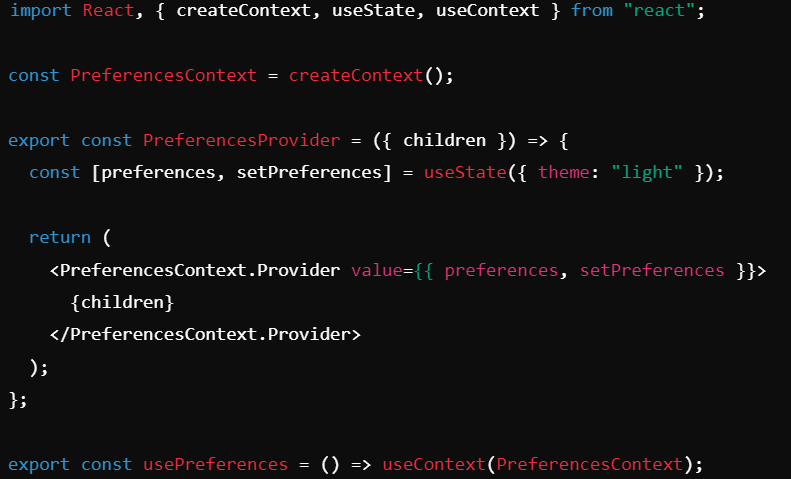
To add a new page or component, you can add a new route within the Router. For example, if you want to add a new profile page for users:

Example: Adding a new Profile Route

In this example, we create a new PrivateRoute for the /profile URL, which is only accessible if the user is logged in

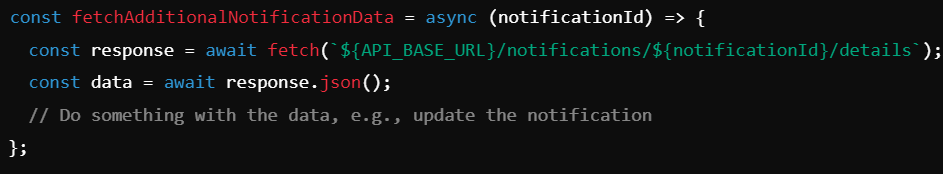
2. Add Global State or Context

If you need to manage additional global state (e.g., for user preferences or settings), you can add a new Context similar to the AuthContext. This ensures the state is accessible across different components.



Example: Adding a Preferences Context

3. Add New Global Functionality (e.g., Notification Logic)

You can add more global functionality like improved notifications or alert management.

Example: Fetching additional data in notifications

You can call this function after fetching notifications or attach it to a button to show more details for each notification.

An example from App.js:

**Key Points in the Example:**

1. **Routing**: Routes are defined for Login, Register, and Dashboard pages. Private routes ensure only authenticated users can access the dashboard.
2. **Notifications**: Notifications are fetched when the user is logged in using the fetchNotifications() function and are updated in the local state.
3. **Logout Handling**: The handleLogout() function manages user logout, with an alert shown for a successful logout.

function AppContent() {

const { user, logout, loading } = useAuth();

const [alert, setAlert] = useState(null);

const [showNotifications, setShowNotifications] = useState(false);

const [notifications, setNotifications] = useState([]);

const API\_BASE\_URL = process.env.REACT\_APP\_API\_URL;

// Function to fetch notifications from the server

const fetchNotifications = async () => {

if (!user) return;

try {

const response = await fetch(`${API\_BASE\_URL}/notifications?userId=${user.id}`);

const data = await response.json();

setNotifications(data);

} catch (error) {

console.error("Error fetching notifications:", error);

}

};

useEffect(() => {

if (user) fetchNotifications();

}, [user]);

// Handle user logout

const handleLogout = () => {

setAlert({ type: "success", message: "Logged out successfully." });

setTimeout(() => {

logout();

setAlert(null);

}, 2000);

};

{/\* Route Definitions \*/}

<Routes>

<Route path="/" element={<PublicRoute><Login /></PublicRoute>} />

<Route path="/register" element={<PublicRoute><Register /></PublicRoute>} />

<Route path="/dashboard" element={<PrivateRoute><Dashboard />} />

<Route path="\*" element={<Navigate to="/" replace />} />

</Routes>

);

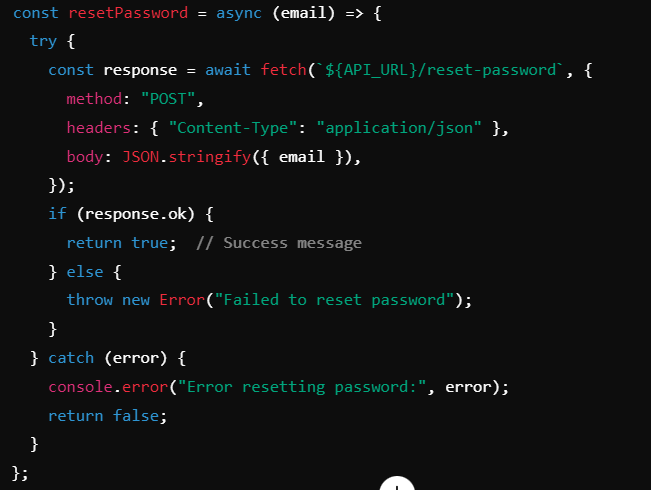
}

Summary:  
• React Router is used for routing between pages like login, registration, and the dashboard.  
• Authentication Context (AuthContext) manages global user state (logged-in user, login, logout).  
• You can easily add new routes for additional components, add new contexts to manage global state, or extend existing components like notifications and alerts.  
• The AppContent component handles the core layout of the app, managing both routing and global UI elements.

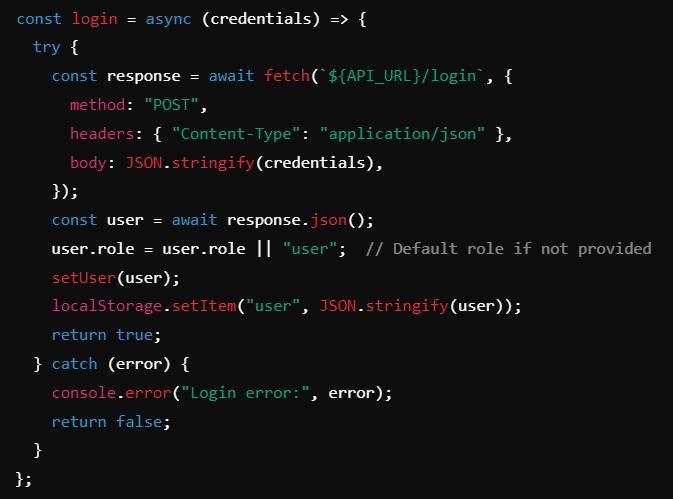
**2. AuthContext.js:**The AuthContext.js file is responsible for managing user authentication in the React frontend application. It uses React's Context API to provide authentication-related data and methods (such as login, logout, and register) to the entire application. This allows any component within the app to access the user's authentication state and interact with the backend for user authentication purposes.

Key Components of AuthContext.js:  
1. Context Creation:  
• A context is created using React.createContext(). This context will be used to share authentication state (like the currently logged-in user) and methods (like login, register, and logout) across the application.  
2. AuthProvider Component:  
• The AuthProvider component acts as a wrapper for the entire application. It contains the authentication logic (like login, register, and logout), and it provides this data and functionality to child components via the context.  
• State Management: The useState hook is used to manage the user's authentication state (whether a user is logged in or not) and the loading state to indicate if authentication is being processed.  
3. Authentication Functions:  
• login: Sends login credentials (username, password) to the backend and, if successful, sets the user state.  
• register: Registers a new user by sending their data to the backend, and sets the user state upon successful registration.  
• logout: Clears the user state and removes the user from local storage when they log out.  
4. Storing and Retrieving User Data:  
• The user authentication state is persisted across page reloads using localStorage. When the AuthProvider is initialized, it checks if there is a user stored in localStorage and restores it to the user state if found.  
5. Providing the Context:  
• The AuthProvider component wraps the entire app and provides the AuthContext.Provider with values like the user, login, register, and logout functions. This allows any child component to access these values by using the useAuth hook.

6.Custom Hook (useAuth):  
• The useAuth hook is a custom React hook that allows any component to easily access the authentication context without manually interacting with AuthContext.

Examples for features that can be added to the AuthContext.js File:  
You can extend the functionality of the AuthContext.js file by adding new authentication-related features, such as password reset, email verification, or adding roles for different types of users.

Example: Adding a resetPassword function



Example: Adding Roles to the User Object

Example from the AuthContext.js Code:

**Key Points in the Example:**

**State Management**: The user and loading states are managed using useState.

**Persistent User**: The user state is persisted using localStorage so that it remains intact across page reloads.

**Login and Register**: Functions login and register send requests to the backend and update the user state based on the response.

**Logout**: Clears the user state and removes the user from localStorage.

export const AuthProvider = ({ children }) => {

const [user, setUser] = useState(null);

const [loading, setLoading] = useState(true);

useEffect(() => {

const storedUser = localStorage.getItem("user");

if (storedUser) {

setUser(JSON.parse(storedUser)); // Restore user from localStorage

}

setLoading(false); // Stop loading after checking localStorage

}, []);

const login = async (credentials) => {

try {

const response = await fetch(`${API\_URL}/login`, {

method: "POST",

headers: { "Content-Type": "application/json" },

body: JSON.stringify(credentials),

});

if (!response.ok) throw new Error("Login failed");

const user = await response.json();

setUser(user);

localStorage.setItem("user", JSON.stringify(user)); // Persist user

return true;

} catch (error) {

console.error("Login error:", error);

return false;

}

};

const register = async (userData) => {

try {

const response = await fetch(`${API\_URL}/users`, {

method: "POST",

headers: { "Content-Type": "application/json" },

body: JSON.stringify(userData),

});

if (!response.ok) throw new Error("Registration failed");

const newUser = await response.json();

setUser(newUser);

localStorage.setItem("user", JSON.stringify(newUser)); // Persist user

return true;

} catch (error) {

console.error("Registration error:", error);

return false;

}

};

const logout = () => {

setUser(null);

localStorage.removeItem("user"); // Clear user from localStorage

};

return (

<AuthContext.Provider value={{ user, login, register, logout, loading }}>

{children}

</AuthContext.Provider>

);

};

**3. Login.js and Register.js:**

The Login.js and Register.js files are React components that manage the user interface for logging in and registering new users. They interact with the authentication logic provided by the AuthContext and communicate with the backend to authenticate users and register new ones.  
Key Components of Login.js:   
1. State Management:  
• The component uses React's useState hook to manage the state of the form inputs (username, password) and also to handle feedback messages like errors or loading states.  
2. Form Handling:  
• The login form captures the user's input (username and password) and handles form submission via an onSubmit handler.  
3. Authentication Interaction:  
• The login function, which is provided by the AuthContext, is called upon form submission. If login is successful, the user is redirected to the dashboard. If it fails, an error message is displayed.  
4. Feedback Handling:  
• The component provides feedback to the user by showing error messages when login fails.  
5. Redirect after Login:  
• Once the user successfully logs in, they are redirected to the dashboard using useNavigate from React Router.  
Key Components of Register.js:  
1. State Management:  
• Similar to Login.js, Register.js uses the useState hook to manage the input fields (username, password, and additional fields like phone number and first/last name) and to track the success or error states.  
2. Form Handling:  
• The registration form collects user details like username, password, and other fields. Upon submission, the form calls the register function provided by the AuthContext to create a new user account.

3. Authentication Interaction:  
• The register function from AuthContext is used to send the user's registration data to the backend. If registration is successful, the user is automatically logged in and redirected to the dashboard.  
4. Redirect after Registration:  
• After successfully registering, the user is redirected to the dashboard.  
  
**4. ElderlyDashboard.js and VolunteerDashboard.js:**  
The ElderlyDashboard.js and VolunteerDashboard.js components are the dashboards for elderly users and volunteers, respectively. These components allow users to manage tasks based on their roles. Elderly users can view tasks they have created, while volunteers can view available tasks and interact with them (e.g., accepting tasks, marking tasks as completed).

Both files share a similar structure, but they are tailored for the specific needs of each user type.

Key Components of ElderlyDashboard.js:

1. State Management:  
• The component uses useState to manage the list of tasks and track the selected task that the user wants to view in more detail.

2. Fetching Data:   
• When the component is mounted, it fetches the list of tasks created by the logged-in elderly user. The tasks are retrieved from the backend using the user's ID.

3. Task Management:  
• The dashboard displays a list of tasks. Each task can be selected for more details, which are displayed in a modal window. The elderly user can view the task's status, see which volunteer accepted it, and track the task's progress.

4. Task Modal:  
• When a task is clicked, more details about the task (such as the volunteer who accepted it, its status, etc.) are shown in a modal. The modal is a reusable component.

Key Components of VolunteerDashboard.js:

1. State Management:  
• The component uses useState to manage the list of available tasks and the selected task the volunteer wants to interact with.

2. Fetching Data:  
• The component fetches tasks from the backend when it is mounted. The tasks shown are those available for the volunteer to accept.

3. Task Management:  
• Volunteers can click on tasks to view more details and perform actions like accepting the task or marking it as completed.

4. Task Modal:  
• When a task is clicked, a modal appears, allowing the volunteer to interact with the task (e.g., accept or complete the task).

**5. TaskList.js:**The TaskList.js file is a React component that handles the display of tasks for both elderly users and volunteers. It dynamically renders tasks based on the type of user and allows users to interact with tasks by clicking on them to view more details or take action (e.g., volunteers accepting tasks, elderly users viewing the status of their created tasks).

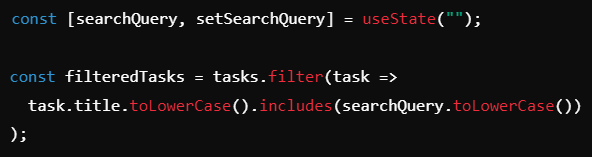
Key Components of TaskList.js:  
1. State Management:  
• The component does not use local state management (useState) since it receives all the data as props. The tasks, onTaskClick, userId, and userType are passed as props from the parent component (e.g., ElderlyDashboard.js or VolunteerDashboard.js).  
2. Rendering Task Lists:  
• The tasks are rendered dynamically based on the userType (either "elderly" or "volunteer"). For example, elderly users will see tasks they have created, while volunteers will see available tasks they can accept.  
• Tasks are grouped into different categories based on their status (e.g., "Open", "Ongoing", "Completed").

3. Conditional Rendering Based on User Type:  
• The tasks are displayed differently depending on whether the user is an elderly user or a volunteer. For example, a volunteer may only want to see tasks that are "Open" (i.e., tasks they can accept), while an elderly user can see tasks they have created and track their progress.

4. Task Click Handler:  
• When a task is clicked, the onTaskClick function (passed from the parent component) is triggered, allowing the parent component (e.g., ElderlyDashboard.js or VolunteerDashboard.js) to display a modal with task details or allow the volunteer to accept or complete the task.

Examples for features that can be added to TaskList.js:

Sorting tasks by creation date



תמונה שמכילה טקסט, צילום מסך, גופן

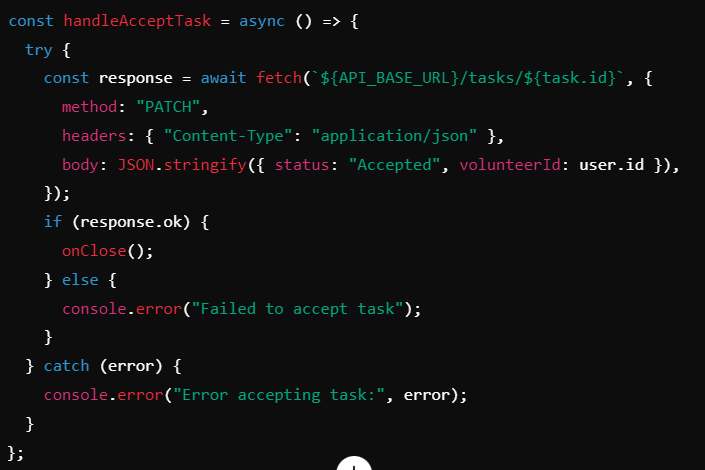
התיאור נוצר באופן אוטומטי

Adding a Search Feature

**6. TaskModal.js:**

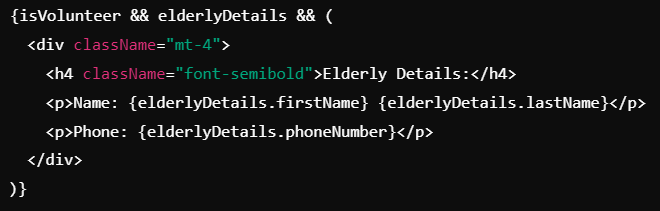
The TaskModal.js file is responsible for displaying detailed information about a specific task when it is selected by either an elderly user or a volunteer. The modal allows the user to interact with the task, depending on their role (e.g., volunteers can accept or complete tasks, while elderly users can confirm completion and provide ratings). The component dynamically renders task details and provides appropriate actions based on the user's type (elderly or volunteer).

Key Components of TaskModal.js:  
1. State Management:

• The component uses React's useState and useEffect hooks to manage the task details (e.g., user details, rating), as well as to fetch related information from the backend.  
• It manages states for user details (e.g., the elderly or volunteer associated with the task), task rating, and modal display logic.  
2. Fetching Data:  
• The component fetches user details (e.g., the elderly user for a volunteer or the volunteer for an elderly user) when the modal is displayed. This is done through asynchronous API calls using the fetchElderlyDetails and fetchVolunteerDetails functions.  
3. Task Interaction:  
• The modal provides different actions based on the user type:  
Volunteers can accept a task, complete a task, or cancel it.  
Elderly users can confirm that the task has been completed and rate the volunteer

Example for volunteers accepting a task

4. Conditional Rendering:  
• The component conditionally renders different sections based on whether the user is a volunteer or elderly. For example:  
Volunteers can see elderly user details and actions for accepting/completing tasks.  
Elderly users can see volunteer details, task progress, and rating options.



Example for conditional rendering

5. Rating System:  
• The elderly user can rate the volunteer after the task is completed. This rating is submitted via an API call, and the volunteer's overall rating is updated.

6. Modal UI:  
• The modal has a backdrop and a close button to allow users to exit the modal without interacting with the task. The layout is responsive and structured with CSS classes for styling.

**7. ChatComponent.js:**

The ChatComponent.js file is responsible for managing the chat functionality between an elderly user and a volunteer within the context of a task. It allows users to send and receive messages in real time, providing an interface for communication specific to a task. The component renders the chat messages, handles message input, and communicates with the backend to send and retrieve messages.

Key Components of ChatComponent.js:  
1. State Management:  
• The component uses React's useState hook to manage the state for messages (both incoming and outgoing), the current message being typed by the user, and potential loading or error states.  
2. Fetching Chat Messages:  
• When the component is mounted, it fetches existing messages related to a specific task from the backend. This is typically done via an API call that retrieves all messages associated with the task's taskId.  
3. Sending a Message:  
• The component provides an input field where users can type a message. Once the message is submitted, it is sent to the backend via an API call and added to the chat. The component handles both the visual updating of the message list and the actual sending of the message to the backend.  
4. Message Rendering:  
• The component iterates through the list of messages and renders each message with relevant information, such as the sender's name, timestamp, and content. Messages are styled differently depending on whether they were sent by the current user or the other user.

5. Auto-Scrolling to Latest Message:  
• When new messages are added, the chat automatically scrolls to the latest message, ensuring that users can always see the most recent conversation without having to manually scroll

6. Message Input and Send Button:  
• The component provides a text input field for the user to type a new message, along with a "Send" button to submit the message.

**8. Alert.js:**The Alert.js file is responsible for displaying alert messages to users. These alerts are typically used to inform users about the success or failure of an action, validation errors, or other important notifications. The component is designed to be reusable, allowing different parts of the application to trigger and display alerts in a consistent manner.

Key Components of Alert.js:  
1. Props:  
• The component receives several props to customize the alert message. These props typically include:  
type: Defines the type of alert (e.g., success, error, warning, info).  
message: The message text that will be displayed to the user.  
2. Conditional Rendering Based on type:  
• The appearance of the alert changes based on the type prop. Different styles are applied depending on whether the alert is a success, error, warning, or informational alert.  
• This is typically done using conditional class names.  
3. Close Button:  
• The component includes a close button (×) that allows users to dismiss the alert. When the close button is clicked, the onClose callback is triggered, if provided, to remove or hide the alert.  
• The onClose callback is optional and can be omitted if the alert is meant to disappear automatically or if no manual close action is required.  
4. Auto Dismiss:  
• Optionally, you can implement an auto-dismiss feature where the alert disappears after a certain amount of time. This could be useful for success messages that should disappear after a few seconds without requiring user interaction.  
5. Styling and Appearance:  
• The alert's appearance is styled based on the type prop and additional CSS classes. The component is usually rendered with a colored background, border, and text to match the alert's type (e.g., green for success, red for error).  
• The component's layout and structure are typically responsive and designed to work across different screen sizes.

**9. Notifications.js:**The Notifications.js file is responsible for displaying notifications to users within the application. These notifications can inform the user about various events, such as new tasks, updates, or general messages. The component handles fetching notifications from the backend, rendering them in a list, and opti006Fnally marking them as read or dismissing them.

Key Components of Notifications.js:  
1. State Management:  
• The component uses React’s useState and useEffect hooks to manage the notifications state, including the list of notifications fetched from the backend, loading state, and error handling.  
2. Fetching Notifications:  
• When the component mounts, it fetches the user's notifications from the backend. This is done by sending a request to an API endpoint and updating the state with the fetched notifications.  
3. Rendering Notifications:  
• The component maps over the notifications array and renders each notification with its relevant details (such as the message, type, and time). It conditionally applies different styles based on the type of notification (e.g., success, info, warning, error).  
4. Marking Notifications as Read:  
• Users can mark notifications as read. Once a notification is marked as read, an API call is made to update its status in the backend, and the local state is updated to reflect the change.  
5. Conditional Styles Based on Notification Type:  
Different types of notifications (e.g., success, warning, error) are styled differently. The component dynamically applies styles based on the notification’s type.

6. Error Handling:

• The component handles errors that may occur during the fetching process or when interacting with the backend (e.g., marking a notification as read).