Bilkent University

Department of Computer Engineering

Senior Design Project

OverSeer

Final Report

Talha Şen
Hakan Sivuk
Ahmet Berk Eren
Cevat Aykan Sevinç
Yusuf Nevzat Şengün

Supervisor: Ayşegül Dündar

Jury Members: Ayşegül Dündar, Selim Aksoy

Contents

Final Report

OverSeer

1. Introduction

OverSeer is a mobile application that aims to remove barriers for visually impaired people. OverSeer navigates people for the places they want to go and warns them towards obstacles they face on their paths. Next, OverSeer provides live support to help them with any problem. They can ask for price information at a supermarket or they can just want a volunteer to show the correct location of an object with the help of live support. OverSeer is fully accessible. Visually impaired users can navigate across the application with ease and the app can communicate using voice features.

In this report, we are going to discuss our final system considering its full life-cycle from requirements to implementation. Then we are going to talk about other elements such as risks and alternatives, our project maintenance plan, how we planned our team-work, our ethical and professional responsibilities and the new knowledge and strategies we learned and followed.

2. Requirements Details

2.1 Overview

For the visually impaired people who will use our application, OverSeer was planned to mainly work through voice commands. However, we realized that to provide such a feature requires complex natural language processing and due to the time constraints, we decided to look into other ways of accessibility. Therefore, OverSeer provides a fully accessible UI for the visually impaired while using voice features of the device for updating the user. Volunteers are provided a more sophisticated UI so that UX is improved.

OverSeer provides two main functionality to the visually impaired: navigation and live stream. Users can search for a place to navigate. The user can enter their desired location. Once they confirm the address, they can either save this place as their favorite or start the navigation process. The app guides the user through voice commands.

Through the navigation, an object detection system works in parallel. Two systems are provided. One system works in Cloud while the other one is implemented in the local environment. The user can choose which model they prefer for obstacle detection. This allows the user to choose between a more performant but Internet consuming service or less performant but Internet friendly service.

Both systems detect obstacles through travel and warn the user if these obstacles become a threat for the user. Besides, a traffic light detection system is used. It aims to detect the current state of the traffic light for pedestrians and inform the user. For example, when trying to cross a road, the object detection system detects the state of the traffic light, and when it lights green, it tells the user to cross the road. Thus, navigation is made safer with obstacle detection.

The users can also ask other people for help. By using live streaming, volunteers can connect visually impaired users' devices to become their helper eye. The application has a live streaming service that allows other people to see the environment of the visually impaired user. All the users will sign in / sign up to the application with their phone number. This allows us to distinguish different users by their unique phone numbers. After any user signs in / signs up for the first time, their session details are saved. This way, they do not have to sign in / sign up again although they can still logout. Next, Visually impaired users can use the navigation and live stream services after logging in while volunteers can advertise themselves as available for help.

2.2 Final Functional Requirements

2.2.1 Navigation

Users are able to tell OverSeer where they would like to travel. They can save their places of interest and navigate to their point of interest either through searching or by the saved places. OverSeer guides users by voice commands for navigation. OverSeer warns users if they are off-path and guides users back to the correct path. Occasionally, OverSeer tells the remaining distance to the point of arrival.

2.2.2 Obstacle Detection

While users are traveling, they must be notified if an obstacle is on their path so that they could avoid potential harm. OverSeer alerts the user if any obstacle is detected. Obstacles can be cars, poles, holes, fire hydrants and any other object on the path. OverSeer does not warn users when these obstacles are not in the user path or within a certain distance to the user.

2.2.3 Traffic Light Detection

Furthermore, traffic lights can be processed to guide the user for crossing the road. Users can wait for the correct light to cross the road. Also, it detects the start and endpoint of the midline of zebra crossing and directs the user accordingly.

2.2.4 Place Discovery

Users are able to tell OverSeer which places they seek at their current location. OverSeer finds the most relevant places the user asks for and provides an option to navigate to these places. OverSeer informs the user about the found places in an accessible manner. A place can be a pharmacy, restaurant, cafe, ATM.

2.2.5 Live Support

Users are able to ask help with live support. The user will be able to live stream their environment using their device camera to a volunteer or their predefined friends, relatives. Visually impaired users can add their friends by phone and request help by specifically choosing them. Moreover, they can ask a volunteer for help available at that moment. Volunteers can advertise themselves as available so that visually impaired users can ask them for help implicitly using OverSeer.

2.2.6 Accessible Control

The UI in OverSeer is fully accessible for the visually impaired users according to the guidelines [1]. The accessibility is powered by talkback in Android systems. Each UI element has its own purpose explained to the user when they navigate to it. Moreover, each group of UI elements state their opinion when the page is loaded or the user navigates on the group. Finally, OverSeer uses the voice features available in Android systems to report either any errors or meaningful information to the user as a feedback.

2.3 Final Non-Functional Requirements

2.3.1 Performance

- **Obstacle Detection Performance**

The Back-end side of the obstacle detection system gets data from the camera, processes it, and finds the obstacles then warns the user. Besides the accuracy of the detection task, these events should be fast in order to avoid potential harm from close obstacles. Therefore, the processing times of the obstacle detection system are really important and the obstacle detection process should be done within 1 seconds.

Some users may prefer a more Internet friendly approach. As a result, along with the backend, another obstacle detection system runs locally although less performant than its Cloud solution. The local system detects and reports obstacles within 500 ms.

- **Live Support Performance**

Live support provides both audio and video communication between the users and the helpers. Hence, the latency of the live support system is an important concern. There must not be large delays more than 5 seconds to maintain healthy communications.

2.3.2 Usability

OverSeer must be easy to use for in-app navigation. Therefore, These voice commands should be simple and effective. Also, OverSeer must avoid ambiguity and inform users about their actions. It should inform users which page they are on and what they can do on it. Each UI component must have an explanation by the Talkback feature defined in the Android systems. There should not be any UI component that does not describe what its purpose is.

2.3.3 Integrity

The integrity of the obstacle detection system is crucial because any miscalculation can lead to an accident. Therefore, the obstacle detection algorithms must be accurate and consistent. Also, the computer vision algorithms may work with errors, so the most reliable computer vision libraries should be used to minimize any kind of error.

2.3.4 Security

The security of both visually impaired and volunteer users are important. Some bad volunteers can mislead or make fun of the visually impaired user. In order to avoid these issues, the live support system should have a report function. So, the users can report any bad behavior of the volunteer and the volunteer can be banned from the application. Also, banned users should not create another account. Therefore, phone numbers will be treated as unique ids to associate users with their accounts.

2.3.5 Availability

OverSeer must be available 7/24 because the users of the application may need assistance anytime. As a result, OverSeer backend services will be deployed to Amazon Cloud Services as it is one of the most reliable services.

2.4 Final Pseudo Requirements

2.4.1 Version Control

Git/Github must be used as a version controller throughout the project. As the group members are familiar with Git, this allows seamlessly fast collaboration between the developers of OverSeer.

2.4.2 Implementation Language

Java and Kotlin will be used for the project. Normally, the project was going to be developed using React Native library. However, our obstacle detection requirements demanded a native solution which could not be done with React Native. Therefore, Android Studio and Java/Kotlin are the main implementation languages. Java is preferred over Kotlin since developers are more familiar with Java. However, some Kotlin code will be used to handle implementation cases where Java is not enough such as in dealing with dependencies.

2.4.3 Target Platform

Since the application is implemented in Android studio, the target platform is Android systems. Our goal used to deploy the project for both platforms. However, due to problems requiring native solutions, we decided that native implementation would be better. IOS support will be discussed in the maintenance plan section.

2.4.4 Frameworks

Python3 and SageMaker API are used in the backend side of the project. Android Studio is used for the frontend side while Gradle is used for building the project.

2.5 Final User Scenarios

- **Selecting a Destination, Navigating to It and Crossing a Road**

A visually impaired user starts the navigation with “Open Navigation” voice command. The app gets a destination from the user, and the app searches saved locations to see if the selected destination is included in them. If not, the app tries to search the destination on the map and if it cannot find it, it asks for another destination. If it does, it marks the destination and the navigation starts. As the user travels to the destination, an obstacle detection system tries to detect obstacles along the way and warn the user if any threatens the user. While traveling, the user may need to cross a road, in which case the app detects the traffic light and checks until it lights green. When it lights green, it notifies the user to cross. This process continues until the user reaches their destination. Navigation stops when the user reaches their destination.

- **Saving a Destination and Using It With Its Name**

OverSeer keeps all the destinations in its memory. When the user wants to go to a destination, the application asks the user if the destination to be saved. The user gives a name to the destination while saving it as frequently used. After this process, the user can access the destination on their saved places.

- **Matching Socks With the Help of Live Support**

A visually impaired user wants to match his/her pairs of socks. The user starts the application and navigates to the live support page using the UI powered with talkback. The user asks for help from his/her pre-saved contact or available volunteers by selecting the help type they require that is read out by talkback.. OverSeer finds an available volunteer and connects him/her to the user.

- **Searching and Advising Locations to the User**

The user gives a location type (eg. supermarket, train station, bar) to the application, then the application searchers for near locations fitting this type. If it does not find one, it asks for another location type to the user. If it does, it gives a location advice to the user. User makes their preference and the application marks the chosen location as a navigation destination and starts the navigation.

- **Volunteer user helping the visually impaired user**

A volunteer user for OverSeer marks themselves as “available”. After this, a visually impaired user can ask for assistance from available volunteers. A notification is sent to the volunteer, which they can accept or decline. If accepted, OverSeer tries to establish a connection between the two users. If connection is failed, the volunteer goes back to the available state, if not, a livestream starts from the visually impaired user to let the volunteer help.

- **Registration Scenario**

The user registers to the app by giving his/her phone number and determining a password. He/she also indicates whether he/she is a visually impaired person or a volunteer. After this process, he/she can login with its phone number and password. Signing in/up one is enough to save their session details.

- **Visually Impaired Person Adding Friends**

A visually impaired person can allow the application to access the phone book. This means that the application could add relatives or friends of the visually blinded user as friends in the application. This way, the visually impaired person may easily ask for help from their friends saved in the application.

- **Live Support with a User from Friend List**

The user selects another user from his/her friend list and if this user is available, make a live support call with him/her.

2.6 Final Use Case Diagrams

2.6.1 OverSeer Services

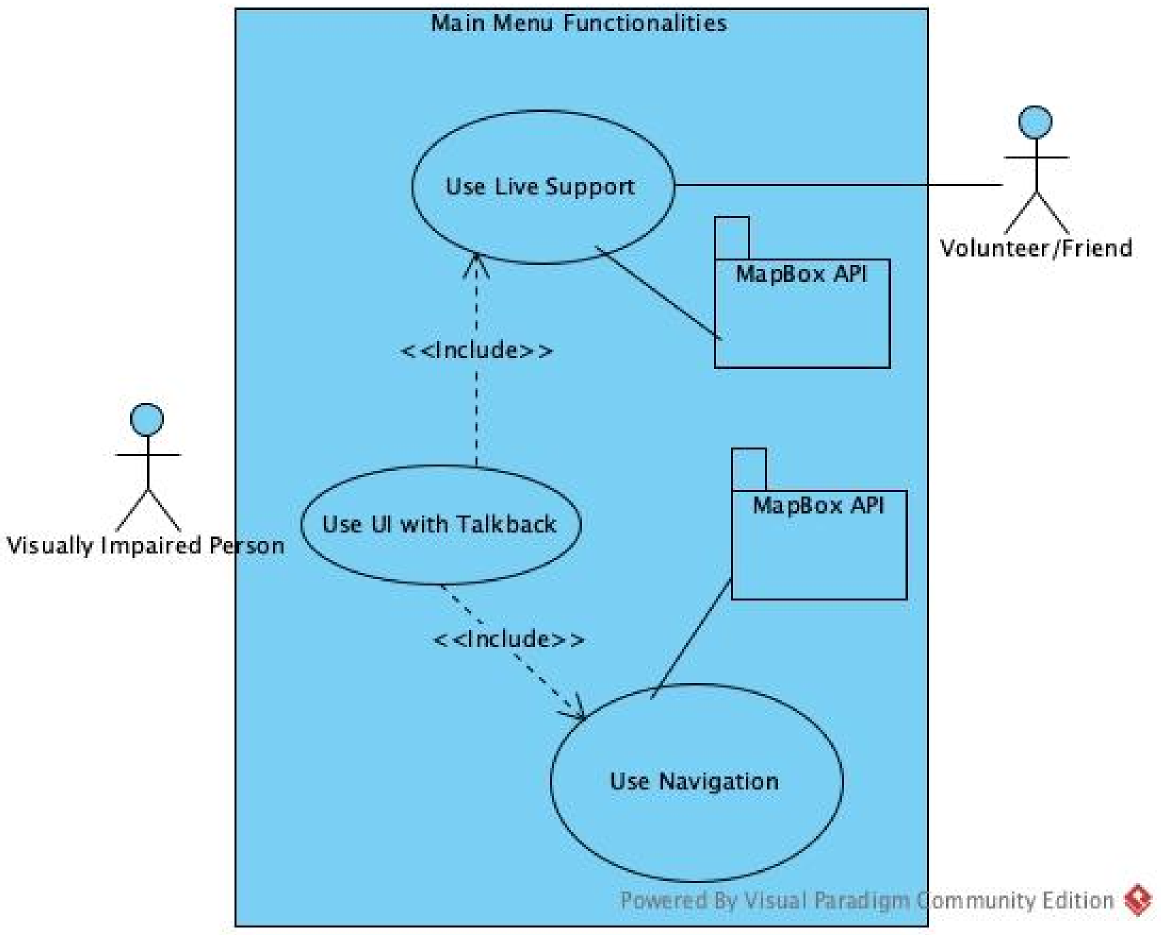

Figure 1. OverSeer Services Use Case Diagram.

Visually impaired users are able to navigate in the app using talkback features. They could use navigation or live support. Volunteer users are capable of using live support as well.

2.6.2 Live Support Use Case Diagram

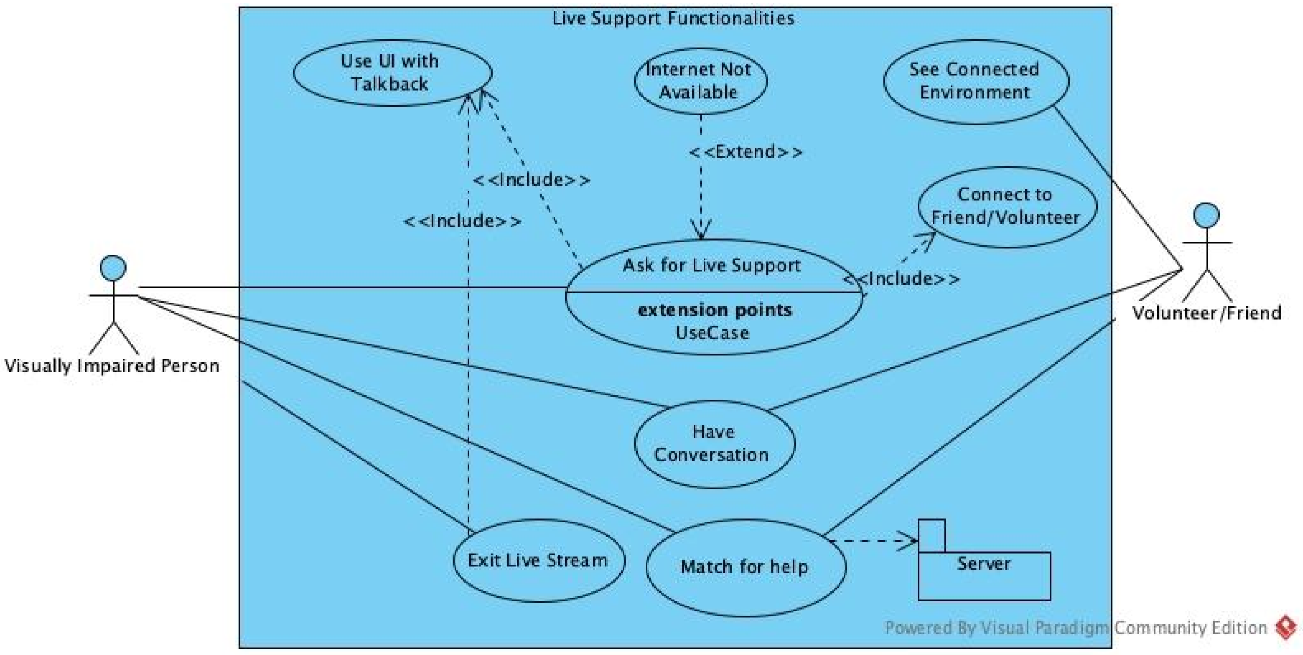

Figure 2. Live Support Use Case Diagram.

Visually impaired users are able to use the UI powered with Talkback to navigate in live support service. They must be able to search for help, whether it is a volunteer or their friend. They can add their friends to OverSeer. Both user types are able to match with each other to communicate while volunteers may see the environment throughout the device camera of visually impaired users. They are able to hear each other after the connection is set up.

2.6.3 Navigation Support Use Case Diagram

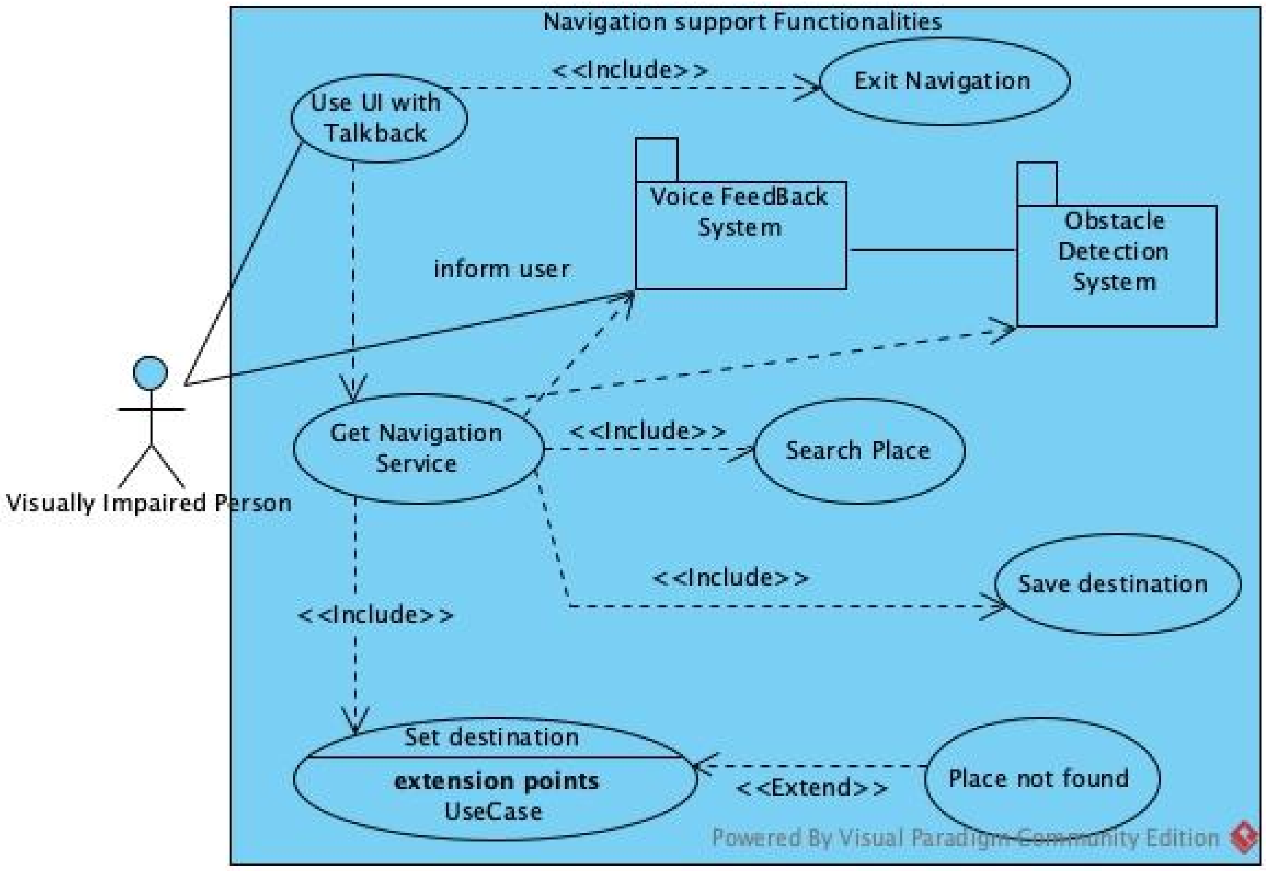

Figure 3. Navigation Support Use Case Diagram.

Visually impaired users are able to use the accessible UI provided with Talkback to navigate in the navigation support. Users can search for a place, save this place, navigate to this place. During navigation, they could get assistance on detecting obstacles.

3. Final Architecture and Design Details

3.1 Proposed Software Architecture

For the visually impaired people who will use our application, OverSeer provides an accessible design. The user can start OverSeer's navigation as it's one of the main functionalities. Next, the user can also ask other people for help. The application has a live streaming service that allows other people to see the environment of the visually impaired user. All the users have to sign to the application with their phone number, and they choose if they are visually impaired, meaning they will use the application for navigation or if they are voluntary helpers, meaning they will use the application to help the visually impaired users through live streaming.

OverSeer provides a software architecture that can provide the required services. The architecture is layered in a way to provide such services. The bottom of the layer is the core of systems. The core systems provide the base functionalities required in other services. These core systems can be used by any other complex systems in OverSeer. Core systems include persistent data management, database management, event system. Next, on top of the core system is the service layer. This layer is where required functionalities such as navigation, accessibility and live stream is provided. These systems define new libraries unique to themselves while using the services of the core systems. At the very top is the input layer where the user interacts with OverSeer. Input layer consists of UI and accessibility. It is responsible for creating events for handling user interaction and displaying the correct response to such actions. Thus, this layered approach in our monolithic architecture allowed us to divide and conquer required functionalities in OverSeer while reusing most of the code written.

3.2 Design Goals

3.2.1 Correct Functionality (Robustness)

The application is used by mainly visually impaired. As a result, the functionalities we provide must be robust and the architecture must be designed accordingly. As a result, we emphasized tests on the implemented services with the software architecture. Moreover, we integrated trustworthy external APIs when implementing such services. We separated each sub system in the architecture to encapsulate their functionalities while decoupling them. This increased the robustness of services since an error in a system is only contained in that system.

3.2.2 Usability / User Experience / Accessibility

Our application must be easy to use for the visually impaired. Therefore, the architecture must comply with accessibility best-practises and guidelines. Accessibility tests were performed to ensure this.

3.2.3 Efficiency

The application must be developed while aiming for efficiency. This is important because the machine learning model and the application must be responsive. When an input is given by the user, they should not wait for the response. This is a must for the machine learning model. While users are navigating to locations, the machine learning model must detect obstacles and report them in time. If the model is not responsive, this would decrease the safety of the OverSeer. The architecture in OverSeer is open for efficiency while preventing premature optimization.

3.2.4 Maintainability

The system must be maintainable. It is going to be used by users in the following years. Each service needs maintenance to have a consistent correct functionality and improved service quality after OverSeer is published. As a result, we aim for an independent state-like system. As a result, if a failure happens in one system, then other systems would be unaffected. This would allow developers to maintain OverSeer as the changes made in one state would not be visible to other states. This state-pattern would also comply with the correct functionality design goal. Services such as live support and navigation are our main states.

3.2.5 Performance

OverSeer will rely on network connectivity on some functionalities such as location suggestion, improved navigation quality, and live support. As a result, network performance is important for OverSeer. The architecture in OverSeer is designed in a way to optimize the connection to the backend.

3.3 Subsystem Decomposition

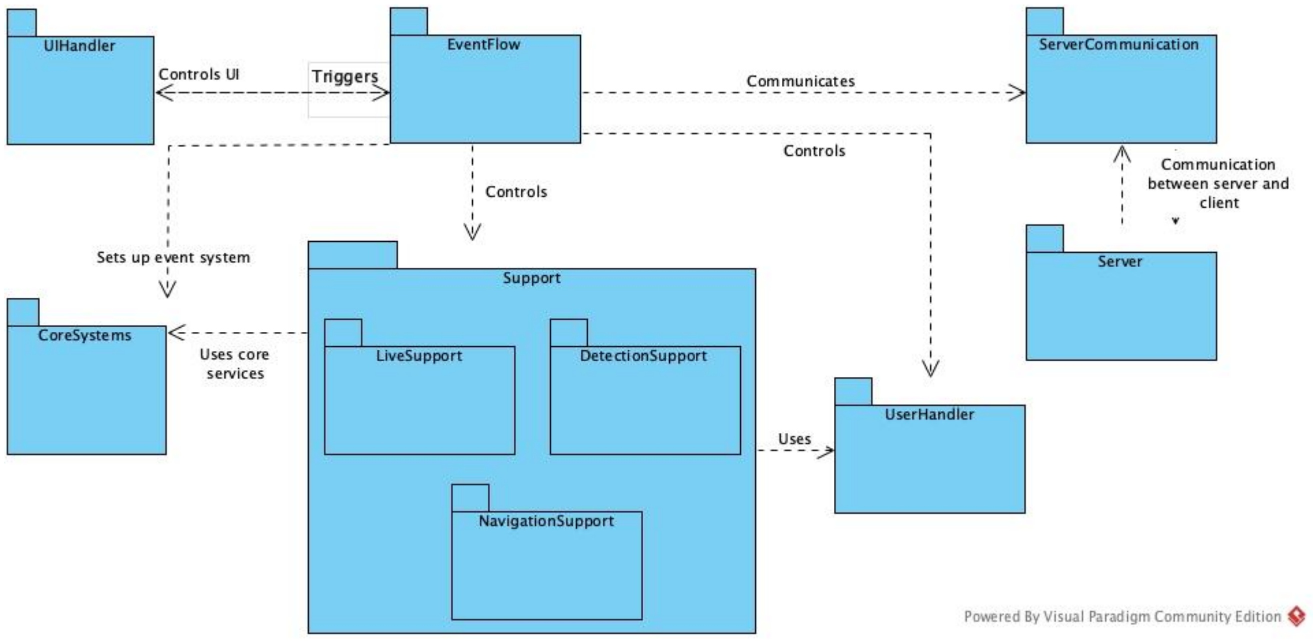

Figure 4. Subsystem Decomposition.

- **UIHandler**

This subsystem is for our UI. This subsystem is responsible for implementing the accessible UI for the visually impaired users while also answering volunteers. As a result, this subsystem would be responsible for handling the UI inputs from both users.

- **EventFlow**

OverSeer would be event-driven. This would allow the systems to be independent. EventFlow will handle the event flow in the system while managing subscribers. When an event is raised, event flow will collect this event and notify any listeners.

- **Core Systems**

This subsystem will be responsible for implementing core services required by support sub systems. These services include persistent data management, event system and database connection.

- **Support**

Main functionalities of our application will be provided by the support subsystem. Users can choose which support they want to use and each support can handle their request using an event-driven approach.

- **Client / Server Communication**

This subsystem is responsible for handling connections between clients and the server. This subsystem would complement other support systems when they need network services.

- **UserHandler**

This subsystem will handle any user-specific task such as login, register, and persistent user data, reporting volunteers.

3.4 Hardware / Software Mapping

- **Detection Support**

Hardware/software mapping of the detection support is rather complex. There is both the client machine and cloud. This means that the detection related software operations are performed on either the user's hardware system or on Cloud. When the detection related software operations are performed on the user’s hardware, we use NPU (Neural Processing Unit) specialized for neural network systems. We use the storage of the user phone for storing model weights for the local model and store weights of the cloud model in SageMaker. User decides which mapping will be used for execution.

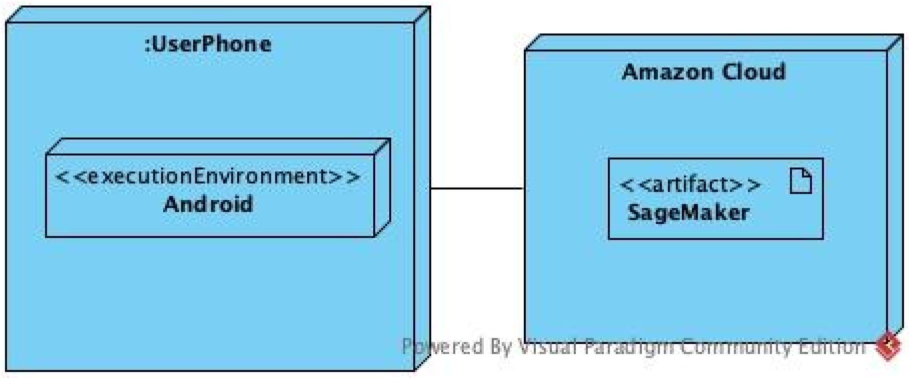

Figure 5. Deployment Diagram for Detection Support

- **Navigation Support**

Hardware/software mapping of the navigation support is simple. There is only the client machine and the navigation-related software operations are performed on the user's phone hardware system. We use the location services of the client machine to find the location.

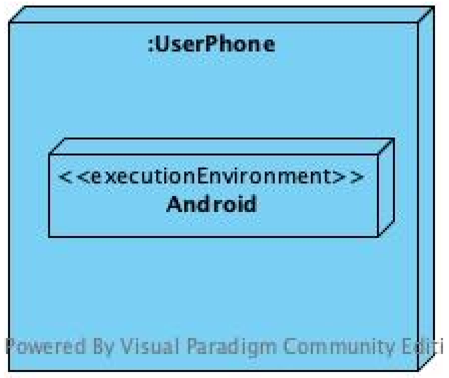

Figure 6. Deployment Diagram for Navigation Support

- **Live Support**

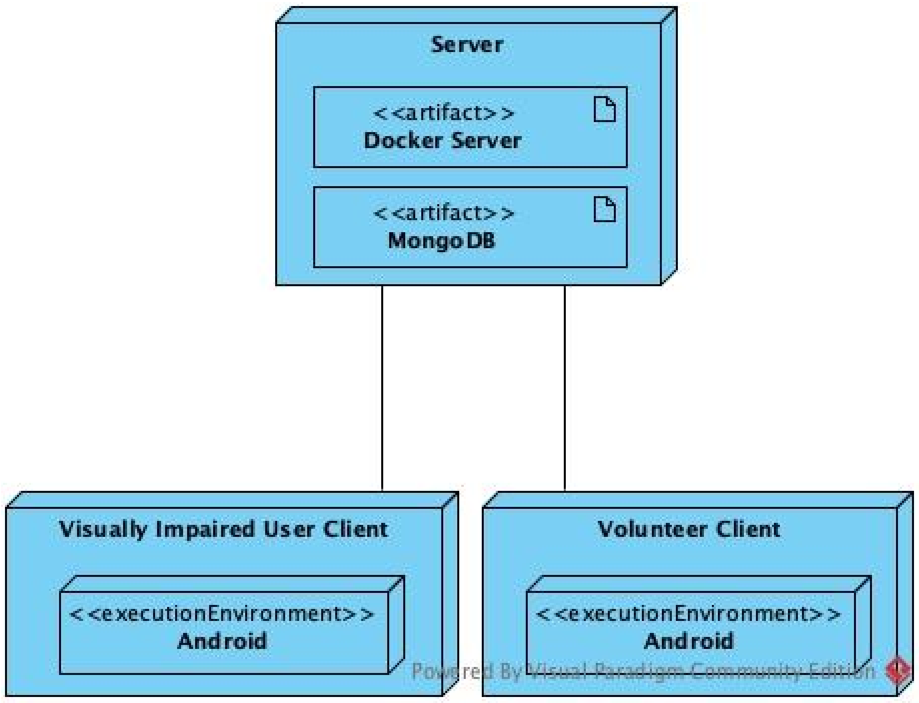

Figure 7. Deployment Diagram for Live Support

Hardware/Software mapping for live support consists of a server and 2 clients which are a visually impaired user and a volunteer. The server operates on an online host and the user interface for each user operates on the client smartphone hardware as a software. The server provides a visual and audio connection between users. The server also stores the IDs of the users.

3.5 Persistent Data Management

Images will be kept in the local hard disk of the users. This is because we believe it is unnecessary to load images from an online source because of performance. If images are kept in local hardware, then this problem is solved. This is a tradeoff as the memory space increases while the performance is optimized.

In addition to this, we also have a database that holds the userIDs and passwords of the registered users. This is because a user must register an account one time and they must use that account to login every other time. If we made the user register an account every time they wanted to use OverSeer, it would reduce the usability and performance. OverSeer has a server that manages all the data related to the other accounts. It is persistent and independent of the application from a local environment.

In the local environment, we store the session of the player using Android system's preference manager. This allowed us to store data related to users. As a result, key information such as session data and settings are stored on the user device.

Apart from these, trained machine learning weights must be stored, because we need them for obstacle and traffic light detection. Since it is a real-time object detection task and it should be fast for producing sufficiently early warning, performance is crucial for detection systems. Therefore, these weights are kept in the both local hardware of the user and in the cloud.

3.6 Access control and security

OverSeer has an accounting system with unique userIDs and passwords. These passwords are kept as hashed inside the main server to avoid account theft. In the register, the phone number of the user is used as a userID. Therefore, if an account is banned from the system the owner of the account can not register again with the same phone. This aims to avoid harmful users to use OverSeer. Consequently, if a user got banned from the system, then they can not reaccess it.

In live support, the volunteer can access only the camera and microphone data of the visually impaired user. On the other hand, visually impaired users can access only the microphone data of the volunteer during the call. Such permissions are asked to the user when they start using OverSeer.

In our obstacle detection system, there are not any different users. Because of this, there is not any special access control for users. As the system is working in a local environment, there are not any security concerns for our obstacle detection system. In the cloud model deployed, connection is secure as secure since we are using Amazon SageMaker. As a result, local hardware only sends the inputs to the SageMaker while getting the output. User information is not sent to the SageMaker. Thus, security is ensured.

3.7 Global software control

Since the server provides services to all users, it is possible to face race conditions when different users try to use the same data. To prevent any problem, the server should be event-driven. It should handle its tasks in an event-driven way where services are executed when the related event is called. In addition to preventing race conditions, event-driven programming is an important way of providing quick and accurate responses to the client. Event-driven programming is supported by many programming languages, and it will be used especially for the communication between server and client.

The local software in the client would be state-driven. This is because separating service logic increases the robustness. Each state would be responsible for handling their own events in a decoupled manner. This way, the software is easier to control and debug. Each state can handle its exceptions and boundary conditions.

3.8 Boundary conditions

- **Initialization**

When users want to use the application for the first time, they are navigated to the login screen where they can either log in to or register an account. After successful authentication, the user is navigated to the main screen where one can start navigation support or request help from a volunteer (visually impaired user's point of view). If they request help, the server sends notifications to available volunteers. When one of them accepts the request, the server initializes the video call. If the user wants to use the navigation support, the target address is asked by the application. After the user enters the information, the route is determined and the navigation service starts.

Also, the server must be initialized. Initialization of the server includes initializing sockets and connection ports, initializing database connection, creating necessary objects, and calling proper listening methods for the server & client communication in SageMaker.

- **Termination**

When the live support service is terminated accidentally (such as internet connection loss), the user waits for a while. If they cannot or do not reconnect, the video call is terminated by the server.

- **Failure**

As correct functionality is one of the most important design goals for us, our system will have a strict event flow to force users to follow the expected events. User inputs will be handled as stated in the correct functionality design goal. The application will not respond to any input other than defined user functionalities.

User authentication failures can happen if the user either enters the wrong username-password combination or if he tries to register an account with a username that already exists in the database. If the application cannot find a volunteer for the visually impaired user, the process must be terminated. Most importantly network failures can happen and any one of the users can lose connection to the internet at any point. In the case of live support, the video call must be terminated and everyone should be directed to the main screen.

Also, since the server must run always, it must be robust in terms of failures. It should have a proper mechanism for handling exceptions and boundary cases. As it should handle these exceptions, it should also send proper error messages to the client to inform the user. For implementing this mechanism, different error and response models are implemented.

4. Development/Implementation Details

4.1 Engineering Standards

It is important for our application to comply with the standards of the software. Standards provide documentation to ensure reliability and quality [2]. Since OverSeer is developed for the visually impaired, it is critical to provide safety, security, and reliability.

Our goal is to comply with ISO standards. ISO ICS 35 provides standards over information technology. By the end of our development life-cycle, we aim to comply with software standards by following ISO ICS 35.080 documentation on quality assurance [3].

4.2 Object Design Tradeoffs

Our software architecture will be monolithic event-driven architecture. This is because monolithic architecture is simple to understand and implement while events enable developers to decouple subsystems. To control and distribute events, there will be an event center where each observable element reports itself. Any object can subscribe to any observable during its lifetime. As a result, when an event is raised, any subscriber can receive it instantly.

Event-driven designs are hard to debug and maintain. However, we believe a decoupled logic is cheaper to modify, thus, outweighing its disadvantages. We will use the component pattern to allow an object to subscribe to multiple events by controlling the full life cycles of concrete Observer objects. The object will have strict control over its components. These components will be encapsulated in a wrapper object, acting as an API for concrete objects to control their event flow.

One of the key design goals of our application is accessibility. Visually impaired users must be able to navigate the app on their own. Considering many accessible UI elements will be exposed to the user, their actions have to be controlled. To separate accessibility logic, and button functionality, the command design pattern will be used.

The application has a singleton service center. OverSeer has two main services: navigation and live stream. Moreover, there are support services such as object detection and accessibility. There should be only one service instance that must act as an API for the control objects. These services will be encapsulated in a singleton factory object, allowing ease of access to any service. Although this increases coupling, the services act as an API. Therefore, depreciated function calls could easily be refactored.

4.3 Packages

4.3.1 Internal Packages

- **Camera2**

Camera2 is the internal hardware package Android provides after the Camera package got deprecated. From the official documentation: "This package models a camera device as a pipeline, which takes in input requests for capturing a single frame, captures the single image per the request, and then outputs one capture result metadata packet, plus a set of output image buffers for the request. The requests are processed in-order, and multiple requests can be in flight at once. Since the camera device is a pipeline with multiple stages, having multiple requests in flight is required to maintain full framerate on most Android devices" [4].

We use this package to obtain a manager and control the back camera of the phone (if it exists and if we get permission to it) to scan the environment, obtain frames and send it to the detection subsystem for prediction and further processes. It is an essential package to our detection support subsystem.

- **Android.Speech.TTS**

Android system library for processing text input to output voice commands. This package is used by the feedback service to alert the user on various activities.

- **Startup**

This package defined under OverSeer includes the classes required in startup such as sign in & up.

- **NavigationActivities**

NavigationActivities package consists of navigation classes that handle the navigation support. The responsibilities include searching for a palace, saving a place and navigating to a location.

- **LiveActivities**

LiveActivities package includes classes related to live support functionality. This package is responsible for handling friends, calling a friend or calling a volunteer. From a volunteer's point of view, volunteers can advertise themselves as available for call and accept incoming calls.

- **Adapters**

Adapters package includes adapters for scroll view UI in OverSeer.

- **Models**

Models package includes model classes for the Adapters package.

- **Systems.BaseSystem**

The BaseSystem package implements the core systems used in OverSeer. Every system derives from the BaseSystem.

- **Systems.FeedbackSystem**

Feedback system package provides support for abstraction of the voice feedback.

- **Systems.EventSystem**

The Event system package provides a global event access point to each class in any package in OverSeer.

- **Systems.Utilities**

Utilities provide general purpose services such as persistent data management and constant key access.

- **Systems.UISystem**

The UI system provides the accessible base UI classes to provide general purpose extensible UI programming.

4.3.2 External Packages

- **Pytorch**

Pytorch is a well-known machine learning library that allows developers to train and use several types of models for many tasks. In this project, Pytorch is used for training machine learning models (in Python). To load these trained model weights and perform detection tasks, Pytorch Android library is used. It is essential for us to use optimized algorithms of this library for a lower inference time and a more efficient system performance.

- **TFLite**

TFLite(TensorFlow Lite) is a set of tools provided by a well-known machine learning library TensorFlow. It allows developers to use machine learning models on mobile devices with different options such as GPU or NNAPI delegation for the machine learning task. In this project, TFLite is used for making use of special hardware called NPU (Neural Processing Unit) on mobile devices which is specialized for neural network systems. It is essential for us to use this special hardware for a lower inference time and a more efficient system performance.

- **Mapbox.Directions**

Mapbox.Directions API can provide point to point navigation routes with several alternatives. The given routes are combined with directions. The directions are separated by turns throughout the route. Additionally, the API provides a textual explanation of each direction along with the star and end coordinates.

- **Mapbox.MapMatching**

Mapbox.MapMatching API provides the real-time information for the user. For example, it provides the direction of the user and matches between the direction and the route. Also, it can provide remaining duration information about the route.

- **Mapbox.Geocoding**

Mapbox.Geocoding API provides basic search capabilities. Searching locations with a keyword is its main capability. It provides several alternatives for the given keyword regarding user’s current location, the popularity of places, and several other metrics. This API also provides special IDs (POI) to places. This way, a place can be represented by a unique ID (POI), and the location can be stored easily.

- **Mapbox.VectorTiles**

  Mapbox.VectorTiles API provides the required visual contents for the application. It includes the map itself mainly, but not limited to that. The API also provides real-time route and location drawing onto the map. This way, the user can follow the route visually, as well, which will be optional in our application.

4.4 Class Interfaces

4.4.1 Base Systems

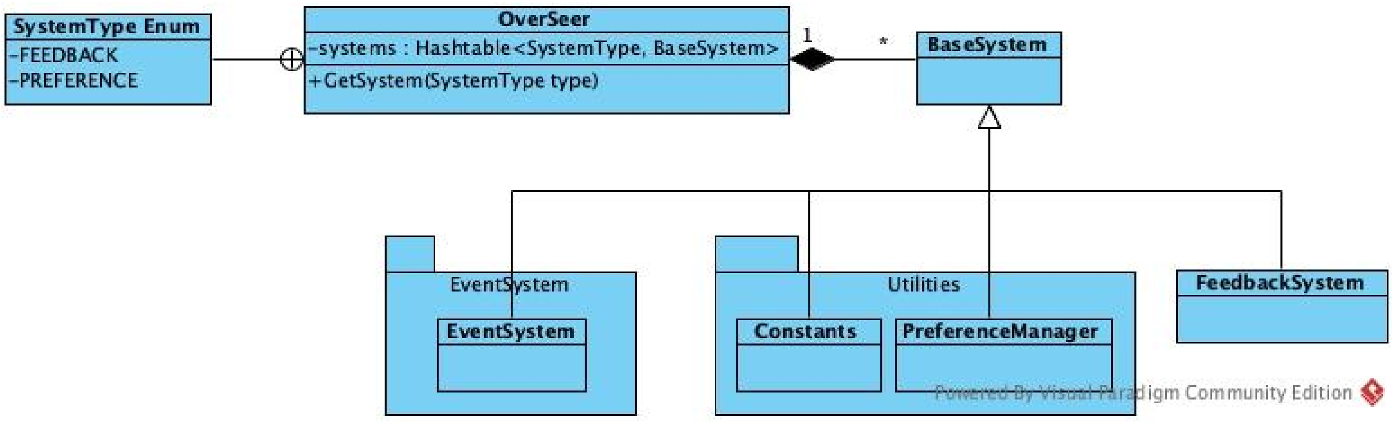

Figure 8. Base Systems Low Level Design

Base systems provide a global access to classes in the application.

- **OverSeer**

Attributes:

- static Hashtable< SystemType, BaseSystem> systems: Stores every system type in a collection for global access.

Methods:

- public static GetSystem(SystemType type): returns the specified system to the caller from the systems.

- **Constants**

  Provides a location for constant keys used across application.

- **PreferenceManager**

Attributes:

- Sharedpreferences sharedPreferences: provides application domain level persistent data management.

Methods:

- public void putBoolean(String key, Boolean value): puts a boolean to the application reserved memory.
- public Boolean getBoolean(String key): gets a boolean from the application reserved memory.
- public void putString(String key, String value): puts a string to the application reserved memory.
- public void getString(String key): gets a string value from the application reserved memory.
- public void clearPreferences(): clears the application reserved memory.
- **FeedbackSystem**

Attributes:

- TextToSpeech tts;

Methods:

- public void NotifyEvent(FeedbackEvent event): notifies the event so that tts can play voice.
- **BaseSystem**

  Provides an abstract inheritance for each system.

4.4.2 Event System

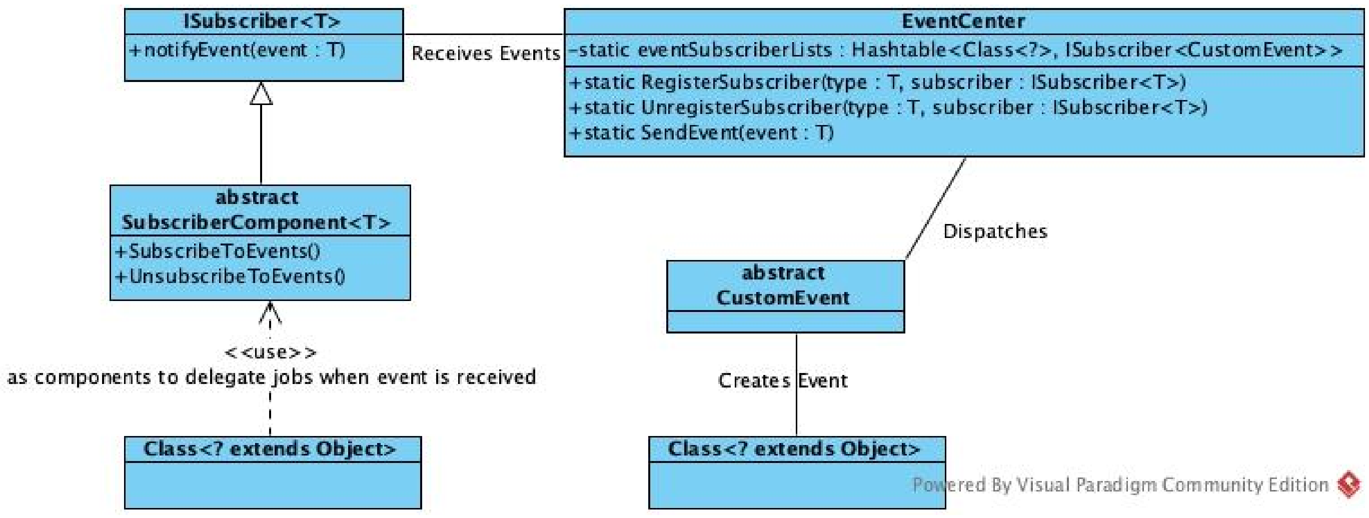
ISubscriber<T>
+notifyEvent(event : T)
Receives Events
EventCenter
-static eventSubscriberLists : Hashtable<Class<?>, ISubscriber<CustomEvent>>
+static RegisterSubscriber(type : T, subscriber : ISubscriber<T>)
+static UnregisterSubscriber(type : T, subscriber : ISubscriber<T>)
+static SendEvent(event : T)
abstract SubscriberComponent<T>
+SubscribeToEvents()
+UnsubscribeToEvents()
Dispatches
abstract CustomEvent
<<use>>
as components to delegate jobs when event is received
Creates Event
Class<? extends Object>
Class<? extends Object>
Powered By Visual Paradigm Community Edition

Figure 9. Event System Low Level Design

Event System is responsible for managing and distributing events.

- **Abstract CustomEvent**

Base for every custom defined event. Empty class.

- **EventCenter**

Attributes:

- static Hashtable< Class<? extends CustomEvent>, HashSet<ISubscriber<CustomEvent>> eventSubscriberLists: Stores every observer of type custom event in the relevant set.

Methods:

- public static <T extends CustomEvent> void RegisterSubscriber(Class<T> type, ISubscriber<T> subscriber): Makes a subscriber (observer) unregister listening to the custom event T.
- public static <T extends CustomEvent> void UnregisterSubscriber(Class<T> type, ISubscriber<T> subscriber): Makes a subscriber (observer) unregister listening to the custom event T.
- public static <T extends CustomEvent> void SendEvent(T event): Sends event to the subscribers.

- **ISubscriber<T>**

Methods:

- public void notifyEvent(T event): Contracts a concrete class to receive events of type T.

- **Abstract SubscriberComponent<T>**

Methods:

- public void SubscribeToEvents(): Subscribes to the events of type T.
- public void UnsubscribeToEvents(): Unsubscribes to the events of type T.

- **Concrete Class<? extends Object>**

Represents that any class having SubscriberComponents may receive events and any class can create any type of event.

4.4.3 UI System

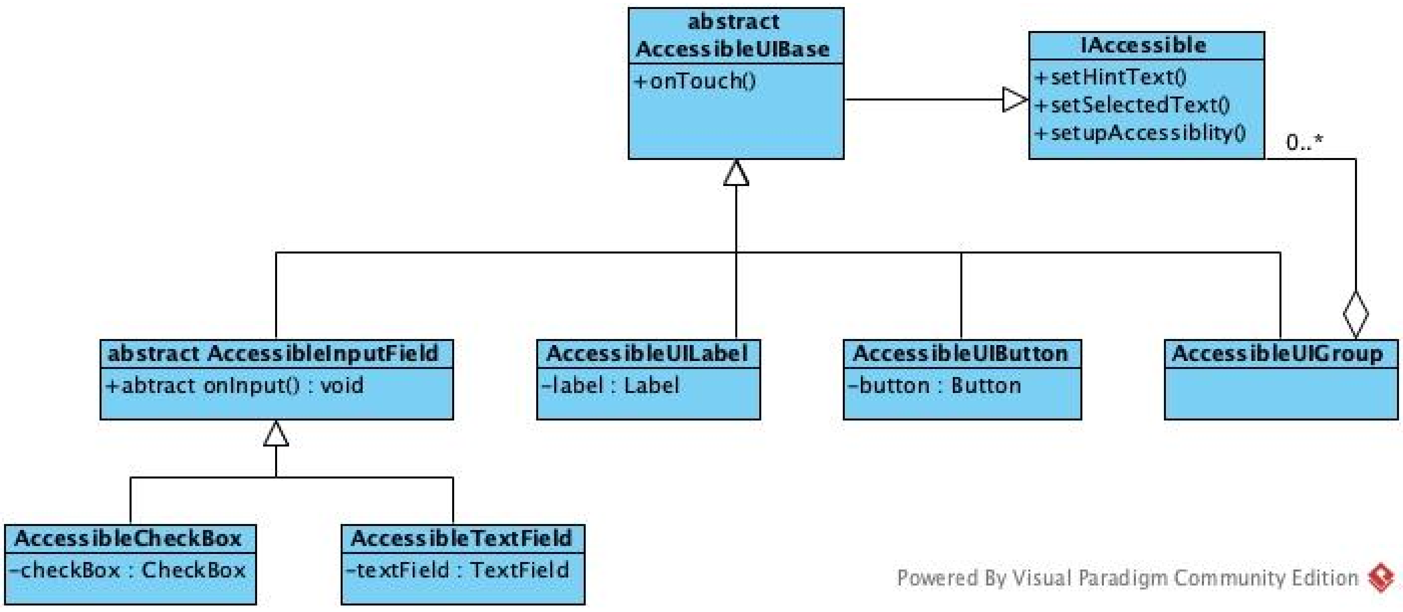

Figure 10. UI System Low Level Design.

- **IAccessible**

Methods:

- public void setHintText(): Receives feedback event. Checks if the system can play the feedback. Else, adds the feedback request to the queue or interrupts the current feedback according to the request.

- public void setSelectedText(): Checks if the android system is available for playing sound.

- public void setupAccessibility(): Receives UI events and calls requested commands.

- **Abstract AccessibleUIBase**

Methods:

- public void onTouch(): Starts accessibility functionalities upon touch.

- **Abstract AccessibleInputField**

Methods:

- public void onInput(): Gets the input from the accessible ui element and makes the relevant input event for the command center.

- **AccessibleCheckBox**

Base class for accessible generic checkbox. The engineer may use this class to create concrete checkboxes to get input on a variety of functionalities.

Attributes:

- Checkbox checkbox: Checkbox ui component.

- **AccessibleTextField**

Base class for accessible generic text field. The engineer may use this class to create concrete text fields to get input on a variety of functionalities.

Attributes:

- TextField textField: TextField ui component.

- **AccessibleUILabel**

Base class for accessible generic label. The engineer may use this class to create concrete labels to navigate the user in-app.

Attributes:

- Label label: Label UI component.

- **AccessibleUIButton**

Base class for the accessible generic button. The engineer may use this class to create concrete buttons to get input on a variety of functionalities.

Attributes:

- Button button: Button UI component.

- **AccessibleUIGroup**

Groups multiple accessible elements to make them one accessible group component.

Attributes:

- IAccessible accessibleComponents: Accessible children components

4.4.4 Startup

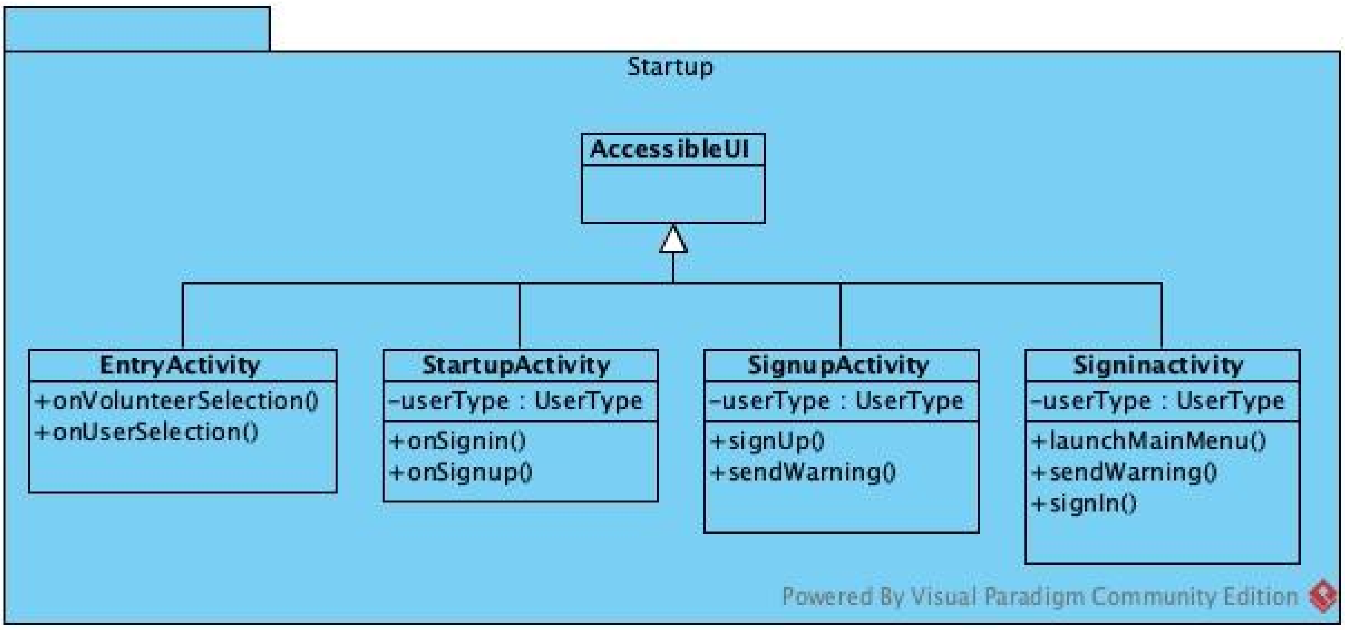

Figure 11. Startup Low Level Design.

- **EntryActivity**

Methods:

- public void onVolunteerSelection(): loads the startup activity for volunteer users.
- public void onUserSelection(): loads the startup activity for visually impaired users.

- **StartupActivity**

Attributes:

- UserType userType: type of the user, indicates volunteer or visually impaired.

Methods:

- onSignin(): loads the sign in activity for the userType.
- onSignup(): loads the sign up activity for the userType.

- **SignupActivity**

Attributes:

- UserType userType: type of the user, indicates volunteer or visually impaired.

Methods:

- public void signUp(): signs up the user.
- public void sendWarning(): sends the stated warning to the user.
- **SigninActivity**

Attributes:

- UserType userType: type of the user, indicates volunteer or visually impaired.

Methods:

- public void launchMainMenu(): launches the main menu for the userType after sign in.
- public void sendWarning(): sends the stated warning to the user.
- public void signIn(): signs in the user.

4.4.5 Navigation Activities

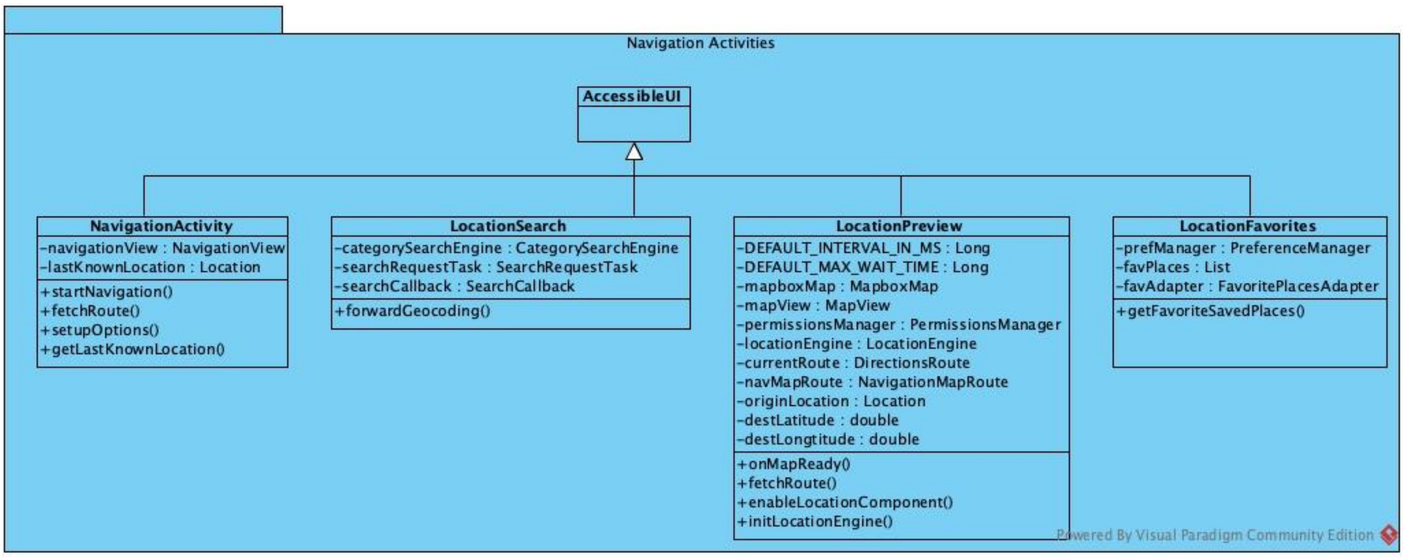
Navigation Activities
AccessibleUI
NavigationActivity
-navigationView : NavigationView
-lastKnownLocation : Location
+startNavigation()
+fetchRoute()
+setupOptions()
+getLastKnownLocation()
LocationSearch
-categorySearchEngine : CategorySearchEngine
-searchRequestTask : SearchRequestTask
-searchCallback : SearchCallback
+forwardGeocoding()
LocationPreview
-DEFAULT_INTERVAL_IN_MS : Long
-DEFAULT_MAX_WAIT_TIME : Long
-mapboxMap : MapboxMap
-mapView : MapView
-permissionsManager : PermissionsManager
-locationEngine : LocationEngine
-currentRoute : DirectionsRoute
-navMapRoute : NavigationMapRoute
-originLocation : Location
-destLatitude : double
-destLongtitude : double
+onMapReady()
+fetchRoute()
+enableLocationComponent()
+initLocationEngine()
LocationFavorites
-prefManager : PreferenceManager
-favPlaces : List
-favAdapter : FavoritePlacesAdapter
+getFavoriteSavedPlaces()
Powered By Visual Paradigm Community Edition

Figure 12. Navigation Activities Low Level Design.

- **NavigationActivity**

Attributes:

- NavigationView navigationView: displays the mapbox navigation view screen. It is the active navigation map.
- Location lastKnownLocation: last known location of the user.

Methods:

- public void startNavigation(): starts the navigation activity between the source location and destiny location.
- public void fetchRoute(): fetches the route from the source of the user location to the destiny location for the navigation activity.
- public void setupOptions(): sets up the options for the navigation MapBox activity.
- public Location getLastKnownLocation(): returns the last known location of the user.
- **LocationSearch**

Attributes:

- CategorySearchEngine categorySearchEngine: provides the list of categories to search for.
- SearchRequestTask searchRequestTask: Communicates the request for the search to the MapBox API.
- SearchCallback searchCallback: handles the callback after a search is made.

Methods:

- public void forwardGeocoding(): forwards the geocode to the MapBox API for searching a location depending on the category.
- **LocationPreview**

Attributes:

- Long DEFAULT_INTERVAL_IN_MS: interval to send requests.
- Long DEFAULT_MAX_WAIT_TIME: maximum wait time for a request.
- MapBoxMap mapBoxMap: map box API to provide a view of the locations on screen.
- MapView mapView: map screen.
- PermissionManager permissionManager: handles permissions.
- LocationEngine locationEngine: handles the location related queries of the user.
- DirectionsRoute currentRoute: encapsulates the route calculated from the source location to the target location and drws it on the graph.

- NavigationMapRoute navMapRoute: the object handling queries related to route calculations.
- Location originLocation: the location indicating the current location of the user.
- Double destLatitude: latitude of the selected target location.
- Double destLongtitude: longitude of the selected target location.

Methods:

- public void onMapReady(): callback for map box API to indicate map services are ready to be used. Used for setting up location details.
- public void fetchRoute(): fetches the route from the MapBox API once the user indicates their target location.
- public void enableLocationComponent(): enables location components by using permissions.
- public void initLocationEngine(): initializes location engine to calculate current location on the world.
- **LocationFavorites**

Attributes:

- PreferenceManager prefManager: preference manager to handle persistent data.
- List<FavPlace> favoritePlaces: a list of saved places indicated by the user.
- FavoritePlacesAdapter favAdapter: adapter to convert favorite place object into an UI representative.

Methods:

- public void getFavoriteSavedPlaces(): returns the list of favorite places of the user and displays.

4.4.6 Live Activities

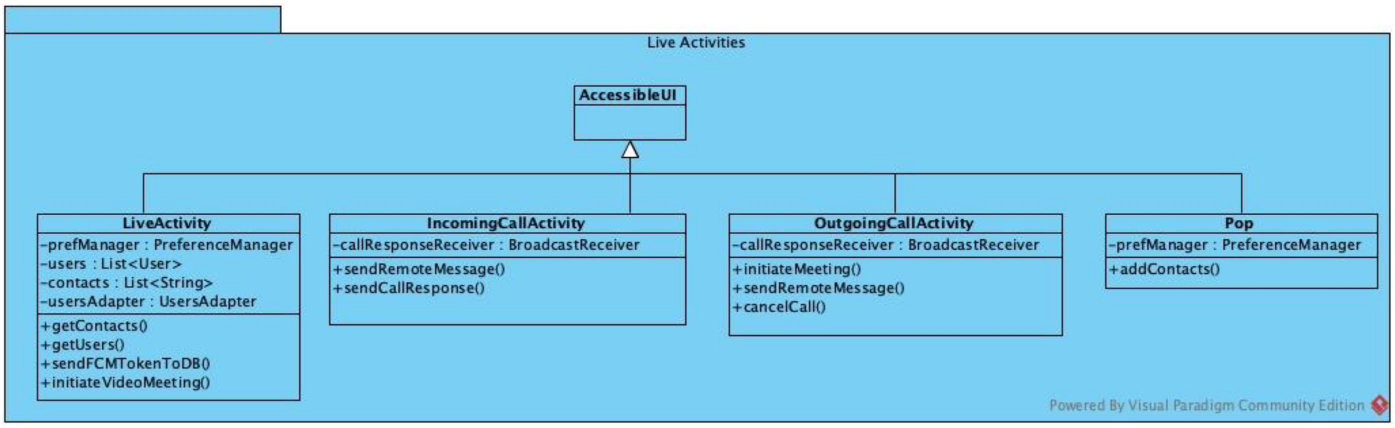

Figure 13. Live Activities Low Level Design.

- **LiveActivity**

Attributes:

- PreferenceManager prefManager: preference manager to handle persistent data.
- List<User> users: list of users that are friends of the user.
- List<String> contacts: list of the names of the friends represented as saved contacts.
- UsersAdapter usersAdapter: Adapter to display a user on UI.

Methods:

- public void getContacts(): initializes the contacts of the user.
- public void getUsers(): initializes the users saved as friends in the current user.
- public void sendFCMTokenToDB(): sends session details to the database.
- public void initiateVideoMeeting(): initiates the video meeting between an user and this user.

- **IncomingCallActivity**

Attributes:

- BroadcastReceiver callResponseReceiver: broadcasts and handles call requests.

Methods:

- public void sendRemoteMessage(): sends a call request to a JitSi user.
- public void sendCallResponse(): sends a response to the call request of a JitSi user.
- **OutgoingCallActivity**

Attributes:

- BroadcastReceiver callResponseReceiver: broadcasts and handles call requests.

Methods:

- public void sendRemoteMessage(): sends a call request to a JitSi user.
- public void cancelCall(): cancels a call request.
- public void initiateMeeting(): initiates the meeting once the callee accepts the request.
- **Pop**

Attributes:

- PreferenceManager prefManager: preference manager to handle persistent data.

Methods:

- public void addContacts(): adds a user as a contact.

4.4.7 Detection Activities

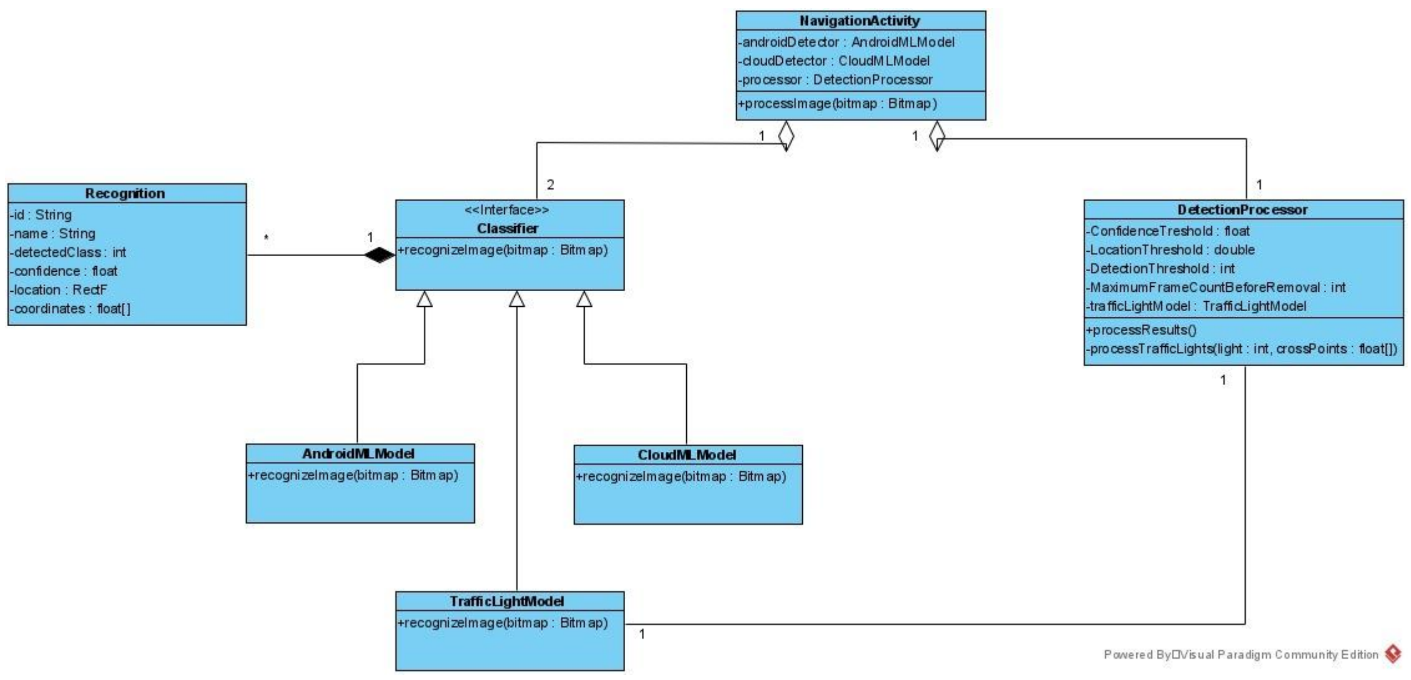
NavigationActivity
-androidDetector : AndroidMLModel
-cloudDetector : CloudMLModel
-processor : DetectionProcessor
+processImage(bitmap : Bitmap)
Recognition
-id : String
-name : String
-detectedClass : int
-confidence : float
-location : RectF
-coordinates : float[]
<<Interface>>
Classifier
+recognizeImage(bitmap : Bitmap)
DetectionProcessor
-ConfidenceTreshold : float
-LocationThreshold : double
-DetectionThreshold : int
-MaximumFrameCountBeforeRemoval : int
-trafficLightModel : TrafficLightModel
+processResults()
-processTrafficLights(light : int, crossPoints : float[])
AndroidMLModel
+recognizeImage(bitmap : Bitmap)
CloudMLModel
+recognizeImage(bitmap : Bitmap)
TrafficLightModel
+recognizeImage(bitmap : Bitmap)
1
1
2
1
*
1
1
1
Powered By Visual Paradigm Community Edition

Figure 14. Detection Activities Low Level Design.

- **Recognition**

Attributes:

- String id: unique id.
- String name: name of the recognize class.
- int detectedClass: class of the detection.
- float confidence: confidence level.
- RectF location: location in the image defined by rectangles.
- float[] coordinates: coordinates of the detected class.

- **Classifier : Interface**

Provides an interface for all classes that contracts them to recognizeImage(Bitmap bitmap) to process images.

- **AndroidMLModel**

Methods:

- public void recognizeImage(Bitmap bitmap): processes the image for local detection support.

- **CloudMLModel**

Methods:

- public void recognizeImage(Bitmap bitmap): sends a request to the cloud model that processes the image and returns the outcome.

- **TrafficLightModel**

Methods:

- public void recognizeImage(Bitmap bitmap): recognizes the traffic lights according to the customized traffic lights model.

- **DetectionProcessor**

Attributes:

- float confidenceThreshold: threshold that drops output when certain confidence is not met.
- double locationThreshold: the threshold for locations once they are found to see if they meet location criteria.
- int detectionThreshold: the threshold for detection that drops certain findings that does not meet this threshold.
- int maximumFrameCountBeforeRemoval: the frame count to remove before processing.
- TrafficLightModel trafficLightModel: the model that processes images for traffic light detection.

Methods:

- public void processResults(): processes each model's result according to the models.
- public void processTrafficLights(int light, float[] crosspoints): processes traffic lights found in the image, determines their current status.

- **NavigationActivity**

Attributes:

- AndroidMLDetector androidDetector: android ML detector model. Used as an API.
- CloudMLDetector cloudDetector: cloud ml detector model. Used as an API.

- DetectionProcessor processor: processor that handles traffic light detection and detection criteria.

Methods:

- public void processImage(Bitmap bitmap): processes the given image for each model and processor.

5. Testing Details

We integrated manual testing versus continuous testing. Although an automated testing environment indicates that the development pipeline is more reliable, our project did not require such complex testing platforms. This is because OverSeer is not a large-scale project. OverSeer is developed by five developers. As a result, each developer is responsible for independent sub-systems. This indicates that each developer could work freely without breaking other subsystems in OverSeer. Although manual testing is more time consuming than continuous testing, creating such a complex continuous testing environment would have been an extravagant task. Therefore, our testing strategy is based on manual testing. However, we plan to include continuous testing for the maintenance of our project in the long scope [5].

5.1 Testing Strategies Adopted

5.1.1 Unit Testing

Our core classes are tested with JUnit. Unit tests allowed us to observe if the classes implement their expected behaviors.

5.1.2 Functional Testing

Each requirement listed in the functional requirements section is tested for acceptance. This ensures that OverSeer delivers the promised functionality to the users. This testing allowed us to find the right UX for our users while providing comfortable services.

5.1.3 Accessibility Testing

This is one of the most important test strategies we followed. Our application is fully accessible for the visually impaired. Therefore, OverSeer has to meet certain accessibility standards. We ensure this by using this testing to test accessibility standards described in the official guidelines [6].

5.1.4 Non-Functional Testing

OverSeer must meet certain technical standards described in non functional requirements such as performance and integrity. As a result, non-functional testing is performed. Simulations are created to profile the performance of OverSeer to determine if a certain non-functional requirement is satisfied. If the results profiled prove that certain areas are under-performing, we optimize such non-functional requirements. This allowed us to avoid premature optimization while providing robust services.

5.1.5 Integration Testing

OverSeer consists of different services provided to the users. This indicates that services are independent of each other. Each service is a distinct subsystem. This allows the developers to implement their subsystems in parallel as subsystems do not have any data flow in between. However, users have to be able to use services in any sequence. Consequently, we performed integrations tests to observe if different services could be run and deployed without disturbing their services.

6. Maintenance Plan and Details

We have three separate plans considering OverSeer. These plans include database, server and application maintenance. Our maintenance strategy includes building a DevOps pipeline while monitoring application performance [7].

6.1 Database Maintenance

OverSeer uses Firebase DB to keep records of the users signed in or signed up. Therefore, it is very important to maintain the database in order to keep it updated. This would make OverSeer services more robust as the data will be kept in a clean database. To accomplish this, developers are going to follow the guidelines on database maintenance including regular backups, optimized queries, data integrity, efficient indexing while adapting change management [8].

6.2 Server Maintenance

The servers in OverSeer are responsible for processing data by custom machine learning models. Therefore, scalability is the key target of the maintenance for the servers. Developers will continuously monitor the performance of the server, its response time and ability to deal with simultaneous requests. Moreover, since the servers contain the machine learning model, models would be maintained as well. Developers would continue training the models, simulating its performance to ensure that the users get a robust service while they are using OverSeer [9].

6.3 Application Maintenance

For the maintenance of our project, developers realize the importance of change management. As a result, today's feature code will be tomorrow's legacy code. In order to deal with the change factor in software, developers will implement a DevOps pipeline to adopt continuous integration and continuous delivery. This would allow developers to integrate continuous redundancy tests.

Developers have used Git for version control and Github Issues for bug management. Furthermore, developers will switch to Jira to track issues and keep a detailed log of bug management history while dealing with change. Therefore, while Git/Github are to be used for source control, Jira will be used to maintain bugs and issues. Moreover, developers will follow the application maintenance best practises while following user feedback. These best practises include updating software to adapt to its environment while dealing with change [10].

7. Other Project Elements

7.1. Consideration of Various Factors in Engineering Design

We have considered various factors in order to make the project valuable and flawless. The first main consideration is about the collaboration of the team. The tasks, both engineering and managing tasks, were divided equally among the group members according to interests and knowledge of the members. The managing tasks were divided into four different parts including arranging meetings, managing engineering jobs, keeping track of reports, communicating with external actors. This allowed us to divide and conquer problems arising in OverSeer. We used Github issues to track our progress and problems encountered throughout CS491/CS492. This allowed us to be transparent in OverSeer life-cycle management, allowing us to design a robust system.

The second main consideration of the project is the user interface which is suited to the visually impaired users. The user interface must be easy to use and also accessible. Therefore, we engineered our design to be controlled by voice commands. However, this was not a realizable objective for the scope of OverSeer. Consequently, we focused on accessibility guidelines for the visually impaired users. Furthermore, we ensured that the results of user actions are reported by using the voice functionality. Last but not least, our engineering design is optimized for accuracy, response time and UX. Therefore, non-functional requirements have an important impact on our engineering design.

The security of the visually impaired people and the volunteers is another topic we considered in our engineering design. Some bad-mannered volunteers can mislead or make fun of the visually impaired user. In order to avoid these issues, the live support system includes a report function. Hence, the users can report any bad behavior of the volunteer and the volunteer can be banned from the application. To ensure such actions have impactful consequences, banned users must not be able to create another account in OverSeer. Therefore, user registration includes unique phone numbers assigned to each user. When a user is banned, their phone number is blocked from accessing OverSeer. More precautions could have been taken such as providing IP ban or hardware ban. However, for the scope of OverSeer, this design was found to be resourceful.

Financial problems are the last main consideration of the project. In order to make a project which has an affordable cost for both the users and the developers, the platform of the application is determined as mobile devices. Also, computer vision is implemented in the application to detect obstacles instead of external sensors. This makes the engineering design of OverSeer both affordable and available for everyone.

7.2. Ethics and Professional Responsibilities

Our topic is about visually impaired users, so we have a lot of ethical and professional responsibilities. One of the biggest ethical responsibilities we have is making our target audience not to feel separated from society. To fulfill this responsibility, we designate requirements accordingly that comply with accessibility guidelines. OverSeer allows visually impaired users to navigate freely and get help whenever they require. This makes visually impaired users live in a society where they can join as individuals.

Developers have dealt with significant professional and ethical issues. Most importantly, the live support system can be abused by malicious people who do not intend to help the user, and we have implemented several plans to deal with this issue. Our precautions include authentication by unique phone numbers and verified supporters. As developers, it is our responsibility to protect users' data and we also need to show that their data will not be used outside their permission or in any malicious way. Another professional issue we have dealt with is having a very accurate object detection system. Our system should make almost no mistake when guiding the user through their travel. Missing an obstacle on a sidewalk that has the potential to hurt the user or labeling a non-obstacle thing as an obstacle that would make the user make unnecessary movements would both be unethical and unprofessional. Similarly, in our navigation system, we should calculate the correct path while making the path as efficient as possible.

In short, we aimed for our application to be the guiding eyes of our visually impaired users. This bears a lot of ethical and professional responsibility. Users must feel secure when using the live support system, they should not be having to do any unnecessary actions while using the application and lastly, they should not feel like they have visual impairment while using our application. Ultimately, we have fulfilled these ethical and professional responsibilities by following best-principle guidelines in accessibility and security.

7.3. Judgements and Impacts to Various Contexts

Throughout the development and design life-cycle of OverSeer, developers have made several judgements that had an impact to various contexts:

Judgement Description:	OverSeer must be accessible. OverSeer is designed to help visually impaired users in their daily life. Therefore, accessibility influenced our various decisions.	
	Impact Level	Impact Description
Impact in Global Context	10/10	The accessibility will ensure that users worldwide can use OverSeer.
Impact in Economic Context	3/10	The accessibility features use Talkback feature in Android systems. Users that do not have an Android phone would not benefit from the services provided by OverSeer.
Impact in Environmental Context	10/10	OverSeer provides voice feedback to users. This increases accessibility in a

		way to warn users if there are any obstacles in their path. Thus, environmental safety is increased for the users.
Impact in Societal Context	10/10	The aim of the application is to ease the daily challenges of visually impaired users. With accessibility, we can reach visually impaired users to help them join society.

Table 1. Accessibility Judgement.

Judgement Description:	OverSer must be secure. There are volunteers that can help the visually impaired users. There can be bad-mannered volunteers. Moreover, both users provide their data to our servers. Security must be ensured. Therefore, security influenced our decisions in various ways.	
	Impact Level	Impact Description
Impact in Global Context	10/10	OverSeer has to provide data security and ensure that visually impaired users are protected from ill-mannered volunteers. Security would allow OverSeer to provide trustworthy services to worldwide users.
Impact in Economic Context	9/10	Since security is a key objective of OverSeer, we used reliable Cloud providers and APIs which have a certain cost. We had to consider various options to choose the best security / cost ratio efficient service.
Impact in Environmental	1/10	Since we are using Cloud providers and APIs,

Context		although our carbon-footprint is high, for the scope of OverSeer, we did not consider security on an environmental sustainability level.
Impact in Societal Context	10/10	Users want secure applications. When they hear an app’s security is breached and their data is stolen, they will not trust the application again. Thus, we prioritized security in OverSeer to have a good impact on a societal level to build trust.

Table 2. Security Judgement.

Judgement Description:	OverSeer must be robust. If certain services crush or when OverSeer does not provide the service it promises when users need it, it will not be successful. Therefore, robustness influenced our decisions in distinct contexts.	
	Impact Level	Impact Description
Impact in Global Context	10/10	Without any robust services, we cannot realize OverSeer. To reach users worldwide, we had to consider robustness in a global context when users worldwide used our services in different conditions.
Impact in Economic Context	8/10	To provide robust services, we had to consider the impact of our solution in an economic context. Most users may not have the best Internet provider. Moreover, their hardware may be old. As a result,

		when designing OverSeer and making decisions, we fairly considered this context.
Impact in Environmental Context	6/10	Users worldwide live in different environments. The machine learning model must detect obstacles regardless of the environment. Thus, to provide robust obstacle detection, we considered this context.
Impact in Societal Context	10/10	If OverSeer does not provide reliable services, the society in which OverSeer operates will reject it. Thus, we had to consider societal context when making decisions on OverSeer.

Table 3. Robustness Judgement.

7.4 Teamwork Details

After we formed our group, each member had been assigned a role on project management. According to these roles, each member had to be effective at:

- Talha Şen: Arranging meetings depending on availability of each member.
- Hakan Sivuk: Managing the project and checking on work done.
- Ahmet Berk Eren: Keeping a track of reports to work on.
- Cevat Aykan Sevinç: Communication with external actors.
- Yusuf Nevzat Şengün: Helping with the management of the project and checking on work done.

This role distribution allowed us to conquer distinct requirements for our group to sustain. However, this distribution does not imply that one was not responsible for other aspects of the project. The assignment was made to ensure that there would be always a member to perceive a responsibility in depth. Every member can help other members and take an active role for each responsibility. Essentially, this is what we have accomplished while developing OverSeer as a group. This would be elaborated on further in the following subsections.

A Google Drive folder that is shared for every member was created. This way, every deliverable other than the base code could be shared between members. This allowed us to access and work on the same report files simultaneously while sharing diagram data such as Visual Paradigm files. This made the process of delivering reports more efficient.

We decided to use Github for storing our code-base while managing version control and have a sustainable development pipeline. We created an organization to represent our project headquarters. In this base headquarter we manage our website project and the design project. The design project allows us to have a rich base of subsystems such as live support, navigation support and object detection. We distributed these subsystems per member for researching and development. To keep track of work done, we agreed on two systems:

- **Weekly Meetings:**

To discuss our progress, we had weekly meetings using Zoom. Each meeting would have a topic for the next week so that we could keep track of our progress while building on top of it. In a way, we have followed a Scrum methodology while working on OverSeer. At the end of each meeting, developers were assigned tasks based on that week's sprint. During each meeting, when there are multiple available works, each work is assigned a weight out of ten by every member. Next, the average weight is calculated and the total weight of the works are determined. Finally, every member is assigned a job and the total weights distributed per member is equalized so that any member would not have an overwhelming amount of work to deliver. This allowed us to optimize our work done per member which improved our delivery rate.

- **Github Projects:**

Each member had an ongoing project. As a result, each project had different requirements and different deliverables. To track every project progress, we decided to use Github Projects. For each project, a representative project topic would be opened on the OverSeer Github repository. For each work to be done, a card was created and assigned to the responsible member. Therefore, every delivery progress could be transparently observed and if there was any problem regarding performance, necessary actions could be decided on to ensure sustainable project development. This transparency allowed us to further trust each other. As a result, we become a solid team.

Our project required us to learn new technological stacks such as React Native, Android Studio, YoloV5 and libraries for navigation, live support. That is the reason why we worked in an Agile manner using Scrum to understand each requirement and continuously build the services. As a result, we created a system to ensure that every member could properly contribute to the project while learning new technological frames.

7.4.1 Contributing and Functioning Effectively on the Team

We believe the key aspect of every software project is the team behind the curtains. We believe that we not only become a functional team, but also a team that shares a culture of inclusion, collaboration and trust. As a team, we have been working on the same projects for two years. This is the reason why we collaborated on the senior design project as the culture we shared is very important to us. Therefore, throughout the life-cycle of OverSeer, each member effectively contributed and functioned in the team in full transparency.

- **Talha Şen**

Talha, along with Hakan, was responsible for mainly in the development machine learning model and arranging meetings. Talha always did the best effort he could. Since he is working a part-time job 3 times a week, he successfully managed his time to keep up with an overwhelming amount of tasks. He has an important role in developing the necessary models that recognizes the potential dangers to the users. He not only developed such a model but also deployed it.

While working on his tasks, Talha always shared his plans with his teammates. He effectively versioned his source code to collaborate with other developers. When other members needed help or there were problems in the project, he always followed an approach that builds trust in the team. By arranging regular meetings in a team, we were able to catch up with each others' work. Without Talha, we would not have been able to develop the key components of OverSeer.

- **Hakan Sivuk**

Hakan is the founder of this project. He came up with OverSeer and other members loved the idea. Therefore, Hakan had an important role in contributing and functioning effectively on the team. Along with Talha, he worked on developing the machine learning models and deploying them to the cloud. He contributed effectively by designing the machine learning models and customising them. Moreover, Hakan has successfully motivated each team member to effectively contribute to the project. He always listened to each member's distress on any uncertainty. By picturing a reliable project development pipeline, he made the team overcome any uncertainty and effectively strengthen the trust between the members.

Hakan works as a part time machine learning engineer at Migros ecommerce. He successfully leveraged his knowledge to the OverSeer project. He integrated the model on Amazon Sagemaker while designing the overall machine learning pipeline with Talha Şen. Together, they arranged meetings with the supervisor to criticize their progress and get the best feedback on the model. By always looking out for his teammates, Hakan created a work environment where each member can happily contribute and function.

- **Ahmet Berk Eren**

Ahmet was responsible mainly for the live support service in OverSeer. He researched, analyzed, and designed the necessary subsystems for live stream. He found that the JitSi API can provide the necessary requirements described for OverSeer. Moreover, he not only implemented the live stream functionality in React Native, but also re-implemented in Android Studio as the environment of the project changed. Ahmet effectively communicated with the team while developing the live support subsystem. He has been fully transparent on the work he has accomplished.

Ahmet always attended meetings. Ahmet also followed the deadlines for each deliverable for the project. As a result, we were able to effectively plan our roadmap and sprints. He has always delivered quality code. He did his best to contribute and function effectively for OverSeer while working a part time job. Members consider themselves very lucky to have such a motivated member who is independent and a self learner.

- **Cevat Aykan Sevinç**

Cevat was responsible mainly for developing the UI, accessibility and core systems in OverSeer. Firstly, Cevat researched if OverSeer could be controlled with voice only to ensure a smooth UX on accessibility. However, he analyzed that such a system would not fit the scope of OverSeer. Therefore, Cevat researched, analyzed and designed the accessibility in OverSeer using accessible UI best-practises. Furthermore, Cevat implemented a global event system which allowed many independent systems in OverSeer to communicate while being decoupled.

Throughout the project, Cevat communicated with external actors to arrange meetings or provide the necessary updates on the project. He always tried to ensure the logistics were met in time. Next, Cevat also helped with the deliverable reports. Thus, he effectively functioned and contributed to OverSeer while working a part time job.

- **Yusuf Nevzat Şengün**

Yusuf was responsible mainly for developing the navigation service in OverSeer. He researched, analyzed and designed how navigation could be implemented in OverSeer. As a result, he integrated the MapBox API which provides the navigation in OverSeer. He was fully transparent on implementing the navigation functionality. He communicated, tested and solved issues arising in navigation successfully.

Yusuf was also responsible for helping with the management of the project. He effectively tracked the project roadmap, analyzed any risks and contributed to the potential solutions in case of any issues arising in the development pipeline. He has managed to accomplish his responsibilities successfully while working a part time job. By having amazing time management skills, he contributed and functioned effectively in OverSeer.

7.4.2 Helping Creating a Collaborative and Inclusive Environment

Each member followed a strong understanding of egoless programming [11]. Since we have been developing projects together for two years, it has not been difficult for us to create such an environment. Although there is a clear division of work among the group members, we had a collaborative environment where people may contribute to other tasks that are not assigned to them originally. For instance, Cevat helped the machine learning team in terms of forming a custom dataset. Also, Yusuf helped Ahmet Berk while he was trying to find a proper live support API for the project. Apart from these, each member tried to contribute to the report writing process as much as possible. This collaborative and inclusive environment inside the group allowed us to deal with tasks easier and optimized the project roadmap for the better.

- **Talha Şen**

Talha always included others in the discussions related to the machine learning model. When Hakan and he needed samples for the model, they allowed other members to label data in a collaborative environment. Moreover, Talha never judged a member and always listened to others respectfully in an inclusive manner. He always attended meetings to collaborate with others on idea design & idea discussions.

- **Hakan Sivuk**

Hakan, the founder of the project, always ensured that the working environment is collaborative and inclusive. He has always invited others to join discussions on machine learning. He always shared their plans with other members. He always listened to other members to solve any problems and never judged a person. He effectively communicated his envision for the project with other members to create a culture of collaboration and inclusion.

- **Ahmet Berk Eren**

Ahmet always shared his problems arising in live stream and allowed others to collaborate with him on solving these issues. By combining his skills with other developers' debugging skills, he created robust systems. He shared his design and findings with other developers consistently, allowing them to contribute to the design & implementation process of the live support subsystem. He never judged a person and discriminated against a member. He shares a great part in creating the collaborative and inclusive culture in the project

- **Cevat Aykan Sevinç**

Cevat has always tried to inform others on his communications with external actors. He has effectively communicated the required logistics by sharing a culture of inclusion and collaboration. By allowing others to share their ideas on logistics, he made sure that the end result does satisfy each member, thus, building trust in the team. He always tried to collaborate with other developers and help them solve problems encountered during the development of OverSeer.

- **Yusuf Nevzat Şengün**

Yusuf, while working on navigation systems, always included others for collaboration. He helped create such an environment by allowing others to join the design decisions made on the navigation system. By sharing his progress in a transparent manner, Yusuf allowed other developers to be involved in the navigation subsystem. This allowed the navigation system to be free of bugs while division of tasks for optimization of the work done. He had an important role in creating the culture of collaboration and inclusion in OverSeer.

7.4.3 Taking Lead Role and Sharing Leadership on The Team

Normally, the formal leader of the project is Hakan. However, taking the lead role doesn't mean handling all the tasks or making decisions by oneself. Instead of it, we shared leadership roles on the team for different parts of the project. For instance, Ahmet is taking a lead role for reports and deadlines. He prepares reports, divides the tasks when it is necessary, and checks the process. But each member contributes to reports equally. Also, as it was mentioned above, different tasks have different leaders. For instance, Talha is responsible for the integration of machine learning models into the application. He did the research and prepared the initial design. However, he also asked the opinions of the group about some parts and also assigned some tasks to the members. In that regard, taking the lead role is a way of focusing on different parts of the project better and directing the group more consciously. This strategy allowed us to divide and conquer various problems arising in the development of OverSeer.

- **Talha Şen**

Lead the team in developing and integrating machine learning models along with Hakan while focusing on different aspects. He effectively planned the tasks and distributed certain work to team members. Moreover, he arranged critical team meetings for various discussions.

- **Hakan Sivuk**

Lead the team in developing and integrating machine learning models along with Talha while focusing on different aspects. Since Hakan is also the founder of the project, he closely tracked each progress done by using a transparent project roadmap.

- **Ahmet Berk Eren**

Lead the team in developing and integrating live support functionality while keeping track of deliverables. He designed the live support subsystem and divided tasks between team members. He also helped in the design process of the navigation subsystem.

- **Cevat Aykan Sevinç**

Lead the team in developing and integrating accessibility and core systems used by every subsystem in OverSeer. He also managed logistics in a way to meet certain quality assurance by the deadlines. He occasionally helped the design and implementation in other subsystems in OverSeer. He also took leadership on communicating with external actors.

- **Yusuf Nevzat Şengün**

Lead the team in developing and integrating live support functionality while keeping track of deliverables. He designed the live support subsystem and divided tasks between team members. He also helped in the design process of the navigation subsystem.

7.4.4 Meeting Objectives

Here is the updated work package table defined in our analysis report:

WP#	Work package title	Leader	Members involved
WP1	Data Science - Model Training, Object Detection	Hakan Sivuk	Talha Şen
WP2	Android Studio - Navigation	Yusuf Nevzat Şengün	Ahmet Berk Eren, Cevat Aykan Sevinç
WP3	Accessibility, UI	Cevat Aykan Sevinç	Ahmet Berk Eren, Yusuf Nevzat Şengün
WP4	Logistics	Ahmet Berk Eren	Hakan Sivuk, Talha Şen, Yusuf Nevzat Şengün, Cevat Aykan Sevinç
WP5	Android Studio - Live Support	Ahmet Berk Eren	Yusuf Nevzat Şengün
WP6	Data Science - Model Format Transformation and Deployment	Talha Şen	Hakan Sivuk, Yusuf Nevzat Şengün

Table 3: List of work packages

- **WP1 Data Science - Model Training, Object Detection**

The objective of this work package was to develop and train a model that can detect obstacles and traffic lights for the visually impaired person. According to the analysis report, two deliverables were required: a custom dataset and the trained Yolov5 model. Both of these deliverables have been met and deployed. Custom dataset was created using a team effort. The model was trained by Hakan and Talha. While they progressed on this work package, it was realized that there were new requirements to overcome. This meant that the deadline of the deliverable would be shifted for two months. Nevertheless, this work package is fully realized with updated functional requirements, complying to the non-functional requirements.

- **WP2 Android Studio - Navigation**

The objective of this work package was to provide navigation support for guiding users. Users had to be able to set their destination and get step by step voice commands until they arrived at their destination.

This work package had three deliverables: navigation system implemented in React Native, a backup mechanism, voice feedback during the navigation. These objectives have been met but they have changed according to our design. Firstly, the navigation system is implemented in Android Studio as the development environment has been changed. Secondly, a backup system has not been implemented yet. It will be added in the future. Thirdly, the navigation system regularly updates the user on their route. As a result, the objectives, excluding a backup system, have been met while dealing with change.

- **WP3 Accessibility, UI**

The original objective of this work package was to include voice recognition to the app. However, as the requirements of this system were researched, it was understood that such a system required a complex technical investment that over-scoped OverSeer. As a result, the objectives had been changed to implement a fully accessible UI according to the accessibility best-principles. This work package is fully realized and the updated objectives have been met.

- **WP4 Logistics**

Originally, this work package only included reports. However, the logistics had to include new deliverables. As a result, each deliverable is boxed under logistics. Our senior design project required several deliverable reports documenting the life-cycle. These reports include: project specification report, analysis report, high-level design report, low-level design report, final report, user manual, a poster, a demo video, three presentations to the expert and jury members. Therefore, the logistics to deliver such work has been ensured and each work is delivered on time. As a result, the objectives of the logistics have been met before the deadlines successfully.

- **WP5 Android Studio - Live Support**

The objectives of this work package include implementing a streaming service where a volunteer can connect to the front camera of the visually impaired person. This work package had two deliverables. First is a system that connects a volunteer and a visually impaired user, regardless of whether they are friends or not. Secondly, the report system that bans bad mannered volunteers. While the first deliverable is met as an objective, the second deliverable is left for future work due to the time constraints we faced in the development of OverSeer.

- **WP6 Data Science - Model Format Transformation and Deployment**

The original objective of this work package included transforming the trained model to Tensorflow Lite so that it can be used efficiently in the application. However, as the platform was changed to Android Studio, it was no longer needed. Thus, the requirements changed in a way that the model was deployed using Amazon Sagemaker. Therefore, the updated objectives of this work package is fully met while delivering deployed models.

7.5 New Knowledge Acquired and Applied

For our object detection model, we decided to use yolov5 and we did not have any prior knowledge about it before. Yolo (You Only Look Once) is an open-source real-time object detection system. We needed to detect objects throughout the user's navigation accurately and fast, and this system fit our requirement. The strategy we planned to apply while learning this was that we were preparing many training datasets to run on this model. While doing this, we planned to change hyperparameters on the model to see if our results improve. When we found a model that satisfied our object detection requirements, we planned to use it in our final application. This allowed us to train a robust machine learning model although we had not had any experience before.

For the application, we decided to use React Native as it was cross platform. Moreover, it was supported by rich libraries of machine learning, navigation and cloud services. As a result, we learnt React Native. However, after the progress demo, we realized that React Native would not provide the best environment when the model would be deployed. This made us change our environment to Android Studio. Consequently, we had to start over. However, we had a strong design and good understanding of the requirements necessary to realize OverSeer. Thus, we quickly learnt Android Studio to leverage our previous experience from React Native. We know that most are not capable of doing such a radical and risky choice in the middle of the semester. However, our adaptation to change by quickly determining problems without them being arising led us to self learn required solutions. Certainly, Bilkent has an important role in this as all of us are senior computer science students who have acquired a set of soft and hard skills throughout its academic environment.

Developers had not had any experience on accessibility before. OverSeer allowed them to learn the daily challenges of the visually impaired people. Thus, developers researched contemporary approaches on accessibility for the visually impaired. Next, developers implemented accessibility best-principles to OverSeer to make it fully accessible.

For the backend part of the project, we decided to use Node.js since it had a good compatibility with React Native, and also it had a rich library and community. However, we decided to use Sagemaker API that uses Python3 to implement backend. To accomplish this, developers first read the documentation of Amazon SageMaker. Next, they deployed their machine learning model to it. Finally, developers learnt how to communicate using the API. Step by step, using a top-down approach, we divided and conquered most of the problems we had not had any experience before by using this strategy. Documentations were one of the biggest learning sources we benefited from.

To integrate a database to our application, we evaluated our choices after we switched to the Android Studio. We found that Firebase Database could be nicely integrated with Android Studio as both are powered by Google. Thus, we created a database schema to support OverSeer users while they are using the application. We applied the skills we gained in CS353 as the developers were taking this course at the same time they started developing OverSeer.

Navigation functionality was possible by using MapBox API. Developers scanned existing libraries. Since there were multiple ones, a chart was made to display the advantages and disadvantages each API provided. As a result, MapBox was seen as a better fit to the requirements needed in OverSeer. Next, developers studied its documentation to quickly prototype it. After this, developers were familiar with the API so that they implemented the required functionalities. Moreover, the live support functionality shares a very similar story. In its case, the API is Jitsi and developers simultaneously studied its documentation while studying MapBox. As a result, both APIs were understood in depth to be customized as needed in OverSeer.

8. Conclusion and Future Work

OverSeer is a project that can be extended. This makes the future work an important aspect of its life-cycle. Firstly, as described in maintenance, our future work consists of building a DevOps pipeline to provide continuous testing. This would allow us to deal with change by ensuring redundancy tests on the legacy code. Moreover, there are some objectives in work packages that have been omitted. The future work involves implementing such objectives to increase the functionality of OverSeer. Moreover, we plan to continuously improve the machine learning models and the pipeline. We have an architecture that is open to extension and easy to maintain. We believe that this creates an advantage for the future work planned for OverSeer.

Developing OverSeer was both fun and challenging. In the limited timespan we had, we have suffered important problems. These problems include external factors such as the other course-loads we had, keeping up with our part time jobs, dealing with the inability to socialize. However, OverSeer allowed us to become more than a team, members sacrificing themselves to help others, to overcome such difficulties while sharing a culture of collaboration and inclusion. We consider ourselves very lucky to find each other, since as a team, we learnt the most from each other while having exceptional time-management skills. The OverSeer project allowed us to be greater, implement the theoretical knowledge we have accumulated over the years, and made our group a family. We are proud of the work done.

9. Glossary

- Android Studio: A framework to create Android applications.
- Firebase: Database services provided by Google.
- JitSi: An API that allows live video streaming.
- JUnit: Test framework available in Java.
- MapBox: An API that allows real time navigation.
- Talkback: Accessibility settings that can be toggled on in Android systems to read out application UI.
- Sagemaker API: Amazon Cloud's machine learning platform.

10. References

[1] "Build More Accessible Apps" *developer.android*, [Online]. Available: https://developer.android.com/guide/topics/ui/accessibility. [Accessed: 25.04.2021].

[2] "What are Standards," *library.rose*, Feb. 01, 2021. [Online]. Available: https://library.rose-hulman.edu/c.php?g=104543&p=888557. [Accessed: 08.02.2021].

[3] "ISO Standards," *iso*, [Online]. Available: https://www.iso.org/standards-catalogue/browse-by-ics.html. [Accessed: 08.02.2021].

[4] "Camera2," *developer.android*, [Online]. Available: https://developer.android.com/reference/android/hardware/camera2/package-summary. [Accessed: 08.02.2021].

[5] S. Pittet, "The Different Types of Software Testing", *atlassian*, [Online]. Available: https://www.atlassian.com/continuous-delivery/software-testing/types-of-software-testing. [Accessed: 25.04.2021].

[6] “Accessibility Principles”, *developer.android,* [Online]. Available: https://developer.android.com/guide/topics/ui/accessibility/principles. [Accessed: 25.04.2021].

[7] “Application Performance MAnagement”, *applicationperformancemanagement,* [Online]. Available: https://www.applicationperformancemanagement.org/software/monitoring-software/. [Accessed: 25.04.2021].

[8] G. A. Larsen, “Top Ten Mistakes When Building and Maintaining a Database”, *databasejournal,* Oct. 13, 2010. [Online]. Available: https://www.databasejournal.com/features/mssql/article.php/3906986/Top-10-Mistakes-When-Building-and-Maintaining-a-Database.htm. [Accessed: 25.04.2021].

[9] B. Dobran, “Top Fifteen Point Server Maintanence Checklist IT Pros Depend On”, *phoenixnap,* Mar. 21, 2019. [Online]. Available: https://phoenixnap.com/blog/server-maintenance-checklist. [Accessed: 25.04.2021].

[10] Orchestrate Technologies, “Application Maintanence and Support, Best Practises” *medium,* Nov. 30, 2015. [Online]. Available: https://medium.com/@Orchestrate/application-maintenance-and-support-best-practices-bef7bad780cc. [Accessed: 25.04.2021].

[11] J. Atwood, “The Ten Commandments of Egoless Programming”, *codinghorror,* [Online]. Available: https://blog.codinghorror.com/the-ten-commandments-of-egoless-programming/.

[12] *Object-Oriented Software Engineering*, Using UML, Patterns, and Java, 2nd Edition, by Bernd Bruegge and Allen H. Dutoit, Prentice-Hall, 2004, ISBN: 0-13-047110-0.