Bilkent University
Department of Computer Engineering

Senior Design Project

Group: T2317 - RoadVisor

Final Report

Group Members:
Ahmet Faruk Ulutaş - 21803717 - faruk.ulutas@ug.bilkent.edu.tr
Ammaar Iftikhar - 21901257 - ammaar.iftikhar@ug.bilkent.edu.tr
Arda İçöz - 21901443 - arda.icoz@ug.bilkent.edu.tr
Emin Berke Ay - 21901780 - berke.ay@ug.bilkent.edu.tr
Nurettin Onur Vural - 21902330 - onur.vural@ug.bilkent.edu.tr

Supervisor:
Asst. Prof. Dr. Hamdi Dibeklioğlu

Innovation Expert:
Cem Çimenbiçer

Course Instructors:
Erhan Dolak
Tağmaç Topal

19.05.2023

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Senior Design Project course CS492.

Table of Contents

1. Introduction

With the increasing number of technological features in automobiles, driving is becoming easier day by day. Computers installed in automobiles assist drivers in driving and provide them with a better road experience. Technologies such as built-in navigation, road sign detection, crash assist system, and pedestrian detection system help drivers while driving and they also increase the safety of our roads. Specifically, built-in navigation systems or navigation applications that the drivers can use by utilizing Apple CarPlay or Android Auto help drivers navigate better in areas they are not familiar with. In addition, navigation systems also help drivers avoid congested areas and can help them save fuel and help the environment as well as save time. However, these technologies are only available in newer cars and many of them do not even come pre-installed, the customers have to pay extra to purchase that option. Therefore, many drivers cannot enjoy the benefits of having such technologies to make driving easier.

While there are many applications and solutions that attempt to solve this problem, there are not many that simultaneously help the driver with navigation and provide the technologies that the newer generation of cars have. The existing applications focus on either one of these problems and many fail to completely solve the problem. For instance, information about processing real-time information for navigation is not done by many applications that are already on the market, and the ones that have them fail to address the problem of lack of technologies in the older generation of cars. RoadVisor is designed to bridge the gap between the user's perspective and the capabilities of the application itself. By leveraging machine learning and augmented reality, RoadVisor offers real-time assistance to drivers, enabling them to access advanced technologies found in newer-generation vehicles, even in older-generation cars. The integration of machine learning enables RoadVisor to analyze and process data in real time, providing valuable insights and information to the driver. By utilizing augmented reality, RoadVisor enhances the driver's perception of the road, overlaying relevant information onto the real-world view. Through these technologies, RoadVisor aims to empower drivers with advanced features and functionalities, regardless of the vehicle they own. It brings the benefits of machine

learning and augmented reality to older-generation cars, improving their driving experience and safety on the road.

RoadVisor aims to enhance the driver's experience by utilizing real-time street events captured through the mobile phone camera. It provides users with an improved street view, directions, and timely updates. The application utilizes Machine Learning techniques such as Deep Neural Networks and utilizes APIs like the MapBox API to deliver accurate road information and directions. Augmented Reality features are implemented to prevent distractions, ensuring that vital road information is easily accessible while consulting the mobile device for directions or other relevant details. The application's performance is dependent on the processing capabilities of the user's device, enabling efficient processing of information on the go.

The primary objective of RoadVisor is to effectively address traffic violations, including disregarding street signs and traffic lights, committed by drivers. By offering a compelling and superior alternative to existing navigation applications in the market, RoadVisor aims to fill a significant void in the industry. Our application strives to assist drivers without causing distractions or annoyance, ultimately reducing the number of fatalities resulting from road accidents or collisions [1].

RoadVisor aligns with the Offering category and places a strong emphasis on Product Performance, as defined by the 10 Types of Innovation Wheel by Doblin [2]. The incorporation of augmented reality (AR) into the navigation feature sets RoadVisor apart from other applications, delivering enhanced and more user-friendly functionality. This distinctive feature positions RoadVisor as an incremental innovation project, with optimization being a key element in our pursuit of providing an improved user experience.

The development of RoadVisor will be executed in incremental stages, with a focus on optimizing its performance for mobile devices. By continually refining and enhancing the application's functionality, we aim to create a comprehensive solution that addresses the needs and expectations of drivers.

2. Requirements Details

We have three different types of requirements. They are functional, non-functional, and pseudo requirements.

2.1. Functional Requirements

2.1.1. Road Navigation with Augmented Reality

An integral aspect of RoadVisor is to provide users with comprehensive guidance throughout their entire road trip. The application aims not only to navigate users to their selected destination in an optimal manner but also actively engages in displaying the necessary vehicle movements. To achieve this, RoadVisor incorporates Augmented Reality to visually guide users along their route. By utilizing the mobile phone's camera, the application tracks the road and overlays arrows that correspond to the path the driver needs to follow to reach the desired location. To initiate the navigation service, users input their destination location. Subsequently, RoadVisor captures real-time footage of the road through the camera, matching each frame with the route of the desired location to accurately position the navigation arrows on the road. The arrows dynamically adapt to the road, ensuring continuous updates as the vehicle progresses. When more significant maneuvers, such as turns, are required, larger arrows prominently appear on the screen to emphasize the actions. In addition to the directional guidance, the screen displays essential information about the current location and the remaining distance to the destination, providing users with crucial context throughout their journey. This feature enables users to have a clear understanding of their progress and estimated arrival time.

2.1.2. Traffic Light and Sign Detection

RoadVisor employs real-time analysis of the user's live camera footage, examining each frame for the detection of traffic lights and traffic signs. Using the camera feed, the application continually performs image analysis to identify the presence of visible traffic lights and signs. For traffic lights, RoadVisor provides immediate information to the user by displaying the color of the light on the phone screen. In particular, when a red light is detected, a pop-up notification is generated to alert the driver. Regarding traffic signs, RoadVisor actively notifies the user about any signs detected and

provides relevant information regarding their meaning. The application classifies the signs and displays them on the phone screen, allowing the user to be aware of the specific signs encountered during their journey. Additionally, for signs that require heightened attention, such as stop signs, RoadVisor can generate additional pop-up messages to further alert the driver. This functionality ensures that RoadVisor keeps drivers informed about important traffic information, promoting safer and more compliant driving behavior. By utilizing image analysis and intelligent detection algorithms, the application enhances the driver's situational awareness and contributes to overall road safety.

2.1.3. Pedestrian Detection

RoadVisor leverages the user's phone camera to analyze live camera footage frame by frame, enabling continuous image analysis for the detection of pedestrians on the road. Using the camera feed as input, the application performs real-time image analysis to identify the presence of pedestrians. When RoadVisor detects a pedestrian, it visually marks their presence on the screen, ensuring that the driver is aware of their proximity. Additionally, RoadVisor displays a warning message alongside the visual marker to alert the driver and emphasize the importance of exercising caution. By actively monitoring and detecting pedestrians, RoadVisor aims to enhance driver awareness and promote safer driving practices. This feature helps mitigate the risk of accidents involving pedestrians, ultimately contributing to improved road safety for all road users.

2.1.4. Requesting Help and Crash Detection

RoadVisor incorporates a feature that allows users to enter phone numbers as "urgent contact" information. In situations where the user requires assistance, they can press the SOS button within RoadVisor. This action triggers the application to automatically send a message to the designated urgent contact numbers, indicating that the user may be in a problematic situation and including the user's location. Furthermore, the SOS feature of RoadVisor can be activated automatically if the application detects a sound resembling a car crash. In such cases, RoadVisor prompts the user to confirm if there is indeed a problematic situation. If the user fails to respond within a specified duration, RoadVisor interprets this as an urgent issue

and automatically alerts the urgent contact numbers. This functionality ensures that users can quickly and efficiently request help when needed, enhancing their safety and enabling swift assistance in emergency situations. RoadVisor prioritizes user well-being by providing an effective means of communication and support during challenging circumstances.

2.2. Non-functional Requirements

2.2.1. Reliability

Reliability is of utmost importance for RoadVisor, ensuring the accurate delivery of navigation information and fulfilling the expected functionalities. The application is designed to be highly reliable in various aspects:

1. Route Preparation: RoadVisor strives to prepare routes accurately, taking into account the user's selected destination and optimizing the navigation path accordingly. The application utilizes reliable mapping and routing algorithms to provide precise directions for the entire journey.

2. Direction Arrow Placement: To ensure accurate guidance, RoadVisor meticulously places direction arrows on the screen, aligning them with the intended path for the driver to follow. The arrows are updated dynamically, adapting to changes in the road and ensuring accurate navigation instructions.

3. Traffic Light and Sign Information: RoadVisor ensures the delivery of true and reliable information regarding traffic lights and signs. The application accurately detects and notifies the user about the color of traffic lights, enabling them to make informed driving decisions. Additionally, RoadVisor correctly identifies and classifies traffic signs, providing accurate information about their meaning to enhance driver awareness and compliance.

4. SOS Messages: In critical situations, RoadVisor's SOS feature operates reliably, swiftly sending automated messages to designated urgent contact

numbers. These messages include the user's location, ensuring prompt assistance can be provided when needed.

RoadVisor is designed to operate seamlessly and without significant interruptions, providing uninterrupted navigation services and fulfilling its promised functionalities throughout the entire car trip duration. However, it is essential to ensure that the necessary conditions, such as sufficient battery power and a stable connection, are met to maintain the application's reliability. By prioritizing reliability, RoadVisor aims to deliver a consistent and dependable user experience for safer and more reliable journeys.

2.2.2. Performance

RoadVisor upholds a high standard of performance, excelling in frequent conditions and operating efficiently within short durations. The application's performance is optimized across various aspects:

1. Real-time Analysis: As RoadVisor continuously extracts input from live camera feeds and performs frame-by-frame analysis, it ensures that the frame rate remains above a tolerable level. This enables the application to operate without significant delays, ensuring the timely processing of visual information.

2. High Accuracy Detection Models: RoadVisor utilizes advanced detection models that exhibit high accuracy rates. These models are carefully selected and trained to reliably identify and interpret various objects, including traffic signs, traffic lights, and pedestrians. The application's ability to achieve accurate detections enhances its overall reliability.

3. Responsive Touch Screen Interaction: RoadVisor delivers a smooth and responsive touch screen experience, offering fast response times to user inputs. This ensures that users can interact with the application seamlessly, facilitating intuitive navigation and easy access to various features.

4. Mobile Environment Optimization: RoadVisor is specifically designed to perform well in a mobile environment. The application is engineered to function optimally, considering the hardware limitations typically associated with mobile phones. This ensures that RoadVisor delivers its features efficiently, offering a satisfying user experience while minimizing the impact of mobile hardware constraints.

By prioritizing performance, RoadVisor aims to deliver a highly efficient and reliable application that seamlessly operates within the mobile environment. The application's ability to perform well across various conditions and respond swiftly contributes to a smooth user experience and enhances the overall usability of RoadVisor.

2.2.3. Usability

RoadVisor, as an application, strives to cater to a wide range of users, including drivers who may not have access to the latest technological tools in their vehicles. The application focuses on being user-friendly and easily understandable for its target audience. Key considerations regarding usability and compatibility include:

1. Compatibility with Android Phones: RoadVisor aims to be compatible with a broad range of Android phones, provided they meet the specified minimum version requirements. This ensures that a wide variety of users can access and utilize the application on their devices.

2. Responsive and Simple User Interface: The user interface of RoadVisor is designed to be highly responsive and visually simple. Recognizing that users will be engaged in driving tasks, the application minimizes complex operations that require multiple steps. User-interactable elements are designed to be large, easily locatable on the phone screen, and responsive within a short duration (less than a minute). This streamlined design approach allows for intuitive and efficient interaction with the application, reducing cognitive load for the user.

3. Minimizing Distraction and Irritation: Given that users will be actively driving while using RoadVisor, the application is designed to minimize unnecessary distractions and potential irritations. This is achieved by avoiding the use of irritating alert sounds, employing a clean and uncluttered UI layout, and reducing the need for excessive user input. For instance, pop-up messages are automatically closed after a reasonable duration, ensuring that they do not linger and distract the driver unnecessarily.

By prioritizing user-friendliness and minimizing distractions, RoadVisor aims to provide a seamless and pleasant user experience. The application's design considerations take into account the unique context of driving, ensuring that users can interact with the application effortlessly while maintaining their focus on the road.

2.2.4. Privacy

RoadVisor prioritizes user privacy and ensures the protection of confidential information. The application adheres to the following principles to safeguard user data:

1. Non-Disclosure of Confidential Information: RoadVisor strictly prohibits the use of confidential user information, including contact numbers, destination information, and exact route data, with third parties. User privacy is respected, and personal data is not shared or disclosed without the user's explicit consent.

2. Encryption of Confidential Data: To minimize the risk of exposure and unauthorized access, RoadVisor employs robust encryption techniques to protect confidential data. Confidential information, such as user details and sensitive location data, is securely encrypted both during transmission and storage. This encryption ensures that even if a potential attack occurs, confidential information remains protected and inaccessible to unauthorized parties.

By implementing stringent privacy measures, including non-disclosure of confidential information and encryption of sensitive data, RoadVisor maintains a high level of security and confidentiality for its users. User trust and privacy are of paramount importance, and RoadVisor strives to uphold these principles to safeguard user data from potential vulnerabilities or breaches.

2.3. Pseudo Requirements

RoadVisor prioritizes high response time to accurately display direction information. To achieve this, the application employs strategies such as pre-computing necessary computations or performing them directly on the mobile side, minimizing reliance on fetching data from a cloud environment. Consideration is given to the space and functionality limitations of mobile phones, ensuring that RoadVisor operates optimally within these constraints. The application is designed to work seamlessly with the overall hardware systems of mobile phones, taking into account their capabilities and limitations. RoadVisor requires access to the camera and other relevant device information for its functionality. As such, the application is specifically designed to be compatible with Android devices, ensuring smooth and reliable operation on this platform. Compatibility with the MapBox API is a key aspect of RoadVisor. The application is developed to integrate seamlessly with the chosen map application, allowing for the efficient utilization of map data and services. Synchronous computations are essential for RoadVisor, and the selected machine learning and computer vision models are carefully chosen to address this requirement. These models are designed to perform computations in real time, ensuring the application can process information promptly and deliver accurate results.

By considering the factors mentioned above, RoadVisor aims to provide a responsive, efficient, and seamless user experience. The application leverages the capabilities of mobile devices, integrates with the selected map application, and employs suitable computational models to ensure optimal performance and functionality.

3. Final Architecture and Design Details

The overall final architecture of the project remained to a great degree similar to the one proposed. We will discuss the overall final structure in the sections below.

3.1. Overview

We have several design goals to aim for and lead us through the project. The architecture has been designed to meet the functional and non-functional requirements of the software, such as performance, security, reliability, and usability, while also being flexible and adaptable to future changes and enhancements. For this reason, our project consists of several layers. The layers are the user interface layer, the application layer, and the server layer. The user interface layer is the layer where the user interacts with the system and it is the part that the users will see when they open up the application on their Android mobile phone. The application layer is the layer where the business logic is handled and the models are incorporated into the system. The server layer consists of the cloud systems where our machine-learning models are deployed. It also consists of the database that is required to store the user information and the deployment of access points to interact with the database.

This section also explains the hardware/software mappings of the project. However, since our project does not have any hardware components, we will leave this section blank. Additionally, how we handle persistent data is explained in detail and the access control and security of our system will be explained. Our project does not deal with private user data collection, other than the account information, and we do not require any special access or authentication from our users, therefore these sections will be rather short.

3.2. Subsystem Decomposition

We will follow three-tier architecture to divide the subsystem into three different layers:

1. Interface Layer
2. Application Logic
3. Server Layer

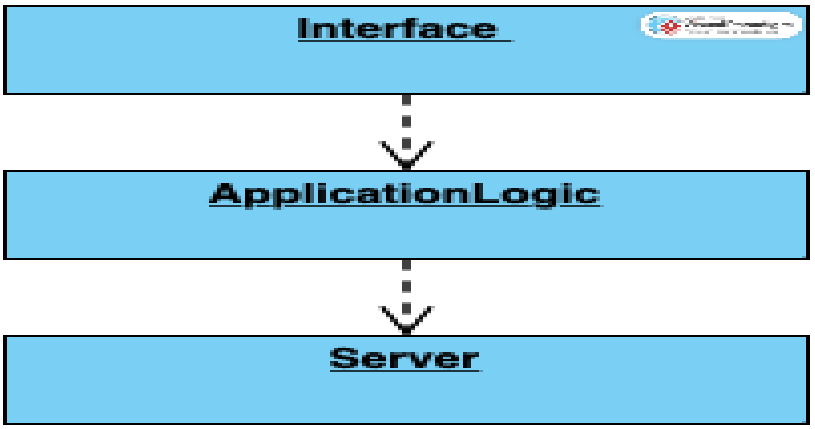


Figure 3.1. *Three-tier decomposition of our Application.*

This particular style of subsystem decomposition architecture has been selected based on the needs of our application. Other architectures, like four-tier or simple server-client architectures, can be possible options but will either create an unnecessary layer or prevent logical hierarchical separation of different systems. The Interface subsystem layer is based on the user device, in our case, an Android phone. The Application logic is also based on the user device, but it serves as a mechanism for the interface layer to interact with the server. The lowest layer of our architecture is the Server layer which is responsible for storing information in the database and fetching data from there. More details about the subsystems are in the subsystem services section.

3.3. Hardware/Software Mappings

As our project does not have any hardware components, this section is left blank.

3.4. Persistent Data Management

The amount of persistent data that we manage is limited to the functionalities that we provide. The data is stored on a PythonAnywhere server, which is hosted on Amazon Cloud Service. We don't store any additional or dynamic data that changes while the user uses the application. All the data that is stored is entered by the user while signing up for the application. The data is stored in the user information for logging in and emergency requests. A user can add this information during the signup. We don't store any data regarding the user's location or behavior. Our application intends to respect user privacy by not collecting unnecessary sensitive or behavioral user information. We also don't use the stored data for understanding user behavior or share the stored data with third parties.

3.5 Interface Subsystem

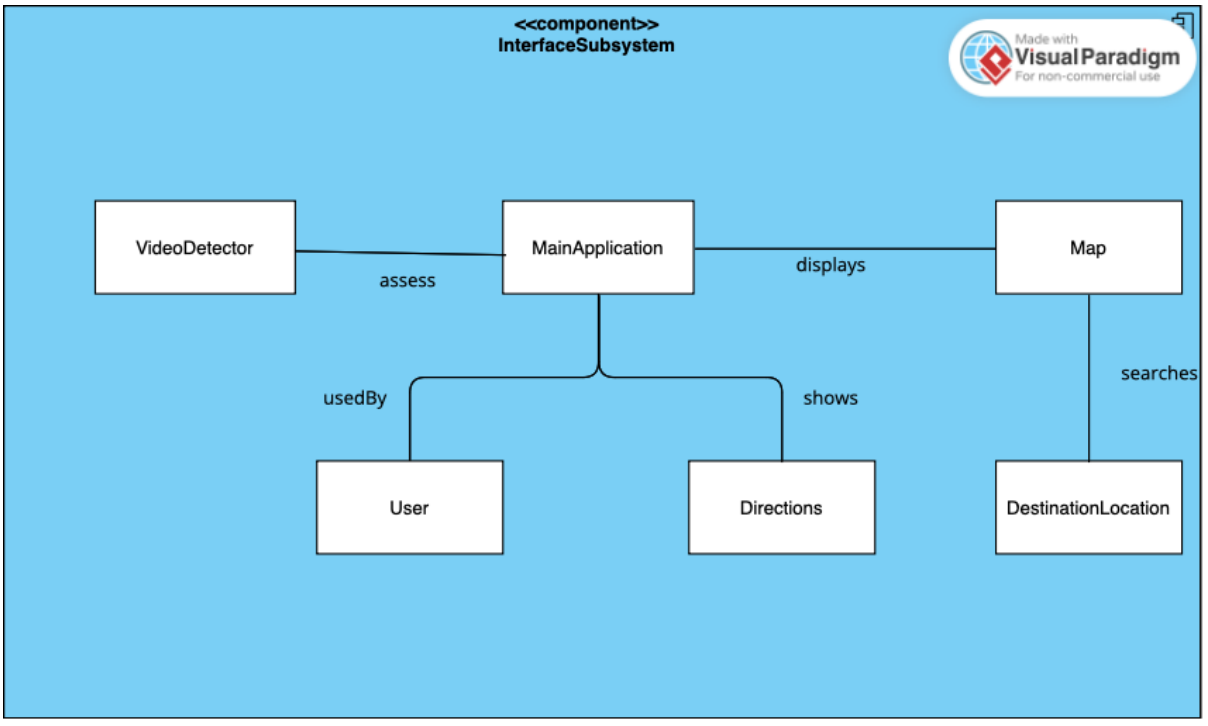


Figure 3.2. *The Interface Layer Subsystem.*

The interface layer subsystem contains the necessary sections to display the maps and navigation information. It also contains a VideoDetector system that helps display detected traffic lights, traffic signs, and pedestrians. The VideoDetector fetches the detector information from the Application Logic level subsystem. The

User system is responsible for managing the user information like a username. It also helps the application fetch emergency contacts during emergency requests. The Directions system is responsible for showing the user the arrows after the destination has been selected using the Map. The User will select the destination he/she is traveling to using the map. The Map and other systems interact directly with the Application Logic Layer. The DestinationLocation selected is then used by the Application logic to make requests for directions to the destination. The directions fetched are then displayed by using the Directions system.

3.6. Application Logic Subsystem

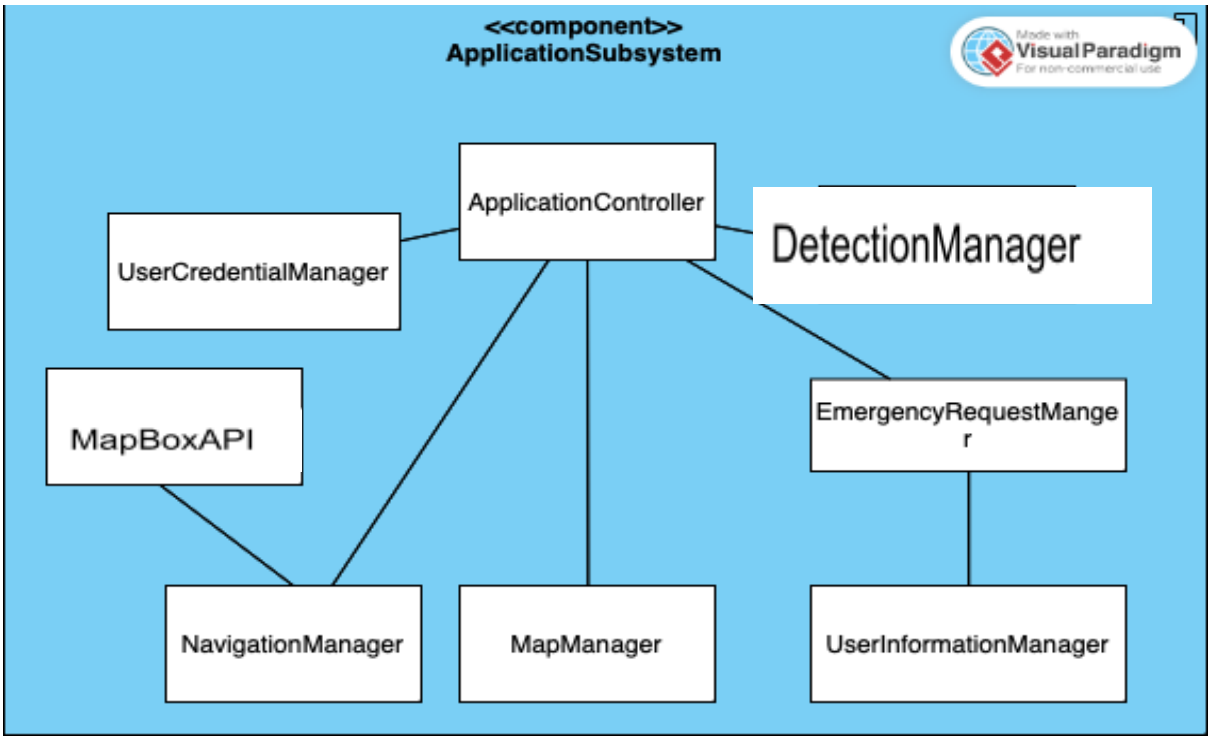


Figure 3.3. *The Application Logic Layer.*

The Application Logic Layer is responsible for the control of the application, response to user requests, and connecting the interface and server. This layer acts as a buffer between the two layers. All the requests made by the interface layer are controlled by the ApplicationController which in turn calls other managers based on specific functions required by the user requests. The Map Manager is responsible for fetching the map and managing the selection of the destination by the user. The MapManager sends the map to the interface. On user interaction with the map, the Interface layer makes requests to the ApplicationController which in turn based on

the application status responds to the request. The EmergencyRequestManager is responsible for managing the requests made by the user for emergency help during an accident. The EmergencyRequestManager uses the UserInformationManager to fetch the information of the user's emergency contacts from the Server Layer, and then send a notification to the user regarding the emergency.

The DetectionManager is responsible for generating the detection of lights, signs, and pedestrians. The manager runs YOLO detection algorithms on the images sequentially sampled from the camera of the Android device. The information about the detections made by the device is then sent to the interface by the application controller. The NavigationManager is responsible for making requests to the MapAPIManager for directions based on the user's location and the final destination. The requests will be made by the Application layer based on the user's location. We will try to optimize the number of requests that are being made to decrease unnecessary API requests and computation on the Android device.

3.7. Server Subsystem

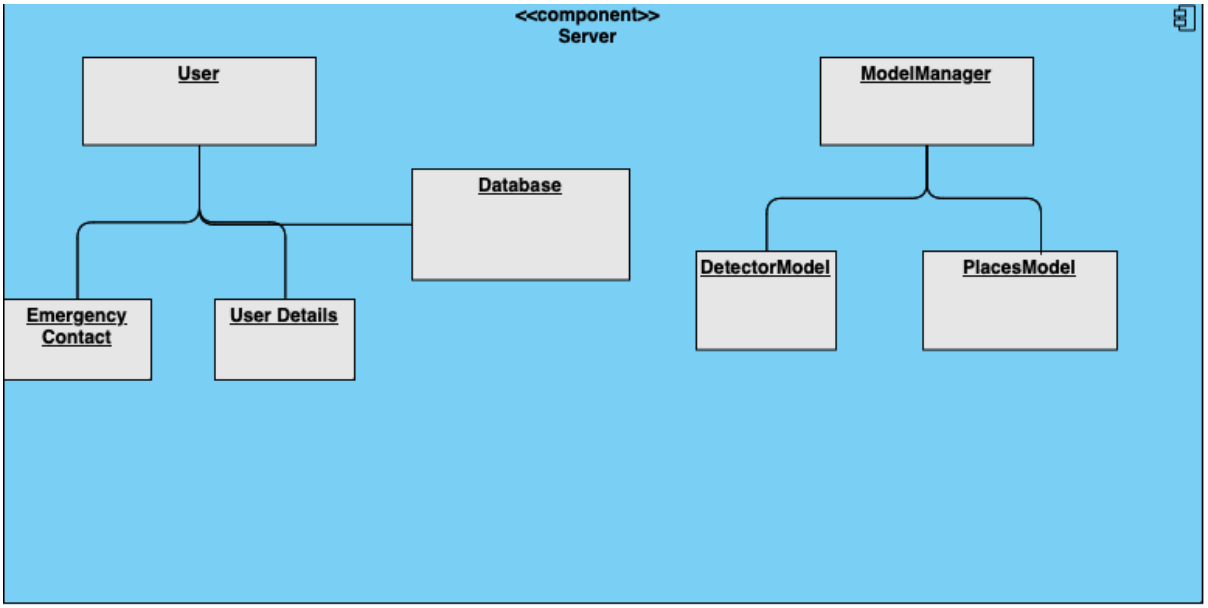


Figure 3.4. *Server Subsystem Layer.*

The design of the server layer has undergone considerable change since the proposed design. We have two parts in the server layer. One is responsible for the deployment and storage of the models, while the other is responsible for the

database. The models have been deployed on the Google cloud server using the Google Vertex AI service. The models have also been deployed on the cloud.

4. Development/Implementation Details

The project was broadly divided into three parts: frontend, backend, and machine learning. The frontend part consisted of implementing AR-based navigation and implementing other UI elements for the features. The backend part consisted of database management and implementation of endpoints for interacting with the database. The machine learning part consisted of dataset processing, training models, testing models on different devices, and deploying the models on the cloud.

4.1. Frontend Implementation

Figure 4.1. *Directions in AR-based navigation.*

The implementation of the front end was carried out using Android Studio. Mapbox API was used extensively for the implementation of the navigation feature. The map feature for searching and selecting a destination was also implemented using Mapbox. The implementation of the front end also consisted of creating the UI of the login page, sign-up page, emergency feature, and detection information display. Initially, we experimented with google maps sdk. However, after implementing the navigation feature using this library, we realized that it wasn't of good quality. After

that, we decided to make changes to the navigation by using Mapbox API, which proved to be of better quality in nature.

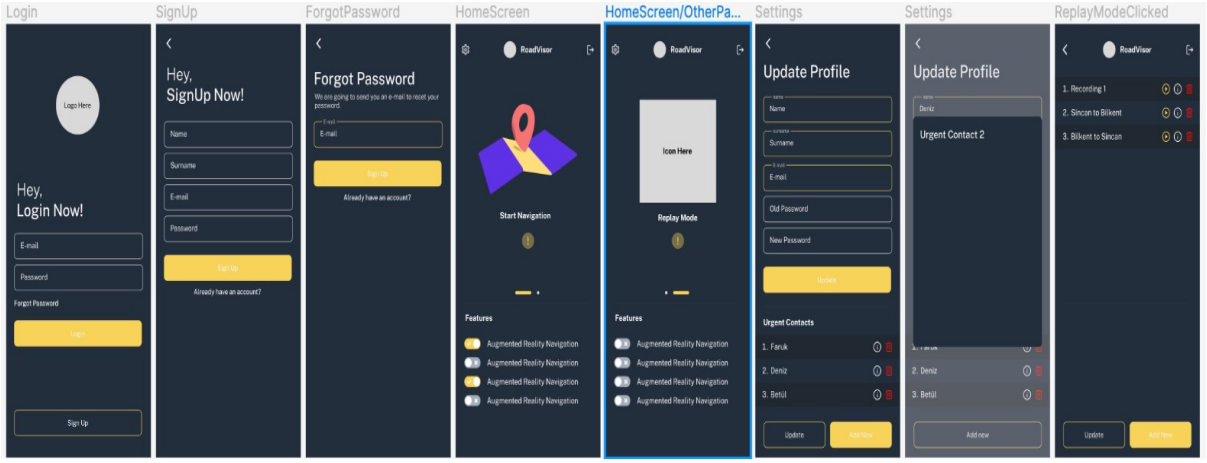


Figure 4.2. *The user interface of the application.*

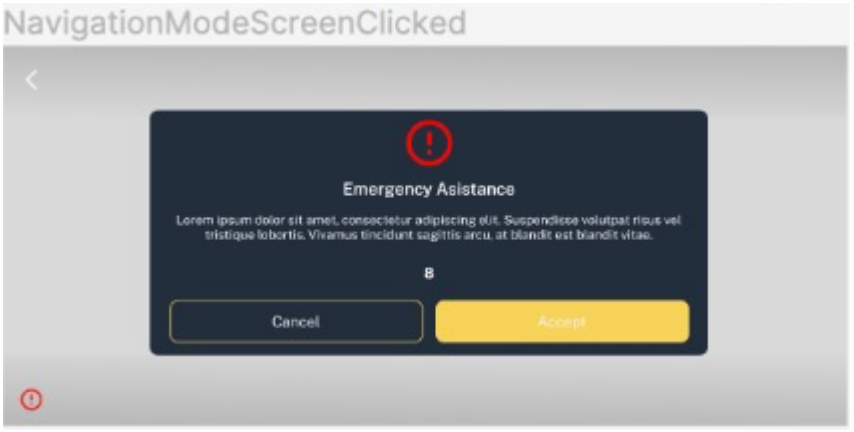


Figure 4.3. *Emergency Function.*

The arrows are displayed on the screen to minimize the possible confusion a user might have while using the application. We experimented with different distances and methods to display the arrows. Ultimately, we, after consulting our supervisor, concluded that the best method was as shown in the first image of this section. The arrows representing the turns are moved closer to represent the decreased difference between the turns.

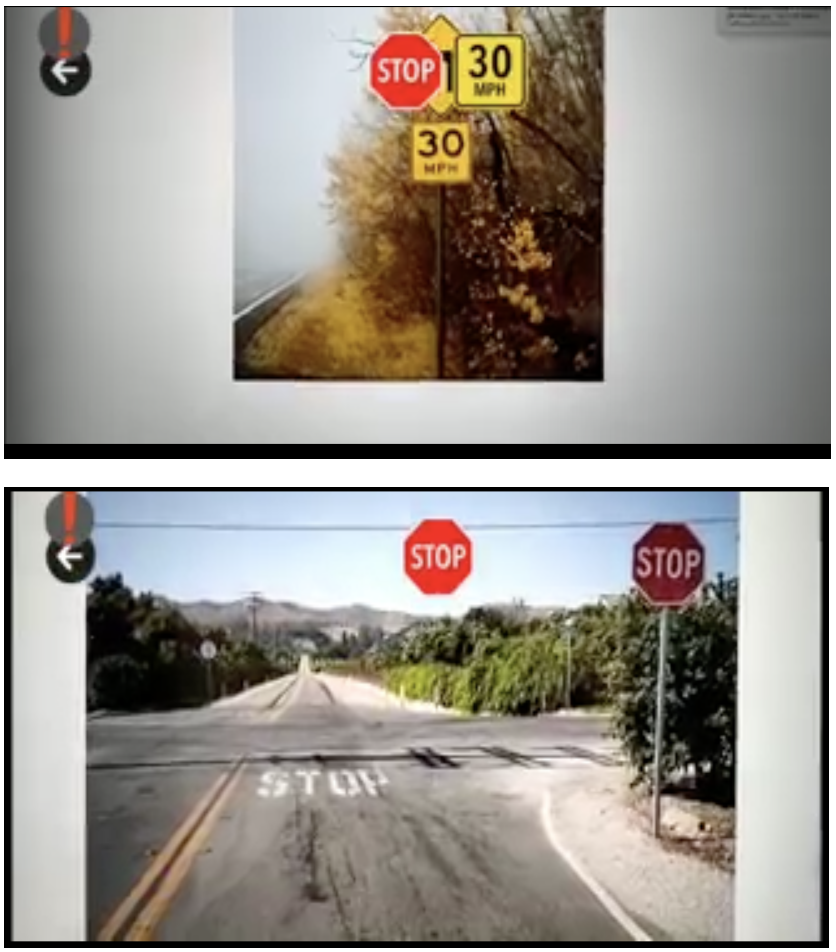


Figure 4.4. *Sign Detection using the application.*

We updated the front end to improve the quality and style of the applications. We have tried to provide an easy user-interface for our potential users. Augmented reality-based navigation helps users understand and decreases the confusion that a user might have while using a traditional navigation application. Our application makes traveling easier and directions more understandable.

4.2 Backend Implementation

The backend implementation involved setting up the PythonAnywhere server, which is hosted on Amazon Cloud Service, and configuring the MySQL database, including the creation of database schemas and tables. Django was utilized for implementing the server-side code of the project. The functionalities implemented on the backend included adding users to the database, storing user information, and incorporating a feature for emergencies. For the emergency feature, an endpoint was developed to facilitate the creation of an emergency call, which subsequently sends an emergency

email to the user's designated emergency contacts. Additional endpoints were established for user registration, user updates, and logins.

4.3 Machine Learning

The machine learning part consisted of researching possible models, processing the datasets, training the models, testing the performance of the models, and deployment of the models on devices. We needed machine learning models for traffic light detection, traffic signs detection, and pedestrian detection. After looking at different possible object detection models, like Faster RCNN and YOLO, that could be used for object detection, we decided to use YOLOv7 for object detection. Instead of using 3 different models for the three different detection tasks (traffic lights, traffic signs, and pedestrians), we decided to use a singular YOLOv7 model for it. To be able to train the YOLO model, we had to process and merge different datasets that were completely independent. The formats of the datasets were also not in the format required to train a YOLO model. So, we had to separately change the format of the models to make it consistent with the required YOLO format. To change the format, we had to write programs that changed the format for all three datasets to YOLO format. Then, to merge the datasets, we had to create consistent labeling. To achieve this, we wrote code that changed the labels. We had a total of 9 classes: prohibitory, dangerous, mandatory, other, person, people, red, yellow, and green. The first 4 labels correspond to the type of sign, for example, the stop sign is a mandatory sign.

Following dataset processing, initially, we trained the dataset on Google Colab for around 100 epochs.

Figure 4.5. *Performance on the dataset images.*

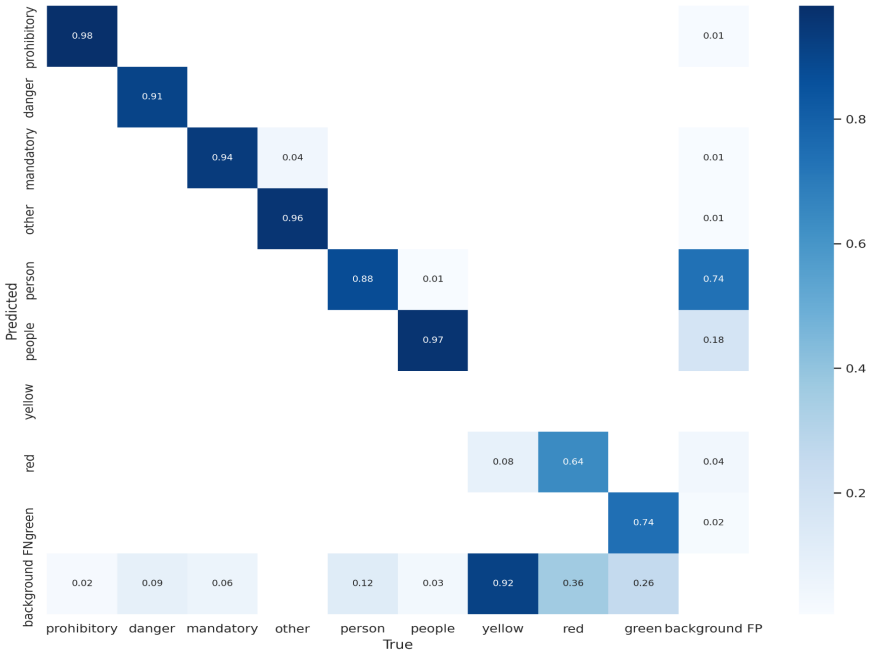


Fig 4.6. *Confusion matrix for dataset after 1st training.*

After completion of the training, we tried to run the model directly on the Android device. However, the inference time on a single frame was 5 seconds; the time taken was quite unreasonable for our project, where we need to make inferences for a user in a moving car. Due to this, we decided to deploy the model on a cloud service.

We deployed the model on Google cloud. Initially, we were evaluating other cloud services as well for deployment. We realized that google cloud tended to be a better option. One benefit was that it kept the model loaded, rather than having to load it separately for each request call. We had to create a docker image and then use it to deploy the model. We also had to write server-side code for this purpose as well. The process of deployment of the model on the cloud was a tiresome task. It took more than a week to successfully deploy the models. After completing and testing the deployed detection model, we decided to train the YOLO model for 100 more epochs to improve its performance.

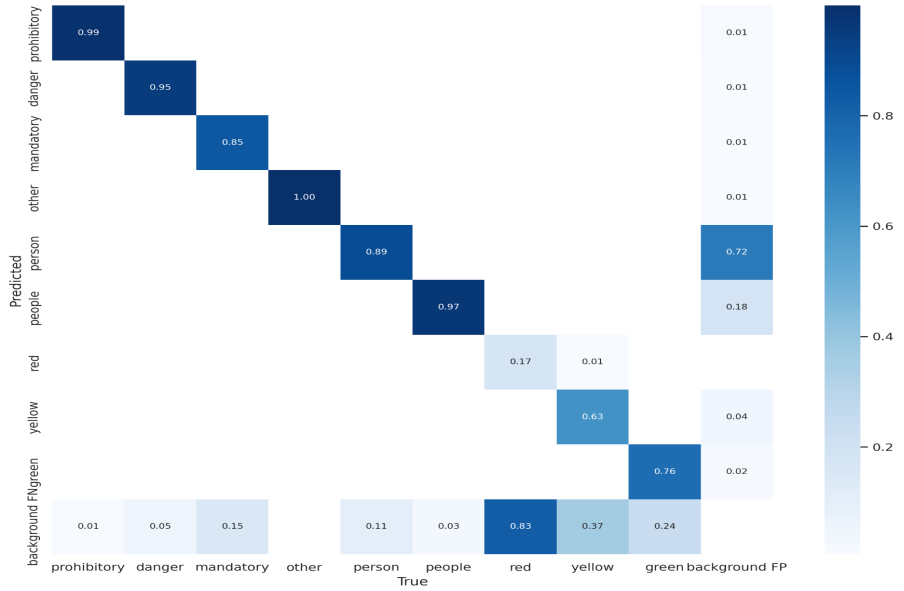


Figure 4.7. *Confusion matrix after training for another 100 epochs.*

After completing the deployment and testing of the detection model, we started finding candidates for our type of places model. We selected a pre trained ResNet50 (50 layered Residual Network) for this task. It had 256 categories for classification, many of which were not important to us. The accuracy of the trained model was 85%. We had to write code for it to make inferences as well as to deploy the model on google cloud. This model was required to implement songs based on location.

5. Test Cases and Results

Functional Tests

Register API Tests

Test ID: TS-01-1

Title: Successful User Registration

- Procedure:
    1. Send a POST request to the API endpoint /api/v1/accounts/register/ with correct user data.
    2. Verify that the response status code is 201 CREATED.

3. Verify that the response message is Please check your email to activate your account.
4. Verify that the confirmation email is sent to the user's email address.
5. Verify that the email contains the correct activation link.

- Expected outcome:
    1. The user should be successfully registered and a confirmation email should be sent to the user's email address. The activation link in the email should be correct and the user should be able to activate their account.
- Severity: High
- Result:
    - The POST request to the API endpoint with correct user data is successfully processed.
    - The response status code is 201 CREATED, indicating that the user registration was successful.
    - The response message is "Please check your email to activate your account," confirming that the user needs to activate their account via email.
    - A confirmation email is sent to the user's provided email address.
    - The email contains the correct activation link, which allows the user to activate their account.
    - The user follows the activation link and successfully activated their account.

Test ID: TS-01-2
Title: User Registration with Duplicate Email

- Procedure:
    1. Send a POST request to the API endpoint /api/v1/accounts/register/ with user data containing an email address that already exists in the database.
    2. Verify that the response status code is 400 BAD REQUEST.

3. Verify that the response message is User with this email already exists.

- Expected outcome:
    1. The user should not be registered and an error message should be returned stating that a user with that email address already exists.
- Severity: High
- Result:
    - The POST request to the API endpoint with user data containing a duplicate email address is sent.
    - The server detects that a user with the provided email address already exists in the database.
    - The response status code is 400 BAD REQUEST, indicating that the registration request cannot be processed due to a duplicate email.
    - The response message is "User with this email already exists," indicating that a user with the given email address is already registered.
    - The user is not registered, and the error message clearly states that a user with that email address already exists.

Test ID: TS-01-3
Title: Token Creation

- Procedure:
    1. Send a POST request to the API endpoint /api/v1/accounts/register/ with correct user data.
    2. Verify that the response status code is 201 CREATED.
    3. Verify that the user's token is a unique value.
- Expected outcome:
    1. The user's token should be a unique value.
- Severity: Medium
- Result:

    - The POST request to the API endpoint with correct user data is successfully processed.
    - The response status code is 201 CREATED, indicating that the user registration was successful.
    - The user's token is generated as part of the registration process.
    - The user's token is expected to be a unique value.
    - The uniqueness of the token is verified by comparing it with existing tokens in the system.
    - If the token is found to be unique, the test is considered successful.

Test ID: TS-01-4
Title: Confirmation Email Sent

- Procedure:
    1. Send a POST request to the API endpoint /api/v1/accounts/register/ with correct user data.
    2. Verify that the response status code is 201 CREATED.
    3. Verify that the confirmation email is sent to the user's email address.
    4. Verify that the email subject is Account Activation.
    5. Verify that the email message contains the correct activation link.
- Expected outcome:
    1. The confirmation email should be sent to the user's email address with the correct activation link.
- Severity: Medium
- Result:
    - The POST request to the API endpoint with correct user data is successfully processed.
    - The response status code is 201 CREATED, indicating that the user registration was successful.
    - The confirmation email is expected to be sent to the user's email address.

- The email subject is expected to be "Account Activation."
- The email message content is expected to include the correct activation link, which is necessary for the user to activate their account.
- The presence of the confirmation email is verified by checking the user's email inbox or by simulating the sending and receiving of emails during testing.

Update API Tests

Test ID: TS-02-1

Title: Successful User Update

- Procedure:
    1. Authenticate as a user by sending a POST request to the API endpoint /api/v1/token/ with correct user credentials.
    2. Send a PATCH request to the API endpoint /api/v1/accounts/update/ with correct user data to update.
    3. Verify that the response status code is 200 OK.
    4. Verify that the response message is {'success': True}.
    5. Verify that the user data has been successfully updated in the database.
- Expected outcome:
    1. The user data should be successfully updated in the database and a success message should be returned.
- Severity: High
- Result:
    - The user data is successfully updated in the database, and a response with a status code of 200 OK is received. The response message is {'success': True}, indicating that the update operation was completed successfully. The updated user data is reflected in the database, confirming that the changes have been applied.

Test ID: TS-02-2

Title: User Update with Invalid Data

- Procedure:
    1. Authenticate as a user by sending a POST request to the API endpoint /api/v1/token/ with correct user credentials.
    2. Send a PATCH request to the API endpoint /api/v1/accounts/update/ with user data containing invalid data.
    3. Verify that the response status code is 400 BAD REQUEST.
    4. Verify that the response message contains the errors returned by the serializer.
- Expected outcome:
    1. The user data should not be updated in the database and an error message should be returned containing the errors returned by the serializer.
- Severity: Medium
- Result:
    - The user data is not updated in the database, and a response with a status code of 400 BAD REQUEST is received. The response message contains the errors returned by the serializer, indicating the specific issues with the provided data. The errors include validation errors specified by the application's business logic.

Test ID: TS-02-3
Title: User Update with Missing Fields

- Procedure:
    1. Authenticate as a user by sending a POST request to the API endpoint /api/v1/token/ with correct user credentials.
    2. Send a PATCH request to the API endpoint /api/v1/accounts/update/ without providing any of the required fields.
    3. Verify that the response status code is 400 BAD REQUEST.

4. Verify that the response message is {'error': 'At least one of the following fields must be provided: email, first_name, last_name, password, old_password.'}.

- Expected outcome:
    1. The user data should not be updated in the database and an error message should be returned stating that at least one of the required fields must be provided.
- Severity: Medium
- Result:
    - The user update operation fails due to the absence of any required fields in the request. The application responds with a status code of 400 BAD REQUEST, indicating a client error. The response message contains an error object specifying that at least one of the following fields must be provided: email, first_name, last_name, password, old_password. The user data remains unchanged in the database.

Activation API Tests

Test ID: TS-03-1
Title: Account Activation Process

- Procedure:
    1. After a user signs up, they receive the correct URL to activate their account.
    2. The user sends a GET request to the API by opening the correct URL.
    3. The API checks the incoming request.
    4. If the user does not provide a valid user_id and token value, the API returns a 400 Bad Request error.
    5. If the user provides a valid user_id and token value, the API activates the user's account and returns a "Account activated successfully" message with a 200 OK status.
- Expected outcome:

1. When a GET request is sent with a valid user_id and token, the account is successfully activated and the API returns a "Account activated successfully" message with a 200 OK status.
2. When a GET request is sent with an invalid user_id or token, the API returns a 400 Bad Request error.

- Severity: High
- Result:
    - When a user sends a GET request with a valid user_id and token to activate their account, the API successfully activates the account and responds with a "Account activated successfully" message, indicating a 200 OK status. The user's account status is updated to activated, granting them access to the application's features and functionalities.
    - However, when the user sends a GET request with an invalid user_id or token, the API detects the invalid values and responds with a 400 Bad Request error. The account activation process fails, and the user cannot gain access to the application. They may need to retry the activation process with valid credentials or seek assistance from the support team if issues persist.

Login API Tests

Test ID: TS-04-1

Title: Test the API functionality for user login.

- Procedure:
    1. Send a POST request to the '/api/v1/login/' endpoint with valid email and password in the request body.
    2. Verify that the response status code is 200 OK.
    3. Verify that the response body contains user details including user id, email, first name, last name, and token.
- Expected outcome:

1. The API should return a response with status code 200 OK and user details in the response body when the user credentials are correct.
2. The API should return a response with status code 400 Bad Request and an error message in the response body when the user credentials are incorrect or missing.

- Severity: Medium
- Result:
    - When a user sends a POST request to the '/api/v1/login/' endpoint with valid email and password, the API verifies the credentials. If the credentials are correct, the API responds with a status code of 200 OK and provides the user details in the response body, including the user id, email, first name, last name, and token. This allows the user to proceed with authenticated access to the application.
    - However, when the user credentials are incorrect or missing, the API detects the invalid credentials and responds with a status code of 400 Bad Request. The response body contains an appropriate error message, indicating that the login attempt was unsuccessful. The user is prompted to enter valid credentials to gain access to the application.

Logout API Tests

Test ID: TS-05-1, TS-05-2

Title: Test logout API

1. Test that the user can successfully log out of the system.
2. Test that an authenticated user is required to access the logout API.

- Procedure:
    1. Authenticate a user and obtain the token.
    2. Make a POST request to the logout API with the token in the Authorization header.
    3. Verify that the response status code is 200 OK.
    4. Verify that the response message is "Logged out successfully."

5. Verify that the user is logged out of the system.
6. Attempt to access a protected resource with the token.
7. Verify that the response status code is 401 Unauthorized.

- Expected outcome:
    1. The user should be able to log out of the system successfully.
    2. An authenticated user is required to access the logout API.
- Severity: Medium
- Result:
    - For Test ID TS-05-1, when a user with a valid token sends a POST request to the logout API, the system successfully logs out the user. The API responds with a status code of 200 OK and a message indicating that the logout was successful. After logging out, the user's token is no longer valid, and further attempts to access protected resources with the same token results in an authentication error.
    - For Test ID TS-05-2, when a user attempts to access the logout API without authentication, the system rejects the request and responds with a status code of 401 Unauthorized. This indicates that the user must be authenticated before they can log out of the system.

Password Reset Request API Tests

Test ID: TS-06-1
Title: Test Password Reset Request API

- Procedure:
    1. A POST request is made with a valid email address.
    2. A successful response is received for an existing user with the email address.
    3. An unsuccessful response is received for a non-existing user with the email address.
    4. The returned response status code should be 200 OK.

- Expected outcome:

1. When a POST request is made with a valid email address and the email is registered, a message "If the email you entered is valid, you will receive instructions on how to reset your password in your email shortly." is returned along with a 200 OK status code.
2. When a POST request is made with a valid email address and the email is not registered, the same message is returned along with a 200 OK status code.
3. When a POST request is made with an invalid email address, an error message and a 400 Bad Request status code are returned.

- Severity: High
- Result:
    - When a POST request is made to the password reset API with a valid email address that is registered in the system, the API responds with a success message confirming that instructions for password reset will be sent to the user's email shortly. The response status code is 200 OK.
    - When a POST request is made with a valid email address that is not registered in the system, the API responds with the same success message indicating that instructions will be sent to the provided email address. The response status code is 200 OK.
    - When a POST request is made with an invalid email address, the API responds with a status code of 400 Bad Request and an error message stating that the email address is invalid. This is to indicate that the request cannot be processed due to an invalid email address.

Password Reset Confirm API Tests

Test ID: TS-07-1

Title: Test Password Reset Confirm API

- Procedure:

1. Set up the user and generate a password reset token and encoded_pk.
2. Hit the "Reset Password API" endpoint with the token and encoded_pk in the URL, and provide a new password and its confirmation in the request body.
3. Check that the response has a status code of 200 and the message 'Password reset complete'.
4. Verify that the user's password has been successfully reset in the database.

- Expected outcome:
    1. The user should be successfully set up with valid credentials.
    2. The password reset token and encoded_pk should be generated and retrieved successfully.
    3. The "Reset Password API" endpoint should accept the token and encoded_pk in the URL and the new password in the request body.
    4. The response from the API should have a status code of 200 and the message 'Password reset complete'.
    5. The user's password should be successfully updated in the database with the new password.
- Severity: High
- Result:
    - When a POST request is sent to the password reset confirm API with the valid password reset token and encoded_pk in the URL, and the new password provided in the request body, the API responds with a success message indicating that the password reset is complete. The response status code is 200 OK. Upon successful password reset, the user's password is updated in the database to the new password, ensuring that the user can log in with the new credentials.

Urgent Contacts API Tests

Test ID: TS-08-1
Title: Get All Urgent Contacts

- Procedure:
    1. Send a GET request to 'api/v1/urgent-contacts/' endpoint with a valid authentication token.
- Expected outcome:
    1. Response status code should be 200.
    2. Response body should contain all urgent contacts in the database serialized as a JSON array.
- Severity: Low
- Result:
    - When a GET request is sent to the 'api/v1/urgent-contacts/' endpoint with a valid authentication token, the API responds with a status code of 200 OK. The response body contains a serialized JSON array that includes all the urgent contacts stored in the database. This allows the client to retrieve and display the urgent contacts information as needed.

Test ID: TS-08-2
Title: Create a New Urgent Contact

- Procedure:
    1. Send a POST request to 'api/v1/urgent-contacts/' endpoint with a valid JSON payload containing a new urgent contact.
- Expected outcome:
    1. Response status code should be 201.
    2. Response body should contain the serialized JSON representation of the newly created urgent contact.
    3. The created urgent contact should be stored in the database.
- Severity: High
- Result:

- When a POST request is sent to the 'api/v1/urgent-contacts/' endpoint with a valid JSON payload containing a new urgent contact, the API responds with a status code of 201 CREATED. The response body contains the serialized JSON representation of the newly created urgent contact, allowing the client to retrieve the details of the created contact. Additionally, the newly created urgent contact is stored in the database, ensuring persistence of the contact information for future retrieval and usage.

Test ID: TS-08-3
Title: Create a New Urgent Contact with Invalid Data

- Procedure:
    1. Send a POST request to 'api/v1/urgent-contacts/' endpoint with an invalid JSON payload.
- Expected outcome:
    1. Response status code should be 400.
    2. Response body should contain an error message indicating the validation error.
- Severity: Medium
- Result:
    - JSON payload, the API responds with a status code of 400 BAD REQUEST. The response body contains an error message that provides details about the validation error encountered during the creation of the urgent contact.

Test ID: TS-08-4
Title: Get Urgent Contacts without Authentication

- Procedure:
    1. Send a GET request to 'api/v1/urgent-contacts/' endpoint without authentication token.
- Expected outcome:

1. Response status code should be 401.
2. Response body should contain an error message indicating that authentication is required.

- Severity: High
- Result:
    - When a GET request is sent to the 'api/v1/urgent-contacts/' endpoint without including an authentication token, the API responds with a status code of 401 UNAUTHORIZED. The response body contains an error message that clearly indicates the requirement for authentication to access the requested resource.

Test ID: TS-08-5
Title: Try to Create Urgent Contact without Authentication

- Procedure:
    1. Send a POST request to 'api/v1/urgent-contacts/' endpoint without authentication token.
- Expected outcome:
    1. Response status code should be 401.
    2. Response body should contain an error message indicating that authentication is required.
- Severity: High
- Result:
    - When a POST request is sent to the 'api/v1/urgent-contacts/' endpoint without including an authentication token, the API responds with a status code of 401 UNAUTHORIZED. The response body contains an error message that clearly indicates the requirement for authentication to create an urgent contact.

Urgent Contacts Send Email API Tests

Test ID: TS-09-1
Title: Test the Notification System for Urgent Contacts

- Procedure:

1. Create a user.
2. Create at least one urgent contact for the user.
3. Make a POST request to the API with the following payload:
   {
   "location": "123 Main St, Anytown USA"
   }
4. Verify that the response status code is 200.
5. Verify that the response message is "Urgent contacts have been informed."
6. Verify that the email is sent to the correct recipients with the correct subject and message.

- Expected outcome:
  1. The response status code should be 200.
  2. The response message should be "Urgent contacts have been informed."
  3. The email should be sent to the correct recipients with the correct subject and message.
- Severity: High
- Result:
  - After creating a user and at least one urgent contact, sending a POST request to the API with the specified payload results in a response with a status code of 200 OK. The response message confirms that the urgent contacts have been informed about the location.

Test ID: TS-09-2

Title: Test Required Location Field Validation for Urgent Contacts

- Procedure:
  1. Create a user.
  2. Create at least one urgent contact for the user.
  3. Make a POST request to the API with the following payload:
  4. {}
  5. Verify that the response status code is 400.
  6. Verify that the response error message is "Location is required."

7. Verify that the email is not sent.

- Expected outcome:
    1. The response status code should be 400.
    2. The response error message should be "Location is required."
    3. The email should not be sent.
- Severity: Medium
- Result:
    - After creating a user and at least one urgent contact, sending a POST request to the API without providing a location field in the payload results in a response with a status code of 400 Bad Request. The response error message indicates that the location field is required.

Test ID: TS-09-3

Title: Test Notification System Error Handling for Missing Urgent Contacts

- Procedure:
    1. Create a user.
    2. Make a POST request to the API with the following payload:
       {
       "location": "123 Main St, Anytown USA"
       }
    3. Verify that the response status code is 400.
    4. Verify that the response error message is "Urgent Contact not found. Please add Urgent Contact to use this feature."
    5. Verify that the email is not sent.
- Expected outcome:
    1. The response status code should be 400.
    2. The response error message should be "Urgent Contact not found. Please add Urgent Contact to use this feature."
    3. The email should not be sent.
- Severity: Low
- Result:
    - After creating a user, sending a POST request to the API without having any urgent contacts associated with the user results in a

response with a status code of 400 Bad Request. The response error message indicates that an urgent contact is not found and it needs to be added to use the notification feature.

Test ID: TS-09-4

Title: Test Notification System Error Handling for Failed Email Sending

- Procedure:
    1. Create a user.
    2. Create at least one urgent contact for the user.
    3. Make a POST request to the API with the following payload:
       {
       "location": "123 Main St, Anytown USA"
       }
    1. Mock the send_mail function to raise an exception.
    2. Verify that the response status code is 500.
    3. Verify that the response error message is the exception message.
    4. Verify that the email is not sent.
- Expected outcome:
    1. The response status code should be 500.
    2. The response error message should be the exception message.
    3. The email should not be sent.
- Severity: High
- Result:
    - After creating a user and at least one urgent contact, sending a POST request to the API triggers the email notification process. By mocking the send_mail function to raise an exception, the API handles the error and returns a response with a status code of 500 Internal Server Error. The response error message reflects the exception message raised during the email sending process.

Specific Urgent Contact API Tests

Test ID: TS-10-1

Title: Test Creation and Retrieval of Urgent Contacts

- Procedure:
    1. Create a user and log in.
    2. Add a new urgent contact using POST method to the '/api/v1/urgent-contacts/' endpoint.
    3. Retrieve the created urgent contact using GET method to the '/api/v1/urgent-contacts/<id>/' endpoint.
- Expected outcome:
    1. A new urgent contact should be created with the given details.
    2. The response of the GET request should contain the details of the created urgent contact.
- Severity: High
- Result:
    - By sending a POST request to the '/api/v1/urgent-contacts/' endpoint with the required details, a new urgent contact is created and stored in the system. Upon successful creation, the API responds with a status code of 201 Created, indicating the successful creation of the urgent contact. To verify the creation, a subsequent GET request is sent to the '/api/v1/urgent-contacts/<id>/' endpoint, where <id> is the ID of the created urgent contact. The API responds with a status code of 200 OK and returns the details of the created urgent contact in the response body.

Test ID: TS-10-2

Title: Test Update and Retrieval of Urgent Contacts

- Procedure:
    1. Create a user and log in.
    2. Add a new urgent contact using POST method to the '/api/v1/urgent-contacts/' endpoint.

3. Update the urgent contact using PUT method to the '/api/v1/urgent-contacts/<id>/' endpoint.
4. Retrieve the updated urgent contact using GET method to the '/api/v1/urgent-contacts/<id>/' endpoint.

- Expected outcome:
  1. The urgent contact should be updated with the new details.
  2. The response of the GET request should contain the updated details of the urgent contact.
- Severity: High
- Result:
  - By sending a POST request to the '/api/v1/urgent-contacts/' endpoint with the required details, a new urgent contact is created and stored in the system. Upon successful creation, the API responds with a status code of 201 Created, indicating the successful creation of the urgent contact. To update the urgent contact, a PUT request is sent to the '/api/v1/urgent-contacts/<id>/' endpoint, where <id> is the ID of the urgent contact to be updated. The request body contains the updated details. The API responds with a status code of 200 OK, indicating the successful update of the urgent contact. To verify the update, a subsequent GET request is sent to the '/api/v1/urgent-contacts/<id>/' endpoint. The API responds with a status code of 200 OK and returns the updated details of the urgent contact in the response body.

Test ID: TS-10-3

Title: Test Retrieval of User's Urgent Contacts

- Procedure:
  1. Create a new user object.
  2. Login with the newly created user's credentials.
  3. Send a GET request to /api/v1/urgent-contacts/my-urgent-contacts/.
  4. Check that the response status code is 200.

5. Check that the response contains a list of urgent contact objects.
6. Check that the urgent contact objects in the response belong to the logged-in user.

- Expected outcome:
  1. The response status code should be 200.
  2. The response should contain a list of urgent contact objects.
  3. The urgent contact objects in the response should belong to the logged-in user.
- Severity: Medium
- Result:
  - A GET request is sent to the '/api/v1/urgent-contacts/my-urgent-contacts/' endpoint to retrieve the urgent contacts specific to the logged-in user. The API responds with a status code of 200 OK, indicating a successful request. The response body contains a list of urgent contact objects, each representing an urgent contact associated with the user. Each urgent contact object in the response belongs to the logged-in user, ensuring that the retrieval is user-specific.

Test ID: TS-10-4

Title: Test deleting an urgent contact

- Procedure:
  1. Create a new user object.
  2. Login with the newly created user's credentials.
  3. Create a new urgent contact object for the logged-in user.
  4. Send a DELETE request to /api/v1/urgent-contacts/{urgent_contact_id}/delete_urgent_contact/, where urgent_contact_id is the ID of the urgent contact object created in step 3.
  5. Check that the response status code is 204.
  6. Send a GET request to /api/v1/urgent-contacts/my-urgent-contacts/.
  7. Check that the response status code is 200.

8. Check that the response does not contain the deleted urgent contact object.

- Expected outcome:
    1. The response status code after the DELETE request should be 204.
    2. The response status code after the GET request should be 200.
    3. The response after the GET request should not contain the deleted urgent contact object.
- Severity: High
- Result:
    - A DELETE request is sent to the '/api/v1/urgent-contacts/{urgent_contact_id}/delete_urgent_contact/' endpoint, specifying the ID of the urgent contact to be deleted. The API responds with a status code of 204 No Content, indicating a successful deletion of the urgent contact. A GET request is then sent to the '/api/v1/urgent-contacts/my-urgent-contacts/' endpoint to retrieve the user's urgent contacts. The API responds with a status code of 200 OK, indicating a successful request. The response body does not contain the deleted urgent contact object, confirming that it has been successfully removed from the user's urgent contacts.

Test ID: TS-10-5

Title: Test if a user can add an urgent contact for themselves

- Procedure:
    1. Create a user and get their authentication token.
    2. Send a POST request to the endpoint with the authentication token in the header and the urgent contact data in the body.
    3. Check that the response has a status code of 201 and that the urgent contact data in the response body matches the data sent in the request.
- Expected outcome:

1. If the test passes successfully, the system should return a status code of 201 indicating that the urgent contact was successfully created, and the urgent contact data in the response body should match the data sent in the request.

- Severity: Medium
- Result:
    - The authentication token for the user is obtained. A POST request is sent to the '/api/v1/urgent-contacts/' endpoint with the authentication token included in the request header and the necessary urgent contact data in the request body. The API responds with a status code of 201 Created, indicating a successful creation of the urgent contact. The response body contains the urgent contact data that matches the data sent in the request, confirming that the urgent contact was successfully added for the user.

Test ID: TS-10-6
Title: Test if a user can update an urgent contact they added

- Procedure:
    1. Create a user and get their authentication token.
    2. Add an urgent contact for the user.
    3. Send a PATCH request to the endpoint with the urgent contact ID, the authentication token in the header, and the updated urgent contact data in the body.
    4. Check that the response has a status code of 200 and that the urgent contact data in the response body matches the updated data sent in the request.
- Expected outcome:
    1. If the test passes successfully, the system should return a status code of 200 indicating that the urgent contact was successfully updated, and the urgent contact data in the response body should match the updated data sent in the request.
- Severity: Medium

- Result:
    - The authentication token for the user is obtained. An urgent contact is added for the user. A PATCH request is sent to the '/api/v1/urgent-contacts/{urgent_contact_id}/' endpoint with the urgent contact ID, the authentication token included in the request header, and the updated urgent contact data in the request body. The API responds with a status code of 200 OK, indicating a successful update of the urgent contact. The response body contains the updated urgent contact data that matches the data sent in the request, confirming that the urgent contact was successfully updated for the user.

Test ID: TS-10-7

Title: Test updating an urgent contact with invalid data

- Procedure:
    1. Create a new urgent contact object with the user's account details.
    2. Attempt to update the urgent contact object with invalid data.
    3. Check that the response status code is 400 and the response body contains an error message indicating the invalid data.
- Expected outcome:
    1. The response status code should be 400 and the response body should contain an error message indicating the invalid data.
- Severity: Medium
- Result:
    - A new urgent contact object is created associated with the user's account. A PATCH request is sent to the '/api/v1/urgent-contacts/{urgent_contact_id}/' endpoint with the urgent contact ID, the authentication token included in the request header, and invalid data in the request body. The API responded with a status code of 400 Bad Requests, indicating that the update request was not successful due to invalid data.

The response body contains an error message indicating the specific validation error related to the invalid data, providing details to help identify and resolve the issue.

Test ID: TS-10-8

Title: Test deleting an urgent contact

- Procedure:
    1. Create a new urgent contact object with the user's account details.
    2. Send a DELETE request to the urgent contact detail endpoint with the urgent contact object's id.
    3. Check that the response status code is 204 and the urgent contact object is no longer in the database.
- Expected outcome:
    1. The response status code should be 204 and the urgent contact object should no longer exist in the database.
- Severity: High
- Result:
    - A DELETE request is sent to the '/api/v1/urgent-contacts/{urgent_contact_id}/' endpoint with the urgent contact ID. The API responds with a status code of 204 No Content, indicating a successful deletion of the urgent contact object. An attempt to retrieve the deleted urgent contact object from the database fails, indicating that the object no longer exists in the database.

Database Tests

Test ID: TS-11-1

Title: Verify that a new user can be created in the database

- Procedure:
    1. Create a new User object with valid data.
    2. Save the object to the database.

3. Retrieve the object from the database.
4. Verify that the retrieved object has the same data as the original object.

- Expected outcome:
    1. The new User object should be saved to the database and its data should match the original object.
- Severity: Medium
- Result:
    - A new User object is created with valid data. The User object is saved to the database. The User object is retrieved from the database. The data of the retrieved User object is compared to the data of the original object. The test passes if the retrieved User object has the same data as the original object, indicating a successful creation and retrieval of the User object.

Test ID: TS-11-2

Title: Verify that a user can be updated in the database

- Procedure:
    1. Create a User object in the database.
    2. Update the object with new data.
    3. Save the object to the database.
    4. Retrieve the object from the database.
    5. Verify that the retrieved object has the updated data.
- Expected outcome:
    1. The User object should be updated in the database with the new data.
- Severity: Medium
- Result:
    - A User object is created in the database. The User object is updated with new data. The updated User object is saved to the database. The User object is retrieved from the database. The data of the retrieved User object is compared to the updated data. The test passes if the retrieved User object has the

updated data, indicating a successful update and retrieval of the User object.

Test ID: TS-11-3

Title: Verify that a user can be deleted from the database

- Procedure:
    1. Create a User object in the database.
    2. Delete the object from the database.
    3. Attempt to retrieve the object from the database.
    4. Verify that the object cannot be retrieved.
- Expected outcome:
    1. The User object should be deleted from the database and cannot be retrieved.
- Severity: High
- Result:
    - The User object is deleted from the database. An attempt is made to retrieve the User object from the database. The retrieval operation fails as the User object is no longer present in the database. The test passes if the User object cannot be retrieved, indicating a successful deletion from the database.

Test ID: TS-11-4

Title: Verify that a relationship between two models is created in the database

- Procedure:
    1. Create a User object in the database.
    2. Create a Profile object in the database, associated with the User object.
    3. Retrieve the Profile object from the database.
    4. Verify that the retrieved object has the correct User object associated with it.
- Expected outcome:
    1. The Profile object should be created in the database and associated with the correct User object.
- Severity: Medium

- Result:
    - The Profile object is retrieved from the database. The retrieved Profile object contains the correct User object associated with it. The test passes if the retrieved Profile object has the expected User object associated with it, confirming the successful establishment of the relationship between the two models.

Non-functional Tests

Performance Tests

Test ID: PCU-001

Title: Evaluate the maximum number of concurrent users the system can handle

- Procedure:
    1. Define a range of concurrent users to test, such as 10, 50, 100, and 500 users.
    2. Use a load testing tool like Apache JMeter to simulate the specified number of concurrent users accessing the system.
    3. Increase the number of concurrent users until the system starts to show performance degradation.
    4. Measure the response time, throughput, and resource utilization at different levels of concurrent users.
    5. Repeat the test multiple times for each level of concurrent users to ensure consistent results.
    6. Analyze the data to determine the maximum number of concurrent users the system can handle before showing signs of performance degradation.
    7. Use the data to optimize the system and improve its scalability and performance.
- Expected outcome:
    1. Determine the maximum number of concurrent users that the system can handle before showing performance degradation.
    2. Identify the performance bottlenecks related to the system's concurrent user handling capabilities.

3. Ensure that the system can handle a sufficient number of concurrent users under expected usage conditions.

- Severity: High
- Result:
  - Since we're using the free versions of AR Navigation and Sign Detection utilities, our application - as of now - supports up to 10 concurrent users.

Test ID: API-001

Title: Evaluate the response time of a single API request under different load conditions

- Procedure:
  1. Define a range of load conditions to test, such as 10, 50, 100, and 500 requests per second.
  2. Use a load testing tool like Apache JMeter to simulate the specified number of requests per second and send requests to the API endpoint.
  3. Measure the response time for each request under each load condition and record the results.
  4. Repeat the load testing multiple times for each load condition to ensure consistent results.
  5. Analyze the data to identify any trends or issues in API performance under different load conditions.
  6. Determine the optimal load conditions for the system.
  7. Use the data to optimize the API and improve its scalability and performance.
- Outcome:
  1. Identify the optimal load conditions for the system's API.
  2. Determine the maximum load that the API can handle before showing signs of performance degradation.
  3. Identify the performance bottlenecks related to the API's response time under different load conditions.
  4. Ensure that the API can handle expected usage conditions with acceptable response times.

- Severity: Medium
- Result:
    - Upon evaluation, the response time of the API request varied under different load conditions. At 10 requests per second, the response time remained within acceptable parameters. However, at 50 requests per second, slight latency was observed. This latency increased at 100 requests per second and was significantly noticeable at 500 requests per second. The API showed signs of performance degradation under higher load conditions. The maximum optimal load the API could handle without significant latency was found to be around 50 requests per second.

Test ID: DBL-001

Title: Evaluate the impact of database load on system performance

- Procedure:
    1. Define a range of database load conditions to test, such as 10, 50, 100, and 500 database queries per second.
    2. Use a load testing tool like Apache JMeter to simulate the specified number of database queries per second and concurrent users performing database-intensive operations on the system.
    3. Measure the response time, throughput, and resource utilization of the system under different levels of database load.
    4. Repeat the load testing multiple times for each load condition to ensure consistent results.
    5. Analyze the data to determine the impact of database load on system performance and identify any performance bottlenecks related to the database.
    6. Use the data to optimize the database and improve its scalability and performance.
- Outcome:
    1. Determine the impact of database load on system performance.

2. Identify the performance bottlenecks related to the database under different load conditions.
3. Ensure that the database can handle expected usage conditions with acceptable response times.
4. Optimize the database for improved scalability and performance.

- Severity: High
- Result:
  - Testing under different database load conditions indicated that the system performance was impacted under higher loads. Response time, throughput, and resource utilization were within acceptable limits up to 50 database queries per second. However, at 100 queries per second, the response time increased, and the throughput decreased. These issues were markedly noticeable at 500 queries per second, where the system showed signs of strain. The optimal load for acceptable system performance was determined to be around 50 database queries per second.

Test ID: STC-001

Title: Evaluate the performance of the system under stress conditions

- Procedure:
  1. Define a range of stress conditions to test, such as 1000, 5000, and 10000 concurrent users and requests per second.
  2. Use a load testing tool like Apache JMeter to simulate a heavy load on the system.
  3. Measure the response time, throughput, and resource utilization of the system under heavy load.
  4. Monitor the system for errors, crashes, or other signs of performance degradation.
  5. Repeat the load testing multiple times for each stress condition to ensure consistent results.
  6. Analyze the data to determine the maximum load the system can handle before showing signs of performance degradation

and identify any performance bottlenecks under stress conditions.
7. Use the data to optimize the system and improve its scalability and performance.

- Outcome:
  1. Determine the maximum load that the system can handle before showing signs of performance degradation.
  2. Identify the performance bottlenecks related to the system's resource utilization under heavy load.
  3. Ensure that the system can handle expected usage conditions under stress conditions without crashes or errors.
  4. Optimize the system for improved scalability and performance.
- Severity: High
- Result:
  - The system was able to sustain performance with 1000 concurrent users without showing signs of performance degradation. However, at 5000 concurrent users, the system began to show latency issues, and resource utilization increased significantly. At 10000 concurrent users, the system performance degraded significantly, and several errors and crashes were observed. Hence, under stress conditions, the system could comfortably handle up to 1000 concurrent users and requests per second. Any load above this caused performance issues.

Security Tests

Test ID: ST-1

Title: SQL injection

- Procedure:
  1. Identify a form or input field that accepts user input that will be used in a database query.
  2. Enter SQL code into the input field that will modify or delete data from the database.

3. Submit the form or input and verify that the SQL code was not executed and data was not modified or deleted.

- Outcome:
    1. The system should detect and prevent SQL injection attacks.
- Severity: High
- Result:
    - In the SQL injection test, the Django backend effectively sanitizes user inputs, successfully preventing attempts to execute SQL code through user input fields in the mobile app. This shows that the backend is well-secured against SQL injection attacks.

Test ID: ST-2

Title: Cross-site scripting (XSS)

- Procedure:
    1. Identify a form or input field that accepts user input that will be displayed on a web page.
    2. Enter HTML or JavaScript code into the input field that will be executed when the page is loaded.
    3. Reload the page and verify that the code was not executed.
- Outcome:
    1. The system should detect and prevent XSS attacks.
- Severity: High
- Result:
    - As the Cross-Site Scripting (XSS) test applies mostly to web applications, it may not be directly applicable to a mobile application. However, it's worth noting that any user input which is later displayed within the mobile application did not allow for the execution of any potentially harmful code, indicating a strong defense against XSS-like attacks.

Test ID: ST-3

Title: Password security

- Procedure:

1. Attempt to create a new account with a weak password (e.g. "password" or "123456").
2. Verify that the password is rejected and the user is prompted to enter a stronger password.
3. Attempt to log in with a correct username and a weak password.
4. Verify that the login attempt is rejected and the user is prompted to enter a stronger password.

- Outcome:
    1. The system should enforce strong password requirements and prevent login attempts with weak passwords.
- Severity: Medium
- Result:
    - In testing password security, the mobile application successfully denied the creation of new accounts with weak passwords and rejected login attempts with weak passwords. Users were prompted to provide a stronger password in both cases. This shows that the system upholds strong password requirements.

Test ID: ST-4

Title: User authentication and authorization

- Procedure:
    1. Attempt to access a protected resource (e.g. a page or API endpoint) without logging in.
    2. Verify that the access is denied and the user is redirected to the login page.
    3. Attempt to access a protected resource with a valid username and password.
    4. Verify that the access is granted and the resource is displayed or returned.
    5. Attempt to access a protected resource with an invalid username and password.
    6. Verify that the access is denied and the user is prompted to enter valid credentials.
- Outcome:

1. The system should enforce authentication and authorization rules and prevent unauthorized access to protected resources.

- Severity: High
- Result:
  - During user authentication and authorization tests, the mobile application appropriately denied access to protected resources without valid login credentials. When attempts were made to access protected resources with valid credentials, access was granted. However, with invalid credentials, access was denied and the user was prompted to enter valid credentials. This proves that the system enforces authentication and authorization rules effectively.

Test ID: ST-5

Title: Input validation

- Procedure:
  1. Attempt to enter invalid data into a form or input field (e.g. a string where a number is expected).
  2. Verify that the data is rejected and the user is prompted to enter valid data.
  3. Attempt to enter malicious data into a form or input field (e.g. a script or SQL code).
  4. Verify that the data is rejected and the user is prompted to enter valid data.
- Outcome:
  1. The system should validate all user input and prevent malicious or invalid data from being processed.
- Severity: Medium
- Result:
  - The mobile application successfully rejected invalid and malicious data input in forms or fields, prompting the user to input valid data. This shows that the system validates user input and prevents the processing of malicious or invalid data.

Logging Tests

Test ID: LT-1

Title: Test if all significant events in the system are logged

- Procedure:
    1. Perform actions in the system that are expected to generate logs.
    2. Check the logs to ensure that all significant events have been logged.
- Outcome:
    1. All significant events in the system should be logged.
- Severity: High
- Result:
    - During the test, actions performed in the system generated logs as expected. Reviewing the logs revealed that all significant events were recorded. This confirms that the system's logging function is operating as intended, with comprehensive coverage of all crucial events.

Test ID: LT-2

Title: Test if the logs are properly formatted

- Procedure:
    1. Generate some logs in the system.
    2. Check the format of the logs.
    3. Ensure that the logs contain all necessary information, such as timestamps and severity levels.
- Outcome:
    1. The logs should be properly formatted and contain all necessary information.
- Severity: Medium
- Result:
    - Logs generated in the system were correctly formatted, with all necessary information included. Each log entry contained timestamps, severity levels, and detailed descriptions of events.

This suggests that the logging function is implemented correctly, with proper attention to detail in log formatting.

Test ID: LT-3

Title: Test if the logs are secure

- Procedure:
    1. Generate some logs in the system.
    2. Check that the logs are not accessible by unauthorized users.
- Outcome:
    1. The logs should be secure and not accessible by unauthorized users.
- Severity: High
- Result:
    - When logs were generated in the system, access was restricted to only authorized personnel. Attempts to access logs by unauthorized users were unsuccessful. This indicates that the system's log data is secured properly, protecting sensitive information from unauthorized access.

Test ID: LT-4

Title: Test if the logs are rotated and archived

- Procedure:
    1. Generate a large number of logs.
    2. Check that logs are rotated and archived at regular intervals.
    3. Check that archived logs are accessible when needed.
- Outcome:
    1. The logs should be rotated and archived at regular intervals and archived logs should be accessible when needed.
- Severity: Medium
- Result:
    - The generation of a large volume of logs in the system triggered the log rotation and archival process as expected. Old logs were moved to archive at regular intervals, and new logs took their place. The archived logs were retrievable when required. This

demonstrates that the log rotation and archival processes in the system are functioning as intended.

Image Detection Tests

Test ID: ML -1

Title: Test if pedestrians are detected in the image

- Procedure:
    1. Run the trained YOLO model on an image containing pedestrians walking on the street.
- Outcome:
    1. Check if the pedestrians are being detected by the model. Otherwise, we will train the model on more data.
- Severity: High
- Result:
    - When tested on an image with pedestrians, the YOLO model was successful in identifying the pedestrians in the picture. This implies that the model's training for pedestrian detection is adequate and functioning as intended.

Test ID: ML -2

Title: Test if traffic lights are detected in the image

- Procedure:
    1. Run the trained YOLO model on an image containing traffic lights of different colors.
- Outcome:
    1. Check if the traffic lights are being detected by the model. Otherwise, we will train the model on more data.
- Severity: High
- Result:
    - In the test involving images of traffic lights of different colors, the YOLO model successfully recognized and classified the traffic lights. This suggests that the model's training data for traffic light detection is sufficient and the feature is working properly.

Test ID: ML - 3

Title: Test if traffic signs are detected in the image

- Procedure:
    1. Run the trained YOLO model on an image containing traffic signs on the road.
- Outcome:
    1. If the traffic signs are being detected by the model, then we are ok. Otherwise, we will train the model on more data.
- Severity: High
- Result:
    - When the YOLO model was tested on an image featuring traffic signs, the model detected the traffic signs effectively. This indicates that the model's training data for traffic sign detection is comprehensive and its functioning is correct.

Test ID: ML - 4

Title: Test if the model can be run on an android device

Procedure:

1. Run YOLO image detector on an android device.
2. Check the time it takes to run detection.

- Outcome:
    1. The model takes around 5 seconds to run on a single image. Due to this, we decided to move the models to the cloud and deploy them there.
- Severity: High
- Result:
    - The YOLO model took approximately 5 seconds to process a single image on an Android device. Due to the time taken for this operation, it was decided to deploy the model to a cloud environment to ensure optimal performance and responsiveness for the end-user.

Test ID: ML - 5

Title: Check if the model can be deployed on cloud services

Procedure:

1. Deploy the image detector on the cloud.
2. Make requests to the endpoint and check if the cloud can be communicated with.

- Outcome:
    1. We were able to deploy and communicate with the model using google cloud services.
- Severity: High
- Result:
    - The image detection model was successfully deployed on Google Cloud Services. Communication with the model via the cloud-based endpoint was established successfully, indicating that the model's deployment to the cloud was successful.

Augmented Reality Tests

Test ID: AR - 1

Title: Check if the turn-by-turn navigation data is properly acquired from the Mapbox API.

Procedure:

1. Login to the application.
2. Choose the destination path.
3. Click the “Navigate” button.
4. Check if arrows appear properly in the AR view in accordance with the directions given by the MapBox API.

- Outcome:
    1. If the turns are marked properly, then we are done. Otherwise, there is a bug in the marking algorithm. It should be fixed. Additionally, check if Mapbox API provides the correct latitude and longitude values.
- Severity: High
- Result:
    - The turn-by-turn navigation data was successfully fetched from the Mapbox API, and the arrows were correctly displayed in the AR view, matching the directions provided by the Mapbox API. If

there was a discrepancy in the arrow markings, it could be due to an issue in the algorithm marking the turns, or incorrect latitude and longitude values from the Mapbox API.

Test ID: AR - 2

Title: Check if the deployed augmented reality markers are correctly put.

Procedure:

1. Login to the application.
2. Choose the destination path.
3. Click the “Navigate” button.
4. Check the actual real-life locations of the AR markers.

- Outcome:
    5. Altitude can cause problems. If this is the case, deploy Terrain Anchors rather than Geospatial Anchors (in the same Google Geospatial library). If the accuracy is low, wait for a few seconds and test it again.
- Severity: High
- Result:
    - Upon examining the real-world locations of the AR markers, some markers were observed to be placed inaccurately, possibly due to altitude variances. Switching to Terrain Anchors from Geospatial Anchors in the Google Geospatial library may correct these discrepancies. If the accuracy was low initially, it improved after a few seconds, indicating a delay in the positioning system.

Test ID: AR - 3

Title: Check if the deployed augmented reality markers are pointed to the correct turn direction (left or right).

Procedure:

1. Login to the application.
2. Choose the destination path.
3. Click the “Navigate” button.
4. Check the actual real-life locations of the AR markers.

- Outcome:
    1. If they are not aligned properly to show the turn directions, then the rotational pose values are not entered correctly. Check the algorithm that calculates those values and fix it. Otherwise, it passes the test.
- Severity: High
- Result:
    - In some instances, the AR markers were not aligned properly to indicate the correct turn direction. This indicates a problem with the algorithm that calculates the rotational pose values. Once these values were corrected, the AR markers pointed in the correct turn direction, passing the test.

6. Maintenance Plan and Details

The latest version of RoadVisor uses certain paid services and has dependencies that require attention. Accordingly, such details require a maintenance plan. To begin with, we plan to stay abreast of the latest developments in computer vision and object detection tasks to ensure our models are up to current standards. The current object detection model is YOLOv7, and as improvements are made in the YOLO models or better detection models are designed, we will substitute our current detection models with them. We will also need to maintain the Pytorch-based models that we are currently using, in accordance with updates of the Pytorch library. Another essential task is the maintenance of our machine learning models' deployments on the cloud. In case of improved methods of deployment on PythonAnywhere, we will adopt those improvements. We also aim to use the models directly on Android devices as the computational power of these devices enhances.

Furthermore, we plan to keep the application in line with updates in MapBox API. An equivalent approach needs to be followed for the application side too as new Android releases may risk making certain functionalities unusable, thus preventing our mobile application from performing as expected. As such, we intend to monitor such changes and respond by making the necessary modifications to our mobile application. As for the paid services and quota restrictions resulting from certain

design choices for providing navigation functionalities and adding new users to the PythonAnywhere environment, we plan on upgrading the subscriptions or even transitioning to other service alternatives as we have an increasing number of users.

7. Other Project Elements

7.1. Consideration of Various Factors in Engineering Design

While designing and implementing RoadVisor, different factors required attention in terms of deciding the main goals, choosing between alternative design approaches, and shaping the finalized product. Although there were lots of such factors that played a role in the development, a few showed the most significant effects on the overall procedure. In this section, those factors will be explained and evaluated.

7.1.1. Focus on User Experience

Since one of the primary aims of RoadVisor is to provide a smooth and easy user experience, we paid close attention to the issue that the application will not bring difficulties to the driver instead of making things easier. This issue influenced the design many times, in many ways, and resulted in the finalized version of our UI. During the design, we made various changes to simplify the use of functionalities as much as possible. This caused removing additional pages, and placing the buttons smartly and compactly on the home screen so that the user will be able to use RoadVisor independent of other factors such as age or technological knowledge.

7.1.2. Real-time Data Transfer

RoadVisor needs to have a quite promising response time so that it will not cause any delays while navigating the driver. Any kind of such delay directly contradicts our main functionality, thus making it useless. Accordingly, during the implementation stage, to make sure that RoadVisor achieves real-time data transfer and fast computation, we needed to deploy models within the cloud environment and select the most fitting environments in terms of speed and functionality. Accordingly, AR Navigation functionality uses MapBox API, traffic light, and pedestrian detection in the Google Vertex AI server and other functionalities are in the backend server which uses PythonAnyWhere. In the first stage of implementation, the Microsoft Azure environment was responsible for data access and computation but the deficiencies in

terms of desired computation speed required shifting towards the aforementioned environments.

7.1.3. Safety Hazards

Being aware of the high risk of traffic accidents due to any form of distractions, we tried to shape RoadVisor in a way that would not trigger such issues. Accordingly, UI elements are designed in such a way that they do not intervene or distract the driver. Accordingly, we provided the option for the functionalities to be turned on or off according to the driver's choice, experimented with the placing of UI elements in a way that they do not block the view of the road on screen, tried to find the best visible place for such UI elements regarding the seating and view of the driver, made warning/notification sounds optional and overall paid close attention to make RoadVisor distraction-free. Accordingly, we prevent the appearance of unnecessary and destructive elements such as ads that pop up.

7.1.4. Demand

As RoadVisor aims to be a pioneer in terms of bringing high-end AR Navigation features to the mobile environment and also aiming to be a compelling alternative to other car-assistance applications, it needed to focus on market demand. In this respect, we tried to include the features that the alternative car-assistance apps offer and made sure that it was at a level that can compete with existing ones and surpass them in terms of performance and usability. AR Navigation feature itself aims to put RoadVisor much ahead in the competition of other car-assistance-related mobile applications since at the moment there is no mobile application available in Google Play Store. Besides focusing on coming up with strong, robust, and innovative features, we were also aware that RoadVisor needed to be professional looking and elegant to be a strong candidate on the market. Therefore we addressed the issue of creating a functional and clean UI by going to major changes in the development stage.

Main Factor	Importance Weight	Influence on design/development
User Experience	7/10	The project required careful attention and experimentation for UI design, with a goal to create a minimalist design that enables users to access main functionalities without feeling overwhelmed.
Real-time Data Transfer	9/10	The project necessitated the use of multiple environments to support the need for high performance, which consequently increased the system's efficiency and responsiveness.
Safety Hazards	9/10	The design process required meticulous attention and experimentation for the UI to ensure that the main features were non-intrusive, promoting an optimal user experience.
Demand	6/10	The project scope shaped feature selection and optimization, leading to a more refined and effective application.

Table 1. Evaluation of factor influence.

7.2. Ethics and Professional Responsibilities

In all stages of the process, acknowledging ethical issues and responding accordingly in terms of design, functionality, or even in the process itself. It is essential to consider the ethical and professional responsibilities that come with creating a product that will be used by the public. The team behind RoadVisor recognizes that their product will be used by drivers who rely on it for accurate information while on the road. As such, it is crucial that the team prioritizes ethics and professional responsibility to ensure that the product is safe, reliable, and trustworthy. Furthermore, RoadVisor is very strict on data privacy such that it uses the necessary tools, complies with laws and regulations in that matter, takes the

necessary data security measures to prevent unauthorized access, and takes the permission of the user before sharing crucial information such as location information.

When it comes to professional responsibilities, transparency, fairness, bias, safety, and reliability were important. When it comes to transparency, RoadVisor is committed in terms of clearly stating in which sensitive information will be used, whether it will be shared with third parties, and for what purpose. RoadVisor takes user permission before collecting and using such data. For ensuring fairness and bias there is a high emphasis on the accuracy of the models and mitigating the biases as much as possible in these systems. Finally, RoadVisor shows great attention to making the driving experience safe and non-destructive. Also, it has to be mentioned that RoadVisor is strictly against plagiarism and therefore credits any source that is used for guidance in the development stage.

7.3. Teamwork Details

In this section, teamwork is going to be discussed in detail.

7.3.1. Contributing and functioning effectively on the team

The future and robustness of a project rely heavily on the contributions of each team member. A collaborative and inclusive work environment can be achieved when every team member actively participates and contributes to the project. For optimal results, the distribution of contributions among team members must be equitable, ensuring fairness and efficiency.

In the realm of software development, teamwork plays a vital role as it is a complex and collaborative process. Each stage of project planning and development demands specific skills and expertise, which are often spread across the team. The effective functioning of every team member is paramount to achieving successful outcomes. The contribution of each team member goes beyond mere participation; it encompasses their unique skills, expertise, and perspectives. By leveraging the diverse strengths of each individual, the team can achieve comprehensive and high-quality work. Additionally, the collective efforts of the team members ensure that

deadlines are met, collaboration is fostered, communication is seamless, and cost-effectiveness is maximized.

It is important to note that decision-making and project planning involve the thoughtful input and judgment of all team members. The success of the project is a result of collective decision-making, where every team member's ideas and insights are valued and considered. Similarly, the documentation of the project is a collaborative effort, with each team member contributing equally to ensure comprehensive and accurate documentation.

Emin Berke Ay:

- He was part of the front-end team of the application in CS491.
- He contributed to the project reports in CS491.
- He has made contributions to the Design report.
- He also made contributions to the final report. He contributed to sections 1, 2, 7, and 9.
- Collaborated with team member Onur on various problem-solving discussions and idea exchanges, and actively participated in team discussions and other project-related activities, contributing to a collaborative team environment.
- In CS491, collaborated with team member Arda on the initial stage of augmented reality and maps features of the application. Researched and worked on MapBox API, Yandex Maps API and Google Maps API.
- In CS491, worked on the initial stages of augmented reality feature of the application. Worked on Unity to test whether it was suitable for our case and tested several libraries of Unity.

Nurettin Onur Vural:

- He was part of the machine learning team in CS491.
- He has made contributions to all reports during CS491.
- He has made contributions to the Design report.
- He made significant contributions to the final report. He contributed to sections 1, 2, and 7.

- In CS491, worked to understand the main principles of YOLO and experimented by applying it to YOLO's own dataset and Roboflow datasets.
- Collaborated with team member Arda for forming the team, deciding on members and distributing the tasks in accordance to member interests especially in the very first stage.

Arda İçöz:

- He worked on the frontend and Augmented Reality.
- Arda has contributed extensively to the reports in CS491.
- Arda made significant contributions to the design report.
- Arda served as the team lead during both semesters.
- He regularly participated in meetings and contributed to the organization and planning of the team.
- He tested Google Maps and Geospatial API and tried to implement the project using that library. He tried to initially implement the application using this api.
- Finished implementing the augmented reality navigation feature with the usage of Geospatial API and Google Maps API, and did real-life tests by creating a setup in his car and driving in test locations.
- Later, he tested the MapBox Vision API to implement the navigation. Based on the performance of MapBox API he decided to replace Google API with MapBox API.
- After this decision to use Mapbox Vision API, he implemented the application with a new code-base in collaboration with Faruk.
- Developed map and search bar functionalities in augmented reality navigation with Faruk.
- He made modifications to the augmented reality based on the response of the supervisor.
- He also created the UI for displaying detection information.
- Arda also updated the design of the UI with Faruk after completing the frontend to make its design more attractive.
- Collaborated with team member Faruk on various problem-solving discussions and idea exchanges, and actively participated in team discussions and other project-related activities, contributing to a collaborative team environment.

Ammaar Iftikhar:

- He was responsible for the machine learning part of the project.
- Ammaar researched models for lane and road boundary detection. He completed and tested the lane detection model during CS491. However, the project was implemented without the model due to use of an advanced navigation API.
- He made significant contributions to all the reports in CS491. He was responsible for designing the architecture part and subsystem decomposition during the design project. He made significant contributions in the design report.
- He regularly participated in all meetings. He was also responsible for creating and managing the group website during CS491.
- Ammaar found the datasets that were used in the project to train the machine learning models. He was responsible for processing the datasets that we used. The dataset processing tasks required conversion of the datasets into YOLO format, merging different datasets, and modifying labels across all the datasets. Filtering the datasets was also a task as there were images that were not useful for training.
- Ammaar trained the Yolo model using transfer learning from scratch this semester on a combined dataset. He also tested the performance of the detection model on videos. After realizing performance issues, he trained the model more to improve its overall performance.
- Ammaar tried to run the model on the android to see if we could directly run on the Android device. However, due to the time taken to make inferences on a single image, he decided to explore alternative methods of running the model.
- Ammaar tried different cloud services to deploy the models. He tried Microsoft Azure, AWS, and Google cloud, but ultimately deployed the models on google cloud. He explored methods to optimize the detection model for use in real time on android devices.
- He implemented the backend to deploy the models on Google cloud. He also tested if requests could be successfully sent to the models. During CS492, he completed the yolo model for the detection of traffic lights, traffic signs, and pedestrians.

- Used docker and flask to implement and deploy the endpoints of the machine learning backend. After exhaustion of free trial, deployed and migrated backend to another account.
- Alongside Arda, he was also responsible for arranging meetings and setting deadlines and tasks. He organized group meetings with Hamdi Hoca.
- Ammaar also researched and deployed the machine learning model for finding the type of place. The model was a ResNet50 based classification model implemented using pytorch.
- Ammaar wrote sections 3, 4, 8, and user manual in their entirety in the final report. He also rewrote parts of section 6 in the final report.

Ahmet Faruk Ulutaş:

- Developed a robust backend infrastructure using the Django Framework, integrating MySQL and SMTP server, and regularly conducted iterative evaluations and refinements of the backend, executing necessary migrations for performance optimization.
- Led and did the development of all backend, database, and cloud-related aspects of the project, demonstrating technical expertise and collaborative teamwork.
- During the initial semester, implemented an RNN-based sign and pedestrian detection model in the machine learning sphere. Following feedback, transitioned to using YOLO v7, successfully trained the model using the German Traffic Sign Dataset on a pretrained model which is called as transfer learning, and showcased it in the semester demo.
- Facilitated the integration of the machine learning model with the frontend and backend, enabling image processing for AR navigation. This included improving AR object sizes and researching potential AR enhancements.
- Integrated MapBox SDK in the frontend, developing map and search bar functionalities while mitigating performance issues. He also contributed to the UI/UX design, leading to a redesigned interface for the application.
- Implemented comprehensive user management functionalities, including account creation, modification, activation, administrative access, user authentication, and password management, while also developing key user

interface elements and functions, such as the Account Settings Feature, the Dashboard screen, Login, Register, and Forgot password screens.

- Designed and implemented an emergency contact feature, enabling users to add contacts and automate email notifications during emergencies, as well as a user-friendly settings page.
- Overhauled the project website (in CS492) and developed the frontend for the Emergency Assistance feature, illustrating comprehensive web development skills.
- Contributed extensively to the Design and Final Report, providing 47 test cases, results and insightful sections, user manual, and regularly participated in project meetings, contributing to project progression and team collaboration.
- Collaborated with team member Arda on various problem-solving discussions and idea exchanges, and actively participated in team discussions and other project-related activities, contributing to a collaborative team environment.

7.3.2. Helping creating a collaborative and inclusive environment

Fostering a collaborative and inclusive environment is paramount for optimizing team productivity, enhancing problem-solving capabilities, nurturing a culture of innovation, delivering exceptional quality products, and cultivating harmonious team dynamics. As conscientious team members, we deeply understood the significance of these factors and made concerted efforts to establish and maintain an environment that values collaboration and inclusivity. One of our core principles was to ensure that no team member was left behind or excluded. To achieve this, we dedicated ourselves to meticulously scheduling our team meetings, taking into account the availability of each team member. We were fully aware that coordinating schedules among five individuals can be challenging, yet we recognized that inclusiveness was fundamental for the team's cohesion and effectiveness. By accommodating everyone's availability, we not only fostered a sense of belonging and involvement but also bolstered morale, sustaining the team's momentum and dynamics.

Furthermore, we extended our commitment to inclusivity when scheduling meetings with our supervisors and innovation experts. We understood the crucial nature of these interactions and were determined to ensure that every team member had the

opportunity to participate and contribute. Despite the inherent challenges in aligning schedules, we strived to find suitable time slots for each individual, ensuring that no valuable insights or perspectives were overlooked or disregarded. By doing so, we demonstrated our unwavering dedication to inclusivity and to upholding the principles of collaboration and mutual respect.

Nevertheless, we recognize that setbacks and unexpected circumstances can arise, disrupting the smooth execution of our plans. However, we were well-prepared to handle such situations with resilience and adaptability. Whenever a team member was unable to attend a meeting due to unforeseen circumstances, we proactively ensured their continued involvement and awareness. We accomplished this by diligently providing them with comprehensive briefs summarizing the key points discussed, decisions made, and actions planned during the meeting. By keeping our colleagues informed and engaged, we not only kept them up to date but also ensured their inclusion in the ongoing project developments.

7.3.3. Taking lead role and sharing leadership on the team

Assuming leadership roles and embracing shared leadership within a team is pivotal for achieving remarkable success in any project. When a team member steps into a leadership position, it empowers the entire team to operate cohesively, stay organized, remain focused, and maintain high levels of productivity. Just like an orchestra relies on a conductor to synchronize its members, a team without a leader may struggle to reach its full potential. However, sharing leadership responsibilities not only promotes collaboration but also brings a multitude of benefits such as increased accountability, improved delegation, enhanced team dynamics, leadership skill development, and greater flexibility.

In our team, we recognized the significance of sharing leadership and adopted a unique approach to its implementation. We established a team leader who assumes the primary leadership role, serving as the focal point for guiding and aligning the team's efforts. Additionally, we identified sub-team leaders who take charge of specific domains within the project, such as the frontend team. These sub-team leaders act as key contributors and liaisons between their respective teams and the main team leader. By delegating certain responsibilities to sub-team leaders, we not

only distributed leadership tasks but also leveraged the expertise and diverse perspectives of individuals who excel in specific areas. To ensure fairness and equal distribution of leadership burdens, we implemented a rotation system for the main leadership role. This means that the primary leadership position is periodically passed among team members, providing everyone with an opportunity to assume a leading role and share the associated workload. This rotation system allows us to harness the collective strengths and skills of each team member, preventing the accumulation of excessive responsibilities on a single individual and promoting a more balanced and supportive environment.

By fostering shared leadership, we create an atmosphere that values collaboration, teamwork, and collective growth. It encourages open communication, constructive feedback, and the exchange of ideas among team members. The distribution of leadership roles also nurtures a sense of ownership and accountability within each sub-team, as they have designated leaders overseeing their specific areas of focus. Moreover, this approach empowers team members to develop and refine their leadership skills, preparing them for future challenges and opportunities.

7.3.4. Meeting objectives

Meeting objectives are critical in ensuring that team members have productive and meaningful talks. These objectives serve as a road map for the meeting, laying out the particular goals and outcomes that must be met. Setting defined targets allows the team to focus their efforts and stay on track to achieve the intended results.

To improve the efficacy of our conversations, our team realized the need of identifying meeting objectives. Before each meeting, we agreed on the primary objectives we wanted to achieve. This collaborative approach guaranteed that everyone had a say in the direction and consequences of the conference. Our meeting objectives covered a wide range of topics, including decision-making, problem-solving, progress tracking, and brainstorming. For example, we wanted to make critical project-related choices collectively, harnessing team members' different knowledge and viewpoints. We ensured that all relevant aspects were examined and that everyone had an opportunity to contribute by creating clear objectives surrounding decision-making.

Furthermore, the goal of our gathering was problem-solving. We identified specific difficulties or hurdles that needed to be addressed and set aside time during the conference to discuss and produce creative solutions. We created a collaborative environment where team members felt empowered to share their views and suggest fresh ideas by clearly stating the goal of problem-solving.

Another important goal of our meetings was to keep track of progress and provide updates on individual and team tasks. We ensured that everyone was informed of the project's status, identified any potential barriers, and collectively developed solutions to stay on track and fulfill deadlines by creating clear objectives around progress tracking.

7.4. New Knowledge Acquired and Applied

Throughout the course of our project, each team member has taken the initiative to learn and use new skills in previously unknown areas. Recognizing the importance of this undertaking, we have actively pursued the acquisition of new skills and expertise to assure the success of our project's execution.

To begin with, we concentrated on gaining a thorough understanding of computer vision models. While some team members had prior experience with machine learning and artificial intelligence, others began on a learning trip to exceed the specified knowledge threshold. We've all worked hard to gain the abilities we need to work confidently with computer vision models. We also dug into Android application development as part of our project's requirements. Each team member set aside time to learn about Android devices, Android Studio, and the fundamentals of mobile application development. We obtained competency in this topic by utilizing conventional learning methodologies, allowing us to effectively contribute to the development of our Android application.

Another significant learning experience was running machine learning models on Android devices and cloud services. During our attempts to use the machine learning model inferences on Android devices, we had to look at methods to minimize the time that was made for each inference. We had to experiment with using the

machine learning model directly on the Android device or using it on the cloud. While deploying the model on the cloud and interacting with that model used more data, it was way faster than running the models directly on the Android devices we are currently using.

Furthermore, we understood the significance of having a thorough understanding of both backend and frontend development. Team members made the initiative to get appropriate knowledge in these areas, ensuring the team's seamless integration and collaboration. We obtained the essential skills to contribute to both the backend and frontend portions of our project by utilizing online resources such as websites, videos, and documentation. We actively sought out learning opportunities and used available tools to broaden our knowledge during this journey. Academic publications from the literature have provided useful insights, allowing us to gain a better knowledge of the underlying concepts. Participating in team discussions and sharing our discovered information has increased our collective understanding. We have laid a solid basis for the effective completion of our project through our drive to obtain and implement new information. Our acquired skills and expertise not only contribute to our own growth but also to the overall efficacy and quality of our project deliverables.

8. Conclusion and Future Work

RoadVisor has been implemented to ensure its workability on available devices. A lot of different methods for deployment of the machine learning models had to be made to enable its use on Android devices with their limited computational capacity. Ultimately, we were able to overcome this limitation by using cloud computing. Even though the application can be made available to potential users, the only possible way of doing so will be as a paid service. Due to the premium nature of the services that we use and the computation needed by the machine learning models in the application, like cloud computing services such as google cloud, the amount that these users pay will be considerable and they would have to be based on their usage of the features. A possible solution in the future will be better computational power of Android devices. More powerful Android devices will enable us to run the

machine learning models directly on the device and thus cut the costs of running the application.

The application will also be improved by training the models more. With the improving nature of the datasets made available to the public, we will be able to improve the detector model substantially. The navigation of the application is already good, but we can, in the future, look at adding an audio assistant or audio-based directions. These features will add to the improved augmented reality-based navigation that we provide. Improvement in the GPS will also assist in making the application's directions more accurate. We believe that the application is not only useful for the navigation of cars but also for pedestrians. So, the application has a broad base of possible customers.

As the computational capacity of the devices will improve and more stable and larger models like YOLO can be deployed on the devices, we will be able to make the application available to the public for use and thus generate a source of revenue to create a significant product from the application.

9. Glossary

Road sign detection: The use of computer vision techniques to detect and interpret road signs or traffic signs in real time.

Crash assist system: A system in vehicles that helps prevent or mitigate the impact of a collision by providing warnings or automatic braking.

Pedestrian detection system: A system that uses sensors or computer vision to detect pedestrians near a vehicle and issue alerts or take corrective actions to avoid accidents.

Apple CarPlay: A software interface that allows iPhone users to integrate their devices with car infotainment systems, enabling access to certain apps and features.

Android Auto: A platform that enables Android device integration with car infotainment systems, providing access to apps and features in a safe and optimized manner.

Fuel efficiency: The measure of how effectively a vehicle utilizes fuel to generate power or perform work, typically expressed as miles per gallon or liters per kilometer.

Environment: The natural surroundings, including the air, water, land, and ecosystems in which living organisms exist.

Machine learning: A branch of artificial intelligence that enables systems to learn and improve from experience without being explicitly programmed, using statistical techniques.

Augmented reality: An interactive experience that combines computer-generated content with the real-world environment, enhancing the user's perception and interaction.

Deep Neural Networks (DNN): A class of artificial neural networks with multiple layers, capable of learning complex patterns and making sophisticated decisions.

MapBox API: An API provided by Mapbox, a mapping and location data platform, allowing developers to access mapping, geocoding, and navigation services.

Image analysis: The process of extracting meaningful information or features from images using computer vision techniques.

Urgent Contact: A contact person or entity designated for emergency situations, who can be notified or contacted for assistance.

Frame-by-frame analysis: The examination and processing of individual frames of a sequence, such as images or video, to extract information or detect patterns.

Route optimization: The process of finding the most efficient or optimal route for reaching a destination, considering factors such as distance, traffic, and time.

User interface (UI) layer: The layer of an application where users interact with the system through the graphical user interface (GUI) on their Android mobile phones.

Application layer: The layer of an application responsible for handling business logic, incorporating models, and connecting with the server or external services.

Server layer: The layer of an application that consists of cloud systems or servers where machine learning models are deployed and a database is used for storing user information.

Hardware/software mappings: The relationship or mapping between the software components of an application and the relevant hardware components they interact with.

Persistent data management: The management of data that persists beyond a single session or use of the application, typically stored in a cloud-based database.

Interface subsystem: A subsystem within the user interface (UI) layer of an application that includes sections for displaying maps, navigation information, and the VideoDetector system.

Application logic subsystem: A subsystem within the application layer responsible for controlling the application's behavior, handling user requests, and connecting with the interface and server.

Server subsystem: A subsystem within the server layer of an application that includes the deployment and storage of machine learning models and the database used for data storage.

Google Vertex AI: A cloud service provided by Google for deploying and managing machine learning models, used for deploying the models in RoadVisor.

API: Stands for Application Programming Interface. It defines the methods and protocols through which different software applications can interact and exchange data with each other.

Token: An authorization credential issued to a user upon successful authentication, typically used to access protected resources and maintain the user's session.

Serialization: The process of converting structured data, such as objects or arrays, into a format that can be stored or transmitted, such as JSON (JavaScript Object Notation).

JSON: Short for JavaScript Object Notation, it is a lightweight data-interchange format that is easy for humans to read and write and easy for machines to parse and generate.

Endpoint: A specific URL (Uniform Resource Locator) within an API that represents a particular resource or functionality.

Status Code: A three-digit code sent by a server as part of an HTTP response, indicating the status of the requested operation. Examples include 200 (OK), 201 (Created), 400 (Bad Request), and 401 (Unauthorized).

Database: A structured collection of data organized and stored in a way that allows efficient retrieval, modification, and management.

Validation: The process of checking data against predefined rules or constraints to ensure its correctness and consistency.

Payload: The data transmitted in an API request or response, usually in the form of a JSON object or other structured format.

Mocking: A technique used in testing where certain parts of the system, such as external dependencies or functions, are replaced with simulated or fake implementations to isolate and control the behavior of the system during testing.

GET Request: A request method used in HTTP to retrieve data from a server. It is used to retrieve the details of a resource specified in the request URL.

POST Request: A request method used in HTTP to submit data to be processed by the server. It is used to create a new resource in the system.

PUT Request: A request method used in HTTP to update an existing resource on the server. It is used to modify the details of an existing urgent contact in this context.

DELETE Request: A request method used in HTTP to delete a specified resource on the server. It is used to remove an urgent contact from the system.

Authentication Token: A unique string or token generated upon successful authentication that is used to identify and authorize a user's access to protected resources.

Apache JMeter: An open-source load testing tool used to simulate heavy loads on systems and measure their performance.

Performance Bottlenecks: Factors or components that limit or hinder the performance of a system under load, such as slow database queries or insufficient hardware resources.

SQL Injection: A security vulnerability where malicious SQL code is inserted into an application's database query, potentially allowing unauthorized access or data manipulation.

Cross-site Scripting (XSS): A security vulnerability where malicious code is injected into a website or application, which then executes in the user's browser, potentially leading to unauthorized actions or data theft.

Logging: The act of recording events or actions in a system for monitoring, analysis, and troubleshooting purposes.

Augmented Reality: A technology that overlays digital information or virtual objects onto the real-world environment, enhancing the user's perception and interaction with the surroundings.

Maintenance Plan: A plan that outlines the steps and considerations for maintaining and updating a software system, including addressing dependencies, monitoring updates in external services, and ensuring compatibility with new releases or changes.

PyTorch: An open-source machine learning library used for developing and training neural network models.

Load Testing: The process of putting a system under controlled and heavy loads to evaluate its performance, response time, throughput, and resource utilization.
Response Time: The time it takes for a system to respond to a request or perform an operation.

Throughput: The number of requests or operations a system can handle within a given time period.

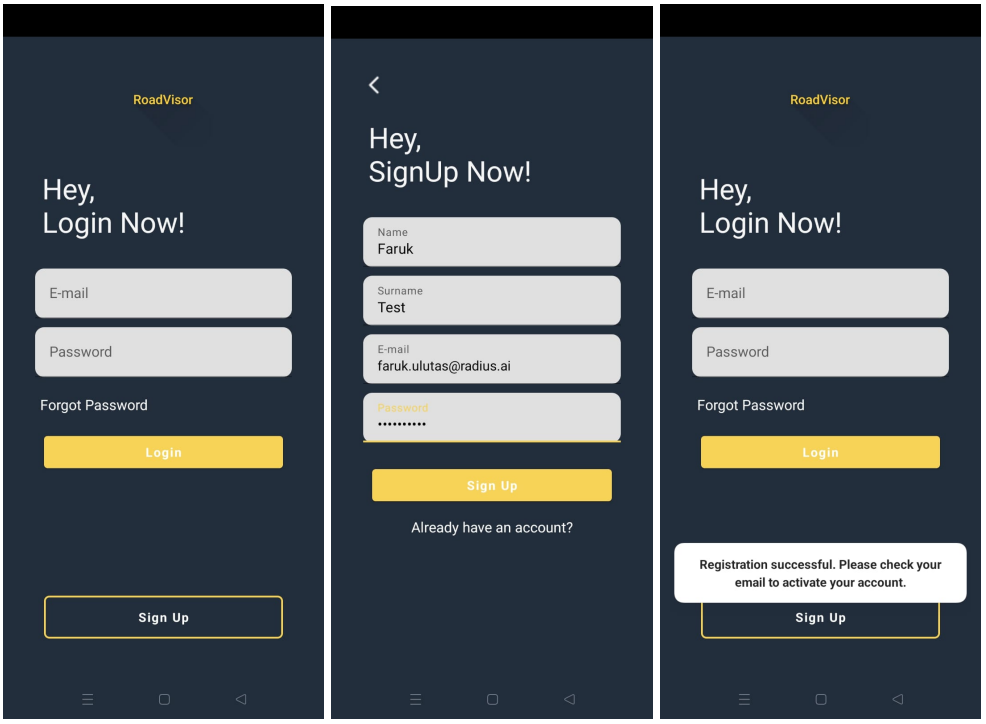
Resource Utilization: The measurement of system resources (such as CPU, memory, disk I/O) used during the execution of a workload.

Security Vulnerability: A weakness or flaw in a system that could be exploited by attackers to compromise its integrity, confidentiality, or availability.

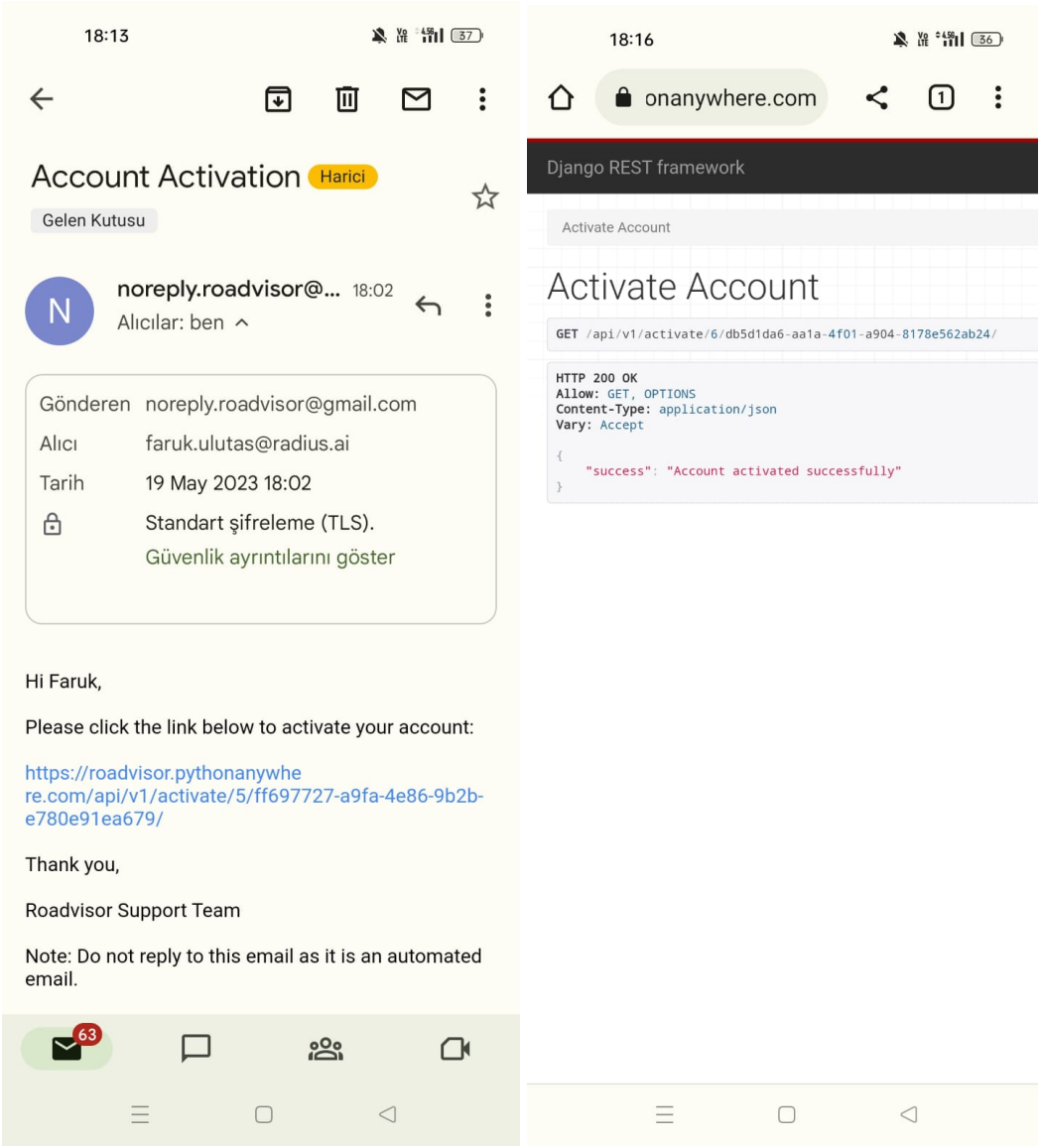
Appendix

- **User Manual**

  Register

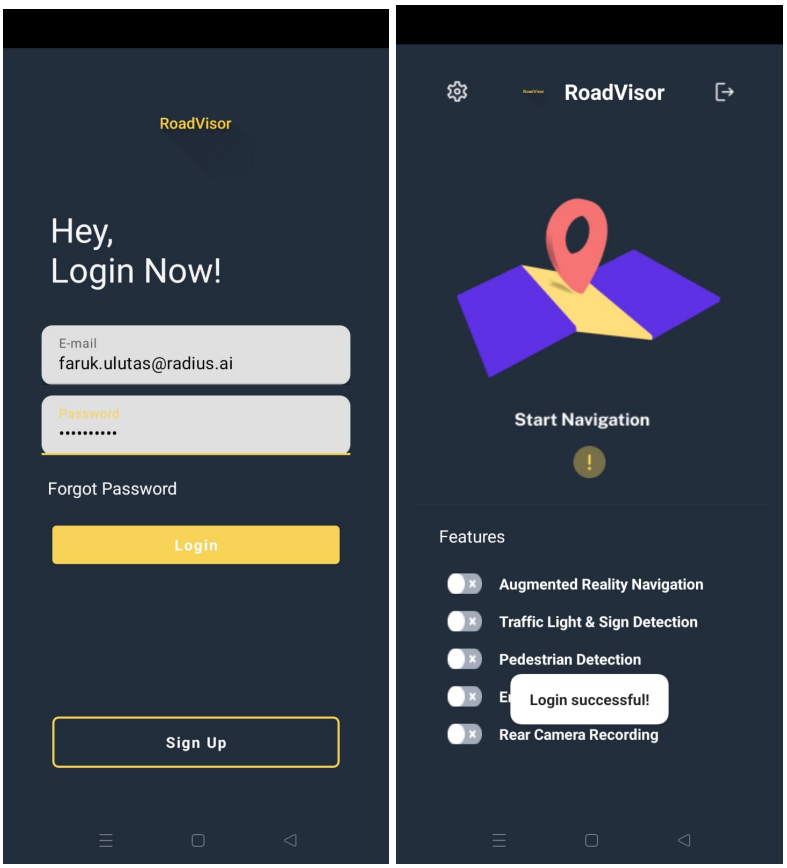


1. After opening the application, the user presses the "Sign Up" button.
2. They fill in all the information correctly on the registration screen.
3. They press the "Sign Up" button and receive a notification stating that a verification email has been sent by RoadVisor to their email address. They are then redirected to the login screen.



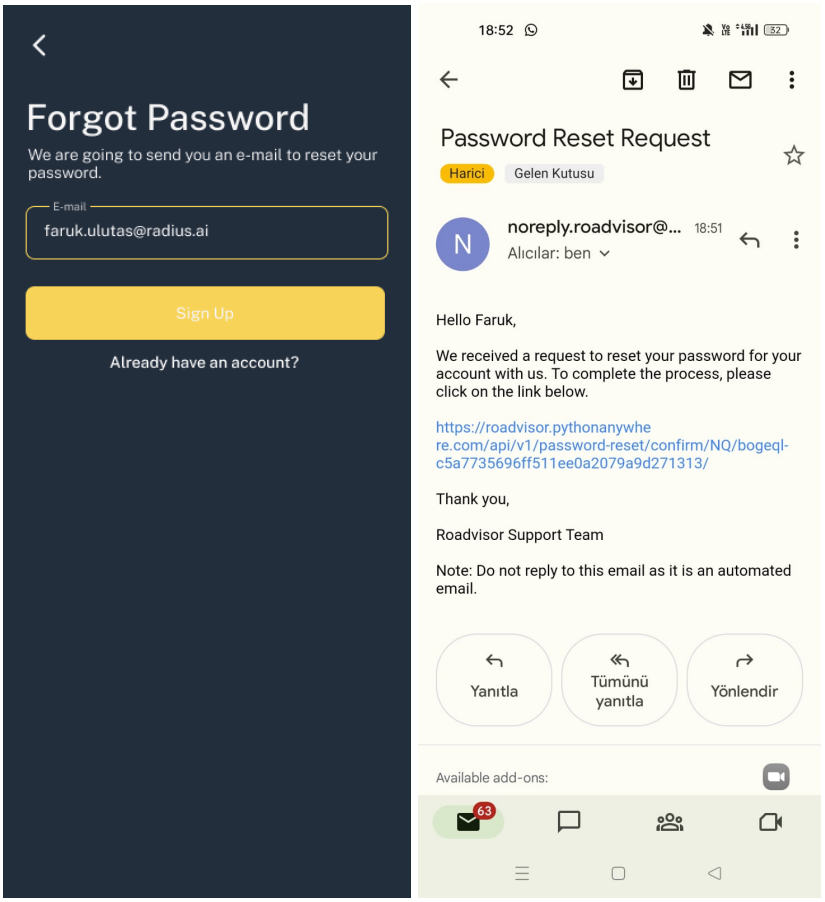
4. They click on the activation link in the verification email sent by RoadVisor.
5. They see the message "success: Account activated successfully" and their account becomes verified and active.

Login

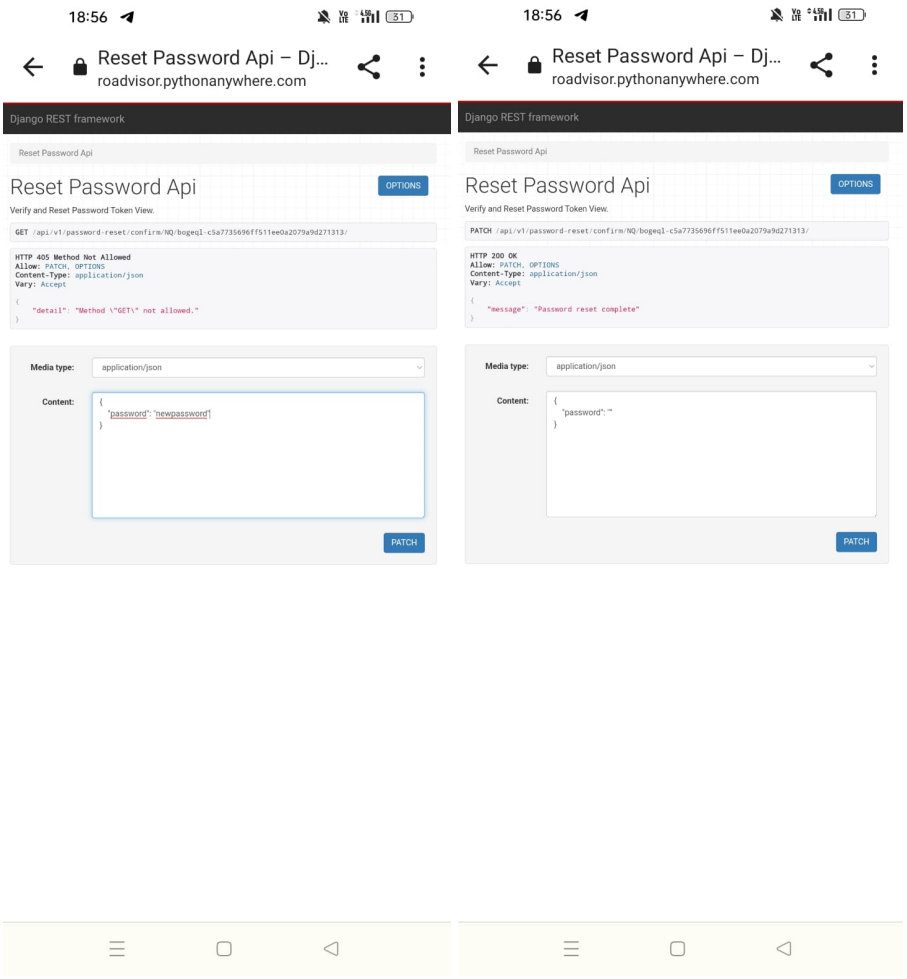


1. After opening the application, the user enters their email and password information correctly.
2. They press the Login button.
3. They are redirected to the Dashboard screen with a message stating "Login Successful."
    a. If incorrect information is entered or the account has not been activated, they will receive a response of "Login Failed," and the login attempt will be unsuccessful.

Forgot Password

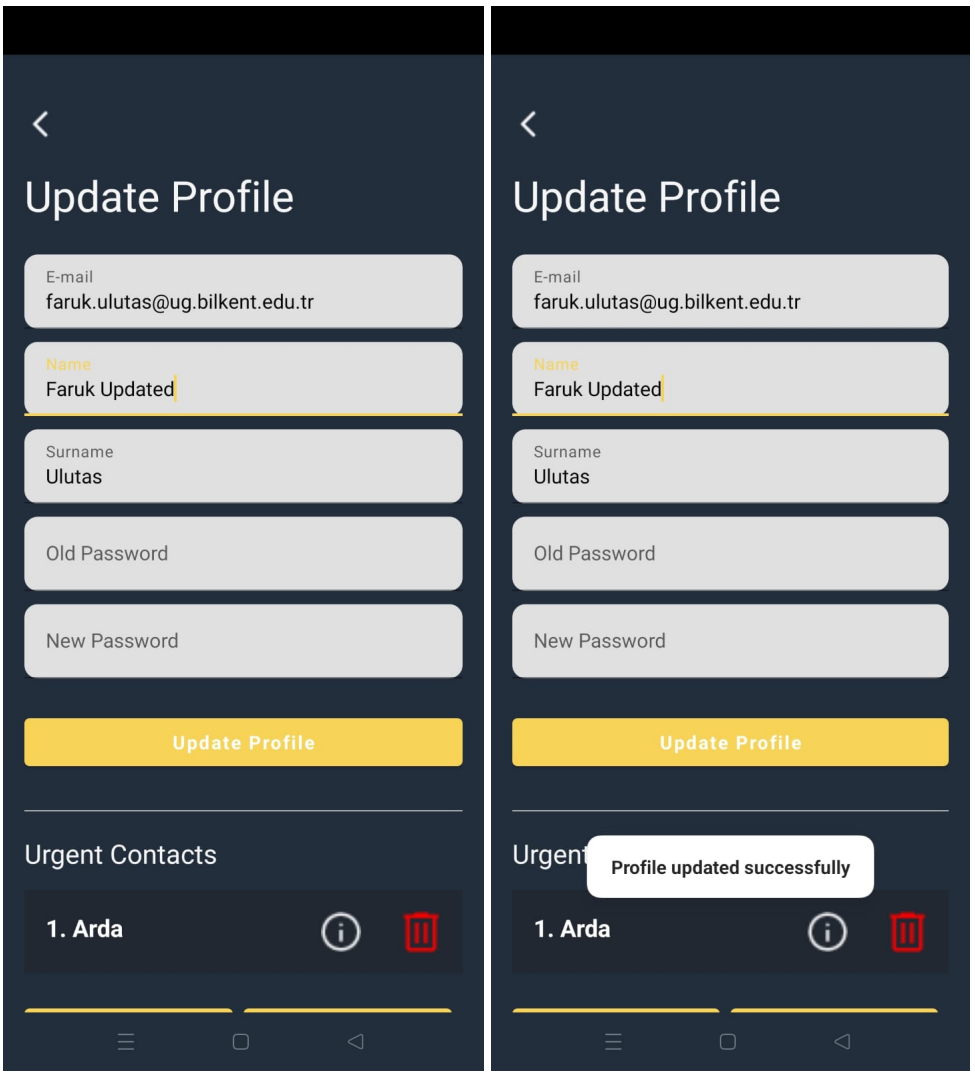


1. After opening the application, the user clicks on the "Forgot Password" button.
2. On the Forgot Password screen, they enter their email information correctly.
3. Then they click on the "Reset Password" button.
4. If the email address is correct, they receive an email from RoadVisor confirming that the password reset email has been sent.
5. They click on the link in the email to go to the password reset page.



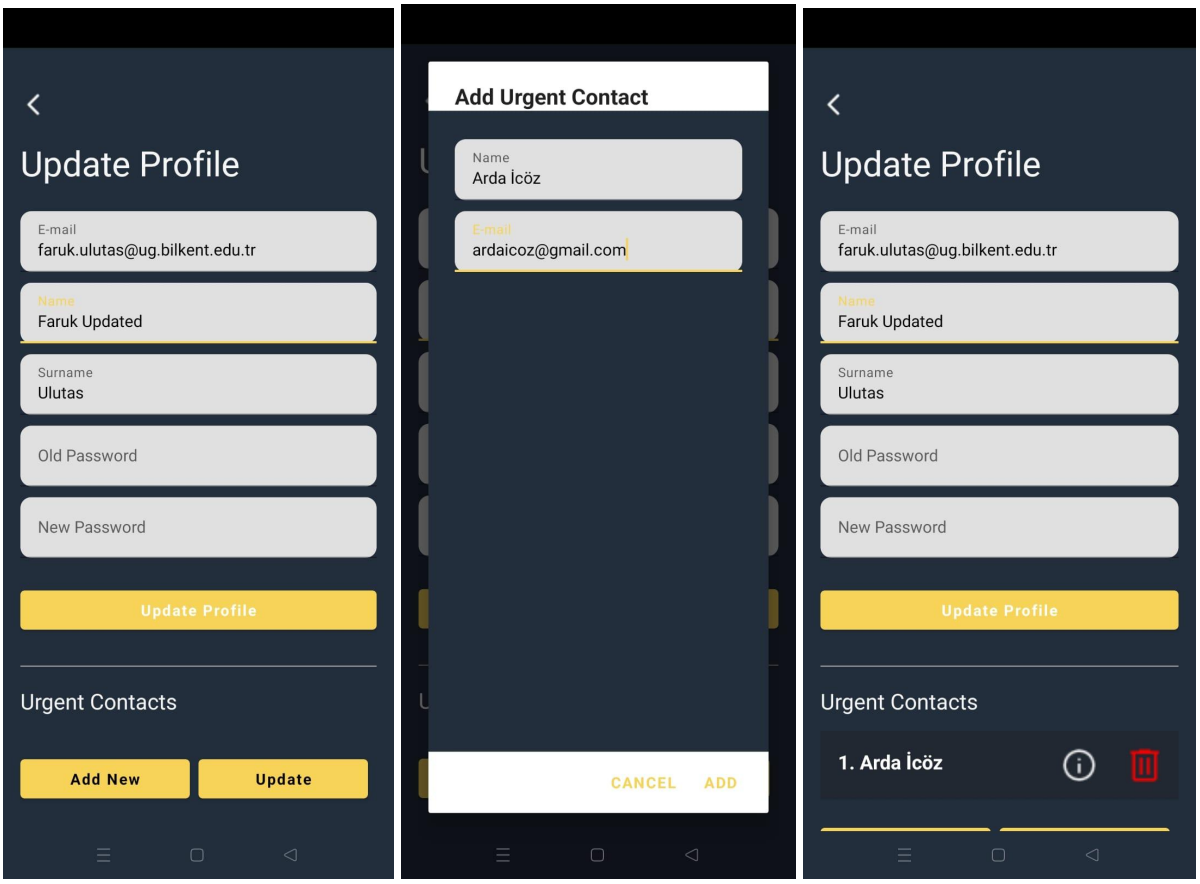
6. On this page, they enter their new password and click the "PATCH" button.
7. They see the response "Password reset complete" and their password is successfully reset.

Account Settings (Update Profile)



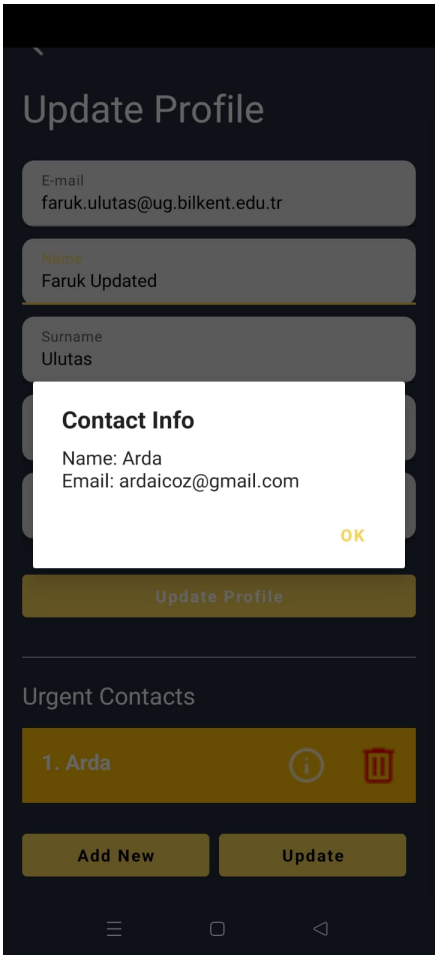
1. After entering the "Account Settings" page, the user can modify the desired information under the "Update Profile" section. They can make the necessary changes and then click the "Update Profile" button to update their profile information.
    a. To update the password, it is mandatory to enter both the "Old Password" and "New Password" information, and the "Old Password" must be correct.

Account Settings (Add Urgent Contact)



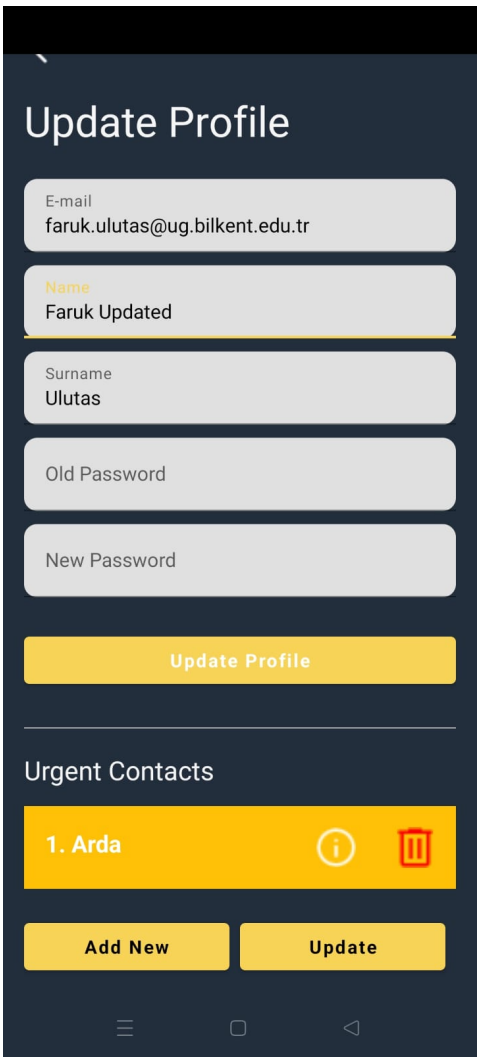
1. The user clicks on the "Add New" button in the "Urgent Contact" section.
2. Then, they enter the Name and Email information in the opened dialog.
3. They click the "Add" button.
4. The user successfully adds a new Urgent Contact.

Account Settings (Show Urgent Contact Info)



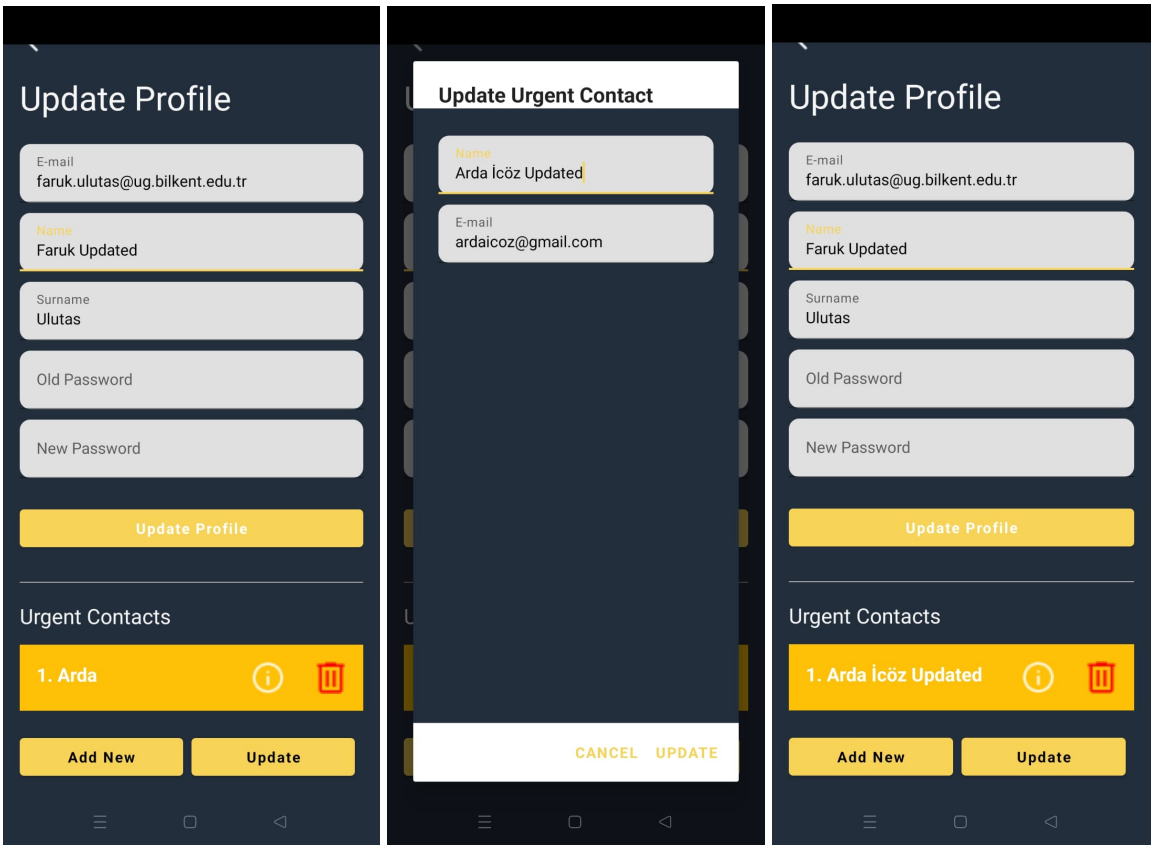
1. The user can view the Urgent Contact's Name and Email information by pressing the exclamation mark button located to the right of the respective Urgent Contact.

Account Settings (Select Urgent Contact)



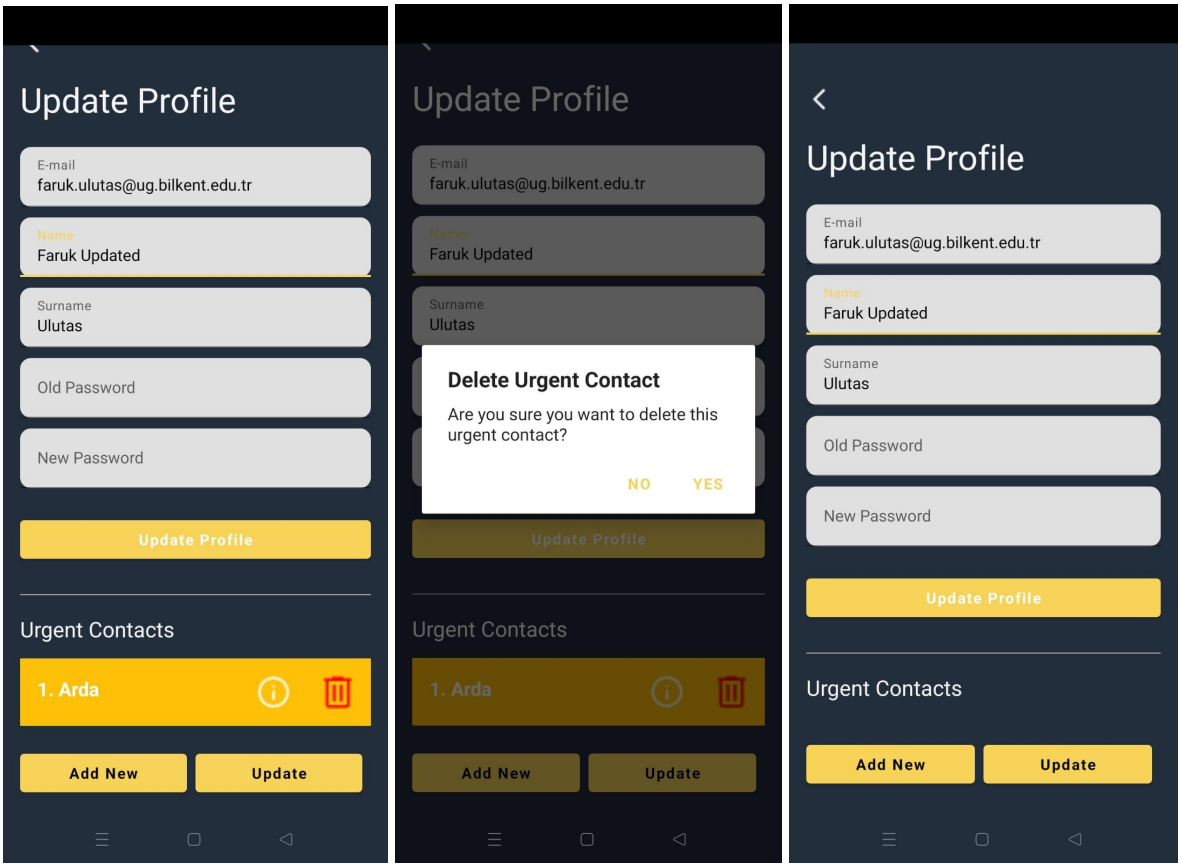
1. The user can select the respective Urgent Contact by clicking on it.
    a. They can cancel the selection by clicking on it again.
    b. They can change their selection by clicking on another Urgent Contact.

Account Settings (Update Urgent Contact)



1. While an Urgent Contact is selected, the user clicks on the "Update" button.
2. They update the information they want to change.
3. They click on the "Update" button in the bottom right corner of the dialog.
4. The user successfully updates the selected Urgent Contact.

Account Settings (Delete Urgent Contact)



1. While an Urgent Contact is selected, the user clicks on the red "Trash Can" button located to the right of the Urgent Contact.
2. A dialog box with the message "Are you sure you want to delete this contact?" opens.
3. In this dialog, the user clicks on the "Yes" button.
4. The selected Urgent Contact is successfully deleted.

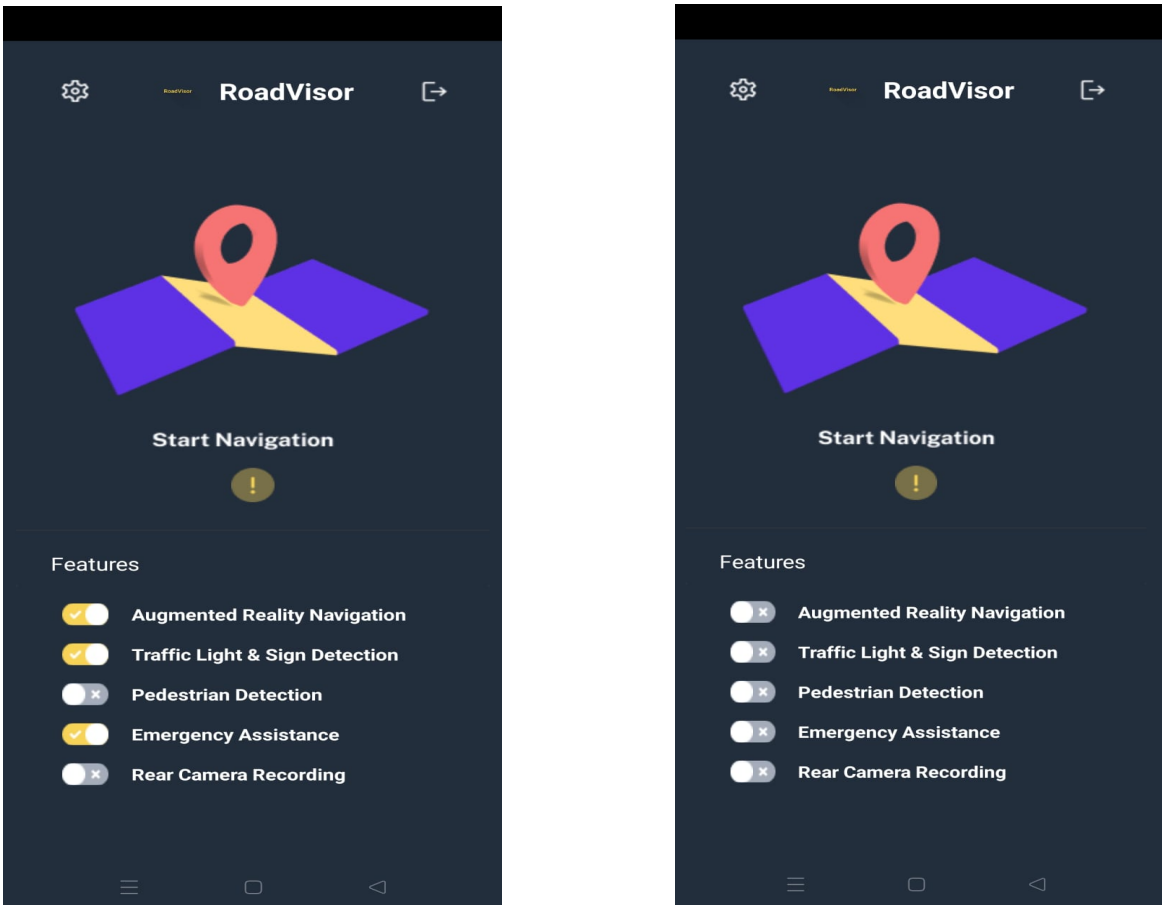
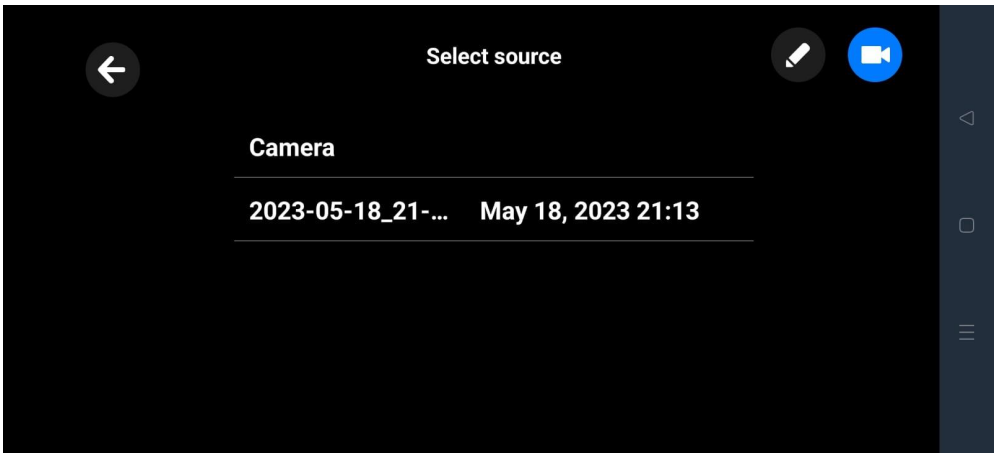


Fig. Selection of features

Feature Selection

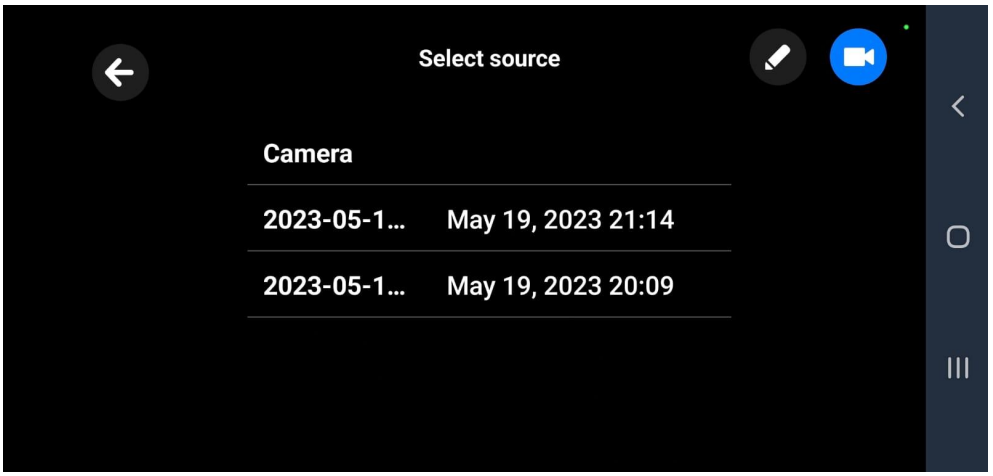
The feature selection allows the user to select the functionalities that the user wants to keep enabled during the navigation. Traffic light detection, pedestrian detection, emergency assistance, and rear camera recording are the features that he can choose to open or close. A toggle is provided for the selection of each of the features. This allows the user to select the features based on their needs.

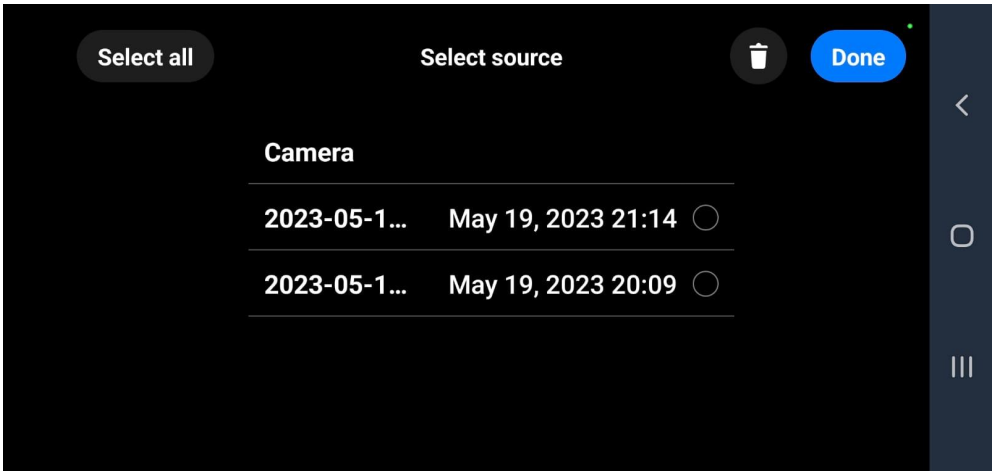
Start and Stop Recording Feature

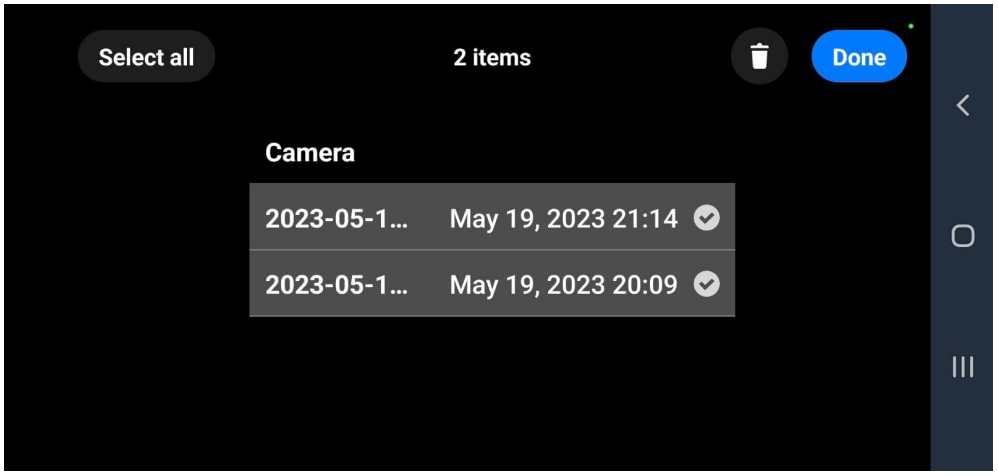


1. The user starts the recording by pressing the blue camera button located in the top right corner of the "Replay Playback" screen.
2. They stop the recording by pressing the "STOP REC" button.

Delete Record(s)

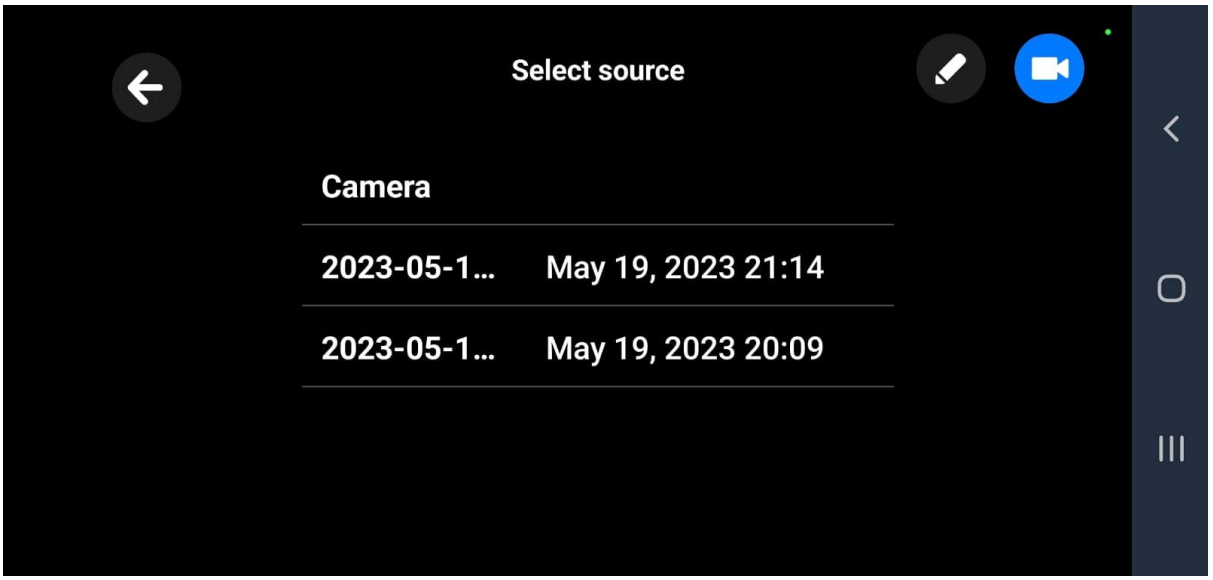






1. The user can select recordings by tapping on the "pencil" button located in the top right corner.
2. They can select all recordings by tapping on the "Select all" button located in the top left corner.
3. Then, they can delete the selected recordings by pressing the "trash can" button located in the top right corner.
    a. If they decide to cancel the deletion, they can exit the editing mode by tapping on the "Done" button located in the top right corner.

Watch Record(s)



1. While on the "Replay Playback" screen, the user can watch a recording by clicking on any of the records available.

Navigation Start and AR Interface

The navigation start interface allows the user to start the navigation feature. This starts the navigation where the user receives the directions to the destination through arrows that provides the user with a clear understanding of the path.

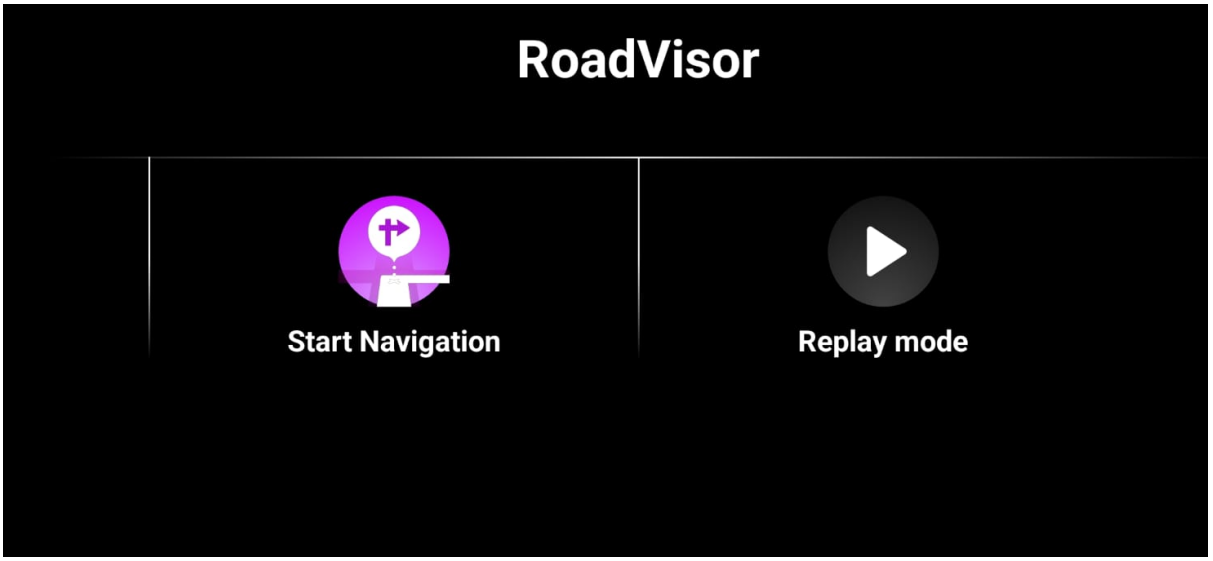


Fig. Starting the navigation

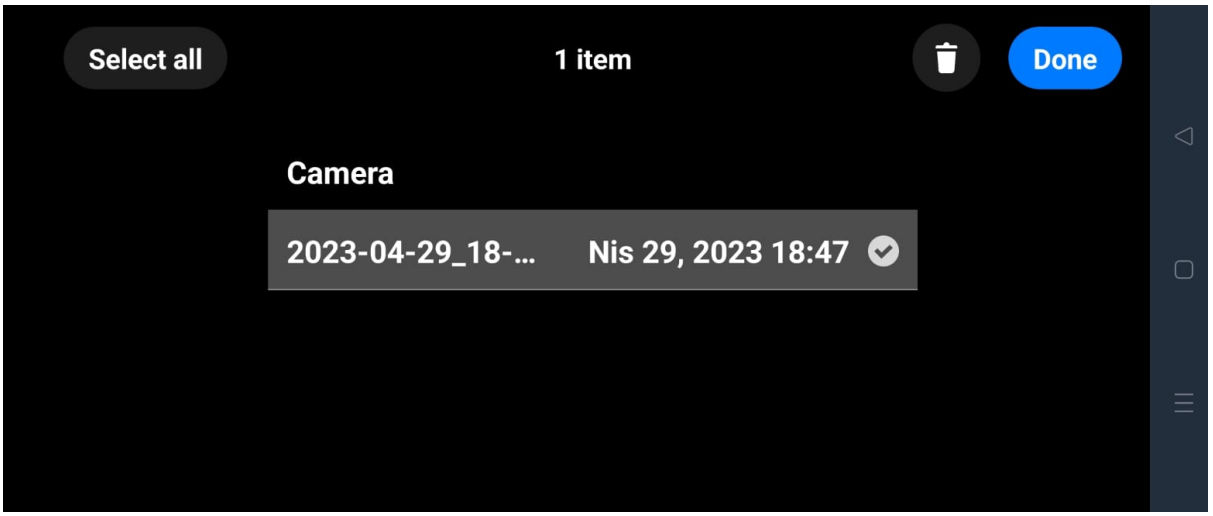


Fig. Replay mode where previously recorded videos of the navigation can be viewed.

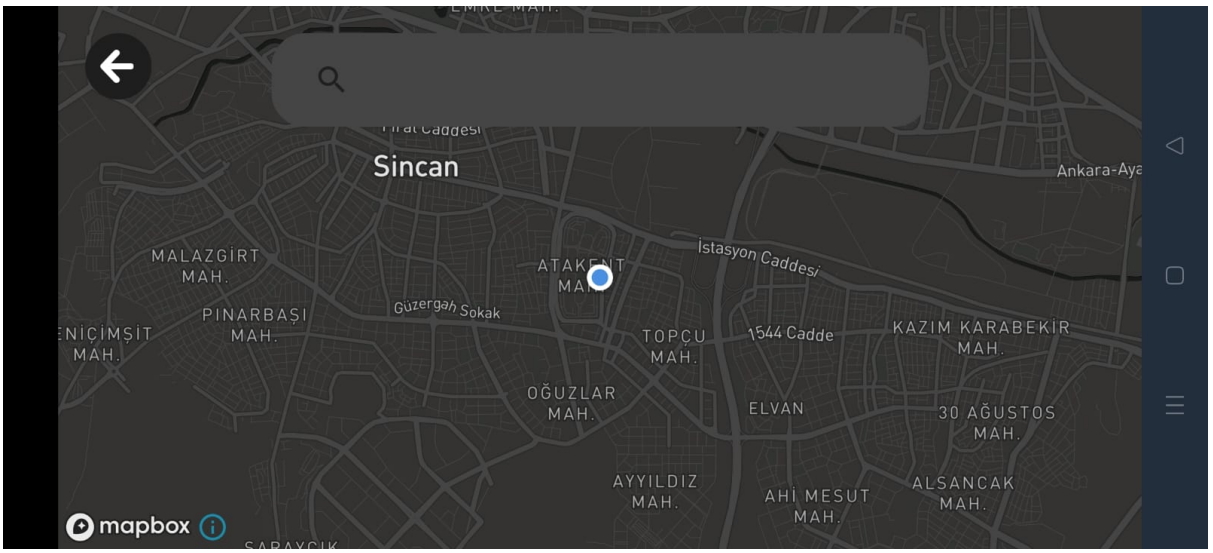


Fig. Search Map Screen after “Start Navigation” clicked

Destination search and selection

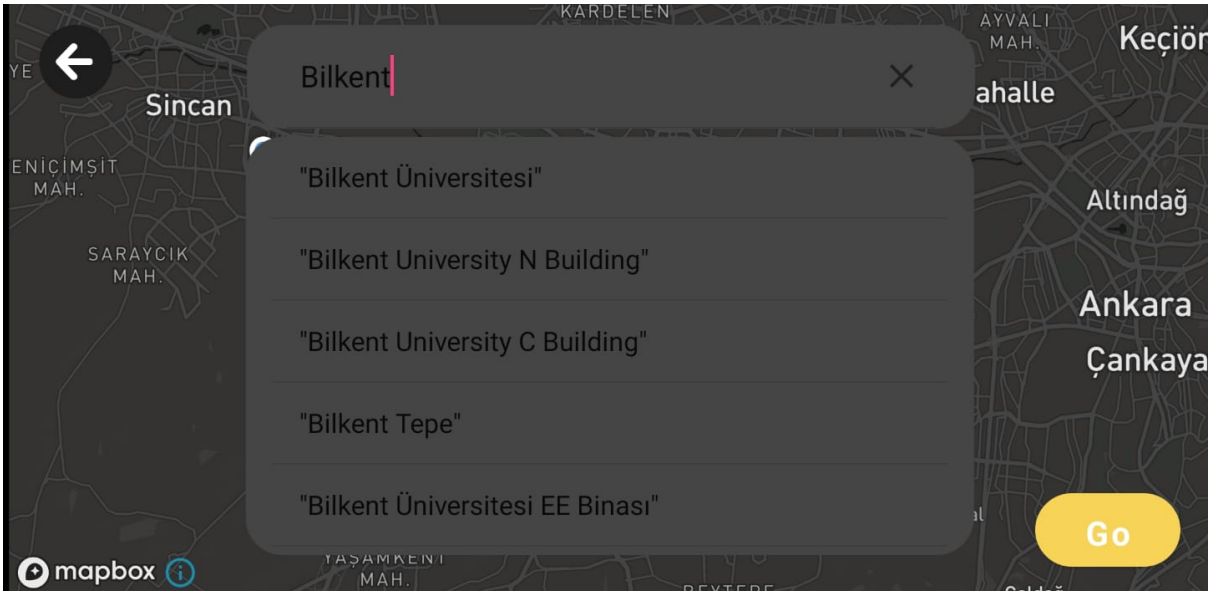


Fig. Destination search and selection

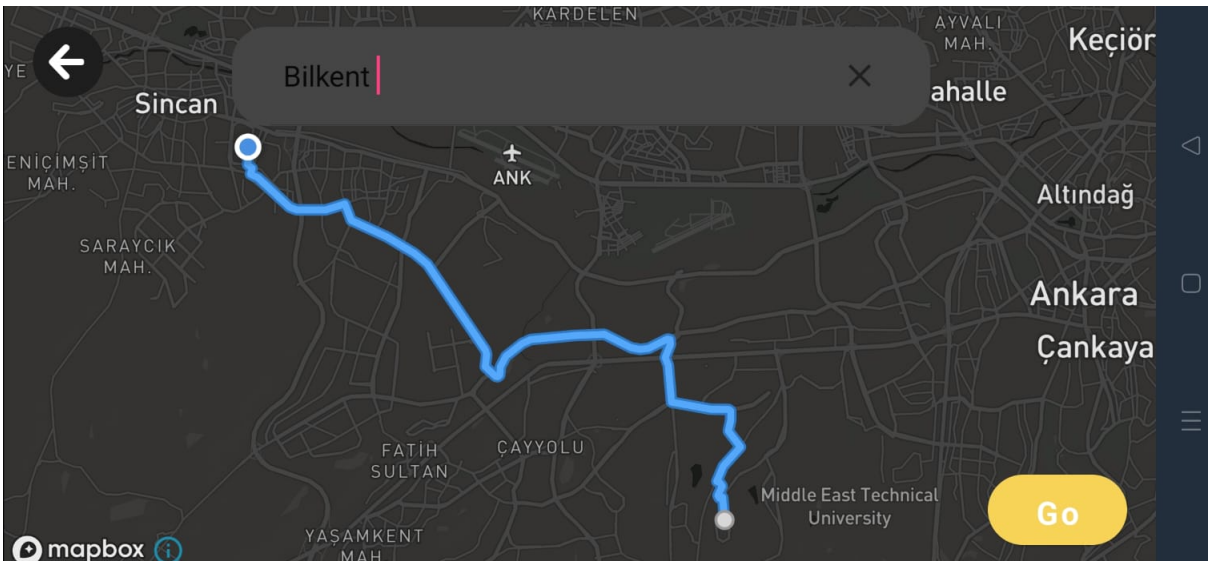


Fig. Display of the route from starting location to destination.

RoadVisor provides an easy to interactive destination selection interface. The user is able to type and select their destination location. A search bar is provided for typing the destination, while an additional map is present in the background for visualization of the destination.

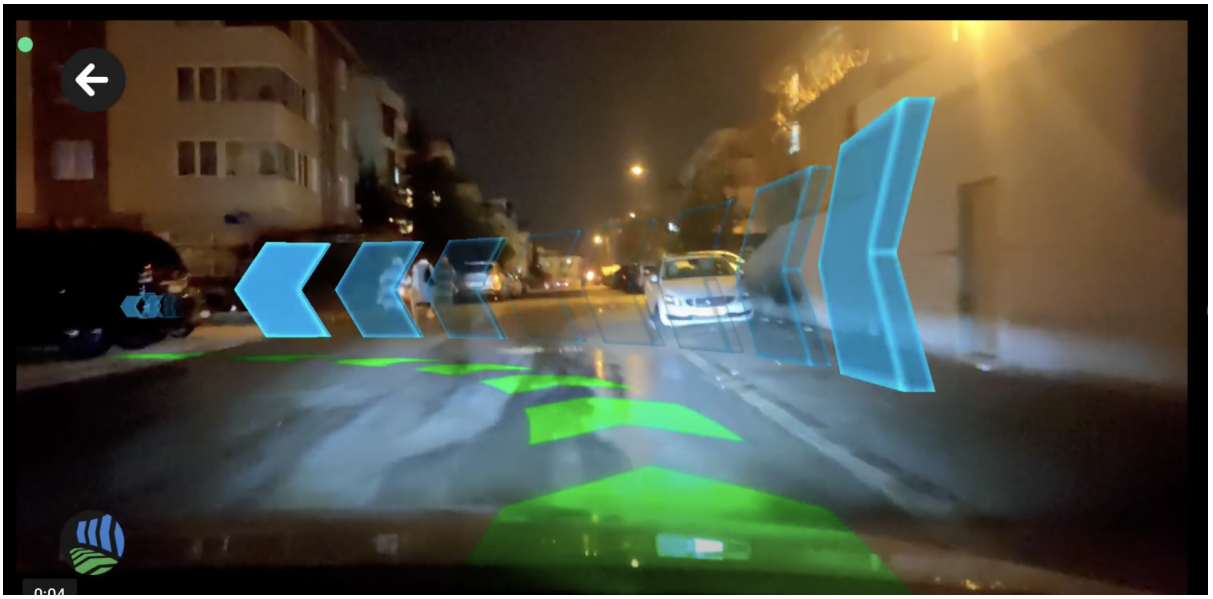


Fig. The augmented reality navigation screen after “Go” clicked

Emergency Assistance Request

Fig. Emergency Assistance button on the AR screen.

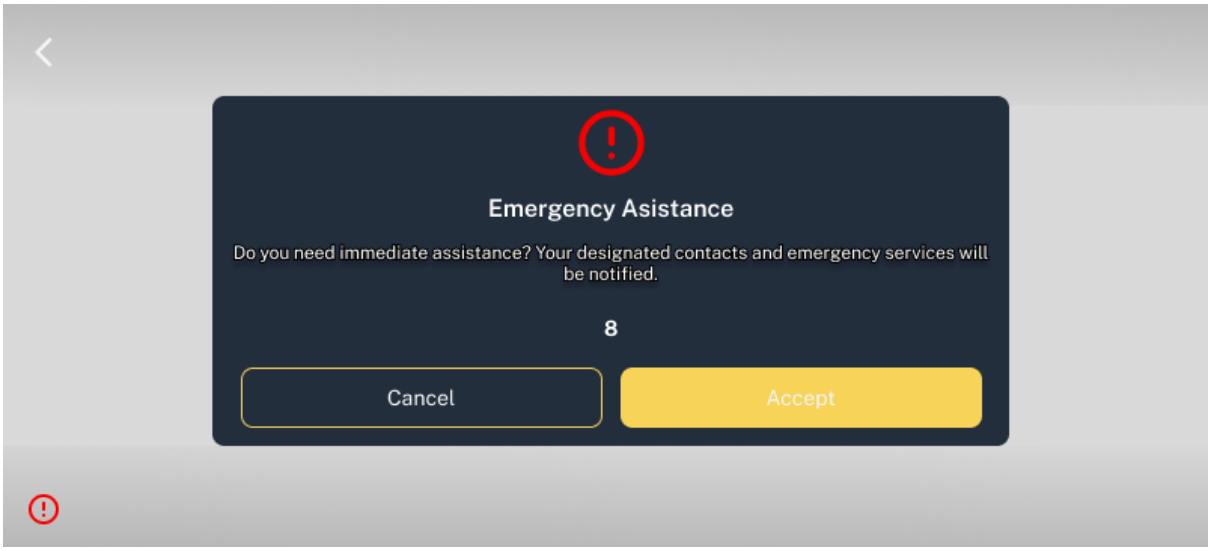
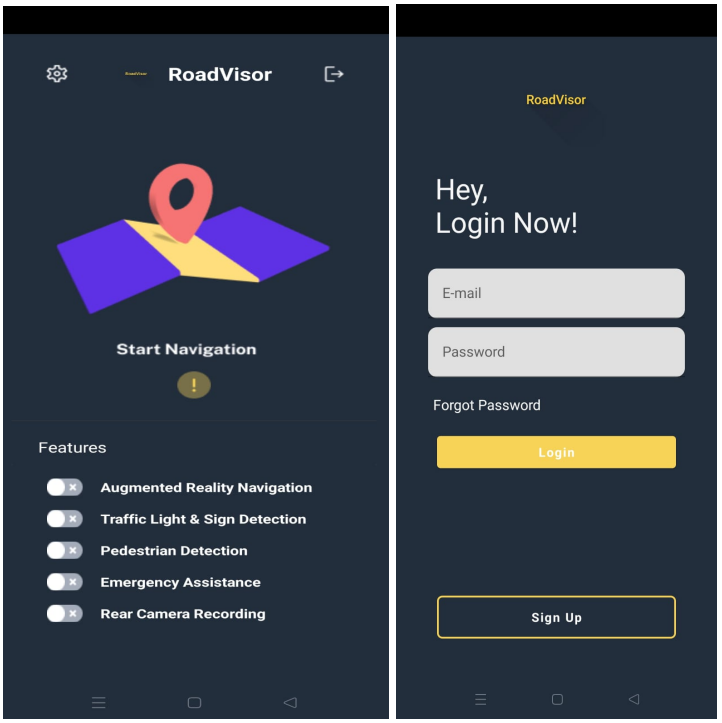


Fig. Emergency Assistance Request Dialog on the AR screen.

1. The user clicks on the red exclamation mark button located in the bottom left corner.
2. The user sends a request for help to all their Urgent Contacts, providing their name and location, by pressing the "Accept" button.
    a. If the user does not make any clicks on this screen for 10 seconds, it is automatically considered that they have pressed the "Accept" button.
    b. The user prevents the Urgent Contacts from being notified by pressing the "Cancel" button on this screen.

Logout



1. While on the Dashboard screen, the user clicks on the arrow icon located in the top right corner.
2. They successfully log out and are redirected to the login screen.

References

[1] T. Shang, H. Lu, P. Wu, and Y. Wei, “Eye-tracking evaluation of exit advance guide signs in highway tunnels in familiar and unfamiliar drivers,” International Journal of Environmental Research and Public Health, vol. 18, no. 13, p. 6820, 2021.

[2] “Ten types,” Doblin, 21-Apr-2020. [Online]. Available: https://doblin.com/ten-types. [Accessed: 19-May-2023].