Gotcha Scooter Mobileye ADD

Amit Moskovitz, Tom Nisim, Alexander Uzelevesky

Table of content

Use Cases

Use Case 1 : Initialization

Use-case 1.a: Server Initialization

- **Actor:** Admin
- **Precondition:** None
- **Parameters:** Configuration File Parameters
- **Actions:**
    - Admin launches server app
    - The server connects to external service (maps service)
    - The server connects & load the DB
    - Default admin is set
    - Default values from Configuration File Parameters are set.

Action	Data	Expected Result
Initialize the System	The configuration file parameters	A system Admin has been set (At least one) and default values from the configuration file were loaded. The server connect & load the DB and connect to maps external service

Use-case 1.b: Raspberry Pi Initialization

- **Actor:** Rider
- **Precondition:** None
- **Parameters:** Configuration File Parameters
- **Actions:**
    - Rider initializes raspberry pi device
    - Raspberry Pi reads data from configuration file
    - Raspberry Pi updates his internal data from the data received from the configuration file

Action	Data	Expected Result
Initialize the System	The configuration file parameters	Raspberry Pi data is updated according to the configuration file

Use-case 2: Change External Service

- **Actor:** Admin
- **Precondition:** Admin is logged in
- **Parameters:** New External Service
- **Actions:**
    - Admin will choose different external services from the list of available services.
    - The system will verify the service is a valid choice.
    - The system will update the Server to use this service.

Action	Data	Expected Result
Change External Service	Valid external service	The system will be updated to use the new external service.
	Invalid External service	The system will alert the Admin that the service is invalid.

Use-case 3: Register new User

- **Actor:** Rider
- **Precondition:** None
- **Parameters:** Email, Password, gender, name, last_name, rp_serial_number,birthday, licence_issue_date
- **Actions:**
    - Rider will enter his credentials.
    - The Server will verify the Email is not being used by another user.
    - If the email is being used already it will alert.
    - The Server will verify the password is secure.
    - The server will check that the rider's raspberry pi serial number is available and is not associated to a another user and alert otherwise
    - The server mark the raspberry pi serial number as unavailable
    - The Server stem will add the new user to the list of active users.
    - The Server will create a mapping between the rp_serial_number and the user

Action	Data	Expected Result
Add New User	Valid Email and password	The Server will add the new user to the list of active users.
	Invalid Password	The Application will alert the User that the password is invalid and will not register the new user
	Invalid Email	The Application will alert the User that the Email is invalid and will not register the new user
	Used email	The Application will alert the User that the Email is already taken by another user and will not register the new user

Use-case 4: Edit User Profile

- **Actor:** Rider
- **Precondition:** Rider is an Active user
- **Parameters:** Age, Gender, type of Scooter, Driver's license issue date
- **Actions:**
    - The Rider will enter his new desired profile information.
    - The Application will verify the validity of the information.
    - The Application will send the new information to the server which will Update the user Profile.

Action	Data	Expected Result
Edit User Profile	Valid new information	The system will update the information on the user profile.
	Invalid Age	The Application will alert the user that the Age is invalid
	Invalid type of scooter	The Application will alert the user that the type of scooter is invalid
	Invalid Driver's license issue date	The Application will alert the user that the Driver's license issue date is invalid

Use-case 5: User Login

- **Actor:** Rider
- **Precondition:** None
- **Parameters:** Email, Password
- **Actions:**
    - The Rider will enter his credentials.
    - The Application will send rider's credentials to the server which will validate the information.
    - If the credentials are valid the user will be logged on, otherwise he will be alerted.

Action	Data	Expected Result
User Login	Valid user credentials	The User will be logged in.
	Incorrect Password	The Application will alert the User that the password he provided is Incorrect .
	Incorrect Email	The Application will alert the User that the Email he provided is Incorrect or not existing

Use-case 6: User Logout

- **Actor:** Rider
- **Precondition:** Rider is logged in
- **Parameters:** None
- **Actions:**
    - The Rider will press the logout button.
    - The Application will send the rider's email to the server which will mark the rider as logged out.
    - If the rider is not logged in, the system will alert it

Action	Data	Expected Result
User Logout	None	The User will be logged out.

Use-case 7: Send Message to Admins

- **Actors:** Rider
- **Precondition:** Rider is logged in
- **Parameters:** Message content
- **Actions:**
    - The Rider will enter his message.
    - The Application will verify the message doesn't contain malicious data and that they are not empty.
    - The application will send the message to the server.
    - The Server will alert the Admin about the message.

Action	Data	Expected Result
Send Message To Admins	Valid message	The Application will send the message to the Server which will send the message to the appropriate Admin
	Invalid/malicious message	The Application will alert the User that the message is Invalid/malicious

Use Case 8 : Start of Riding

Use-case 8.1: Suggest safest routes

- **Actors:** Rider
- **Precondition:** Rider is logged in, system is connecting to maps external system
- **Parameters:** destination address
- **Actions:**
    - The rider will start a ride using the rider app
    - The rider will enter his destination address
    - The system will get the user location via phone GPS
    - The system will send to the External Maps Service the destination address and the driver location
    - The External Maps Service will send to the system a list of routes from the driver location to the destination address.
    - The system will rate each route according to its level of safety according to defined indicators
    - The system will show the driver the safest routes

Action	Data	Expected Result
Suggest Safest routes	Destination address does not exist	The system will show to the user a specific error message
	Legal destination address	The System will show the driver the possible safest routes.

Use Case 9 : During the ride

Use-case 9.1: receiving road video shooting

- **Actor: None**
- **Precondition:** The Raspberry pi is connected to a camera
- **Parameters:** None
- **Actions:**
    - The Raspberry Pi gets the next frame from the camera every 0.5 (configuration file) second
    - The Raspberry Pi check the clarity of the video
    - If the video shooting is unclear , the Raspberry Pi gets another road video shooting from the camera

Receiving road video shooting	Clear video shooting of the road	The Raspberry Pi perform Use Case 9.1.2
	Unclear video shooting of the road	The Raspberry Pi gets another road video shooting from the camera

Use-case 9.2: Identify stationary objects

- **Actor:**
- **Precondition:** A deep learning process of the system for identifying stationary objects according to a collection of images of such objects is done.
- **Parameters:** Road video shooting(Use Case 9.1)
- **Actions:**
    - The Raspberry Pi analyze the frame to detect stationary objects in the rider path - Potholes in the road, trees, electricity poles
    - The system calculate the distance between the rider and the object, and the scooter's current speed
    - When the rider reaches the location where he is supposed to receive an alert (according to the previous section and predetermined parameters), the system creates an alert

Identify stationary objects	The video shooting of the road which does not contain stationary objects in the rider's path.	No alert tone will be heard.
	The video recording of the road which contains stationary objects in the rider's path.	An alert tone will sound for each stationary object detected in the path.

Use-case 9.3: Identify Type of riding surface

- **Actor:**
- **Precondition:** A deep learning process of the system for identifying road surface type according to a collection of road surface types and their classifications is done
- **Parameters:** Road surface data
- **Actions:**
    - The Raspberry Pi analyzes the road surface data to detect the type of the surface. The identification is done according to the deep learning process that was carried out (precondition).

Action	Data	Expected Result
Identify type of riding surface	Road surface data - roadway	The Raspberry Pi identify the type of road surface successfully
	Road surface data - sidewalk	The Raspberry Pi identify the type of road surface successfully
	Road surface data - other	The Raspberry Pi failed to identify the type of road surface

Use Case 10 : End of riding

Use-case 10.1: Sending data to the server

- **Actor:**
- **Precondition:** Use Cases 9.2-9.4
- **Parameters:**
- **Actions:**
    - The Raspberry Pi retrieves data of the rides that were not sent due to communication problems
    - The Raspberry Pi try to send the data from the previous section and the data of the new ride - the data collected in Use Cases 9.2-9.3 to the server
    - In case of a communication problem, the RP saves the data of the new ride to a file together with the data of the last rides that have not yet been sent to the server

Action	Data	Expected Result
Sending data to the server	new ride data. Proper connection to the server	new ride data & last rides that have not yet been sent to the server sends to the server successfully
	new ride data. Improper connection to the server	new ride data saves to a file and will be sent to the server in the future

Use-case 10.2: Saving information about the locations of the hazards

- **Actor:**
- **Precondition:** connection with the DB, connecting to maps external system, detection of object mark as a hazard (Use Cases 9.2 - 9.3)
- **Parameters:** ride id, list of hazards
- **Actions:**
    - The server traverse on hazards list
    - If the hazard is already exist - the server update his size
    - else - the server creates and saves to the DB the new hazard data - location, city, type, size

Action	Data	Expected Result
Saving information about the locations of the dangers.		The coordinates and city which the object detects are saved to the DB with the specific object ID.

Use-case 10.3: Saving riding data

- **Actor:**
- **Precondition:** connection with the DB , Use Cases 9.2-9.4.
- **Parameters:** None
- **Actions:**
    - The system saves the collected riding data (Glossary) to the DB .

Action	Data	Expected Result
Saving riding data		Riding data saved to the DB successfully.

Use-case 10.3: users rating

- **Actor:** System
- **Precondition:** Use Case 10.3
- **Parameters:** None
- **Actions:**
    - The system retrieved from DB the data about the ridings taken by the connected user (Use Case 14).
    - The system rates the registered user by the data from the previous section, by the data of the current ride and by the terms from Appendix C.

Action	Data	Expected Result
Users rating		The system rates the users successfully, The ratings match the terms from appendix C.

Use-case 11: View Riding history

- **Actor:** User
- **Precondition:** connection with the DB, User is connected to the system
- **Parameters:** None
- **Actions:**
    - The system retrieved from DB the data about the ridings taken by the connected user (Use Case 10.3).
    - The system shows the data from the previous section to the connected user.

Action	Data	Expected Result
View Riding history	None	Riding data showed to the user.
	None	User does not have any riding history, specific message shown to the user :"No Riding History To Show"

Use-case 12: Delayed notifications

- **Actor:** Admin
- **Precondition:**None
- **Parameters:** Notification content, the recipient user details
- **Actions:**
    - The admin sends the notifications content to the recipient user.
    - If the user is connected to the system – he will get the notification right away.
    - Else, the notification will be displayed when logging in.

Action	Data	Expected Result
Users rating – connected User	Notification content, user email(User is not connected to the system)	The user gets the notification right away and can view the notification content.
User rating - Guest	Notification content, user email(User is connected to the system)	When logging in, the user gets the notification and can view its content.

Use-case 13: Admin view user's data - ridings, rating and profile

- **Actor:** Admin
- **Precondition:** Use Case 10.3, connection with the DB
- **Parameters:** User details, data type (ridings, ratings, profile)
- **Actions:**
    - The system retrieved from the DB the data type of the user.
    - The system shows the information from the previous section to the admin.

Action	Data	Expected Result
Admin view user's data - ridings, rating and profile	Valid user email,Valid data type	The admin gets the data type of the user successfully.
	Invalid user email,Invalid data type	Specific error message shown to the admin

Use-case 14: Manage User

Use-case 14.1 : Remove user

- **Actor:** Admin
- **Precondition:** User is registered to the system.
- **Parameters:** User email
- **Actions:**
    - The admin sent a request to remove the user.
    - The system verifies that the admin is logged in.
    - The system verifies that the user is registered to the system.
    - The user will be removed from the system.
    - The admin will receive feedback from the system.

Action	Data	Expected Result
Remove user	registered user email	user has been removed successfully from the system.
	unregistered user email	The user does not exist in the system.
	invalid email	invalid input.

Use-case 14.2 : Edit user profile

- **Actor:** Admin
- **Precondition:** User is registered to the system.
- **Parameters:** User email, new email, new name, new address.
- **Actions:**
    - The admin sends a request to edit the user profile.
    - The system verifies that the user is registered to the system.
    - The user will be removed from the system.
    - The admin will receive feedback from the system.

Action	Data	Expected Result
Edit user profile	registered user email, valid data.	user profile has been changed successfully.
	unregistered user email.	The user does not exist in the system
	invalid email/name/address	invalid input

Use-case 15 : Add award

- **Actor:** Admin
- **Precondition:** None
- **Parameters:** List of rider's emails, Coupon code, due date.
- **Actions:**
    - The admin sent a request to add an award.
    - New award with the admin emails, and the emails of the riders who are eligible to receive the award added to the system
    - All the registered users who are eligible to receive the award will get a private message from the system with the award details.

Use-case 16 : Set system configurations

- **Actor:** Admin
- **Precondition:** None
- **Parameters:** System configuration data
- **Actions:**
    - The admin sends a request to set the system configurations.
    - The system will check validation of the parameters.
    - If the data is valid, the configuration will be setted

Use-case 17: Manage Admin

Use-case 17.1 : Add admin

- **Actor:** Admin
- **Precondition:** None
- **Parameters:** Email, phone.
- **Actions:**
    - The system will verify the appointer admin is logged in
    - The system will add the new admin details.
    - The system will save the admin who appoints the new admin.
    - The system will send an email to the new admin with an admin-app installation link & app guide.

Action	Data	Expected Result
Add admin	valid phone & valid email	admin added successfully.
	invalid phone / email.	invalid input.

Use-case 17.2 : Remove admin

- **Actor:** Admin
- **Precondition:** There are at least two admins.
- **Parameters:** Email.
- **Actions:**
    - The system will verify the appointer admin is logged in
    - The system will verify that the admin has the permission to remove this admin.
    - The system will record this action.
    - The system will remove the admin and not support his access to the system as admin anymore.

Action	Data	Expected Result
Remove admin	Admin email	admin removed successfully.
	non-admin email	there is no admin with this email.
	invalid email	invalid input.

System Architecture

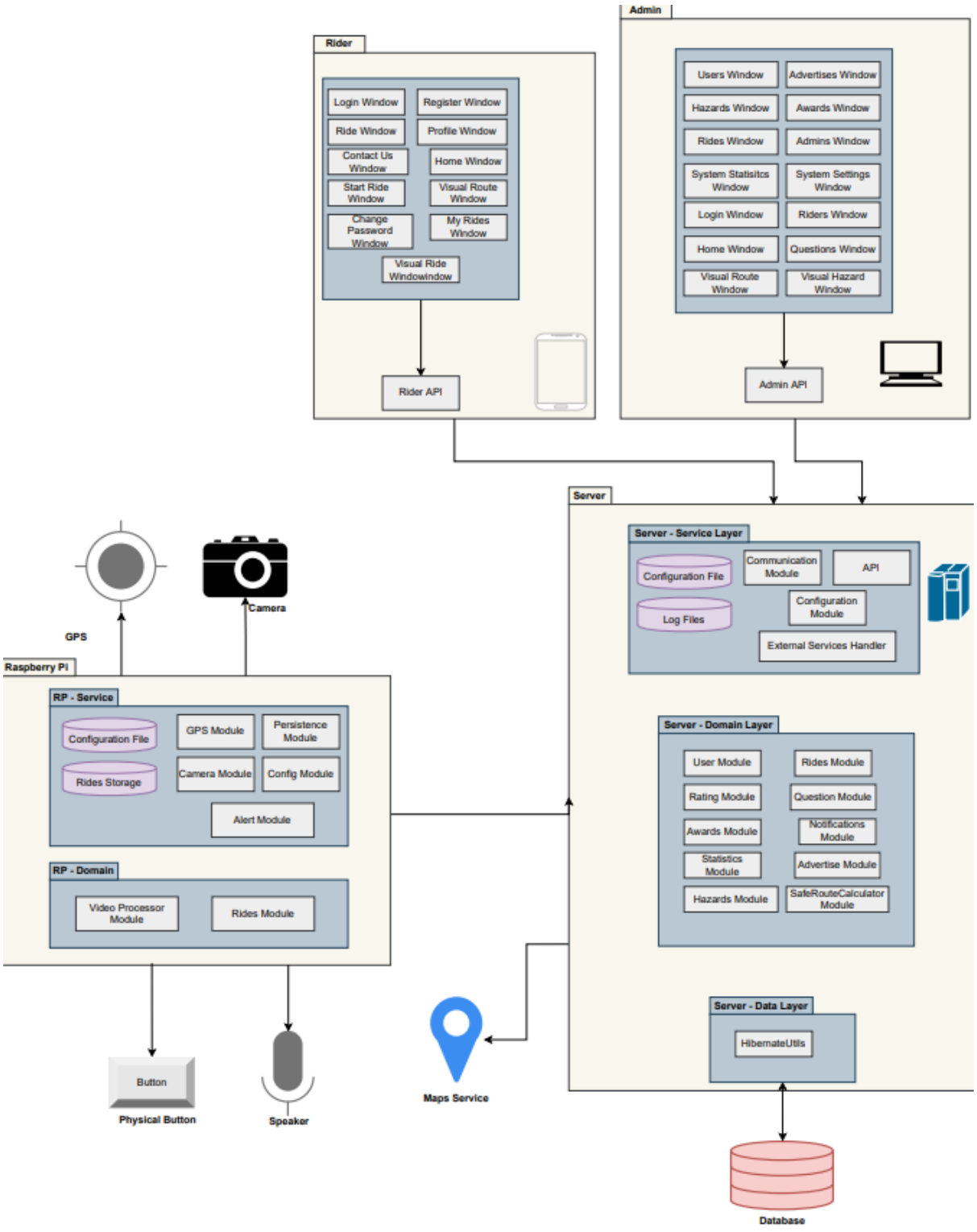


Figure 1 : System Architecture

System components:

- **Scooter** - The user's personal scooter, intended for private use only.
- **Camera** - The Raspberry Pi Camera Module 2 replaced the original Camera Module in April 2016. The v2 Camera Module has a Sony IMX219 8-megapixel sensor (compared to the 5-megapixel OmniVision OV5647 sensor of the original camera).
- **GPS -** The BE-220 GPS Module using the latest 10th generation chip.
  Data Level : TTL or RS-232, BE-220 GPS Module has 4M Flash to save configuration,
  Baud Rate : 4800bps to 921600bps, Default 38400bps
- **Raspberry Pi** - The component composed on the scooter, which includes Internet and GPS connections units, has the function of analyzing the frames coming from the camera, alerting if necessary, and sending data collected during the trip to a server.
- **Server** - A server that runs in a cloud environment, the unit that receives requests from the various components in the system and is responsible for saving the data and handling the various requests
- **Rider Application** - A client-side component in a mobile environment containing the sub-components: window display, communication, GPS-based location and navigation management.
- **Admin Application** - A client-side component in the Windows PC environment responsible for manipulating the information in the system, controlling the activation of various functions to change the system states.

Data Model

Description of Data Objects

Ride		
PK	ride_id	int
	rider_email	string
	date	Date
	city	String
	start_time	Time
	end_time	Time
	origin	Tuple<int, int>
	destenation	Tuple<int, int>

Ride

The object describes a ride performed by a rider registered in the system
for any such object will be save:
rider_id - the id of the rider perform the ride
date - the date the ride was made
city - the city where the ride was made
start_time - the exact time the ride started
end_time - the exact time the ride ended
origin - the exact coordinates of the origin location
destination - the exact coordinates of the destination location

Figure 2.1 : Ride Object Description

StationaryHazard		
PK	id	int
	ride_id	int
	location	Tuple<int, int>
	city	String
	type	int - (HazradType)
	size	double
	rate	int

StationaryHazard

The object describes a stationary object that appeared on a certain riding route.
The object can be a hole in the road, a light pole, an electric pole, etc
A type is basically described by the 'type' field (HazardType), in addition for any such object will be saved:
ride_id - the id of the ride in which the object was detected
location - the exact coordinates of the object
city - the city where the object is located
size - the length , width ,height of the object
rate - the rating of the hazard calculated according to the rules in appendix B

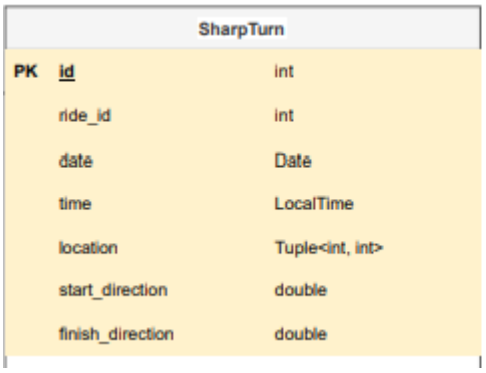
Figure 2.2 : Hazard Object Description

Brake		
PK	id	int
	ride_id	int
	date	Date
	time	LocalTime
	location	Tuple<int, int>
	start_speed	double
	finish_speed	double

Brake

The object describes a riding action in which the rider perform braking
for any such object will be save:
ride_id - the id of the ride where the action happened
date - the date the riding action was made
location - the exact coordinates in which the action was made
start_speed - the speed of the rider before the brake
finish_speed - the speed of the rider after the brake

Figure 2.3 : Brake Object Description

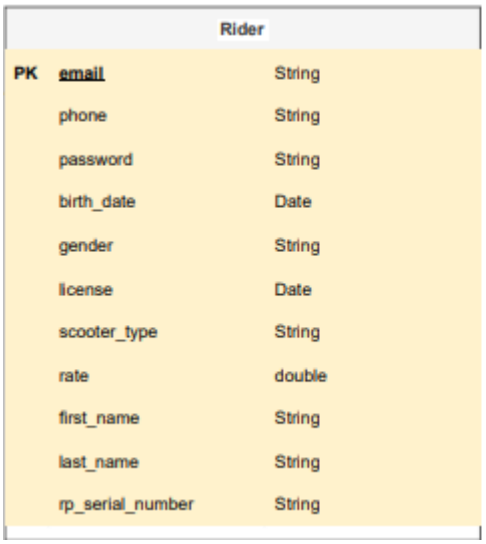


SharpTurn		
PK	id	int
	ride_id	int
	date	Date
	time	LocalTime
	location	Tuple<int, int>
	start_direction	double
	finish_direction	double

SharpTurn

The object describes a riding action in which the rider perform sharp turn
for any such object will be save:
ride_id - the id of the ride where the action happened
date - the date the riding action was made
location - the exact coordinates in which the action was made
start_direction - the direction of the ride before the turn
finish_direction - the direction of the ride after the turn

Figure 2.4 : Sharp-Turn Object Description

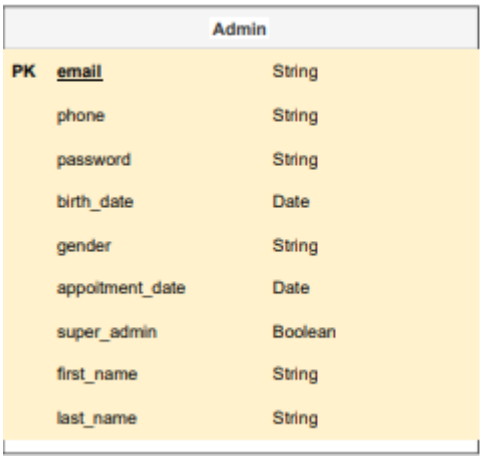


Rider		
PK	email	String
	phone	String
	password	String
	birth_date	Date
	gender	String
	license	Date
	scooter_type	String
	rate	double
	first_name	String
	last_name	String
	rp_serial_number	String

Rider

The object describes a registered rider
for any such object will be save:
email , name, phone number, hash value of the password, birthdate, gender
license - the date the license was issued
scooter type - the type of rider's scooter
rate - the rating of the rider calculated by the rules in appendix C

Figure 2.5 : Rider Object Description

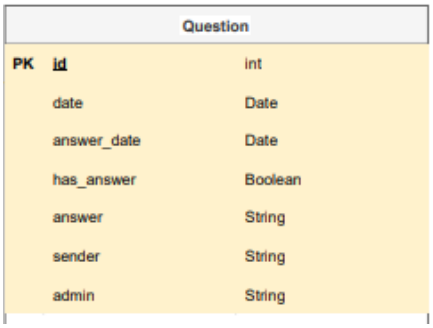


Admin		
PK	email	String
	phone	String
	password	String
	birth_date	Date
	gender	String
	appoitment_date	Date
	super_admin	Boolean
	first_name	String
	last_name	String

Admin

The object describes an Admin of the system
for any such object will be save:
email , name, phone number, hash value of the password, birthdate, gender
appoitment_date - the date the admin was appointed
location - the exact coordinates in which the action was made
super_admin - wether the admin is super admin or not

Figure 2.6 : Admin Object Description



Question	
PK id	int
date	Date
answer_date	Date
has_answer	Boolean
answer	String
sender	String
admin	String

Question

The object describes a question send by rider to one of the admins.
for any such object will be save:
date - the date in which question was send
answer_date - the date in which admin answer the question - NULL if the admin didn't answer yet
has_answer - wether the admin answer the question or not
answer - the answer of the admin for the question , NULL
if the admin didn't answer yet
sender - the email of the sender rider
admin - the email of the admin who is the recipient of the question

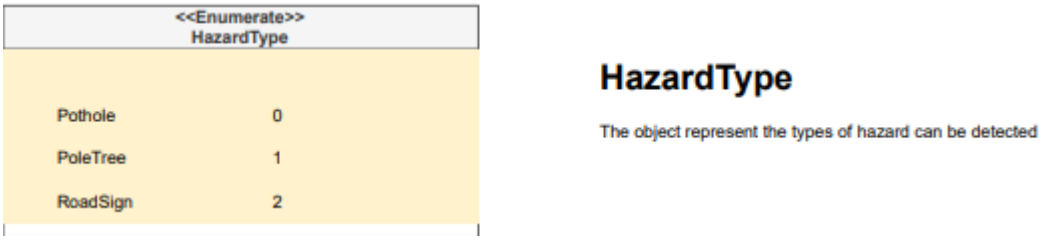
Figure 2.7 : Question Object Description

Advertise	
PK id	int
start_date	Date
final_date	Date
owner	String
message	String
user_clicks	int
photo	String
website	String

Advertise

The object describes an advertise in the system.
for any such object will be save:
start_date - the date in which the advertise was published
final_date - the date in ehich the advertise stopped being published
owner - the email of the advertiser
message - the advertising itself
user_clicks - number of system users that clicks the advertise
photo - the path to the photo of th advertise
website - the link for the advertise website

Figure 2.8 : Advertise Object Description

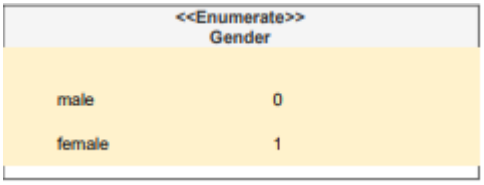


<<Enumerate>> HazardType	
Pothole	0
PoleTree	1
RoadSign	2

HazardType

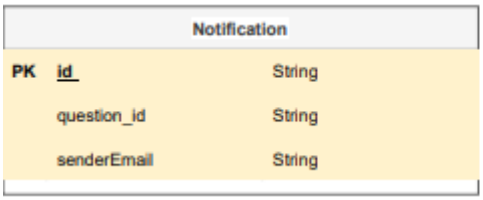
The object represent the types of hazard can be detected

Figure 2.9 : Hazard Type Object Description



<<Enumerate>> Gender	
male	0
female	1

Figure 2.10 : Gender Object Description



Notification

The object describes an alert of a specific question.
Admins and riders can view the question content via this alert
for any such object will be save:
question_id - the id of the question for which the alert is sent
sender_email - the email of the notification sender

Figure 2.11 : Notification Object Description

Data Objects Relationships

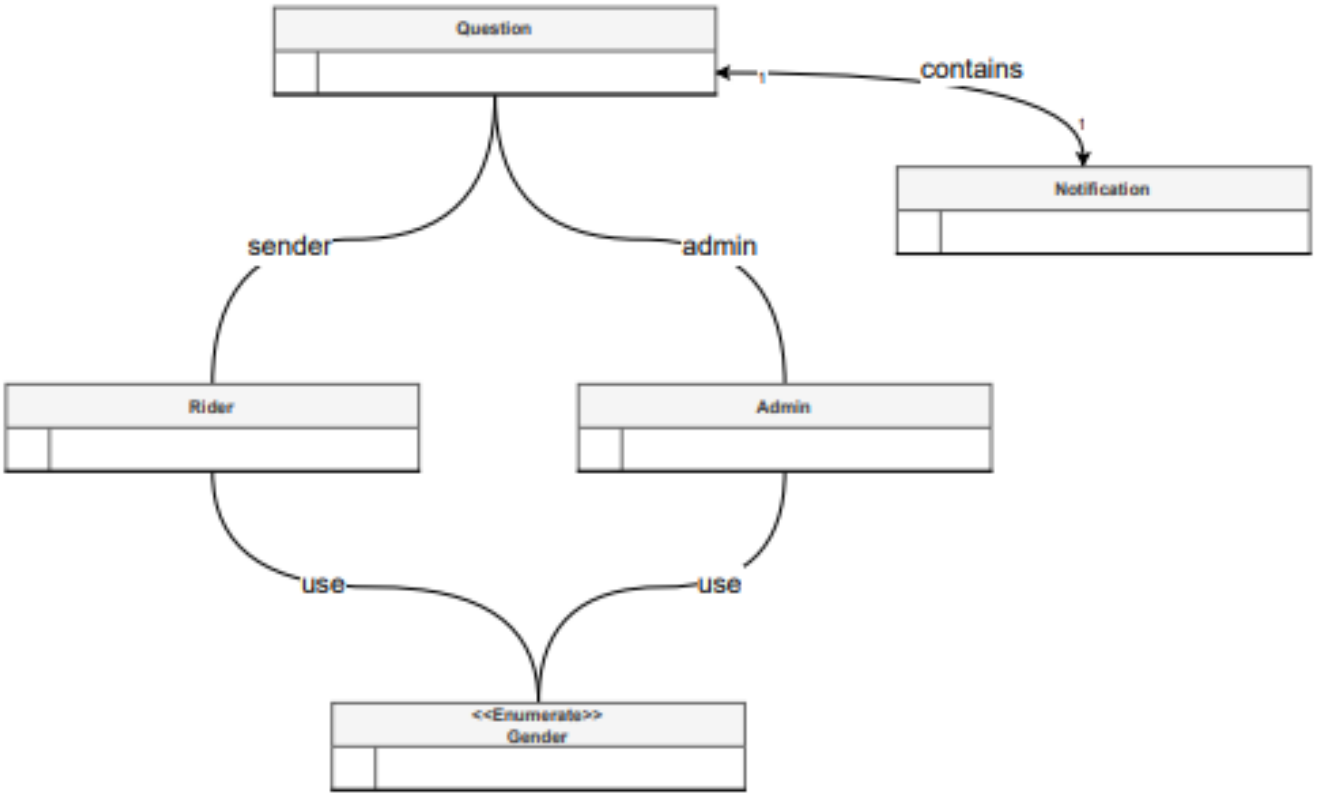


Figure 2.12 : Data object Relationships -Part A

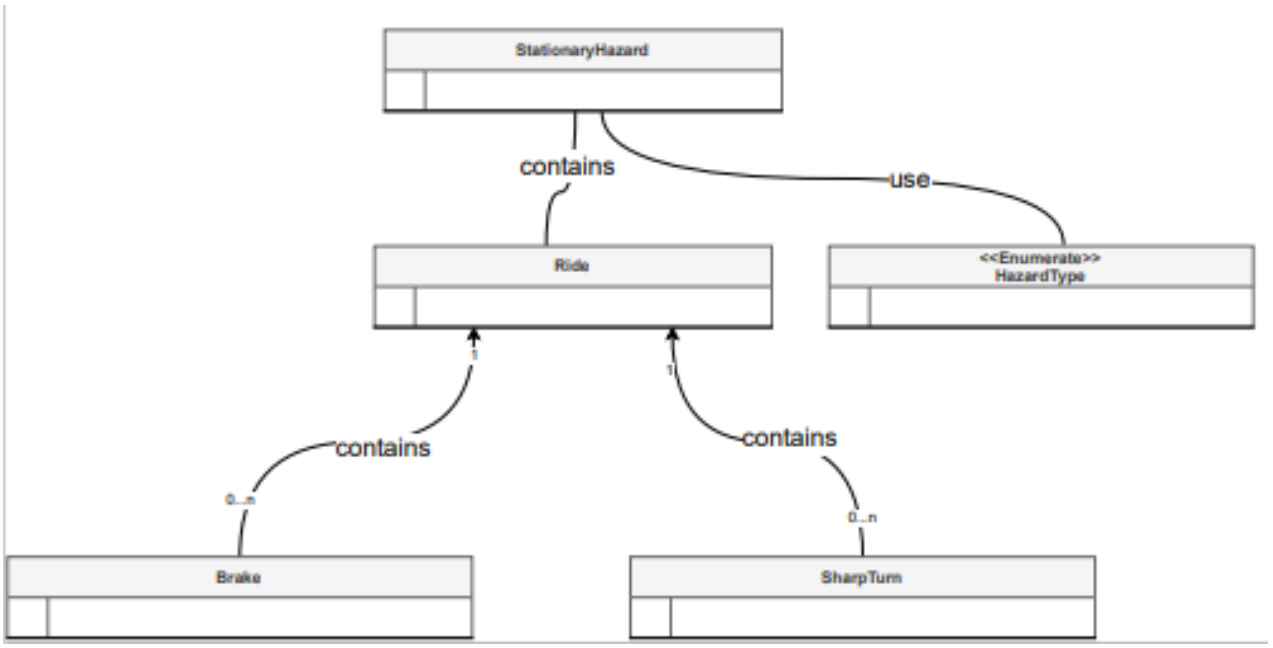


Figure 2.13 : Data object Relationships -Part B

ERD

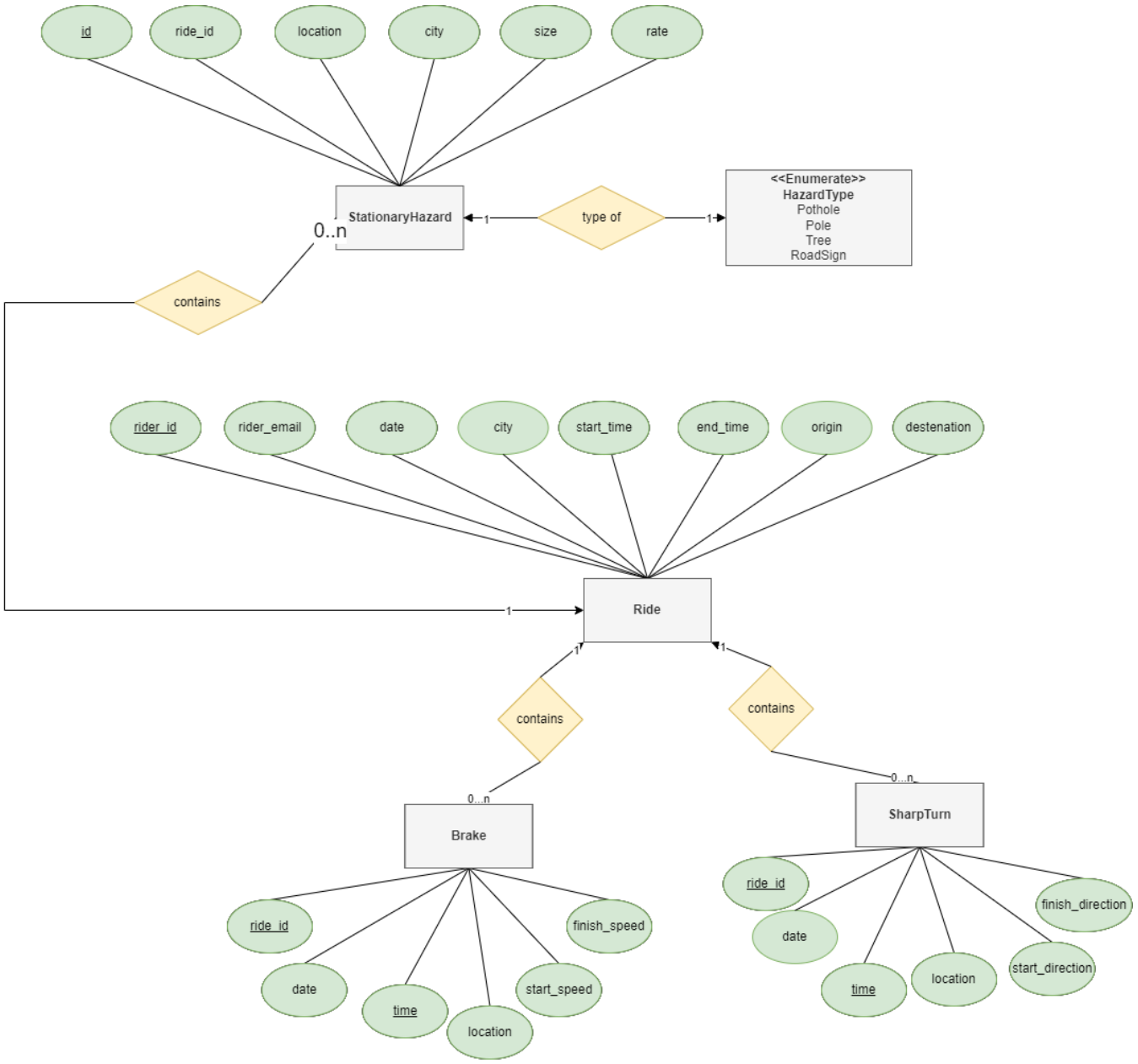


Figure 3.1 : ERD part 1

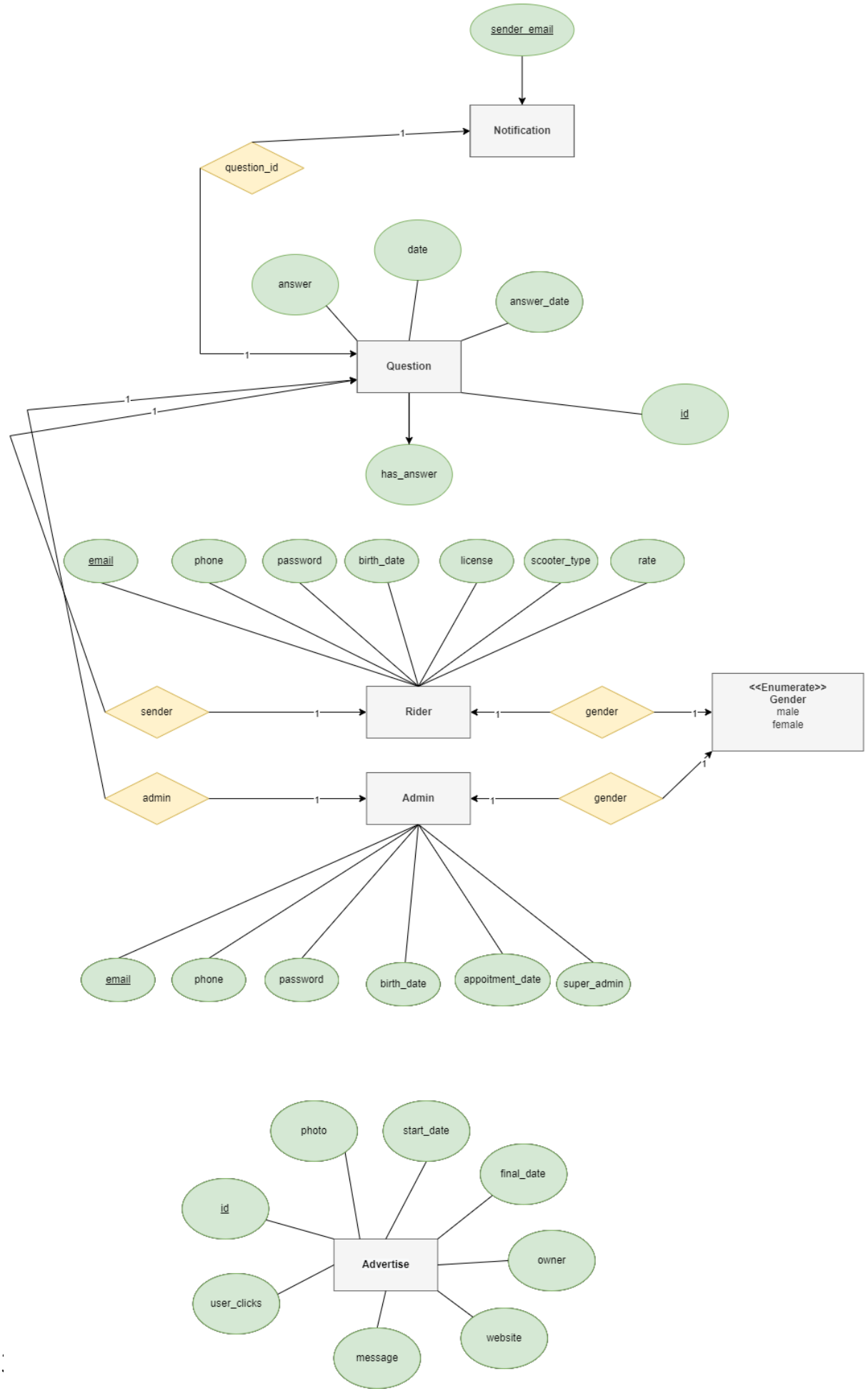


Figure 3.2 : ERD part 2

Behavioral Analysis

Events

- **Server Initialization Event** - Upon system initialization it will load information from a configuration file, and will fetch information from the database if it exists.
  If there is no information in the database which means it's the first initialization of the system then the system will simply create all classes and appoint default admin.

- **Raspberry Pi Initialization Event** - Upon raspberry pi initialization it will load information from a configuration file, start the camera and start recording video.

- **Server Shutdown Event** - Upon receiving a shutdown event the Server will finish any computations that were in the process of being done, save all information to the database, write information regarding the shutdown to the logger and shutdown gracefully.

- **Finish Ride Event** - Upon finishing a ride the raspberry pi will gather all of the information he gathered during the ride and will send the information to the server which in turn will update the system according to information.
  The server will save the information about new hazards that were discovered during the ride, update rider rating based on the rider behavior during the ride and will store information about the ride in the system such as: date, source, destination.

- **Hazard Detected Event** - Upon identification of a hazard while driving, the raspberry pi will store information regarding the hazard such as: location, type, time of detection.
  The raspberry pi will accumulate the information during the ride and will send it to the server upon finishing the ride (Finish Ride Event)

- **Start Ride Event** - Upon Pressing the button by the rider, the raspberry pi will start analyze the video and detect hazards on the road.
  Pressing the button again will cause the raspberry pi to finish the ride : save the data from the current ride and send it to the server

States

- **Server up** - A state where all system components are supposed to work, when the server provides its services and responds to every request, the RP works as an independent unit and at the end of the calculation (trip) sends the data to the server.
  For this mode, the developers should run the server application on the cloud and provide the host & port data.

- **Server up with no data** - A state who describes the first use of the system, when no data from the database should be load,
  the system will be in this mode in two cases:
  when the server is running for the first time
  after a developer called the 'reset' method.

- **Server down** - A state that each RP unit should work independently, when the ride is finished, the RP will save the data about the ride and wait to send it until a connection is established.
  In this state, the rider application should display a message explaining that the service is currently unavailable while the admin application should provide an explanation of the reason for the service interruption.
  For this mode, the developer should apply the 'shut down' method.

- **RP Connect -** A state when the RP is successfully connected to the server and transmit data after finish rides.

- **RP Disconnect** - A State who consist of the two following sub-states:
  1. The RP has been initialized by the company, although the user did not complete the registration process, that is, the user has received his personal serial number from the company but has not yet completed the registration process in the rider application.
  2. The RP fails to establish a connection to the server.

  In this situation, the RP system should work independently but the server should warn of data loss in this case.

- **RP Malfunction** - A situation where the RP is malfunctioning, in this situation the care of a technical person capable of performing tests on all the different systems of the RP is required.

Changing system states methods:

- **Reset:**
  precondition : ‘Server up’ mode
  postcondition : ‘Server with no data’ mode

- **Shut down:**
  precondition : ‘Server up’ mode
  postcondition : ‘Server with no data’ mode

- **Register:**
  precondition : ‘RP Disconnect’ mode
  postcondition : ‘RP Connect’ mode

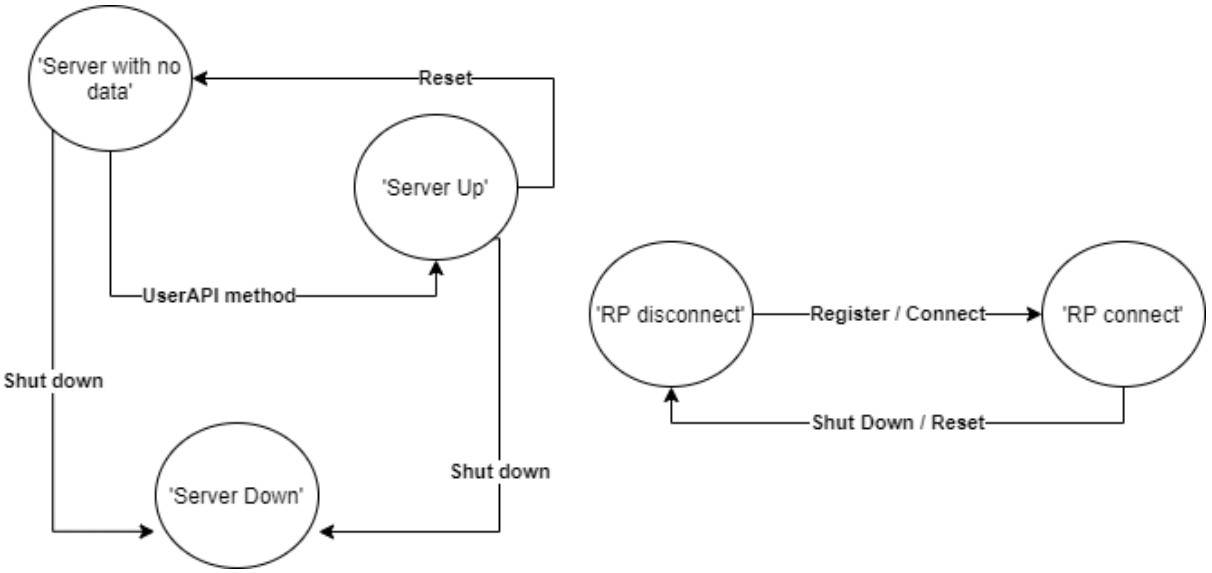


Figure 4 : States Diagram

Sequence Diagrams

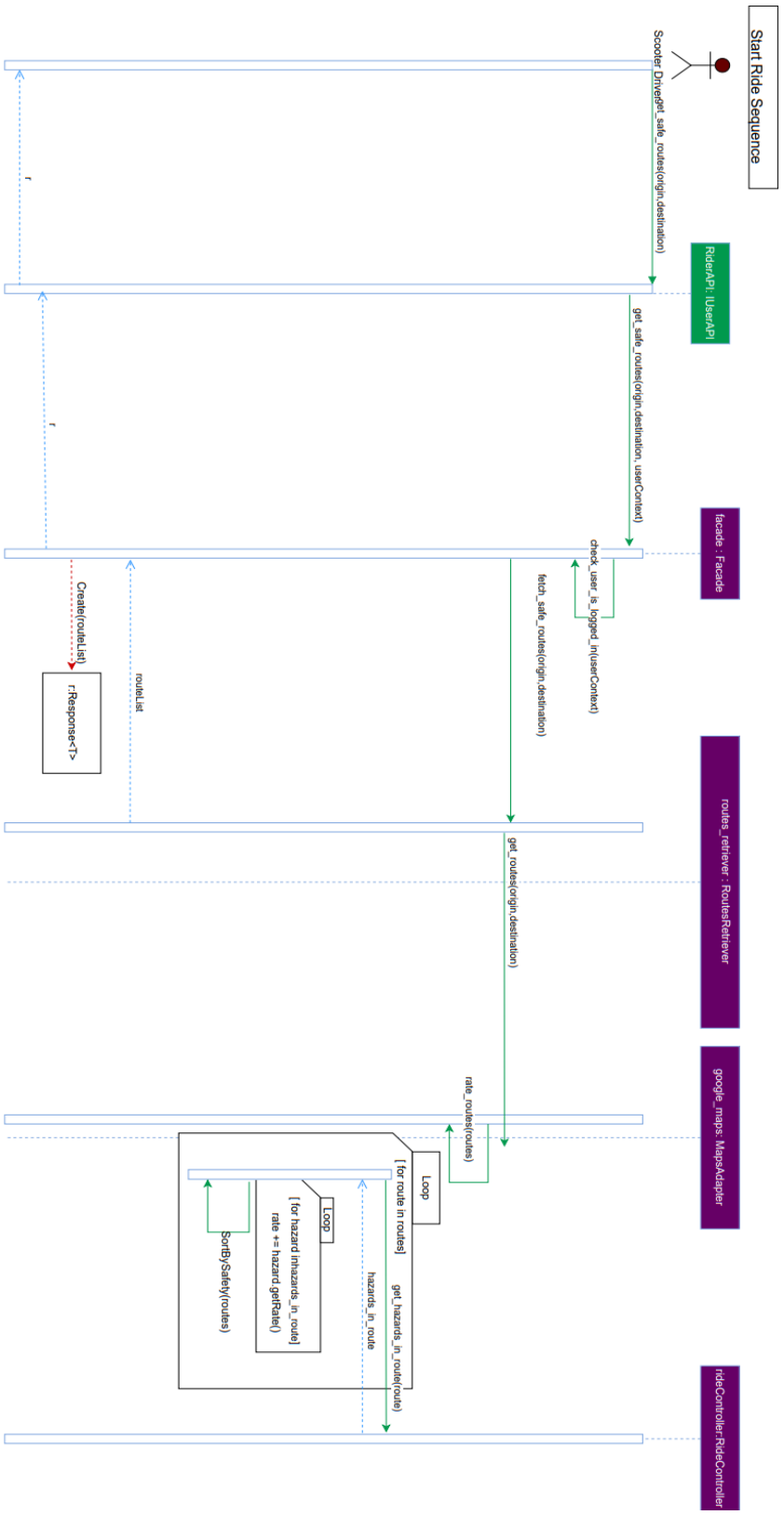


Figure 4 : Sequence Diagram

Object-Oriented Analysis

Class Description

Packages

Internal Packages

Server-side:
The server side consists of several layers that are designed vertically in our system,
the layers that exist on the server are:

- **API Interfaces** - contains interfaces for requests from the various systems: admin, rider and Raspberry Pi.
- **Service layer** - the interface layer between the external components and the business layer, including the following packages:
    - Communication package - the package that is responsible for managing the communication with the various components of the system, includes implementation of the API using the Rest API.
    - External Services package - have the responsibility for connect and send requests to the Google maps api, including handling responses range answers, implemented with proxy and adapter.
    - Initialization package - includes the configuration files and the "server initialization" class that includes the following key methods:
        - Set static vars - initialize static variables from a configuration file.
        - Connect external services - a method responsible for creating a connection to the external services (Google Maps) using connection details found in a configuration file.
        - Create rp config file - a method that creates a configuration file that the server will send to the various components for a default initialization of settings.
        - Connect database - a method that is responsible for creating a connection to the database using connection information found in a configuration file.

- **Business layer** - holds most of the logic in the system, includes the specific algorithms for rating users and dangers.
  including the following packages:
    - Advertise Module - A module who manages the data about the advertisements in the system.
    - Hazards Module The module that handles requests to add, edit and delete hazards, in addition, contains enum Hazard Type who sets the domain of hazards the system supports.
    - Questions Module -The module who handles questions from the users to admins, and notify when an answer arrives from an admin.

- Rating Module - the core of the server-side, implements the ideas about rating users and hazards in our system.
  Main classes:
  - Hazard Rate Calculator - The role of this class is to rate the various dangers that may be found in a rider's path.
    According to the danger score, which is based on various parameters such as the width of the danger in relation to the width of the road,
    The system will be able to estimate the safety level of different roads.
  - User Rate Calculator - The role of this class is to rate the level of safety of a rider based on the trips he has made.
    According to the rating of the riders, it will be possible for the system administrators to receive a business situation assessment regarding transactions with the various riders.
  - Routes Retriever- The class who should get the data from google maps and sort the routes according to the hazards rates inside each route.
- Users Module - The unit which manages both users & admins.

- **Data access layer** - a layer that is the separation between the database and the system, manages the execution of queries to the database.

- The **Utils package** is used by the different layers in the code and its role is to provide static function services.
- A **logger package** that is divided into three loggers that write to text files:
  - Error logger - responsible for documenting all errors that the system detected and handled
  - System logger - responsible for documenting system initialization, contacting external services, creating a connection with the database, initializing configuration files
  - Server logger - records all the requests the server received.

Raspberry Pi:
Raspberry Pi contains different layers:

- **Service layer** - its role is to initialize the system settings according to configuration files and create a connection with the server, through which the information will be transferred. Contains the two central modules:
  1. Communication package
  2. Configuration package
- **Domain layer** - implements the main logic of the component and its role is to receive input from the camera, process the image and alert if necessary. Contains the following packages:
  1. Alerts module - responsible for outputting input to the user by various means
  2. Camera module - responsible for receiving input from the camera
  3. GPS module - responsible for receiving the location of the component

4. Image analysis module - the main logic of the system, its role is to receive an image and return an answer - "is there a danger in the image".

   This module includes the following main classes:

   - Hazard Detector - The role of this class is to process frames and decide if the frame contains hazard, in addition, the class should classify the hazard from the range of possibilities that the system supports.
   - Road Detector - This class should classify the route based on the first frame that arrived from the camera.

External Packages

ultralyticsplus provides the potholes detection model called YOLO, the model is trained and compatible with PIL image of size (640,640) as input.
The model detects pothole in the frame and marks its boundaries

OpenCV provides a real–time optimized Computer Vision library, tools, and hardware. It also supports model execution for Machine Learning.

React Native combines the best parts of native development with React, a best-in-class JavaScript library for building user interfaces.

Java Spring Framework provides a comprehensive programming and configuration model for modern Java-based applications - on any kind of deployment platform.

Gpsd is a service daemon that monitors one or more GPSes or AIS receivers attached to a host computer through serial or USB ports, making all data on the location/course/velocity of the sensors available to be queried on TCP port 2947 of the host computer.
https://maker.pro/raspberry-pi/tutorial/how-to-use-a-gps-receiver-with-raspberry-pi-4

Google-maps-services-java This library brings the google maps api web services to our server-side Java application.
https://github.com/googlemaps/google-maps-services-java

Hibernate is an object–relational mapping tool for the Java programming language. It provides a framework for mapping an object-oriented domain model to a relational database. Hibernate handles object–relational impedance mismatch problems by replacing direct, persistent database accesses with high-level object handling functions.

Class Diagram

Link To Draw.io :

https://drive.google.com/file/d/1F1jfwhJMyfHAKvEk57Lb5p482_jjP0A4/view?usp=sharing

User Interfaces

Rider Interfaces

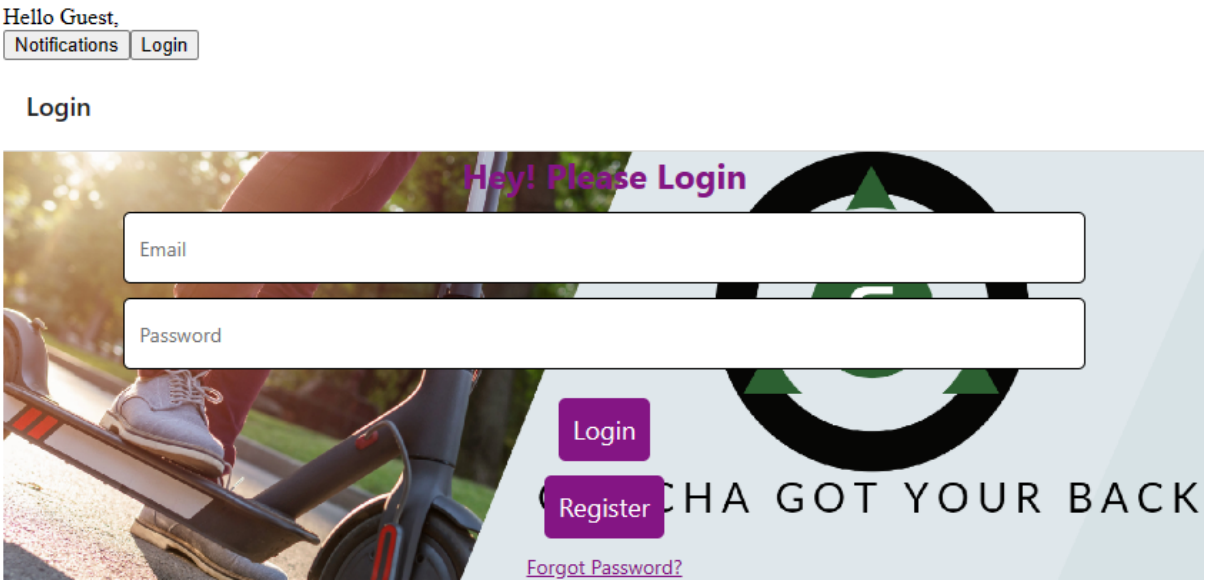


Figure 6.1 : Login window

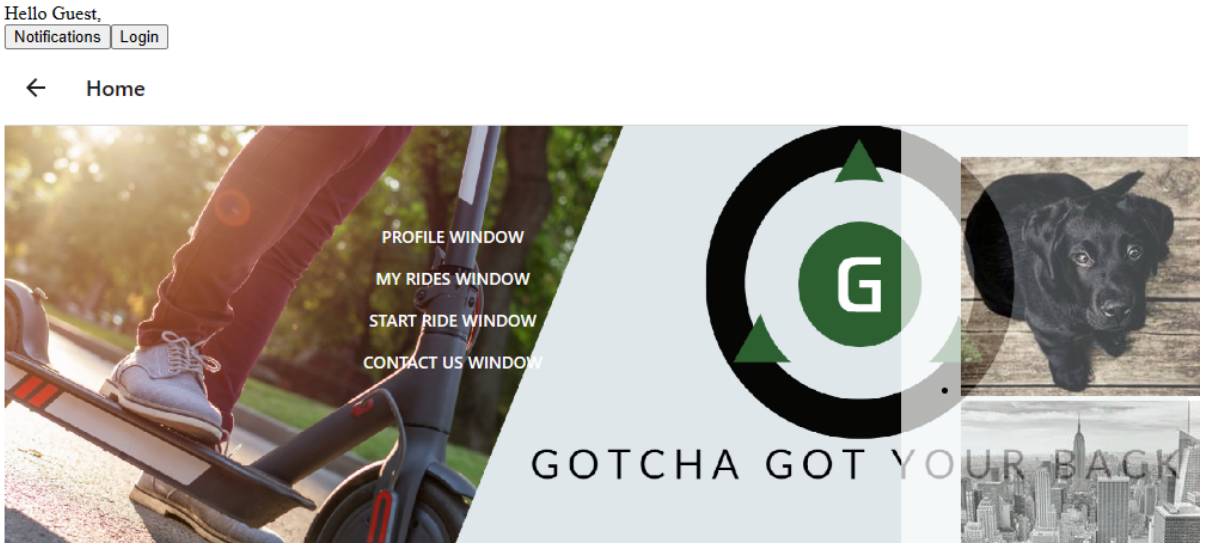


Figure 6.2 : Home window

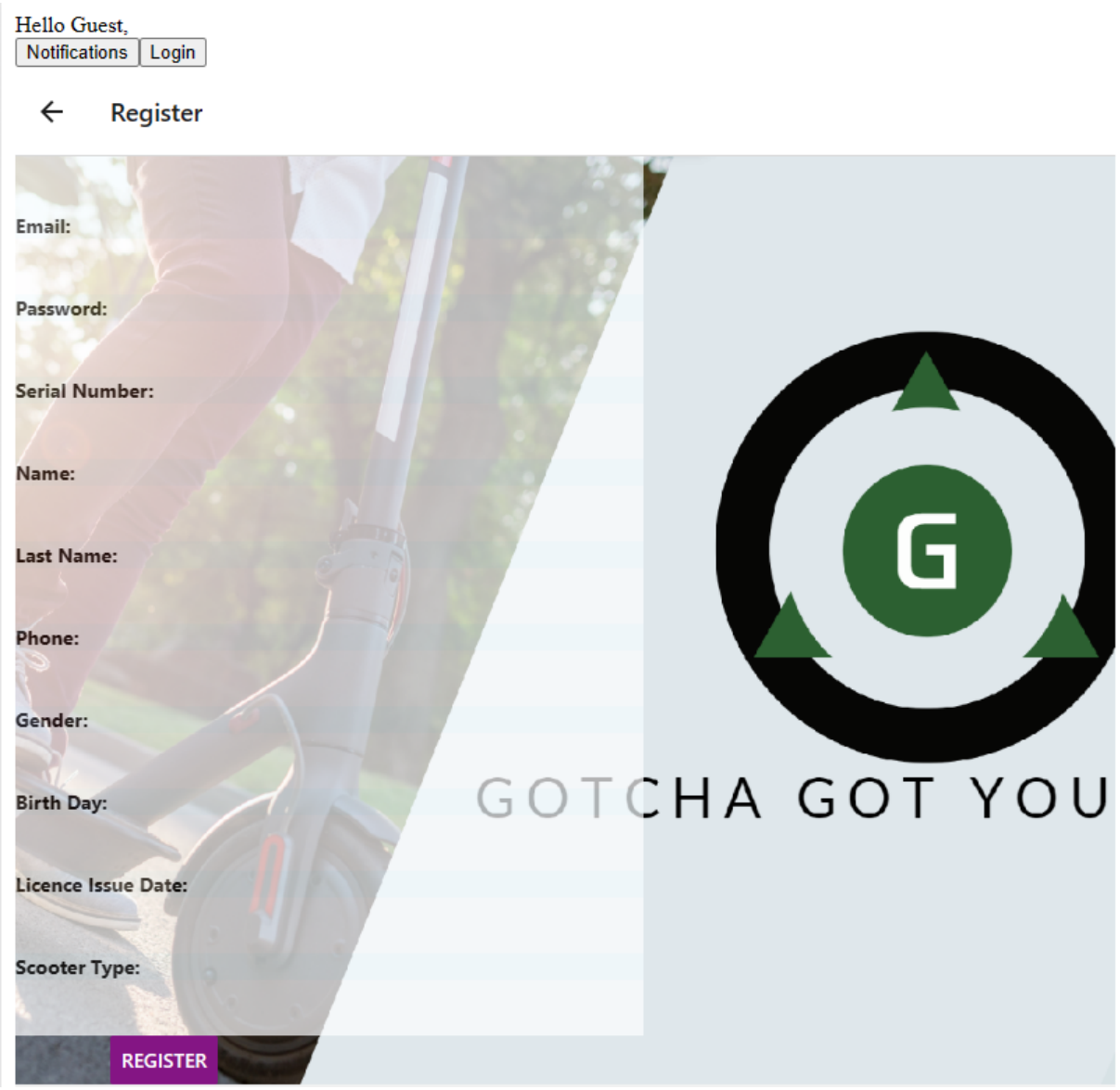


Figure 6.3 : Register window

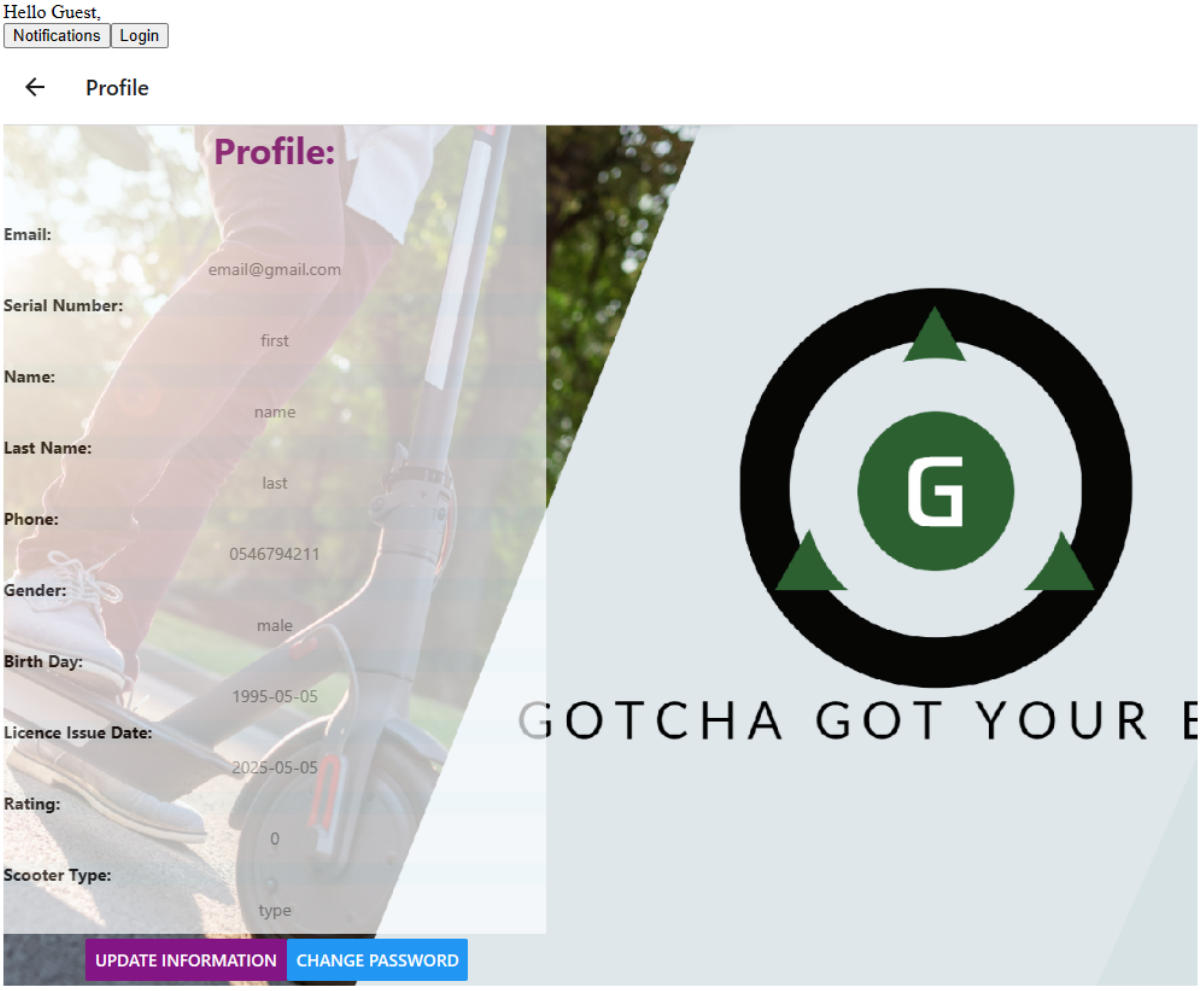


Figure 6.4 : Profile window

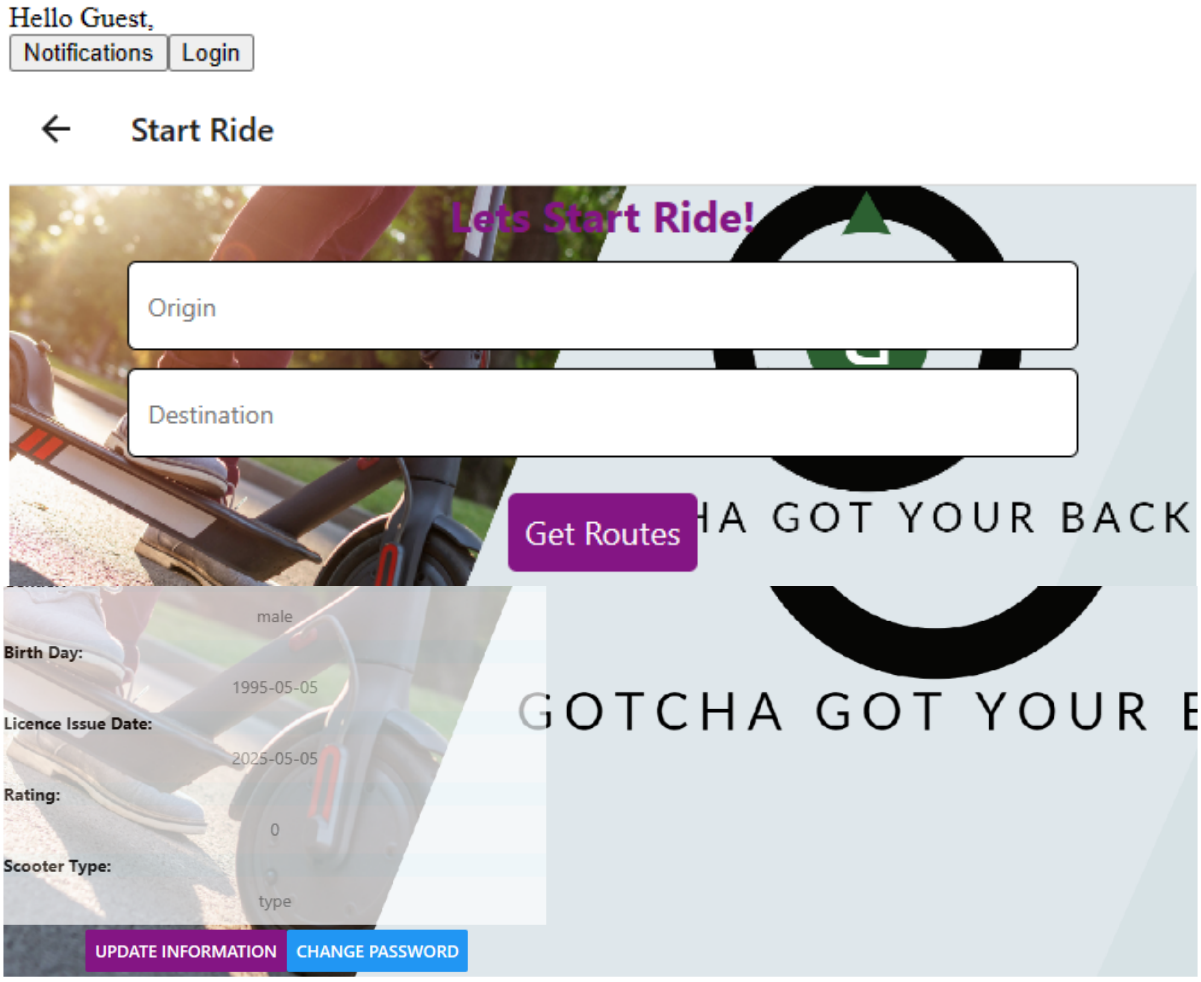


Figure 6.5 : Start Ride window

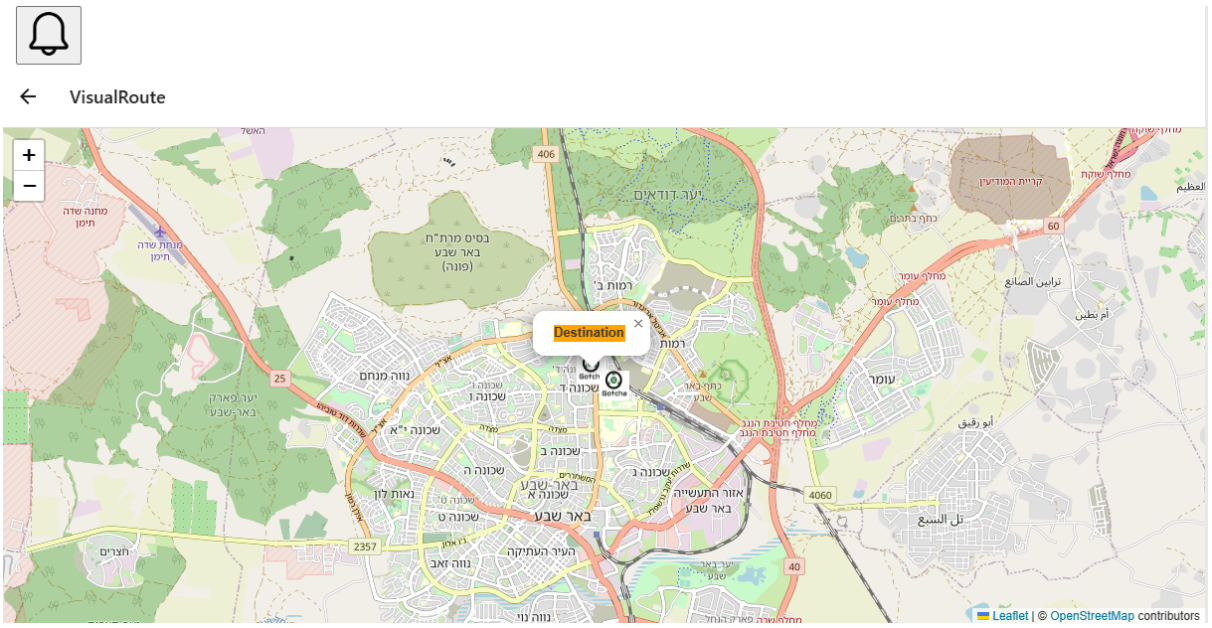


Figure 6.6 : Visual Route window

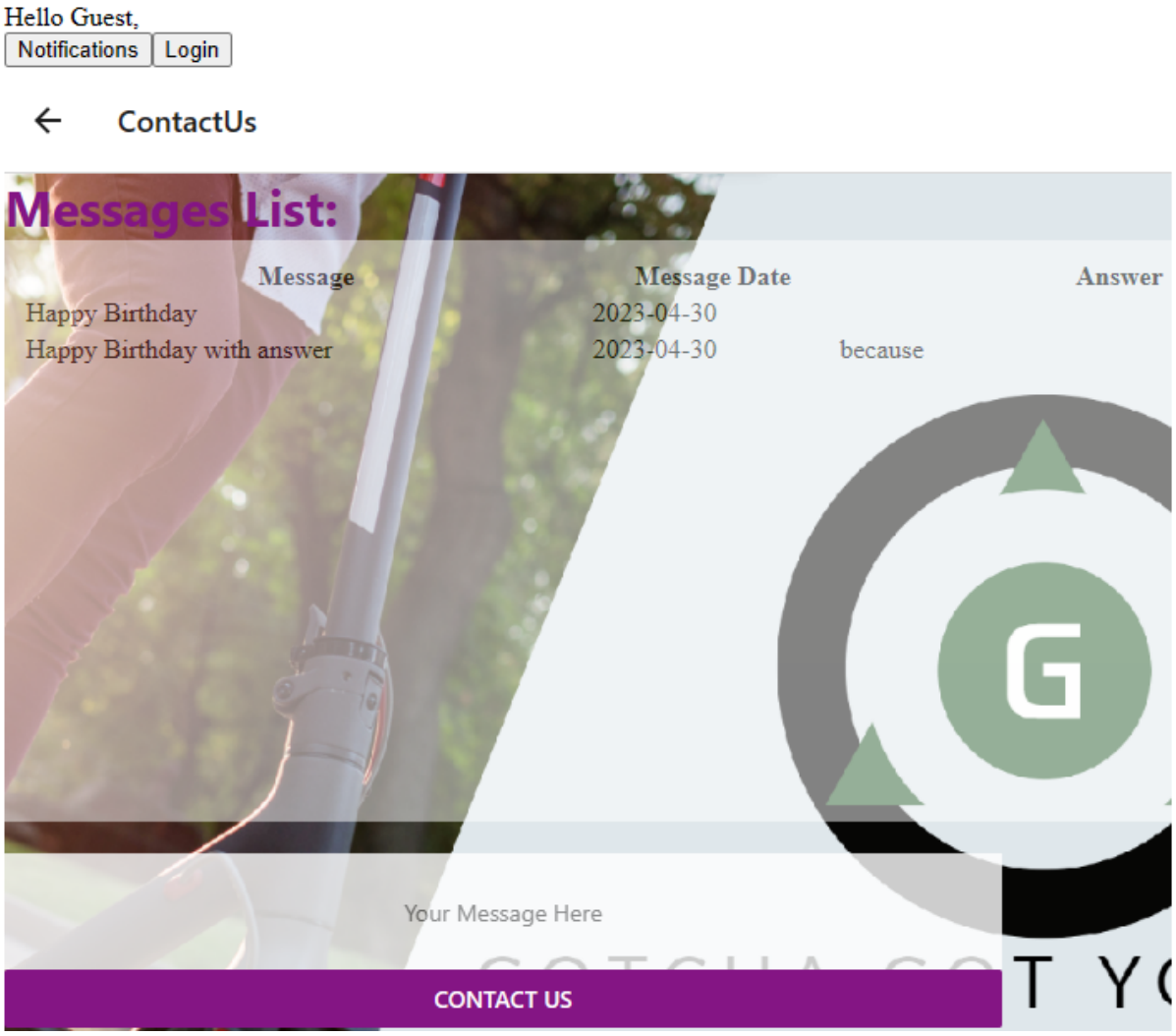


Figure 6.7 : Contact Us window

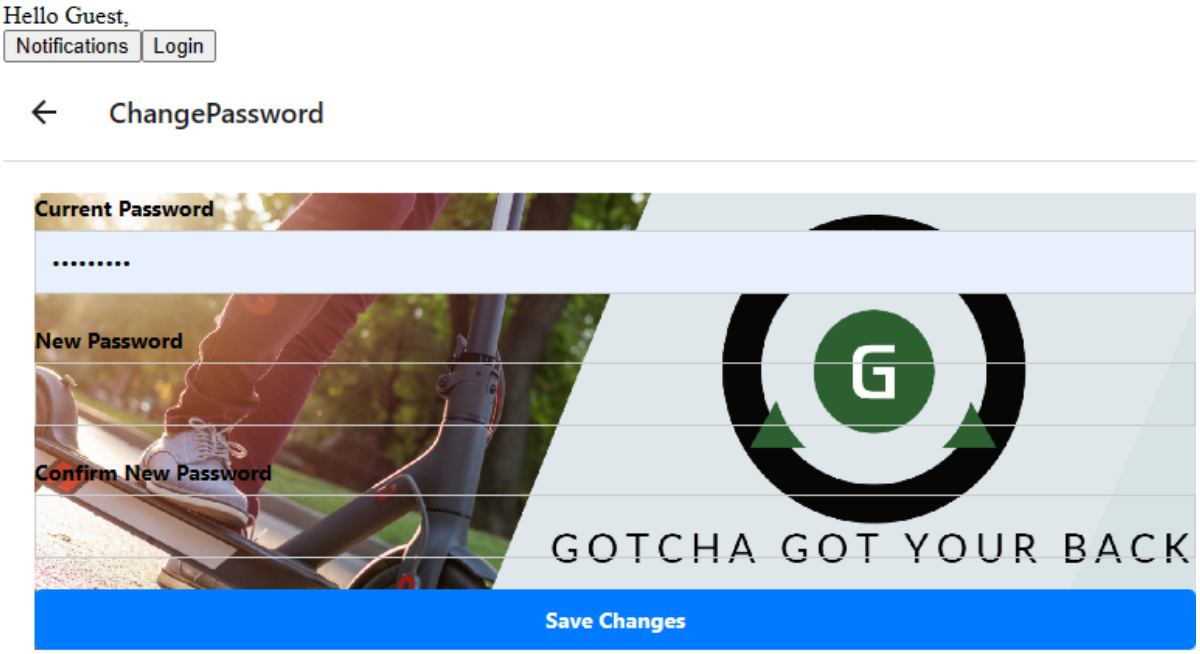


Figure 6.8 : Change Password window

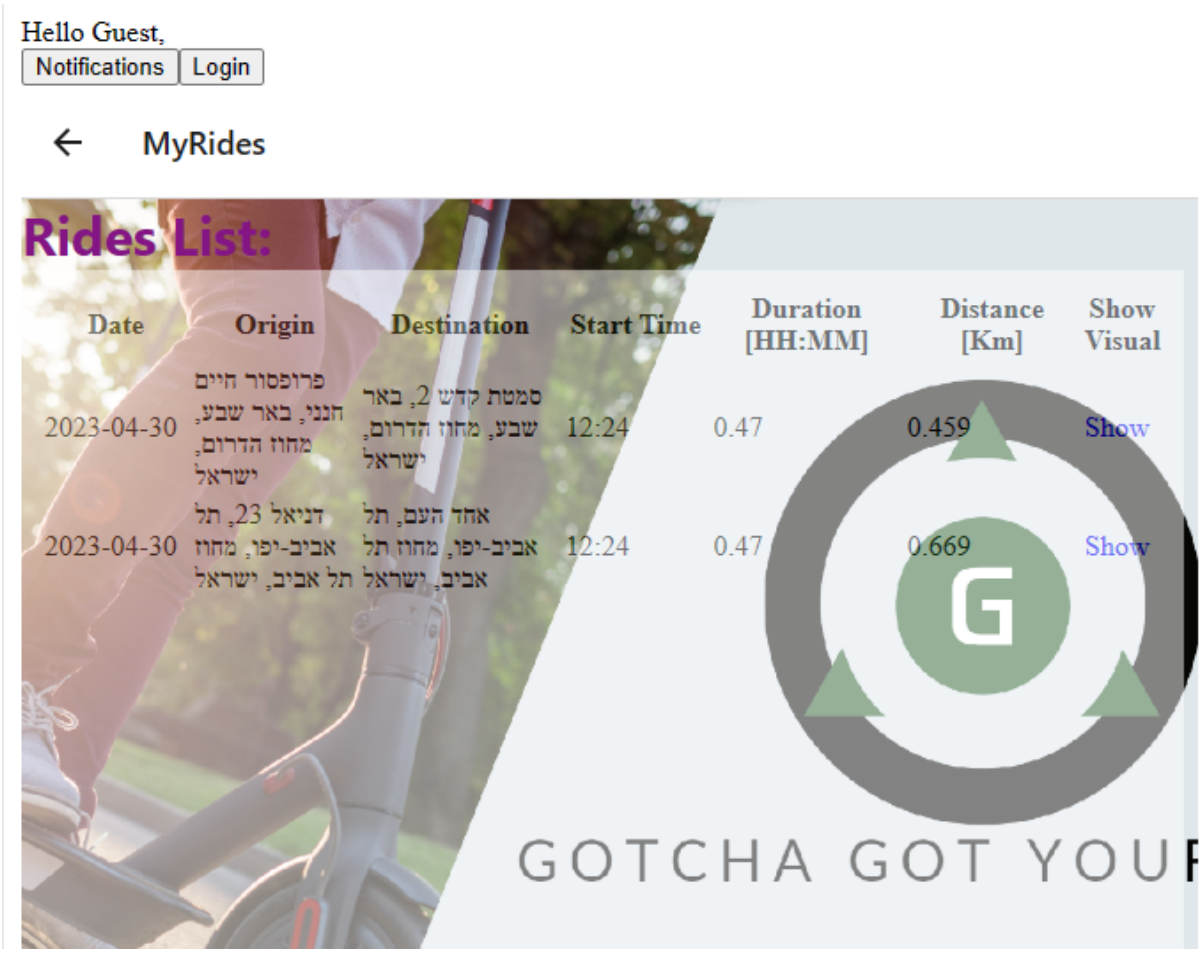


Figure 6.9 : My Rides window

Hello Guest,
Notifications Login

← VisualRide

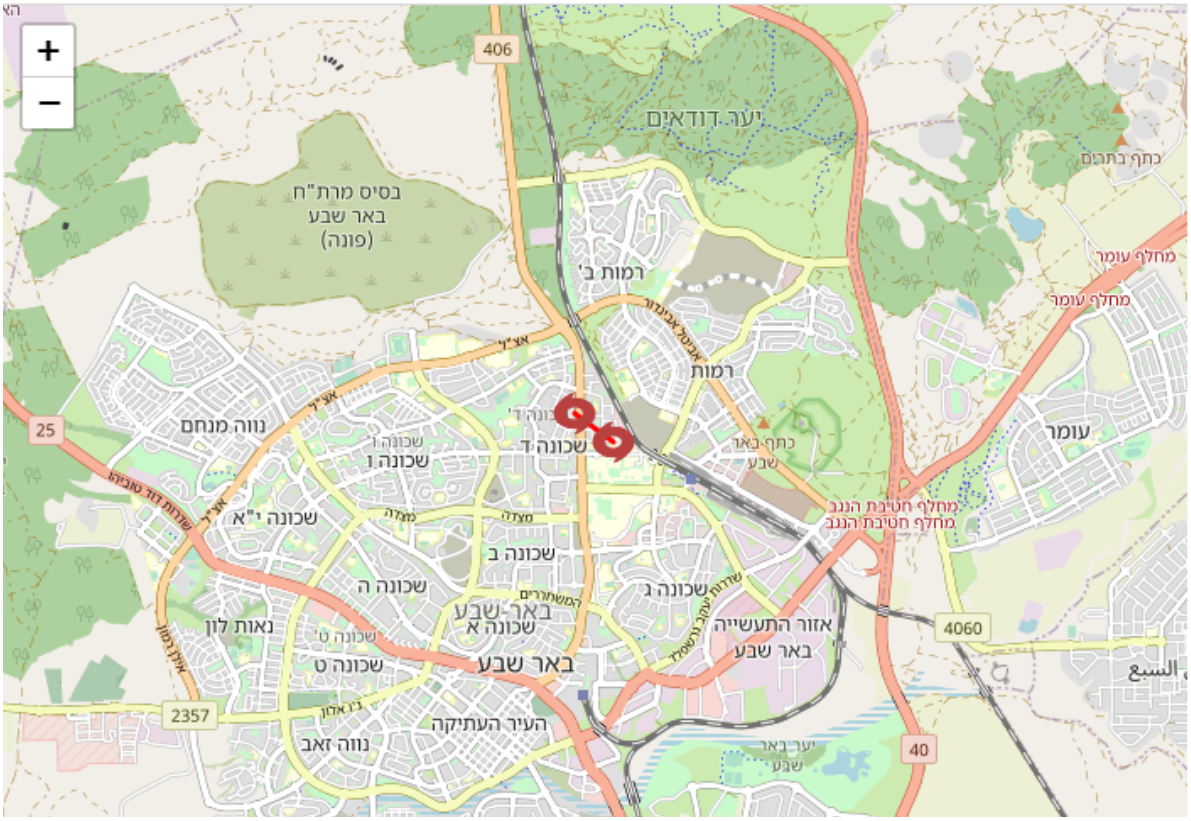


Figure 6.10 : Visual Ride window

Admin Interfaces

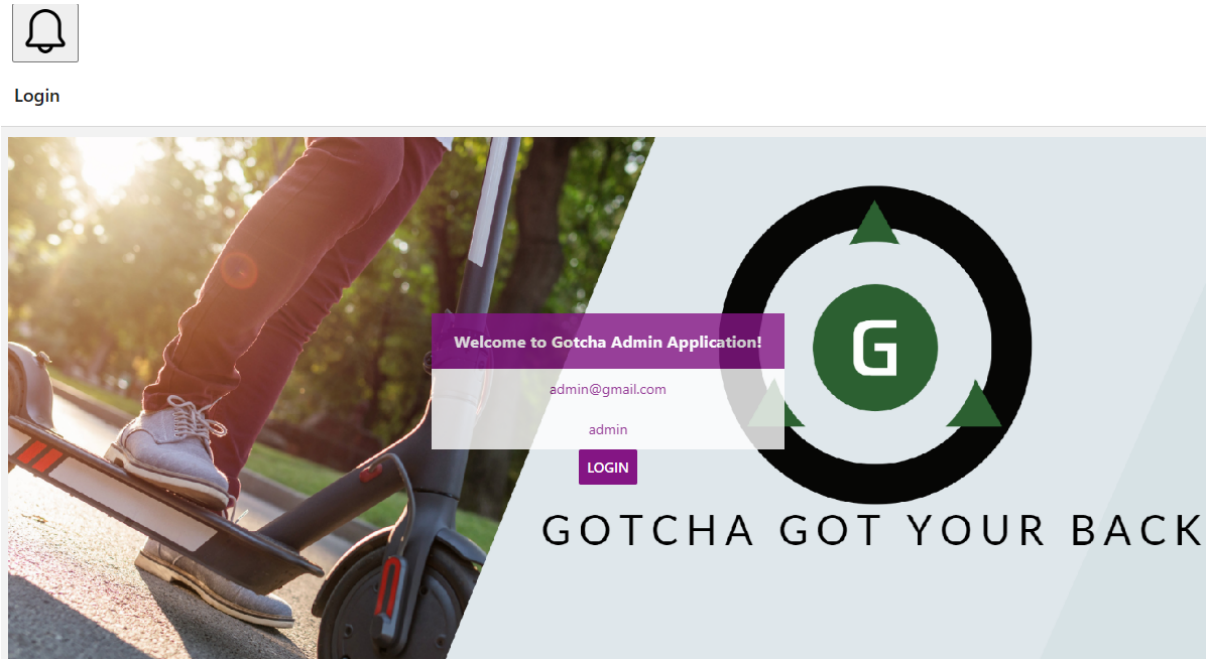


Figure 7.1 : Login window

Figure 7.2 : Home window

Figure 7.3 : Riders window

Figure 7.4 : Admins main window

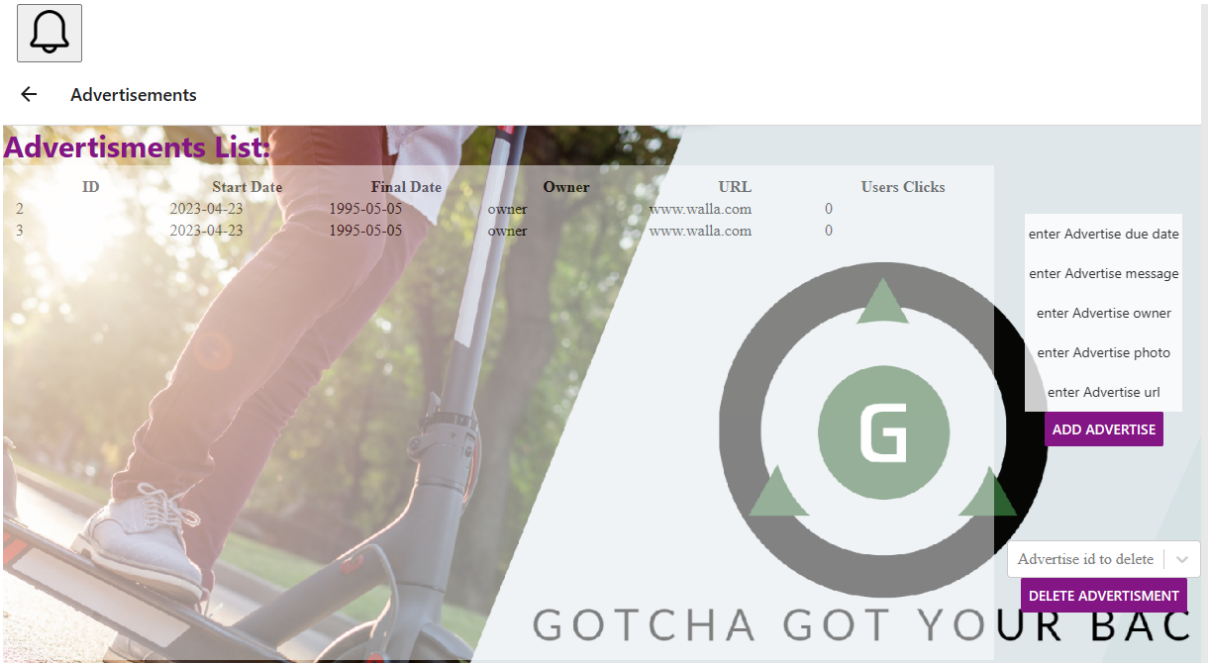


Figure 7.5 : Advertisements window

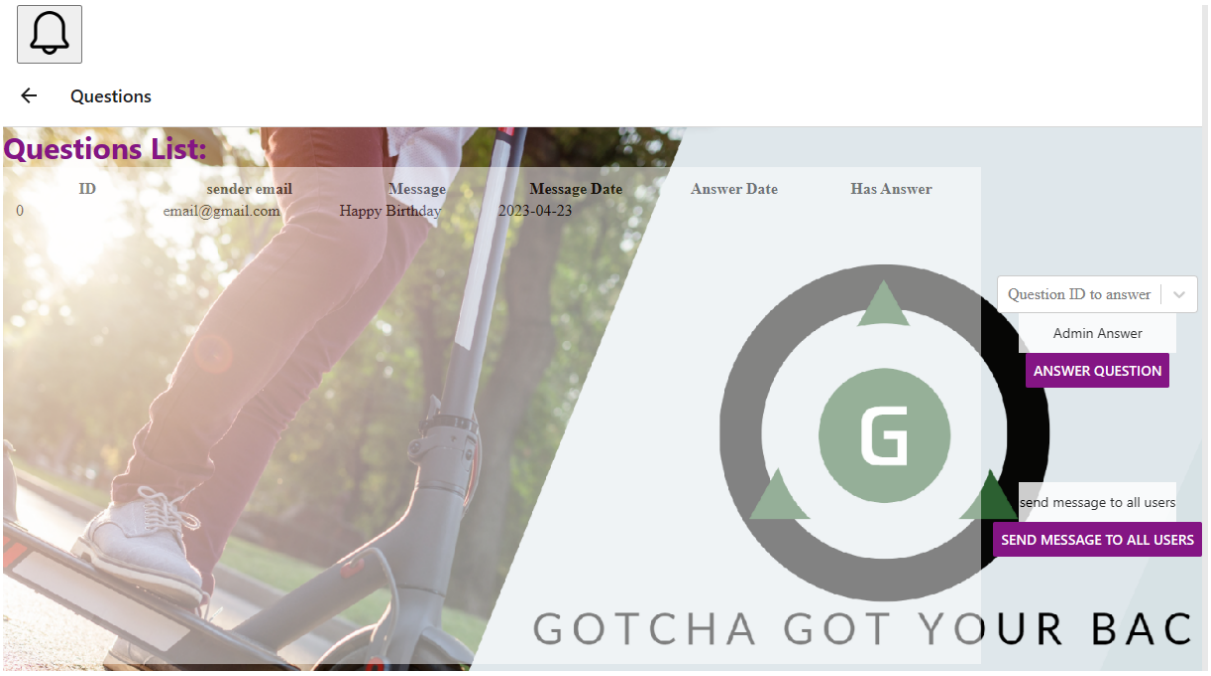


Figure 7.6 : Questions window

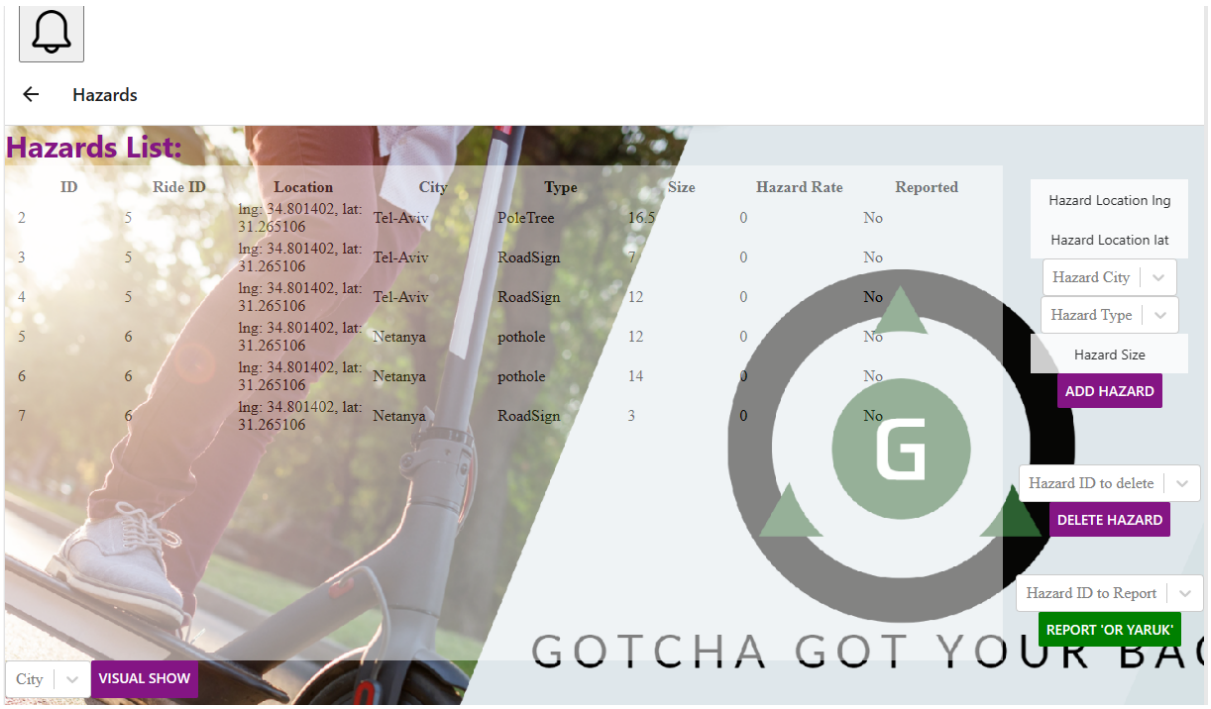


Figure 7.7 : Hazards window

Figure 7.8 : Rides window

Figure 7.9 : Statistics window

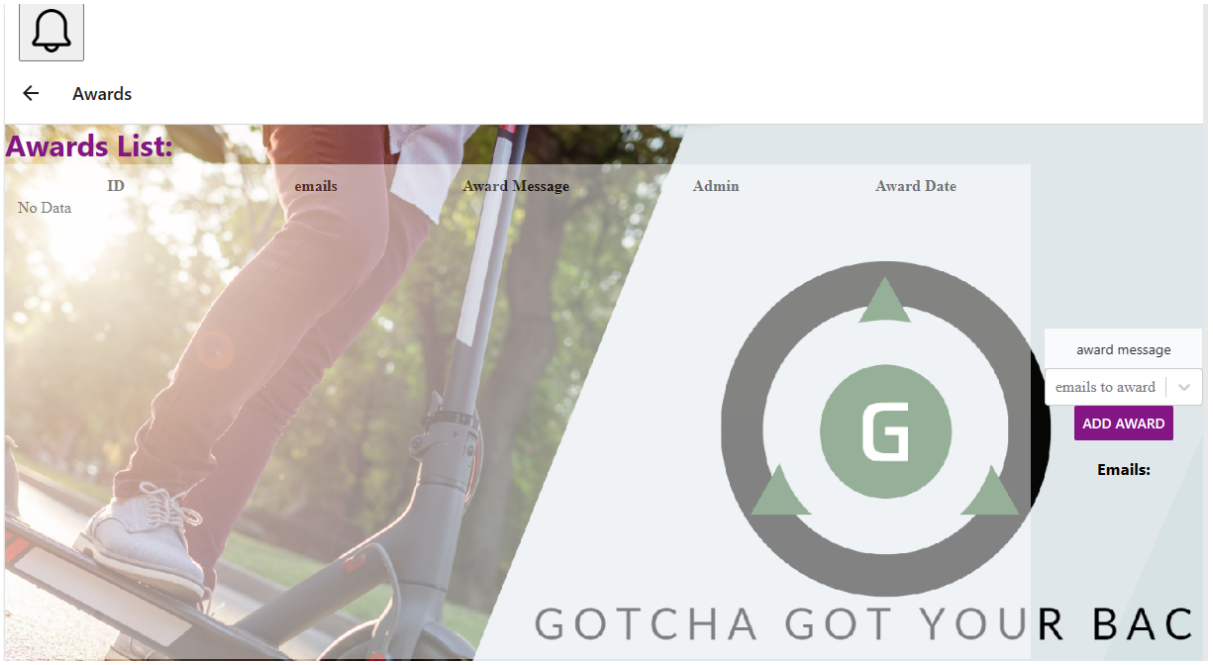


Figure 7.10 : Awards window

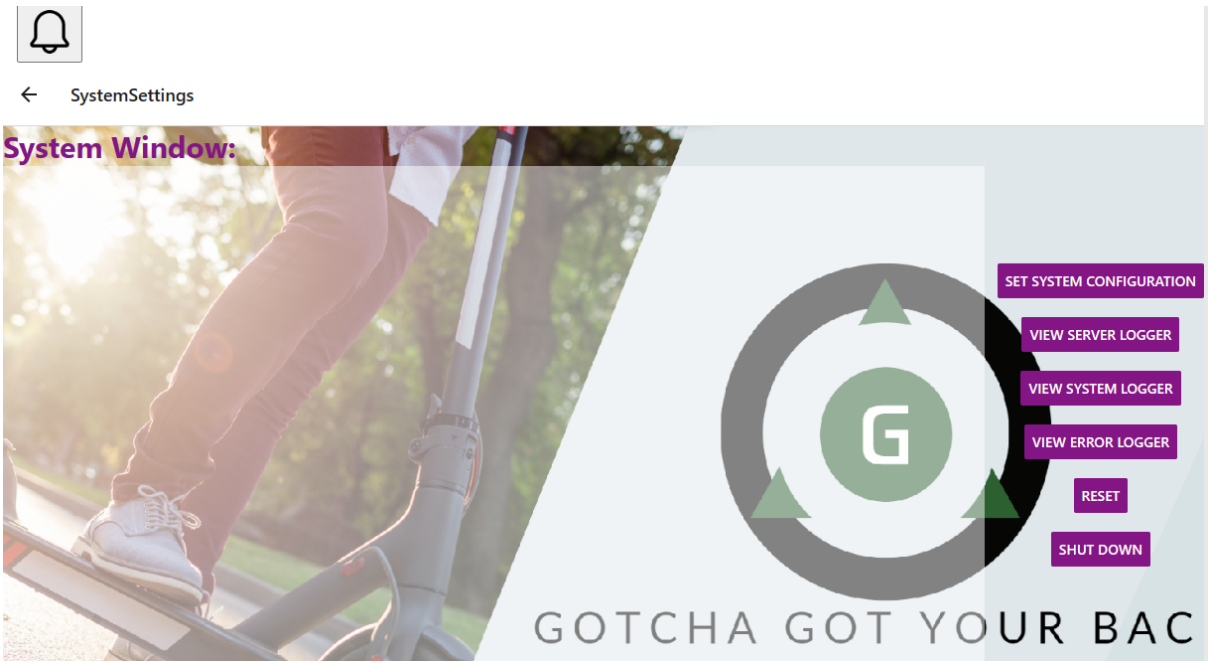


Figure 7.11 : System Setting window

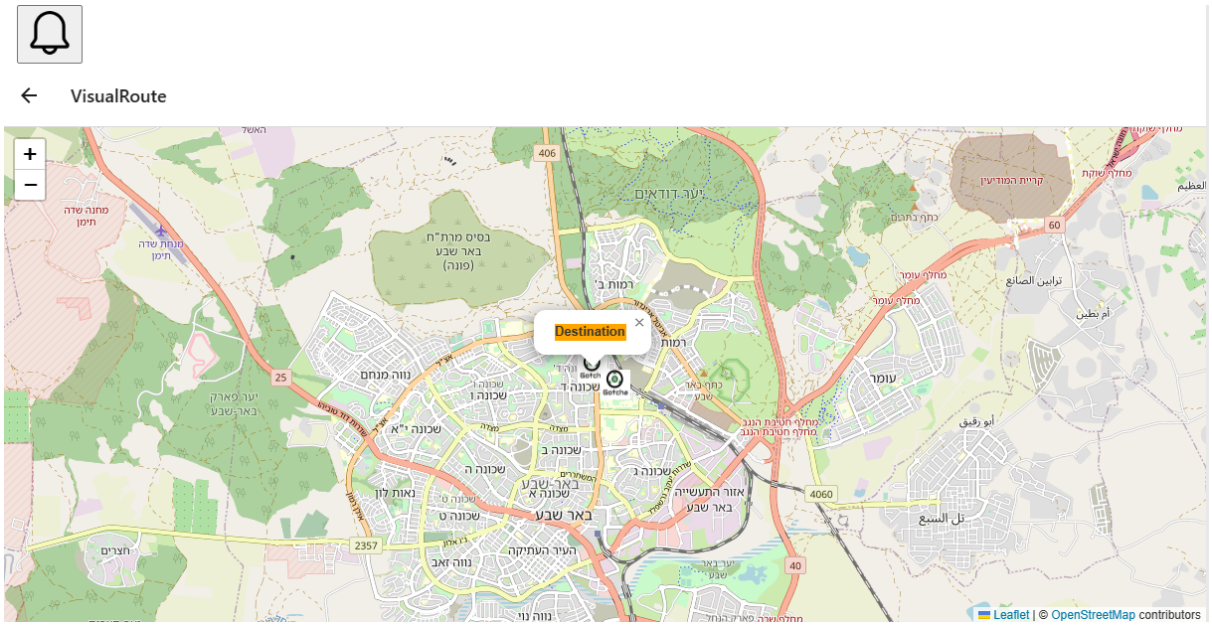


Figure 7.12 : Visual Route window

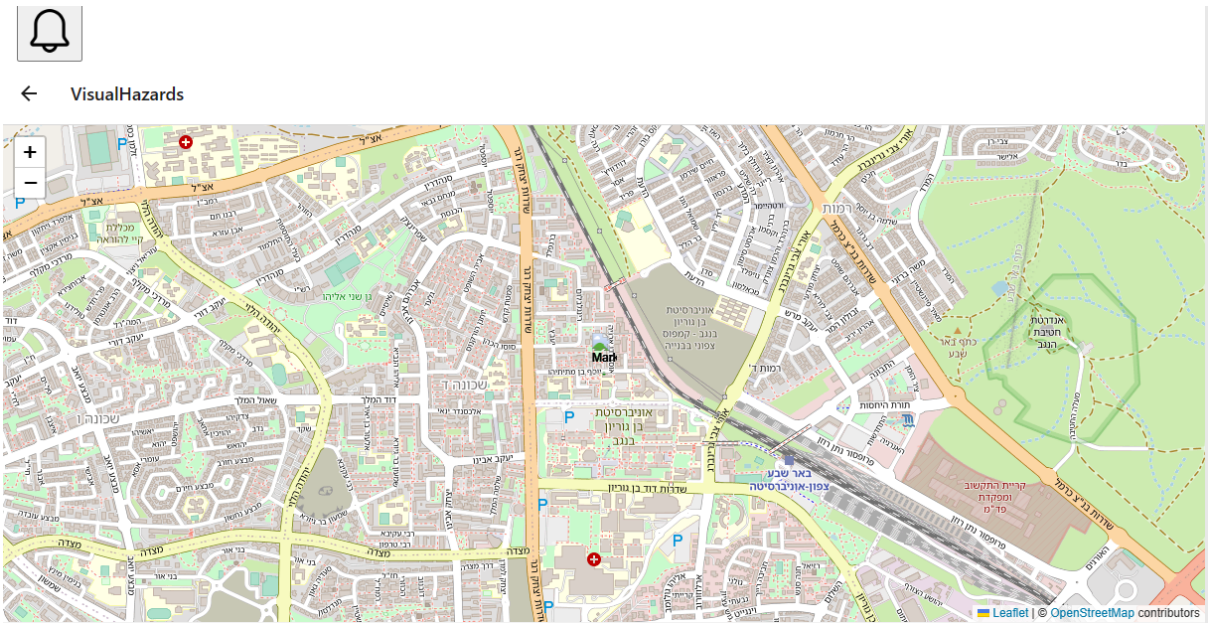


Figure 7.13 : Visual Hazard window

Figure 7.14 : Set Configuration window

Testing

1. Unit test:

Our unit tests plan for the following classes & modules according to preconditions and postconditions without considering the other system components.

Server:

- Advertise module - advertise controller: add, remove & edit advertisements.
- Hazards module - hazards controller: add, remove & edit hazards.
- Questions module - question controller: add, remove, edit & answer questions.
- Users module - users controller: add admin, change password, login, logout & register.
- Rating module -rate controller: rate rider, rate hazard.
- Utils - check all methods.
- Logger - write to loggers.
- Password manager - encrypt and decrypt passwords.

Raspberry Pi:

- Image processor module - detect all hazards types in a frame.
- Alert module - verify verbal and visual outputs from RP (manual).
- Camera Module - we will be testing the “get next frame” method.
- GPS module - we use Gpsd module, therefore, we will based on the package tests.

2. Acceptance tests:

The acceptance testing plan for the project will be performed per the usage scenarios using the data tables containing happy, sad and bad scenarios.
In accordance with the actors that activate the action, we will separate the tests into tests that are activated by each actor when the admin application, the rider application and the Raspberry Pi will activate the usage scenarios.

As part of our testing program, we will abstractly describe the Facade class on the server that implements the "complete functions" (that is, functions that are activated from another component and activate a set of sub-functions), as an endpoint that initiates the use case scenario.

For this tests, we intended to implement the following design patterns combination for separate system tests & implementation:

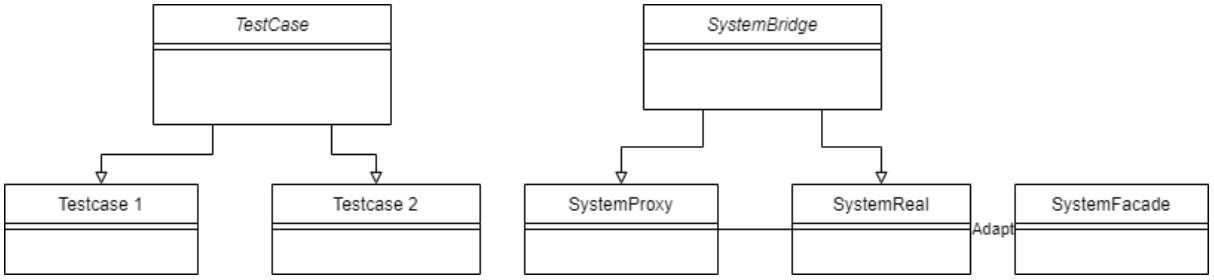


Figure 9.1 : Acceptance tests design

3. Integration tests:

Server - we will use bottom up methodology to test our integration,
We will use Mocks for testing each unit, and incrementally connect the separate units until we get complete integration of the server.

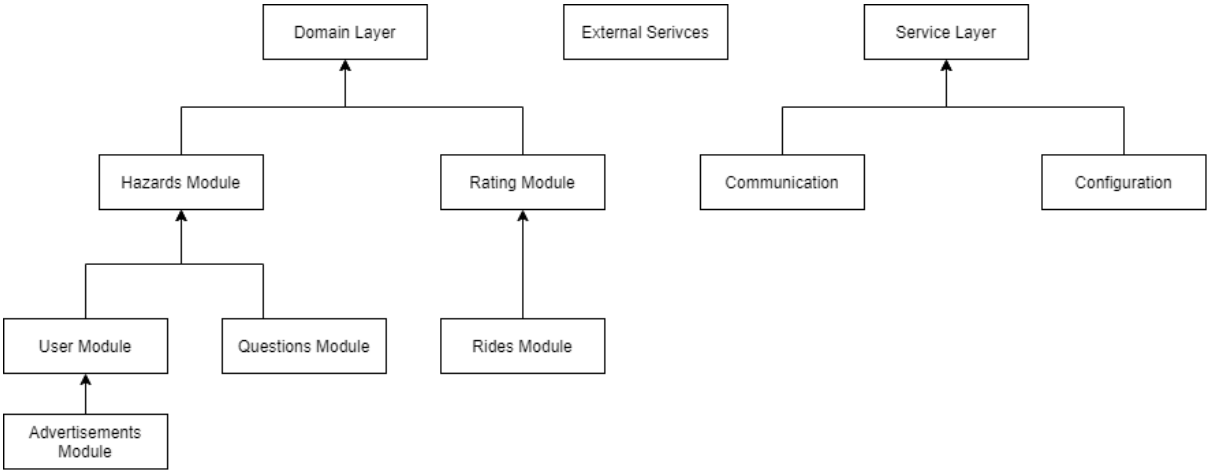


Figure 9.2 - Server bottom up integration tests design

Raspberry Pi - we will use bottom up methodology to test our RP integration, our main challenge is to test Camera & Alert modules automatically.
In addition, our video process module will test with our tagged images hoard.

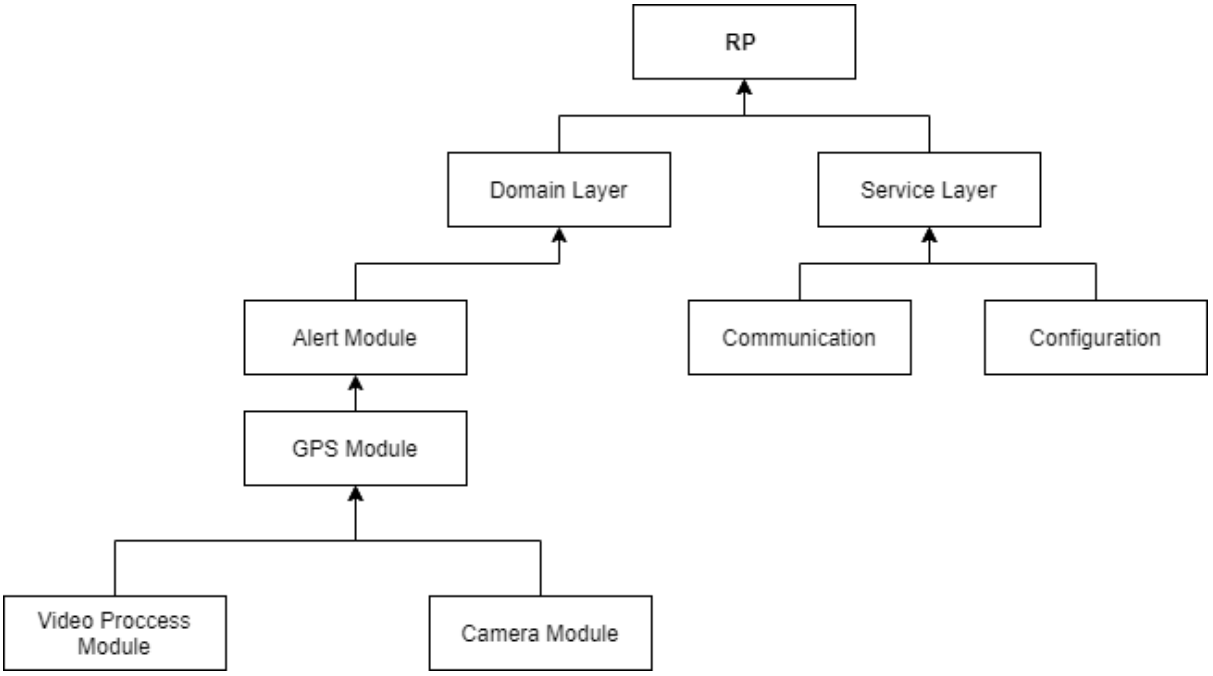


Figure 9.3 - RP bottom up integration tests design

4. System tests:

Our system tests will focus mostly on raspberry pi before we release it, and we will test the use cases of which rider is the actor.

In these tests, we will use all the real system components and units(except external services) and test all the Raspberry pi functionality.

The company should give us a road with all hazards types and we expected to get an alert for each one of them.

In this test, we use all the different options for Raspberry pi's configuration.

In addition, our team will check the admin & rider applications with different data for each use case.

5. Load tests:

According to 37 + 38 requirements, we will implement load tests in the server which includes handling 50,000 register requests(not at the same time), and 10,000 login requests at the same time - which means 100 login requests per minute during 10 minutes.

As of this writing, we plan to automatically examine the log files by searching and counting error messages, but other technological options will be examined.

Functional Requirements Test Plan:

1) **System Initialization -** Req 1
    a) **Case 1**
        i) **input:** None
        ii) **Expected Result:** System config is the same as described in configuration file
2) **Change External System** - Req 2
    a) **Case 1**
        i) **input:** valid external system was picked
        ii) **Expected Result:** System will now use the new external system
    b) **Case 2**
        i) **input:** invalid external system was picked
        ii) **Expected Result:** Exception is thrown

3) **Register -** Req 3
    a) **Case 1**
        i) **input:** valid credentials
        ii) **Expected Result:** new user successfully created
    b) **Case 2**
        i) **input:** invalid email address
        ii) **Expected Result:** Exception was thrown
    c) **Case 3**
        i) **input:** invalid password
        ii) **Expected Result:** Exception was thrown
    d) **Case 4**
        i) **input:** invalid raspberry pi serial number
        ii) **Expected Result:** Exception was thrown
    e) **Case 5**
        i) **input:** invalid birth date
        ii) **Expected Result:** Exception was thrown
    f) **Case 6**
        i) **input:** invalid license issue date
        ii) **Expected Result:** Exception was thrown
    g) **Case 7**
        i) **input:** invalid scooter type
        ii) **Expected Result:** Exception was thrown

4) **Edit user Profile** - Req 4
    a) **Case 1**
        i) **input:** valid new user information
        ii) **Expected Result:** user information is changed to the new information
    b) **Case 2**
        i) **input:** invalid age
        ii) **Expected Result:** Exception was thrown
    c) **Case 3**
        i) **input:** invalid type of scooter
        ii) **Expected Result:** Exception was thrown
    d) **Case 4**
        i) **input:** invalid driver's license date
        ii) **Expected Result:** Exception was thrown
5) **Login -** Req 5
    a) **Case 1**
        i) **input:** valid credentials
        ii) **Expected Result:** user is logged in
    b) **Case 2**
        i) **input:** invalid email address
        ii) **Expected Result:** Exception was thrown
    c) **Case 3**
        i) **input:** invalid password
        ii) **Expected Result:** Exception was thrown
6) **Logout -** Req 6
    a) **Case 1**
        i) **input:** None (User is logged in)
        ii) **Expected Result:** user is logged out
    b) **Case 2**
        i) **input:** None (User is not logged in)
        ii) **Expected Result:** Exception was thrown

7) **Send Message To Admins** - Req 7
    a) **Case 1**
        i) **input:** valid question information
        ii) **Expected Result:** new question is created and admins are notified about it
    b) **Case 2**
        i) **input:** invalid title
        ii) **Expected Result:** Exception was thrown
    c) **Case 3**
        i) **input:** invalid message body
        ii) **Expected Result:** Exception was thrown

8) **Start Ride** - Req 8
    a) **Case 1**
        i) **input:** valid destination
        ii) **Expected Result:** a request to the External service is made and list of possible routes is returned to the rider
    b) **Case 2**
        i) **input:** invalid Destination address
        ii) **Expected Result:** Exception was thrown
9) **Show Driving Instructions** - Req 9
    a) This requirement will be manually tested by entering a valid destination and starting the ride and then we will be able to see if the map is displayed to user

10) **Detect Stationary Hazards** - Req 10
    a) **Case 1**
        i) **input:** A photo with no hazards
        ii) **Expected Result:** Raspberry pi didn't detect an hazard
    b) **Case 2**
        i) **input:** A photo with hazards
        ii) **Expected Result:** Raspberry pi detected the hazard correctly

11) **Detect Mobile hazards** - Req 11
    a) **Case 1**
        i) **input:** A photo without mobile hazards
        ii) **Expected Result:** Raspberry pi didn't detect an hazard
    b) **Case 2**
        i) **input:** A photo with hazards
        ii) **Expected Result:** Raspberry pi detected the hazard
12) **Detect Riding Surface** - Req 12
    a) **Case 1**
        i) **input:** A photo with a sidewalk surface
        ii) **Expected Result:** Raspberry pi detected it was a sidewalk
    b) **Case 2**
        i) **input:** A photo with a road surface
        ii) **Expected Result:** Raspberry pi detected it was a road
    c) **Case 3**
        i) **input:** A photo with an unknown surface
        ii) **Expected Result:** Raspberry pi didn't detect the surface

13) **Save information about hazards** - Req 13
    a) **Case 1**
        i) **input:** A new hazard
        ii) **Expected Result:** Information about a new hazard has been created and stored
    b) **Case 2**
        i) **input:** An existing hazard
        ii) **Expected Result:** Information about the hazard is updated if needed
14) **Fill Riding experience questionnaire** - Req 14
    a) **Case 1**
        i) **input:** Valid information
        ii) **Expected Result:** Information is stored and admins are notified about user experience
    b) **Case 2**
        i) **input:** Invalid description
        ii) **Expected Result:** Exception is thrown
15) **Save Ride information** - Req 15
    a) **Case 1**
        i) **input:** Invalid user id in the ride
        ii) **Expected Result:** Exception is thrown
    b) **Case 2**
        i) **input:** Invalid ride date
        ii) **Expected Result:** Exception is thrown
    c) **Case 3**
        i) **input:** Invalid ride date
        ii) **Expected Result:** Exception is thrown
16) **View Rides history** - Req 16
    a) **Case 1**
        i) **input:** User does not have rides history
        ii) **Expected Result:** An empty history is returned
    b) **Case 2**
        i) **input:** User has rides history
        ii) **Expected Result:** Entire history is returned
17) **User Rating based on ride data** - Req 17
    a) **Case 1**
        i) **input:** A ride where the user ignore hazards and didn't drive safely
        ii) **Expected Result:** User rating is lowered using the formulas provided in the ARD
    b) **Case 2**
        i) **input:** A ride where the user reacted to the hazards and drove safely
        ii) **Expected Result:** User rating is increased using the formulas provided in the ARD

18) **Delayed Notification** - Req 18
    a) This requirement will be manually tested by sending a message to a user (rider or admin) that is logged out, then logging in with this user credentials and verifying the notification was received when the user became logged in.
19) **View User Data** - Req 19
    a) **Case 1**
        i) **input:** a rider tried to view someone else's data
        ii) **Expected Result:** Exception is thrown
    b) **Case 2**
        i) **input:** admin is trying to view an existing user information
        ii) **Expected Result:** Information is returned to the admin
    c) **Case 3**
        i) **input:** admin is trying to view information of a user that doesn't exist
        ii) **Expected Result:** Exception is thrown
20) **Delete And Edit users** - Req 20
    a) **Case 1**
        i) **input:** a rider tried to edit someone else's data
        ii) **Expected Result:** Exception is thrown
    b) **Case 2**
        i) **input:** a rider tried to delete someone else's user
        ii) **Expected Result:** Exception is thrown
    c) **Case 3**
        i) **input:** admin is trying to delete a user that doesn't exist
        ii) **Expected Result:** Exception is thrown
    d) **Case 4**
        i) **input:** admin is trying to delete an existing user
        ii) **Expected Result:** the user account is deleted
    e) **Case 5**
        i) **input:** admin is trying to edit an existing user with valid information
        ii) **Expected Result:** the user account is updated
    f) **Case 6**
        i) **input:** admin is trying to edit an existing user with invalid information
        ii) **Expected Result:** Exception is thrown
21) **Add award to user** - Req 21
    a) **Case 1**
        i) **input:** admin is trying to add award to non existing user
        ii) **Expected Result:** Exception is thrown
    b) **Case 2**
        i) **input:** admin is trying to add award to an existing user
        ii) **Expected Result:** User receives a notification with information about the award

22) **Set system configuration** - Req 22
    a) **Case 1**
        i) **input:** admin is trying to edit a non existing configuration
        ii) **Expected Result:** Exception is thrown
    b) **Case 2**
        i) **input:** admin is trying to edit a configuration with invalid data
        ii) **Expected Result:** Exception is thrown
    c) **Case 3**
        i) **input:** user is trying to edit a configuration
        ii) **Expected Result:** Exception is thrown
    d) **Case 4**
        i) **input:** admin is trying to edit a configuration with valid data
        ii) **Expected Result:** configuration is updated and server is updated with new configuration
23) **Remove admin** - Req 23
    a) **Case 1**
        i) **input:** regular admin is trying to remove admin
        ii) **Expected Result:** Exception is thrown
    b) **Case 2**
        i) **input:** super admin is trying to remove a non existing admin
        ii) **Expected Result:** Exception is thrown
    c) **Case 3**
        i) **input:** super admin is trying to remove an existing admin
        ii) **Expected Result:** the admin is removed
    d) **Case 4**
        i) **input:** user is trying to remove an admin
        ii) **Expected Result:** Exception is thrown
24) **Add admin** - Req 23
    a) **Case 1**
        i) **input:** regular admin is trying to add admin
        ii) **Expected Result:** Exception is thrown
    b) **Case 2**
        i) **input:** super admin is trying to add an existing admin
        ii) **Expected Result:** Exception is thrown
    c) **Case 3**
        i) **input:** super admin is trying to add a non existing admin
        ii) **Expected Result:** the admin is added
    d) **Case 4**
        i) **input:** user is trying to add an admin
        ii) **Expected Result:** Exception is thrown

25) **Filter trips** - Req 24
    a) **Case 1**
        i) **input:** Admin entered fast travel threshold
        ii) **Expected Result:** only rides with maximum travel velocity above threshold is returned
    b) **Case 2**
        i) **input:** Admin entered change of speed threshold
        ii) **Expected Result:** only rides with amount of speed changes after alerts above threshold is returned
    c) **Case 3**
        i) **input:** Admin entered amount of breaking threshold
        ii) **Expected Result:** only rides with amount of breakings threshold is returned
26) **Detailed Explanations** - Req 25
    a) This requirement will be tested both in unit tests (i.e checking exception messages are informative), and manually tested by trying to perform some actions on the application and verifying we got information about the successes or failures.
27) **View Statistics** - Req 26
    a) **Case 1**
        i) **input:** Admin entered invalid dates
        ii) **Expected Result:** Exception is thrown
    b) **Case 2**
        i) **input:** Admin has entered valid dates
        ii) **Expected Result:** Statistics about the system between the dates provided are returned
28) **Save Information to External DB** - Req 27
    a) This will be done using both integration tests and manual testing.

        Integration tests:

        Perform multiple operations on the system that will cause the system to store information in the DB such as:

        1) Register user
        2) Send question to admins
        3) end a ride with some ride data
        4) add admin
        5) add hazard
        etc…

        afterwards we will try to fetch data from the DB and verify that the data was saved properly.

        Manual Testing:

        Perform some actions on the server, then send a shutdown signal which will cause the server to store all information to the DB and shutdown.
        afterwards initialize the server again and verify the information was retrieved from the DB properly.

Non Functional Requirements Test plan:

Requirement	How we are going to test
27	We will activate the object detection system 4 times (for each type of alert) on a video of a route containing a dangerous object (which has been tested as such). For each session, we will change the configuration settings so that the system creates a different type of alert. The expected result is that the type of alert that the system generates corresponds to the type of alert that it was supposed to generate
28	We will run the hazard detection system on all videos containing hazards and measure the distance between the hazard and the scooter rider (RP) while the alert is activated. A distance greater than the 'Minimum Distance To Alert' is an unwanted result, another result is a result we would like and expect to receive
29	We will run the hazard detection system on all the videos that contain hazards and measure the duration of time when the alert is activated, we would like this time to correspond to the value set by the admin in seconds
30	We will activate the 'fetch_safe_routes'function from the 'RoutesRetriever' object that will be activated during the 'start_ride' function from the Facade, we will expect that the 'fetch_safe_routes' function will return a list containing routes of size equal to 'Number Of Routes' value (Appendix A)
31	We will sort our tagged videos into 2 groups, one will contain videos that contain a large amount of hazards (to simulate peak time), the other group will contain videos that contain an amount of hazards that corresponds to normal activity time. The amount of danger required for each group is calculated based on previous data we have collected. We will run the object detection system on both groups, for the simulated peak time group we will expect a response time (from the moment the hazard is detected until the alert is activated) of 1 second. For the group simulating regular activity time,we will expect a response time of 0.1 seconds . Also, we will run the system on all the videos we have, calculate the average response time and expect a value of 0.5 second.
32	we will run the application on web, ios, android and expect that the application's color is green in all platforms
33	We will upload a new advertisement to the application, go to the advertisement page of the new advertisement we uploaded, and check if the details we entered, including the URL of the advertisement are correct. In addition, we will enter the advertisements page and count (manually) the amount of advertisements and we will expect the value to be the same as the

	amount of advertisements in the the Database
34	We will activate the start_ride function from a scooter containing RP and drive a certain way We will expect that the ride screen will show us the riding speed at any given moment.
35	We will activate the system and after an extended period of time (several days or weeks) during which many test operations will be performed except from turning off the system by the administrators, we will expect that the system will work properly
36	We will create initiated multiple failures in the system, and we will expect normal operation of the system (no crashing) and that the system will display a detailed message about the problem
37	We will run a loop 10000 times, in each iteration a new thread simulating a client will be created, the client will login to the system and activate the main function'strat_ride'. we will expect normal operation of the system, that is, without crashes and malfunctions
38	The registration process in our system will be done manually by a technician in order to link the RP to a specific user, so we will not check this requirement
39	We will manually check this requirement by looking at the DB in the 'USERS' table, we will observe that the 'PASSWORD' column will only have hash values, we will also make sure that when creating a new user the password is not saved at any stage in a variable but only its hash value
40	We will manually check this requirement by making sure that no buttons appear on the application client screens that enable admin functionality
41	We will initialize the system, after the initialization we will (automatically) compare the data included in the configuration file with the configuration file itself. A complete match of the data is expected
42	We will run the server on a Linux operating system. We will expect to valid operation of the system
43	We will run the administrator application on a Windows OS(version 10 or higher). We will expect to valid operation of the system and clear visibility of all the user interface elements
44	we will run the rider application on web, ios, android and expect that the application windows will fit the device on which they run in terms of display and position in relation to the margins, thus all the user interface elements are visible and clear
45	We will create initiated failures in the communication between the system components, and we will expect normal operation of the system (no crashing) and

	that the system will display a detailed message about the problem
46	We will create initiated failures in the external services (using a proxy) , and we will expect normal operation of the system (no crashing) and that the system will display a detailed message about the problem
47	We will activate the camera on a scooter in a route that contains certain objects that are less than 15 meters away with the shooting direction, we will expect to see all these objects in the video transmitted to the RP
48	We will activate several functions in the system, including functions with incorrect arguments that cause an error, after the activation we will observe at the error log and the event log and check whether all the actions we performed are mentioned in the event log and whether all the actions that generated an error are mentioned in the error log
49	We will login to the system as an admin and try to view the error log and event log, we will expect to be able to view them
50	we will run the application on web, ios, android and expect that the application’s language is English in all platforms