Bilkent University
Department of Computer Engineering

Senior Design Project

T2320

BilRide

Design Project Final Report

21702587, Turgut Alp Edis, alp.edis@ug.bilkent.edu.tr
21703556, İdil Yılmaz, idil.yilmaz@ug.bilkent.edu.tr
21602059, Dilay Yiğit, dilay.yigit@ug.bilkent.edu.tr
21702331, Doğukan Ertunga Kurnaz, ertunga.kurnaz@ug.bilkent.edu.tr
21801861, Funda Tan, funda.tan@ug.bilkent.edu.tr
Prof. Dr. İbrahim Körpeoğlu
Erhan Dolak and Tağmaç Topal

2023-05-18

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 CS491/2.

Contents

Detailed Design Report

Project Short-Name: BilRide

1 Introduction

Transportation has been a problem at Bilkent University since only some students have a private vehicle, and rings are not at bus stops on time. Therefore, many students suffer from this issue. Before the pandemic, gas prices were low so that students could choose to come with their cars, the latency of the rings was somehow acceptable, and even if carpooling was not expected, students could choose to carpool as a transportation option. However, during and after the pandemic, gas prices becomes high. Therefore, the students who choose their private car as a transportation option start to choose public transportation. Also, carpooling usage is at its lowest point due to health concerns, and the latency of the rings is at its highest level. New students may not know about carpooling and do not choose this option; they choose rings as a means of transportation. However, the schedule of the rings can delay up to one hour. Therefore, the hardship of the students who do not have car increases. Even though the municipality of Ankara provides EGO buses as a solution, they only come occasionally and become expensive for students. So, students only choose these buses unless there is another option. As a result, we propose a solution to fix the transportation problem. Our application is called BilRide. BilRide is a mobile application aiming to solve the need for more ring shuttles, high gas prices, the lack of hitchhiking culture problems, and increased carbon emissions.

This design project final report includes requirements details, final architecture and design details, consisting of subsystem decomposition, persistent data management and access control and security, global software control and boundary conditions, development and implementation details which consists of object design trade-offs, engineering standards, packages and class interfaces, test cases divided as functional and non-functional and their results, maintenance plan and details, other project elements and conclusion and future work.

2 Requirements Details

2.1 Overview

BilRide is a mobile app that aims to solve the issues caused by the lack of ring shuttles and hitchhiking culture at Bilkent University. Also, it tries to transform the transportation environment of Bilkent University by allowing individuals to socialize with each other via the application. Thus, BilRide can pioneer the change in transportation at Bilkent University.

There are some alternatives to our application, for example, BlaBlaCar [1]. Even though they are already in the market, they have some areas for improvement in functionality and usability. We focus on these points more.

Besides, we will bring a new option to the pickup choice of the passenger. BlaBlaCar offers only pickup points to the users. However, in our application, the passenger can also be picked up from the current location and the pickup points.

In addition, we want to bring to the market a sustaining innovation by adding unique features. These are checking if the user is a Bilkent University member to provide more safety, providing both carpooling options and ring bus ETA's for a complete solution to our users, and creating an environment for socializing with exchanging information about their ride, Spotify playlists, and gamification the experience with a badge system.

Thus, with the sustaining innovations, BilRide will be one of the appropriate traveling apps for the students and alumni of Bilkent University.

2.2 Functional Requirements

2.2.1 Functionalities of the Application

2.2.1.1 All Users Functionalities

- The application should provide users with the logging-in and/or signing-up options.

- The application should provide users with a map view, including some information, such as routes, and the current location of the cars, passengers, and shuttles.
- The application should provide a customizable user profile page for all users.
- The application should show the progress for each badge to each user.
- The application should show the earned badges in the user profiles.
- The application should show users how much gasoline they save after each ride and how much gasoline they save in total on the profile page.
- The application should allow all users to create a team with other users to create a common Spotify playlist and chat with each other before the ride.
- The application should allow all users to join open or private teams with an invitation.

2.2.1.2 Passenger Functionalities

- The application should show nearby routes of the drivers to the user from the map.
- The application should show the route, available quota, gender of the car's other passengers (if it is a problem for the user), whether the gasoline price is needed, the driver's rating, picture, and name for each nearby car to the user.
- The application should allow passengers to filter the routes according to their gender preferences, the gas price and the total number of passengers in the car, and the driver's rating.
- The application should allow passengers to choose the gender of the other passengers (if they need to) they wish to travel with.
- The application should allow passengers to send requests to nearby drivers with a selection of pickup points or their current locations.

- The application should allow passengers to rate the driver and give feedback about the journey after the ride.
- The application should show the user's current approximate location of the shuttle.
- The application should show the occupancy rate of the upcoming shuttle.

2.2.1.3 Driver Functionalities

- The application should show the current requests from the passengers to the driver with selected options, such as their pickup points, the gender they prefer to travel with, and whether they pay some portion of the gasoline price.
- The application should suggest the driver's route with the least gas consumption.
- The application should allow drivers to choose the number of people they want to travel with and the route, then inform the system about the choices.

2.2.2 Functionalities of the Server

- The system should keep information about all drivers, such as name, Bilkent ID number, rating, and routes informed to the app.
- The system should keep information about all passengers, such as name, Bilkent ID number, and stops where they want to join the ride.
- The system should keep information about the shuttle, such as the estimated location and occupancy rate.
- The system should keep the information about the teams, such as the member information.

- The system should keep the requests for joining the team.
- The system should keep the location information about each user in case of emergency, such as accidents.

2.3 Non-functional Requirements

In this section of the report, we will discuss the non-functional requirements of the project in detail. We choose the non-functional requirements in the scope of our application's domain.

2.3.1 Usability

Usability is an important non-functional requirement for our application because our application aims to reach everyone in the university. The application should be easy to use by design. All of the main functionalities of our application should be accessible by at most 3 touch gestures. The user interface will be designed to be friendly to people from all backgrounds, regardless of their understanding of technology. The application should not require any previous experience, and it should be easy to learn for novice users. The user should feel satisfied while using the application with intuitiveness.

2.3.2 Supportability

Supportability is another important non-functional requirement for our application because our application should be able to work as designed on different platforms. The application should be able to run on both Android (10.0 and above) and iOS (13.0 and above) operating systems. The application should be able to run smoothly with average hardware requirements. The application may require external

installation of Google App Services for Chinese versions of some Android Mobile Phones.

2.3.3 Maintainability

The application should be developed with future improvements and extensions in mind. Object Oriented Software Development techniques will be used during implementation. The application's codebase will be utilized with the version control tools like Git, which enables our team to work on multiple branches. That's how our team can continue development while the application is still usable. The deployment of our application will be automated with the help of GitHub Actions; when a new release happens, the action will automatically distribute the app package to the remote servers where our users can update their apps on the air. That is how the application will be easier to maintain.

2.3.4 Performance

The application should be loaded from a cold boot in less than thirty seconds. In-app pages should load in under three seconds. The application should send the user's information to the application's backend servers in less than 3 seconds.

2.3.5 Security

In our application, there will be a database in which we store all the user's personal data, such as; student id, full name, password, mail address, route information, etc. Sensitive ones, such as passwords, should be stored encrypted with a non-reversible hash function in the application database. Other sensitive information related to the user itself will be stored and processed according to the law requirements, such as KVKK and GDPR. The backend functionality of the application should be only accessible by the application itself. The application should

sanitize dangerous codes from the user-inputted fields and have a rate-limit functionality to prevent DoS attacks. The password should be at least eight characters in length. The application should automatically disable the user's account if the user tries to log in with the wrong password three times. The application should require re-login if the user has not opened the application for more than five days or the user changes the password. Since the application will not process transactions over it, there will be no further security checks other than the implementation guide of 3rd party payment provider. The application package will be compiled with some obfuscation techniques as best practices to prevent the decompilation of our application package.

2.3.6 Scalability

Since the application's implementation process was designed considering the scale-out approach, scalability would not be difficult. The only thing we may have to scale is the backend API, and the database in the future depends on the user count we will have. The application backend will not use microservice for simplicity, but we will try to Dockerize the API to enhance security and scalability.

2.4 Pseudo Requirements

- GitHub will be used throughout the project as a main Version Control platform.
- Dart programming language and Flutter UI SDK will be used in the project's frontend.
- Application should be able to run on both Android (+10.0) and iOS (+13.0) platforms.
- Python programming language and Flask will be used for the backend side of the project.

- To provide real-time communication (chatting), WebSockets will be used while implementing the chat feature.
- To provide location services and maps, Google Maps API will be used.

3 Final Architecture and Design Details

3.1 Overview

BilRide will mostly work through text and location inputs from the users. It will be a mobile app that works on Android and iOS operating systems. Though the design mainly focuses on iOS mobile phones, it is also compatible with Android since it is developed in Flutter. BilRide is a simple client-server application. It uses Dockerized Flask backend with a Postgresql database to store and manipulate the data, defined as the server, and users request the data from the server using the Flutter application, defined as clients.

3.2 Subsystem Decomposition

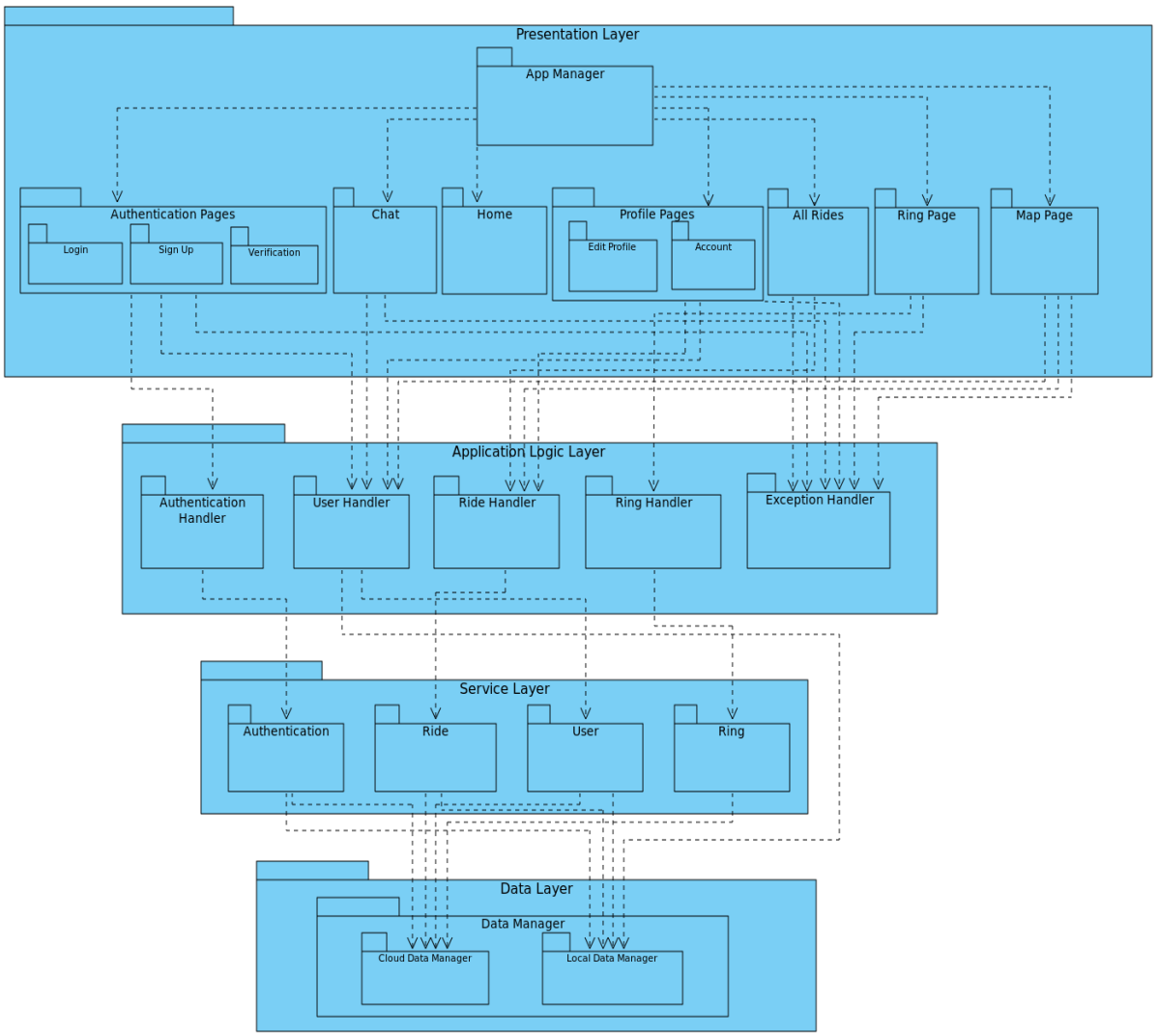

Figure 1: *Subsystem Decomposition Diagram*

3.3 Persistent Data Management

There are mainly four types of data in the project. The first type is the ride data. All necessary information about the ride, such as name, maximum capacity, start location, end location, driver, and passengers, is stored in the cloud database, and the data is seen by the user whenever the user is near the start location. The second type is user data. Necessary information about the user, such as name, surname, email, user type, and phone number, is stored in the cloud database, while the authorization token of the user is stored in the local device. When user-related action occurs in the application, such as logging in,

signing up, and editing a profile, the user data is taken from the cloud database and it is used by the application. The third type of data is ring data. Ring times, start and end locations are stored in the cloud database, and when the ring page is opened by the user, the ring service takes this data. The last type of data is chat data. All chat data is stored in the cloud server in encrypted format due to the security concern of the users. Also, this data will never be shared with any third party program because of the KVKK and GDPR. When the chat room is opened, this encrypted data is decrypted by the service and loaded to UI via the app manager.

3.4 Access Control and Security

We designed our application by using a security-by-design approach. Also, we enforced that the application uses a zero-trust policy in its critical infrastructures. In our application, there will be a database in which we store all the user's personal data, such as; student id, full name, password, mail address, route information, etc. Sensitive ones, such as passwords, should be stored encrypted with a non-reversible hash function in the application database. Other sensitive information related to the user itself will be stored and processed according to the law requirements, such as KVKK and GDPR [2]. The application's backend functionality should only be accessible by the application itself. We will be going to use all security-related flags in our HTTP requests. Our API endpoints are behind the WAF, and they are protected by the JWT authentication mechanism. The application should sanitize dangerous codes from the user-inputted fields and have a rate-limit functionality to prevent DoS attacks. The password should be at least eight characters in length. The application should automatically disable the user's account if the user tries to log in with the wrong password three times. The application should require re-login if the user has not opened the application for more than five days or the user changes the password. Since the application will not process transactions over it, there will be no further security checks other than the implementation guide of 3rd party payment provider. The application package will be compiled with some obfuscation techniques as best practices to prevent the decompilation of our application package.

3.5 Global Software Control

While the application is used by many users, the same data might travel at the same time to the users. Therefore, database and application can show some bugs and lags. To prevent these unexpected conditions, the server is event-driven so that server responses the client's requests quickly and with the accurate data. It is used for the client-server applications like our application, BilRide.

3.6 Boundary Conditions

- **Initialization**

  Since Flutter is used to develop the project, it can be downloaded from Google Play Store for Android and App Store for IOS devices. After downloading the application, it is opened with clicking the icon that represents the application.

  When it is opened first time, the landing page displays the brief information about the application. After the landing page, the user can move to the login page by clicking get started button. If the user signup previously, he/she can log in via filling information in login page. If he/she has not signed up yet, can register by clicking signup and navigates to the register page. After the login process is done, the user see the home page with request permission for the location pop up since it is the first time. After the permission is granted, the user can navigate to chat, rings and profile page. Also, the user can create or find pool according to his/her user type.

  The internet connection is required to use location services and API services, if the connection is not found, the connection not found page emerges with the option of retry connection for the user.

- **Termination**

  Very few data is kept on the device to give better performance. After the user logs out from the application, these data is deleted and he/she goes back to the login screen. If the user quits from the application by changing the screen and comes back later, the token for the application is refreshed automatically so that the user does not have to log in again.

- **Failure**

  The users of BilRide may counter problems in limited conditions. Only bad or lost internet connection and old phone may lead to some failures. The application still has some bugs that can result in failures but it is resolved when closing and opening the app. Also, if the server is down, the application is crashed since the users cannot use any of the functions inside the application.

4 Development/Implementation Details

4.1 Object Design Trade-offs

4.1.1 Usability vs Feature Complexity

Bilride is designed to be easy to use. Main features of the application are accessed by at most 3 touch gestures. Therefore, features need to have limited complexity at some point so that they remain easy to use. More complex features may improve the functionality but the application may become challenging for the users.

4.1.2 Supportability vs Performance

BilRide is aim to be supported by two large mobile platforms, Android and iOS, so we used Flutter to develop our application. However, it did not provide as good performance as the native development applications such as Android Studio and Swift.

4.1.3 Maintainability vs Flexibility

BilRide is designed to be easy to maintain with version control tool Git, deployment tool Github Actions and Object Oriented Software Development techniques. We use common components for each page in the app so that it is easier to maintain. However, customizing the application according to user preferences consists of limited options, such as dark mode. Thus, the application does not offer very customizable components and features.

4.1.4 Performance vs Consumption

Backend and frontend communication is designed to be established less than 3 seconds. However, it may vary according to the internet speed and the performance of the mobile phone. It is compatible with the lowest possible Android and iOS versions. However, since BilRide is designed to be give high performance, it may lead to be high battery, memory or storage consumptions for the mobile phones that has lower version of Android or iOS. Consumptions are optimized but we may need to optimize more in the future.

4.1.5 Security vs User Convenience

Since the application needs to be store some sensitive information such as location, mail address and passwords, it has to be secured. Verifications are implemented for the security of the app. However, these security steps may cause inconvenience users.

4.1.6 Scalability vs Cost

The application is designed to be scalable so that the users can use it 7/24. However, when the user count of the application increases, it may lead to additional costs for the cloud services, load balancers and infrastructure.

4.2 Engineering Standards

This report follows the Unified Modeling Language (UML) standards to visualize the design of the system and IEEE referencing are used in the report for all of the citations.

4.3 Packages

4.3.1 Pages

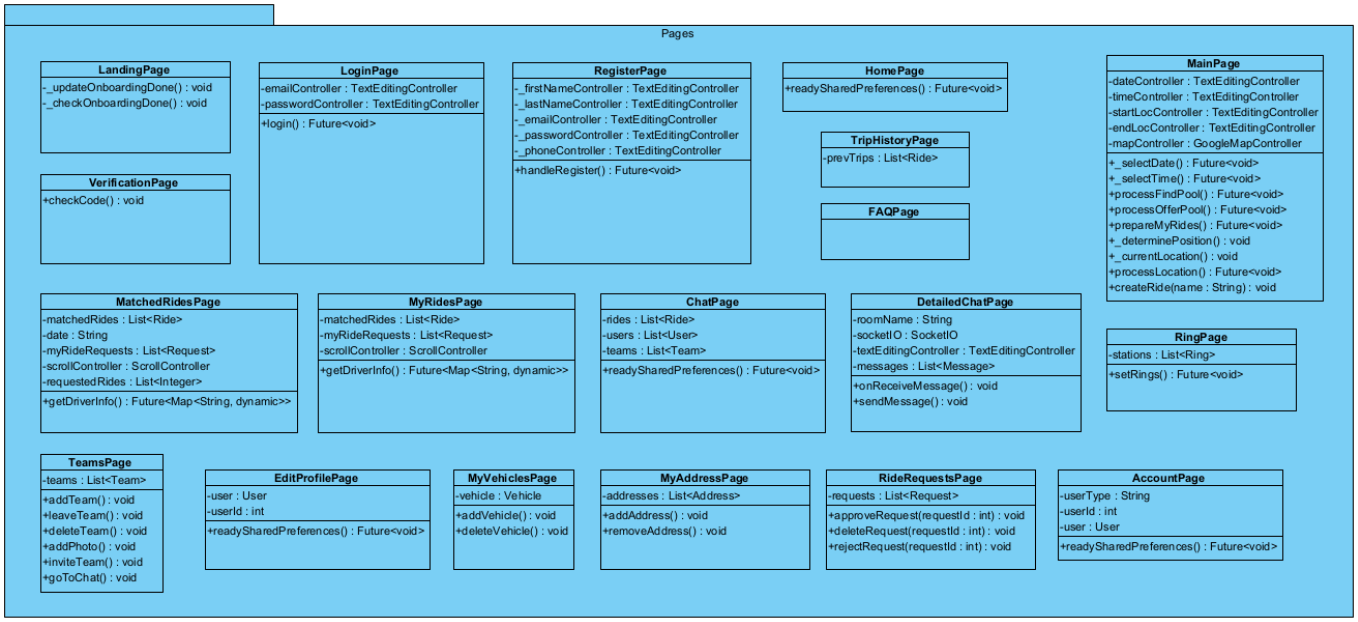

Figure 2: *Pages Package*

LandingPage: First page when the user opens the app first time. Displays landing Page and gives a brief introduction about the application.

LoginPage: Displays login page for the users. App comes back to this screen after the user logs out.

RegisterPage: Displays the register page. If the user does not have account, he/she can move to this page from login page.

VerificationPage: Displays the verification page that allows users to enter the verification code that is sent to their emails.

HomePage: Displays bottom navigation bar and list of pages for the users. It allows to navigate to main page, chat page, ring page and account page.

MainPage: Displays home page for the users. Allows users to find pool, see their rides and create pool.

TripHistoryPage: Displays the previous trips page for the users.

FAQPage: Displays the FAQ Page for the users.

MatchedRidesPage: Displays the map page with founded pools for passengers.

MyRidesPage: Displays the map page with the pools that the user will attend to or user has.

ChatPage: Displays the chat page with the users, teams and rides that the user can chat with.

DetailedChatPage: Displays the user chat page that can send and receive messages.

RingPage: Displays the ring page that allows users to see the upcoming rings to the given stop.

AccountPage: Displays the account page that the users can see their information and multiple options such as vehicles, addresses, previous trips, FAQ, Feedback, Logout

EditProfilePage: Displays the edit profile page that allows users to change their personal information.

MyVehiclesPage: Displays the vehicles page that the users can add, edit, delete or see their vehicles.

MyAddressPage: Displays the address page that the users can add, edit, delete or see their addresses.

RideRequestPage: Displays the ride request page for the rides of the drivers.

TeamsPage: Displays the team page that allows users to see and manage their teams.

4.3.2 Managers

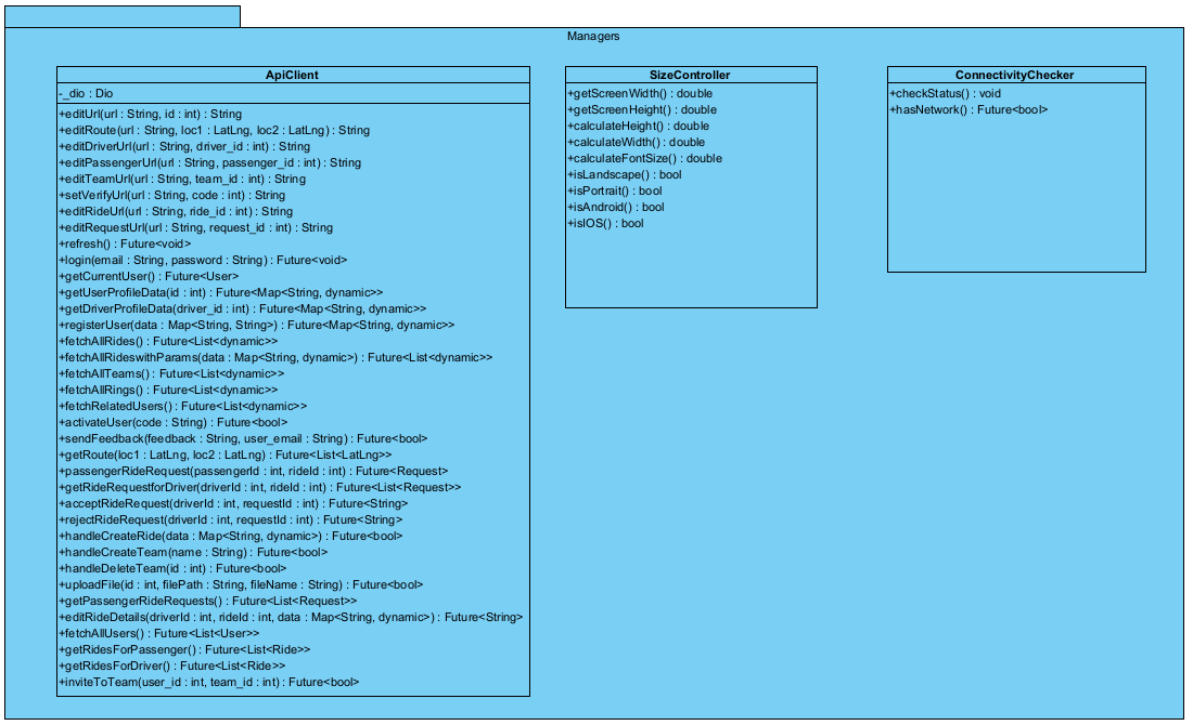

Figure 3: *Managers Package*

ApiClient: Manages the API requests inside the application and sets the responses according to the models.

SizeController: Calculates the size for the components in the pages to be compatible for each type of screen.

ConnectivityChecker: Checks the internet connection of the application.

4.3.3 Models

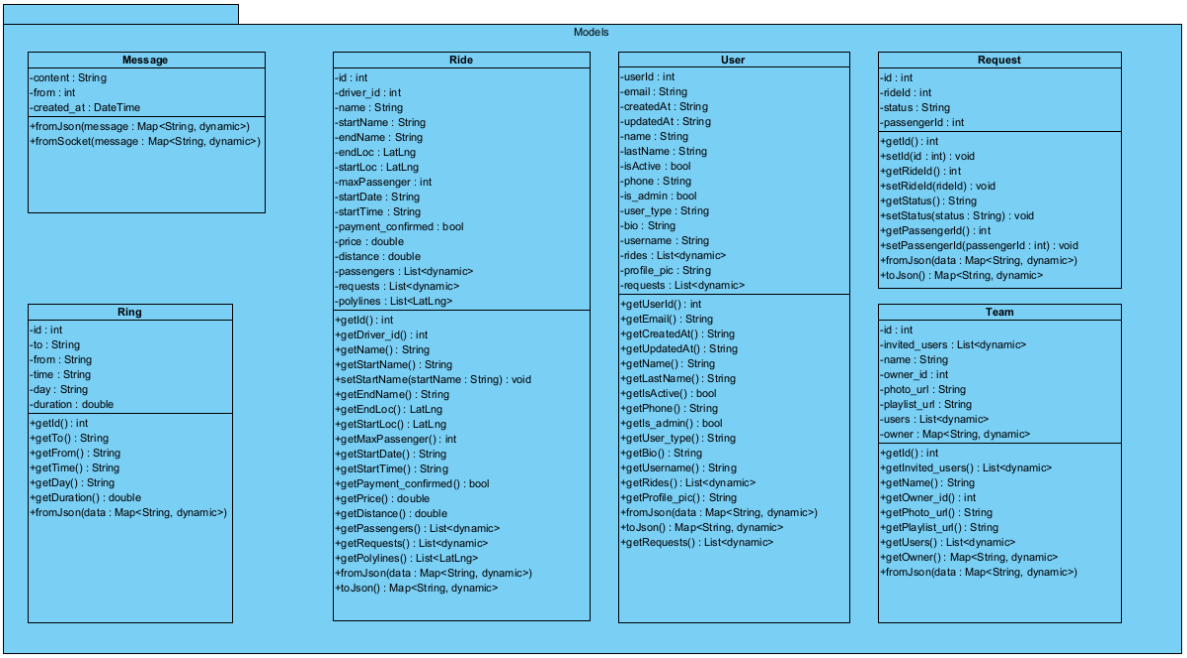

Figure 4: *Models Package*

Message: Contains information about the chat messages to build effectively as chat item.

Ride: Contains the ride information to manage ride responses better

User: Contains the user information to manage the users effectively.

RideRequest: Contains the request information to manage the response better.

Ring: Contains the ring information to manage the rings better.

Team: Contains the team information to easily manage the team.

4.4 Class Interfaces

LandingPage	First page when the user opens the app first time. Displays landing Page and gives a brief introduction about the application.	
Methods	_updateOnboardingDone() : void	Updates the onboarding completed status and navigates to LoginPage.
	_checkOnboardingDone() : void	Checks if the onboarding is completed. If it is completed previously, navigates to LoginPage.

VerificationPage	Displays the verification page that allows users to enter the verification code that is sent to their emails.	
Methods	checkCode() : void	Checks whether the given code is correct or not. If it is correct, it navigates to LoginPage.

LoginPage	Displays login page for the users. App comes back to this screen after the user logs out.	
Attributes	emailController : TextEditingController	Holds the email input from the user.
	passwordController : TextEditingController	Holds the password input from the user.

Methods	login() : Future<void>	Checks the credentials and allow user to log in to the application if the credentials are correct. Otherwise, displays error message.

RegisterPage	Displays the register page. If the user does not have account, he/she can move to this page from login page.	
Attributes	_emailController : TextEditingController	Holds the email input from the user.
	_passwordController : TextEditingController	Holds the password input from the user.
	_firstNameController : TextEditingController	Holds the name input from the user.
	_lastNameController : TextEditingController	Holds the last name input from the user.
	_phoneController : TextEditingController	Holds the phone input from the user.
Methods	handleRegister: Future<void>	Checks the credentials and allow user to register to the application if the credentials are proper. Otherwise, displays error message.

HomePage	Displays bottom navigation bar and list of pages for the users. It allows to navigate to main page, chat page, ring page and account page.	
Methods	readySharedPreferences: Future<void>	Gets the data from the storage to ease the navigation between pages.

TripHistoryPage	Displays the previous trips page for the users.	
Attributes	prevTrips : List<Ride>	The previous trips that user has done.

FAQPage	Displays the FAQ Page for the users.

MainPage	Displays home page for the users. Allows users to find pool, see their rides and create pool.	
Attributes	dateController : TextEditingController	Holds the date input from the user.
	timeController : TextEditingController	Holds the time input from the user.
	startLocController : TextEditingController	Holds the start location input from the user.
	endLocController : TextEditingController	Holds the end location input from the user.

	mapController : GoogleMapController	Holds the map controller for Google Maps.
Methods	_selectDate() : Future<void>	Sets the date for selected value.
	_selectTime() : Future<void>	Sets the time for selected value.
	processFindPool() : Future<void>	Gets the matched pool results for the passenger.
	processOfferPool() : Future<void>	Sets the values for creating ride for the driver.
	prepareMyRides() : Future<void>	Gets the rides for the driver and passenger. Also, get the pending requests for the passenger.
	_determinePosition() : void	Sets the camera location according to the current location of the user.
	_currentLocation() : void	Gets the current location of the user.
	processLocation() : Future<void>	Gets the new location if the user change his/her location.
	createRide(name: String): void	Creates the ride with given ride name and the parameters from the controllers.

MatchedRidesPage	Displays the map page with founded pools for passengers.	
Attributes	matchedRides: List<Ride>	Holds the ride results from the MainPage.
	date: String	Holds the date from the MainPage.

	myRideRequests: List<Request>	Holds the requests for the rides for the passenger.
	scrollController: ScrollController	Control variable for the scroll.
	requestedRides: List<int>	Holds the requested rides for the passenger to help to track.
Methods	getDriverInfo(): Future<Map<String,dynamic>>	Gets the driver info of the ride.

MyRidesPage	Displays the map page with the pools that the user will attend to or user has.	
Attributes	matchedRides : List<Ride>	Holds the rides of the driver from the MainPage.
	myRideRequests : List<Request>	Holds the requests for the rides of the driver.
	scrollController : ScrollController	Control variable for the scroll.
Methods	getDriverInfo() : Future<Map<String,dynamic>>	Gets the driver info of the ride.

ChatPage	Displays the chat page with the users, teams and rides that the user can chat with.

Attributes	rides : List<Ride>	Holds the rides for the ride chat.
	users : List<User>	Holds the users for the private chatr.
	teams : List<Team>	Holds the teams for the team chat.
Methods	readySharedPreferences() : Future<void>	Sets the lists for the chat items and DetailedChatPage.

DetailedChatPage	Displays the user chat page that can send and receive messages.	
Attributes	roomName: String	Holds the name of the room for chat.
	socketIO: SocketIO	Holds the SocketIO instance that allows users to chat
	textEditingController: TextEditingController	Holds the chat message
	messages: List<Message>	Holds the lists of messages to set the message history
Methods	onReceiveMessage(): void	Receives messages of the room.
	sendMessage(): void	Sends message to the room or other user.

RingPage	Displays the ring page that allows users to see the upcoming rings to the given stop.	
Attributes	stations : List<Ring>	Holds the stations for ring tracking and filtering.
Methods	setRings() : Future<void>	Sets the stations for ring tracking.

TeamsPage	Displays the team page that allows users to see and manage their teams.	
Attributes	teams : List<Team>	Holds the teams for the ride chat.
Methods	addTeam(): void	Creates a team for the user.
	leaveTeam(): void	Leaves from the team if the user is member of the team.
	deleteTeam(): void	Deletes the team if the user is the owner of the team.
	addPhoto(): void	Adds team photo for the team.
	inviteTeam(): void	Invites users to the team.
	goToChat(): void	Navigates to the team chat.

EditProfilePage	Displays the edit profile page that allows users to change their personal information.

Attributes	user: User	Holds the user object that will be edited
	userId: int	Holds the id of user.
Methods	readySharedPreferences() : Future<void>	Sets the userId and user object to display and edit.

MyVehiclesPage	Displays the vehicles page that the users can add, edit, delete or see their vehicles.	
Attributes	vehicle: Vehicle	Holds the vehicle object to display, add or delete.
Methods	addVehicle() : void	Adds the vehicle and driver can select the vehicle for the pool.
	deleteVehicle() : void	Deletes the vehicle from the driver.

MyAddressesPage	Displays the address page that the users can add, edit, delete or see their addresses.	
Attributes	addresses : List<Address>	Holds the addresses to display, add or delete.
Methods	addAddress() : void	Adds address and user can select the address as the start or location.
	deleteAddress() : void	Deletes the address from the user.

RideRequestsPage	Displays the ride request page for the rides of the drivers.	
Attributes	requests : List<Request>	Holds the requests to display, add or delete.
Methods	approveRequest(requestId : int) : void	Approves the request for the ride of the driver. Drivers can only approve their rides.
	deleteRequest(requestId: int): void	Deletes the request of the passenger. Passengers can only their requests.
	rejectRequest(requestId: int): void	Rejects the request for the ride of the driver. Drivers can only reject their rides.

AccountPage	Displays the account page that the users can see their information and multiple options such as vehicles, addresses, previous trips, FAQ, Feedback, Logout	
Attributes	userType : String	Holds the type of the user to create driver or passenger specific menu.
	userId : int	Holds the userId for authentication.
	user : User	Holds the user object to retrieve some general information.
Methods	readySharedPreferences() : Future<void>	Sets the attributes when the page is loading.

ApiClient	Manages the API requests inside the application and sets the responses according to the models.	
Attributes	_dio: Dio	Holds Dio instance to make API requests.
Methods	editUrl(url : String, id : int) : String	Edits the url for user requests.
	editRoute(url : String, loc1 : LatLng, loc2 : LatLng) : String	Edits the url for route requests.
	editDriverUrl(url : String, driver_id : int) : String	Edits the url for driver requests.
	editPassengerUrl(url : String, passenger_id : int) : String	Edits the url for passenger requests.
	editTeamUrl(url : String, team_id : int) : String	Edits the url for team requests.
	setVerifyUrl(url : String, code : int) : String	Edits the url for verification requests.
	editRideUrl(url : String, ride_id : int) : String	Edits the url for ride requests.
	editRequestUrl(url : String, request_id : int) : String	Edits the url for ride request requests.
	refresh() : Future<void>	Refreshes the auth token inside the application.
	login(email : String, password : String) : Future<void>	Allows user to log into the application.
	getCurrentUser() : Future<User>	Gets the current user profile and stored locally.
	getUserProfileData(id : int) : Future<Map<String, dynamic>>	Gets the user data from the id parameter as Map Property.

	getDriverProfileData(driver_id : int) : Future<Map<String, dynamic>>	Gets the profile data of the driver with given driver_id
	registerUser(data : Map<String, String>) : Future<Map<String, dynamic>>	Registers the user for the application with given data.
	fetchAllRides() : Future<List<dynamic>>	Gets all rides from the database.
	fetchAllRideswithParams(data : Map<String, dynamic>) : Future<List<dynamic>>	Gets the filtered rides from the database.
	fetchAllTeams() : Future<List<dynamic>>	Gets all teams of the user from the database.
	fetchAllRings() : Future<List<dynamic>>	Gets all rings from the database.
	fetchRelatedUsers() : Future<List<dynamic>>	Gets the related users, teams and rides of the user.
	activateUser(code : String) : Future<bool>	Activates the user.
	sendFeedback(feedback : String, user_email : String) : Future<bool>	Sends feedback with feedback and contact information to developer team.
	getRoute(loc1 : LatLng, loc2 : LatLng) : Future<List<LatLng>>	Gets the polyline data between two locations.
	passengerRideRequest(passengerId : int, rideId : int) : Future<Request>	Gets the request of passenger for the ride.
	getRideRequestforDriver(driverId : int, rideId : int) : Future<List<Request>>	Gets the list of requests for the ride of the driver.
	acceptRideRequest(driverId : int, requestId : int) :	Accepts the request for the ride.

	Future<String>	
	rejectRideRequest(driverId : int, requestId : int) : Future<String>	Rejects the request for the ride.
	handleCreateRide(data : Map<String, dynamic>) : Future<bool>	Creates a ride with given data for the driver.
	handleCreateTeam(name : String) : Future<bool>	Creates a team with a given name for the user.
	handleDeleteTeam(id : int) : Future<bool>	Deletes the team that has the id of given id.
	uploadFile(id : int, filePath : String, fileName : String) : Future<bool>	Uploads a picture for the team.
	getPassengerRideRequests() : Future<List<Request>>	Gets the list of requests of the passenger.
	editRideDetails(driverId : int, rideId : int, data : Map<String, dynamic>) : Future<String>	Edits the ride parameters for given ride id.
	fetchAllUsers() : Future<List<User>>	Gets all users from the database.
	getRidesForPassenger() : Future<List<Ride>>	Gets all rides of the passenger.
	getRidesForDriver() : Future<List<Ride>>	Gets all rides of the driver.
	inviteToTeam(user_id : int, team_id : int) : Future<bool>	Sends team invitation to a user.

SizeController	Calculates the size for the components in the pages to be compatible for each type of screen.

Methods	getScreenWidth() : double	Gets the screen width of the device.
	getScreenHeight() : double	Gets the screen height of the device.
	calculateHeight() : double	Calculates the screen height with some ratio.
	calculateWidth() : double	Calculates the screen width with some ratio.
	calculateFontSize() : double	Calculates the font size with some ratio.
	isLandscape() : bool	Returns true if the screen is landscape.
	isPortrait() : bool	Returns true if the screen is portrait.
	isAndroid() : bool	Returns true if the device is Android.
	isIOS() : bool	Returns true if the device is iOS.

ConnectivityChecker	Checks the internet connection of the application.	
Methods	checkStatus() : void	Check the internet connection status.
	hasNetwork() : Future<bool>	Indicates whether there is an internet connection or not.

Message	Contains information about the chat messages to build effectively as chat item.	
Attributes	content : String	Holds the content of the message.
	from : int	Holds the user_id that is indicated where the message comes.
	created_at : DateTime	Holds the creation date of the message
Methods	fromJson(message : Map<String, dynamic>)	Creates message object from json data.
	fromSocket(message : Map<String, dynamic>)	Creates message object from socket data.

Ride	Contains the ride information to manage ride responses better.	
Attributes	id : int	Holds the id of the ride.
	driver_id : int	Holds the driver_id of the ride.
	name : String	Holds the name of the ride.
	startName : String	Holds the start name of the ride.
	endName : String	Holds the end name of the ride.
	endLoc : LatLng	Holds the end location of the ride.

	startLoc : LatLng	Holds the start location of the ride.
	maxPassenger : int	Holds the max passenger of the ride.
	startDate : String	Holds the start date of the ride.
	startTime : String	Holds the start time of the ride.
	payment_confirmed : bool	Holds the payment status of the ride.
	price : double	Holds the price of the ride.
	distance : double	Holds the distance of the ride.
	passengers : List<dynamic>	Holds the passengers of the ride.
	requests : List<dynamic>	Holds the requests of the ride.
	polylines : List<LatLng>	Holds the directions of the ride.
Methods	getId() : int	Gets the id.
	getDriver_id() : int	Gets the driver id.
	getName() : String	Gets the name.
	getStartName() : String	Gets the start name.
	setStartName(startName: String) : void	Sets the start name to new value.

	getEndName () : String	Gets the end name.
	getEndLoc() : LatLng	Gets the end location.
	getStartLoc() : LatLng	Gets the start location.
	getMaxPassenger(): int	Gets the maximum passenger.
	getStartDate() : String	Gets the start date.
	getStartTime() : String	Gets the start time.
	getPayment_confirmed() : bool	Gets the payment status.
	getPrice(): double	Gets the price.
	getDistance() : double	Gets the distance.
	getPassengers() : List<dynamic>	Gets the passengers.
	getRequests() : List<dynamic>	Gets the requests.
	getPolylines() : List<LatLng>	Gets the directions.
	fromJson(data : Map <String, dynamic>)	Creates a ride object from json data.
	toJson() : Map<String, dynamic>	Converts the ride object to json data.

User	Contains the user information to manage the users effectively.	
Attributes	userId: int	Holds the id of the user.
	email : String	Holds the email of the user.
	created_at : String	Holds the creation date of the user.
	updatedAt: String	Holds the last update date of the user.
	name : String	Holds the name of the user.
	lastName : String	Holds the last name of the user.
	isActive : bool	Holds the activity status of the user.
	phone : String	Holds the phone number of the user.
	is_admin : bool	Holds the admin status of the user.
	user_type : String	Holds the user type of the user.
	bio : String	Holds the short description of the user.
	username : String	Holds the username of the user.
	rides : List<dynamic>	Holds the rides of the user.
	requests: List<dynamic>	Holds the requests of the passenger.
	profile_pic : String	Holds the profile picture url of the user.

Methods	getUserid(): int	Gets id of the user.
	getEmail() : String	Gets email of the user.
	getCreatedAt() : String	Gets creation date of the user.
	getUpdatedAt(): String	Gets last update date of the user.
	getName() : String	Gets name of the user.
	getLastName() : String	Gets last name of the user.
	getIsActive() : bool	Gets activity status of the user.
	getPhone() : String	Gets phone number of the user.
	getIs_admin() : bool	Gets admin status of the user.
	getUser_type() : String	Gets the user type of the user.
	getBio(): String	Gets the bio of the user.
	getUsername() : String	Gets the username of the user.
	getRides() : List<dynamic>	Gets the rides of the user.
	getProfile_pic() : String	Gets the profile picture url of the user.
	getRequests(): List<dynamic>	Gets the requests of the passenger.

	fromJson(data : Map <String, dynamic>)	Creates a user object from json data.
	toJson(): Map <String, dynamic>	Converts user object to json data.

Request	Contains the request information to manage the response better.	
Attributes	id : int	Holds the id of the request.
	rideId : int	Holds the ride id of the request.
	status : String	Holds the status of the request.
	passengerId : int	Holds the passenger id of the request.
Methods	getId() : int	Gets the id of the request.
	setId(id : int) : void	Sets the id of the request.
	getRideId() : int	Gets the ride id of the request.
	setRideId (rideId) : void	Sets the ride id of the request.
	getStatus() : String	Gets the status of the request.
	setStatus(status : String) : void	Sets the status of the request.
	getPassengerId(): int	Gets the passenger id of the request.

	setPassengerId(passengerId: int) : void	Sets the passenger id of the request.
	fromJson(data : Map <String, dynamic>)	Creates a request object from json data.
	toJson() : Map<String, dynamic>	Converts a request object to json data.

Ring	Contains the ring information to manage the rings better.	
Attributes	id : int	Holds the id of the ring.
	to : String	Holds the arrival place name of the ring.
	from : String	Holds the departure place name of the ring.
	time : String	Holds the departure time of the ring.
	day : String	Holds the departure day of the ring.
	duration : double	Holds the duration of the ring.
Methods	getId() : int	Gets the id of the ring.
	getTo() : String	Gets the arrival place name of the ring.
	getFrom() : String	Gets the departure place name of the ring.
	getTime() : String	Gets the departure time of the ring.

	getDay() : String	Gets the departure day of the ring.
	getDuration() : double	Gets the duration of the ring.
	fromJson (data : Map <String, dynamic>)	Creates a ring object from json data.

Team	Contains the team information to easily manage the team.	
Attributes	id : int	Holds the id of the team.
	invited _users: List<dynamic>	Holds the invited user list of the team.
	name : String	Holds the name of the team.
	owner_id : int	Holds the owner id of the team.
	photo_url : String	Holds the photo url of the team.
	playlist_url : String	Holds the playlist url of the team.
	users: List<dynamic>	Holds the member user list of the team.
	owner : Map <String, dynamic>	Holds the owner information of the team.
Methods	getId() : int	Gets the id of the team.

	getInvited_users() : List<dynamic>	Gets the invited users list of the team.
	getName() : String	Gets the name of the team.
	getOwner_id() : int	Gets the owner id of the team.
	getPhoto_url() : String	Gets the photo url of the team.
	getPlaylist_url() : String	Gets the playlist url of the team.
	getUsers() : List<dynamic>	Gets the user list of the team.
	getOwner() : Map<String, dynamic>	Gets the owner information of the team.
	fromJson(data : Map <String, dynamic>)	Creates a team object from json data.

5 Test Cases and Results

5.1 Functional Testing

Test ID: BR01

Test Type: Functional

Summary: Verify that the user can log in the application successfully

Procedure:

- Launch BilRide.
- Check if the text fields on the login page are editable, visible, and selectable.
- Check if the login button is clickable and visible.
- Check if the user sees the error box after filling in text fields with incorrect credentials and clicking the login button.
- Check if the user can see the homepage after filling in the text fields with the correct credentials and clicking the login button.

Expected Results:

- User can see the homepage after filling in text fields with the correct credentials and clicking the login button.
- User can see the error box after filling text fields with incorrect credentials.

Severity: Critical
Date Tested: 19/05/23
Result: Pass 2, Fail 0

Test ID: BR02

Test Type: Functional

Summary: Verify that a new user can successfully sign up for the application

Procedure:

- Launch BilRide.
- Click on the "Sign Up" button on the homepage
- Fill in all the required fields for registration, including name, email, password, and phone number
- Click on the "Register" button

Expected Results:

- The user is registered successfully, navigated to the verification page.

Severity: Critical
Date Tested: 19/05/23
Result: Pass 1, Fail 0

Test ID: BR03

Test Type: Functional

Summary: Verify the email verification mechanism successfully works

Procedure:

- Check if the text fields in the verification page are editable, visible, and selectable.
- Check if the continue button is clickable and visible.
- Check if the user sees the error message after filling the text field with the incorrect code and clicking the continue button.
- Check if the user can see the approval message and the login after filling the text field with the correct code and clicking the continue button.

Expected Results:

- User can see the approval message and the login page after filling the text field with the correct code and clicking the continue button.
- User can see the error message after filling the text field with incorrect code.

Severity: Major
Date Tested: 19/05/23
Result: Pass 2, Fail 0

Test ID: BR04

Test Type: Functional

Summary: Verify “Forgot Password” page elements’ visibility, editability, selectability, clickability, error, and approval messages.

Procedure:

- Check if the text fields in the forgot password page are editable, visible, and selectable.
- Check if the continue button is clickable and visible.
- Check if the user sees the error message after filling all text fields incorrectly and clicking the continue button.
- Check if the user sees the error message after filling the email code text field with incorrect code and other text fields correctly and clicking the continue button.
- Check if the user sees the error message after not filling retype password field same as the new password field and filling in the email code correctly, and clicking the continue button.
- Check if the user sees the error message after filling new password not the same as retype password field and filling in the email code correctly, and clicking the continue button.
- Check if the user can see the approval message and the login after filling text fields correctly and clicking the continue button.

Expected Results:

- User can see the approval message and the login page after filling in text fields correctly and clicking the continue button.
- User can see the error message after filling the email text field with incorrect code and other text fields correctly and clicking the continue button.
- User can see the error message after filling new password text field not the same as retype password text field and filling in the email code correctly and clicking the continue button.
- User can see the error message after filling retype password text field not the same as new password text field and filling email code correctly and clicking the continue button.
- User can see the error message after filling all text fields incorrectly and clicking continue button.

Severity: Major
Date Tested: 19/05/2023
Result: Pass 5, Fail 0

Test ID: BR05

Test Type: Functional

Summary: Verify that the user can be redirected to the main page and validate that all the required elements for the main functionality are visible.

Procedure:

- Check if the user logs in to the application successfully.
- Verify that the user is redirected to the main page.
- Validate that the main page contains the following elements:
- A map displaying the user's current location and the location of nearby rides.
- "Find Pool" button to search for available rides.
- "Offer Pool" button to create a new ride.
- "All Rides" button to view all rides in the system.
- “Current Location” button to mark the user’s current location to map

Expected Results:

- The system successfully redirected the user to the main page.
- All elements on the main page should be visible and correctly displayed.
- The map is responsive and can map the user's current location.
- Validation messages should be displayed if there are any errors in loading the page.

Severity: Major
Date Tested: 19/05/23
Result: Pass 4, Fail 0

Test ID: BR06

Test Type: Functional

Summary: Verify that the user can successfully offer a pool

Procedure:

- Launch BilRide.
- Log in to the system using valid credentials
- Click on the “Offer Pool” option from the slider menu
- Fill in all the start location, drop location, date and time, and maximum seat numbers
- Click on the “Offer Pool” button

Expected Results:

- A new pool is created successfully and displayed on the user's dashboard of “My Rides”.

Severity: Major

Date Tested: 19/05/23
Result: Pass 1, Fail 0

Test ID: BR07

Test Type: Functional

Summary: Verify that the user can successfully browse a pool and view results.

Procedure:

- Launch BilRide.
- Log in to the system using valid credentials
- Click on the “Find Pool” option from the slider menu
- Fill in all the start location, drop location, date and time, and maximum seat numbers
- Click on the “Find Pool” button

Expected Results:

- Results of the query is executed and displayed on user’s dashboard. The app should display a list of query results with all the necessary details about the pools.

Severity: Major
Date Tested: 19/05/23
Result: Pass 1, Fail 0

Test ID: BR08

Test Type: Functional

Summary: Verify that users’ rides are displayed successfully to the user.

Procedure:

- Launch BilRide.
- Log in to the system using valid credentials
- Clicking the “My Rides” button from the main page.

Expected Results:

- The user should be able to log in to the application successfully.
- The "See My Rides" page should be displayed.
- All available rides should be displayed on the page. Each ride should display starting point, destination, date and time, and the number of available seats.
- Rides should be listed chronologically, with the most recent ride appearing first.

- Users should be able to filter rides by start and end locations, date and time, and the number of available seats.
- Users should be able to click on a ride to view more details, including driver information and driver rating.

Severity: Major
Date Tested: 19/05/23
Result: Pass 5, Fail 1
Notes: Filtering is not implemented. Expected result 5 failed.

Test ID: BR09

Test Type: Functional

Summary: Verify that the user can join an existing pool.

Procedure:

- Launch BilRide.
- Log in to the system using valid credentials
- Search for an existing pool using the “Find Pool” option or browse through available rides by clicking the “All Rides” button.
- Click on a ride from the list
- Click on the "Join" button for the desired route.
- Click on the "Join Ride" button

Expected Results:

- User successfully requests the selected ride, and a confirmation message is displayed on the screen.

Severity: Major
Date Tested: 19/05/23
Result: Pass 6, Fail 0

Test ID: BR10

Test Type: Functional

Summary: Verify that the user can approve a pool join request.

Procedure:

- Launch BilRide.
- Log in to the system using valid credentials
- Go to “Profile Page” from the bottom menu
- Click “My Rides”
- Click “Incoming Requests”
- Approve a request by clicking the “Approve” button for request.

Expected Results:

- The request status changes from "Pending" to "Approved"
- The user who requested the ride is notified of the approval

Severity: Major
Date Tested: 19/05/23
Result: Pass 1, Fail 1
Notes: The user who requested the ride is not notified of the approval.

Test ID: BR11

Test Type: Functional

Summary: Verify that the user can decline a pool join request.

Procedure:

- Launch BilRide.
- Log in to the system using valid credentials
- Go to “Profile Page” from the bottom menu
- Click “My Rides”
- Click “Incoming Requests”
- Declined a request by clicking the “Decline” button for a request.

Expected Results:

- The request status changes from "Pending" to "Declined"
- The user who requested the ride is notified of the decline

Severity: Major
Date Tested: 19/05/23
Result: Pass 1, Fail 1
Notes: The user who requested the ride is not notified of the rejection.

Test ID: BR12

Test Type: Functional

Summary: Verify that the user can cancel a pool join request.

Procedure:

- Launch BilRide.
- Log in to the system using valid credentials
- Go to “Profile Page” from the bottom menu
- Click “My Rides”
- Click “Pending Requests”
- Cancel a request by clicking the “Cancel” button.

Expected Results:

- The request status changes from "Pending" to "Canceled"
- The user to whom the request was sent should be notified of the cancellation.

Severity: Major
Date Tested: 19/05/23
Result: Pass 1, Fail 1
Notes: The user to whom the request was sent is not notified of the rejection.

Test ID: BR13

Test Type: Functional

Summary: Verify that the user can edit his/her rides.

Procedure:

- Launch BilRide.
- Log in to the system using valid credentials
- Go to “Profile Page” from the bottom menu
- Click “My Rides”
- Click on the “Edit Ride” button
- Update the details of the pool
- Click on the "Save Changes" button.
- Verify that the pool details have been updated.

Expected Results:

- The user should be able to edit ride details successfully and the updated information should be displayed correctly.

Severity: Minor
Date Tested: 19/05/2023
Result: Pass 1, Fail 0

Test ID: BR14

Test Type: Functional

Summary: Verify that the user can delete his/her rides.

Procedure:

- Launch BilRide.
- Log in to the system using valid credentials
- Go to “Profile Page” from the bottom menu
- Click “My Rides”
- Select the route to be canceled from the list
- Click on the "Cancel" button
- Confirm the cancellation by providing a reason for cancellation (if required)

Expected Results:

- The selected route is canceled successfully and removed from the user's “My Rides” list.
- Any other users who had booked seats on the route are notified about the cancellation.

Severity: Major
Date Tested: 19/05/23
Result: Pass 1, Fail 1
Notes: Users are not notified when the cancellation happens.

Test ID: BR15

Test Type: Functional

Summary: Verify that the user can cancel his/her rides.

Procedure:

- Launch BilRide.
- Log in to the system using valid credentials
- Go to “Profile Page” from the bottom menu
- Click “My Rides”
- Select the route to be canceled from the list
- Click on the "Cancel" button
- Confirm the cancellation by providing a reason for cancellation (if required)

Expected Results:

- The selected route is canceled successfully and removed from the user's “My Rides” list.
- Driver who owns the ride is notified about the cancellation.

Severity: Major
Date Tested: 19/05/23
Result: Pass 1, Fail 1
Notes: Driver who owns the ride is not notified about the cancellation.

Test ID: BR16

Test Type: Functional

Summary: Verify if the system displays the ring timetable and estimated location accurately.

Procedure:

- Launch BilRide.
- Log in to the system using valid credentials
- Click on the “Ring” button from the bottom menu

- Verify that the timetable displayed shows the correct departure and arrival times of the ring for each stop.
- Click on a specific stop on the timetable to view the estimated arrival time at that stop.
- Verify that the estimated time of arrival is displayed accurately.

Expected Results:

- The system should display the ring timetable accurately.
- The timetable should show the correct departure and arrival times for each stop.
- The estimated time of arrival at each stop should be displayed accurately.

Severity: Critical
Date Tested: 19/05/23
Result: Pass 2, Fail 1
Notes: The estimated time of arrival is not seen for each stop.

Test ID: BR17

Test Type: Functional

Summary: Verify that the user can view their profile page after logging in.

Procedure:

- Launch BilRide.
- Log in to the system using valid credentials
- Go to “Profile Page” from the bottom menu
- Verify that the user is redirected to their profile page.

Expected Results:

- The user should be able to view their profile page without any errors.

Severity: Major
Date Tested: 19/05/23
Result: Pass 1, Fail 0

Test ID: BR18

Test Type: Functional

Summary: Verify that users can edit their profile information.

Procedure:

- Launch BilRide.
- Log in to the system using valid credentials
- Click on the “Profile” button from the bottom menu

- Click on the user name.
- Edit the user’s profile information.
- Click on the "Save" button.
- Verify that the changes are saved successfully.

Expected Results:

- The user's profile information should be updated without any errors.

Severity: Minor
Date Tested: 19/05/23
Result: Pass 0, Fail 1
Notes: Profile information is not editable.

Test ID: BR19

Test Type: Functional

Summary: Verify that the user can edit their vehicle information.

Procedure:

- Launch BilRide.
- Login with valid credentials.
- Click on the “Profile” button from the bottom menu
- Click on the "My Vehicles" button.
- Edit the user's vehicle information.
- Click on the "Save Changes" button.
- Verify that the changes are saved successfully.

Expected Results:

- The user's vehicle information should be updated without any errors.

Severity: Minor
Date Tested: 19/05/23
Result: Pass 0, Fail 1
Notes: Vehicle information is not editable.

Test ID: BR20

Test Type: Functional

Summary: Verify that users can edit their saved addresses.

Procedure:

- Launch BilRide.
- Login with valid credentials.

- Click on the “Profile” button from the bottom menu
- Click on the “Manage Address” button from the list menu.
- Click on the "Edit Saved Addresses" button.
- Edit the user's saved addresses.
- Click on the "Save" button.
- Verify that the changes are saved successfully.

Expected Results:

- The user's saved addresses should be updated without any errors.

Severity: Minor
Date Tested: 19/05/23
Result: Pass 0, Fail 1
Notes: Saved address information is not editable.

Test ID: BR21

Test Type: Functional

Summary: Verify that the user can see their trip history.

Procedure:

- Launch BilRide.
- Login with valid credentials.
- Click on the “Profile” button from the bottom menu
- Click on the "Previous Trips" button from the list menu.
- Verify that the user can see their trip history.

Expected Results:

- The user should be able to see their trip history without any errors.

Severity: Minor
Date Tested: 19/05/23
Result: Pass 0, Fail 1
Notes: Previous Trips button is not clickable.

Test ID: BR22

Test Type: Functional

Summary: Verify if the user can edit teams.

Procedure:

- Launch BilRide.
- Login with valid credentials.
- Click on the “Profile” button from the bottom menu
- Click on the “My Teams” option from the list menu.
- Update the team information.
- Save the updated information.
- Verify that the information is updated and displayed correctly.

Expected Results:

- The user should be able to edit teams successfully.
- The updated information should be displayed correctly.

Severity: Minor
Date Tested: 19/05/23
Result: Pass 2, Fail 0

Test ID: BR23

Test Type: Functional

Summary: Verify if the user can create teams

Procedure:

- Launch BilRide.
- Login with valid credentials.
- Click on the “Profile” button from the bottom menu
- Click on the “My Teams” option from the list menu.
- Click on the “Create Team” button.
- Write team information.
- Select users to add to the team.
- Click on the “Create” button.
- Verify the team is created successfully, and the user is directed to the team dashboard

Expected Results:

- The user should be able to edit teams successfully and the updated information should be displayed correctly.

Severity: Minor
Date Tested: 19/05/23
Result: Pass 1, Fail 0

Test ID: BR24

Test Type: Functional

Summary: Verify if the owner sends an invitation to user for joining a team

Procedure:

- Launch BilRide.
- Login with valid credentials.
- Click on the “Profile” button from the bottom menu
- Click on the “My Teams” option from the list menu.
- Click on the “Invite” button which is indicated as “+” sign at the top right corner.
- Browse from the users list
- Click “Invite” to the user from the users list.
- Verify the invitation is created and sent successfully and the user is notified.

Expected Results:

- Invitation is created and sent successfully.
- The user is notified.

Severity: Minor
Date Tested: 19/05/23
Result: Pass 1, Fail 1
Notes: The user is not notified when the invitation is created and sent.

Test ID: BR25

Test Type: Functional

Summary: Verify if the user can leave a team successfully.

Procedure:

- Launch BilRide.
- Login with valid credentials.
- Click on the “Profile” button from the bottom menu
- Click on the “My Teams” option from the list menu.
- Browse from the teams list
- Click "Yes" to confirm leaving the team.
- Verify that the user is no longer a member of the team.

Expected Results:

- The user can leave the team successfully.
- The user should not have any access to the team after leaving.

Severity: Minor
Date Tested: 19/05/23
Result: Pass 2, Fail 0

Test ID: BR26

Test Type: Functional

Summary: Verify if the user can delete a team successfully.

Procedure:

- Launch BilRide.
- Login with valid credentials.
- Click on the “Profile” button from the bottom menu
- Click on the “My Teams” option from the list menu.
- Verify that the “Delete” option is only displayed for the teams where the user is in the owner position.
- Verify that the team is deleted and all of the users are removed from the team.

Expected Results:

- The team is successfully deleted from the system.
- Team no longer appears in any “Teams” list.
- All team members are notified that the team has been deleted.

Severity: Minor
Date Tested: 19/05/23
Result: Pass 2, Fail 1
Notes: Team members are not notified that the team has been deleted. Expected result 3 failed.

Test ID: BR27

Test Type: Functional

Summary: Test if the users can access the FAQ page and validate that the information is relevant and useful.

Procedure:

- Launch BilRide.
- Login with valid credentials.
- Click on the “Profile” button from the bottom menu
- Click on the "FAQ" button from the list menu.

Expected Results:

- The user should be able to access the FAQ section.
- Verify that the information is displayed correctly.
- Validate the “FAQ” page has relevant and useful information.

Severity: Minor
Date Tested: 19/05/23
Result: Pass 3, Fail 0

Test ID: BR28

Test Type: Functional

Summary: Test if the users can log out from the application successfully.

Procedure:

- Launch BilRide.
- Login with valid credentials.
- Click on the “Profile” button from the bottom menu.
- Click on “Logout”.
- Verify if the user logged out successfully.

Expected Results:

- The user should be logged out of the app successfully.
- The user should be redirected to the login page.

Severity: Major
Date Tested: 19/05/23
Result: Pass 2, Fail 0

Test ID: BR29

Test Type: Functional

Summary: Test whether users can see the chat page.

Procedure:

- Check if the user logs in to the application successfully.
- Check if the chat button at the bottom navigation bar is selectable and visible.
- Check if the user’s chat is visible and clickable as a list view item on the chat page.

Expected Results:

- User can log in to the application successfully.
- User can see the chat button at the bottom navigation bar.
- User can click the chat button.
- User can see the chat page.
- User can see the chats as a list view item on the page.

Severity: Major
Date Tested: 19/05/23
Result: Pass 5, Fail 0

Test ID: BR30

Test Type: Functional

Summary: Test whether users can enter the chat room.

Procedure:

- Check if the user logs in to the application successfully.
- Check if the chat button at the bottom navigation bar is selectable and visible.
- Check if the user's chats are visible and clickable as a list view item on the chat page.
- Check if the selected chat is visible as a separate page from previous messages.

Expected Results:

- User can log in to the application successfully.
- User can see the chat button at the bottom navigation bar.
- User can click the chat button.
- User can see the chat page and the chats as list view item in the page.
- User can see the selected chat room with previous messages.

Severity: Major
Date Tested: 19/05/23
Result: Pass 5, Fail 0

Test ID: BR31

Test Type: Functional

Summary: Test if the user can send the message to another user.

Procedure:

- Check if the user logs in to the application successfully.
- Check if the chat button at the bottom navigation bar is selectable and visible.
- Check if the user's chats are visible and clickable as a list view item on the chat page.
- Check if the selected chat is visible as a separate page with previous messages.
- Check if the text field at the bottom of the page is visible, editable, and selectable.
- Check if the send button at the right of the text field is visible and clickable.
- Check if the user filled the text field with characters.
- Check if the user clicks send button.
- Check if the application sends the message to the server.

Expected Results:

- User can log in to the application successfully.
- User can see the chat button at the bottom navigation bar.
- User can click the chat button.
- User can see the chat page and the chats as list view items on the page.
- User can see the selected chat room with previous messages.
- User can select and type any message in the text field at the bottom of the page.
- User can click send button at the right of the text field.
- Server can receive the message sent by the user.

Severity: Major
Date Tested: 19/05/23
Result: Pass 8, Fail 0

Test ID: BR32

Test Type: Functional

Summary: Test if the users can receive messages from other users.

Procedure:

- Check if the user logs in to the application successfully.
- Check if the chat button at the bottom navigation bar is selectable and visible.
- Check if the user's chats are visible and clickable as a list view item on the chat page.
- Check if the selected chat is visible as a separate page.
- Check if the user receives the messages sent by the other user in the chat room.

Expected Results:

- User can log in to the application successfully.
- User can see the chat button at the bottom navigation bar.
- User can click the chat button.
- User can see the chat page and the chats as list view items on the page.
- User can see the selected chat room.
- User can receive messages sent by the other user in the chat room.

Severity: Major
Date Tested: 19/05/23
Result: Pass 6, Fail 0

5.2 Nonfunctional Testing

Test ID: BR33

Test Type/Category: Non-Functional / Performance Testing

Title: Load Testing

Procedure of Load Testing:

- Check the response time the user receives while using the application and evaluate the response time.
- Check the response time of the application when multiple users are trying to use the application in the actual environment.
- Compare the response times and evaluate the maximum load of the application.
- Evaluate the performance of the database load before the application starts behaving unexpectedly or the application starts working slower.

Expected Results:

- Response time is lower than 1 second.
- Response time does not increase by 5 seconds when multiple users try to use the application simultaneously.
- Application should support a load of at least 1000 users simultaneously.
- Database load should remain as lower as possible to prevent any unexpected work of the application.

Severity: Major
Date Tested: 19/05/23
Result: Pass 4, Fail 0

Test ID: BR34

Test Type/Category: Non-Functional / Performance Testing

Title: Stress Testing

Procedure of Stress Testing:

- Check the system's performance on low memory or low disc space on clients and servers. Repeat this test with different numbers of memory and disc space.
- Check the system's performance when multiple users try to take the same action/transaction on the same data.
- Check the system's performance when multiple clients are connected to the servers with different workloads.
- Evaluate the maximum load the application can handle.

Expected Results:

- The system should work properly under low memory or low disc space circumstances.
- The system should not allow multiple users to take the same transaction on the same data.

- The system should work properly when multiple clients are connected to the servers with different workloads.
- The maximum load of the application should be proportional to the average user actively using the application.

Severity: Major
Date Tested: 19/05/23
Result: Pass 4, Fail 0

Test ID: BR35

Test Type/Category: Non-Functional / Performance Testing

Title: Volume and Capacity Testing

Procedure of Volume and Capacity Testing:

- Check the behavior of the system when a large amount of data is involved with the software and note the number of application failures.
- Evaluate the database size according to the application usage and the amount of data to store.
- Determine whether the application's capability will be enough in future loads or not.

Expected Results:

- The application should behave normally when more than the average amount of data is involved in the software.
- The database size should be proper according to the application usage and the number of users.

Severity: Major
Date Tested: 19/05/23
Result: Pass 2, Fail 0

Test ID: BR36

Test Type/Category: Non-Functional / Performance Testing

Title: Recovery Testing

Procedure of Recovery Testing:

- Check whether there is a way to return back when the application behaves unexpectedly/abnormally.

Expected Results:

- The user can go back to the normal state when there is an error and there should not be any data lost.

Severity: Critical
Date Tested: 19/05/23
Result: Pass 1, Fail 0

Test ID: BR37

Test Type/Category: Non-Functional / Performance Testing

Title: Reliability Testing

Procedure of Reliability Testing:

- Evaluate the time (or time estimation) that takes when the user is trying to go back to normal state.
- Check when the user returns to the normal state, are there any data lost or not.

Expected Results:

- The user can go back to the normal state when there is an error and there should not be any data lost.
- Returning to the normal state should not be long enough to make the user wait too long.

Severity: Major
Date Tested: 19/05/23
Result: Pass 2, Fail 0

Test ID: BR38

Test Type/Category: Non-Functional / Security Testing

Title: Access to Application

Procedure of Access to Application Testing:

- Use the whole application with the admin role and analyze which actions are authorized and which are not. Examine whether your role has permission to take an action for which it is not authorized.
- Repeat the previous test with Driver and Passenger roles and examine the role and the permission relation to whether there is a security leak or not.

Expected Results:

- Users in different roles should not be able to perform actions for which they are not authorized.

Severity: Critical

Date Tested: 19/05/23
Result: Pass 1, Fail 0

Test ID: BR39

Test Type/Category: Non-Functional / Security Testing

Title: Data Protection

Procedure of Data Protection Testing:

- Check the roles (admin, driver, and passenger) and their rights in the field of reading and writing the data. Inspect if a user in the passenger role can access the privileges of the driver role.
- Check if the data is kept encrypted in the database, such as the users' password.
- Examine the user credentials. Test cases are, checking whether the correct user's profile is opened when the user is logged into the application. After logging in, check that the correct data is read on the previous rides' view page.

Expected Results:

- Different users should not be able to both read and edit data for which they are not authorized.
- The user must not have the privileges of another user.
- The user should not be able to access the information of another user without his/her authorization.

Severity: Critical
Date Tested: 19/05/23
Result: Pass 3, Fail 0

Test ID: BR40

Test Type/Category: Non-Functional / Security Testing

Title: Brute-Force Attack

Procedure of Brute-Force Attack Testing:

- Check if the application allows trying to login with different passwords in a row using the same user id.
- Check and examine the number of login attempts before the account is blocked.

Expected Results:

- In 3 wrong login attempts, the account should be blocked for a certain period of time.

Severity: Critical
Date Tested: 19/05/23
Result: Pass 1, Fail 0

Test ID: BR41

Test Type/Category: Non-Functional / Security Testing

Title: SQL injection and XSS

Procedure of SQL injection and XSS Testing:

- As with login or sign-in pages, check whether the input received from the user is filtered before being stored in the database.
- Check if the size of the inputs is limited.
- Check if the input formats are checked. For example, when an input other than email format is entered where the email address is to be entered, examine whether the correct error is given.
- Check if there is an XSS filter when the user injects malicious script as input.

Expected Results:

- The user must not be able to inject malware as input.
- The user's inputs must pass XSS filters before being sent to the backend and database.
- Input type and data must be compatible with each other.

Severity: Critical
Date Tested: 19/05/23
Result: Pass 3, Fail 0

Test ID: BR42

Test Type/Category: Non-Functional / Security Testing

Title: Session Management

Procedure of Session Management Testing:

- Check if the session is terminated after a certain amount of time has elapsed.
- Check the maximum lifetime of the sessions.
- Check if the user can have multiple simultaneous sessions.
- Evaluate the session termination duration when the user logs out from the application.

Expected Results:

- If the user has been inactive for a certain period of time, the session must be terminated.

- A user should have at most one session simultaneously.

Severity: Critical
Date Tested: 19/05/23
Result: Pass 1, Fail 2
Notes: Session refresh is implemented but the session is not terminated when the user is inactive for a while

Test ID: BR43

Test Type/Category: Non-Functional / Security Testing

Title: Error Handling

Procedure of Error Handling Testing:

- Check for HTTP error codes whether error codes are accurate according to the case or not.
- Check if error messages include sensitive and critical information.

Expected Results:

- Error messages should not contain sensitive and critical information. It should not reflect information about the structure in the backend.
- Error messages should not contain information about endpoints and the backend functions to which they are directed.

Severity: Critical
Date Tested: 19/05/23
Result: Pass 2, Fail 0

Test ID: BR44

Test Type/Category: Non-Functional / Usability Testing

Title: Remote Usability Testing

Procedure of Remote Usability Testing:

- Check if the users can use the application in different locations in Turkey.

Expected Results:

- The application supports working with the locations inside Turkey.

Severity: Major
Date Tested: 19/05/23
Result: Pass 1, Fail 0

Test ID: BR45

Test Type/Category: Non-Functional / Usability Testing

Title: Automated Usability Testing

Procedure of Automated Usability Testing:

- Test scripts are written to test the use case of the application.
- The application is tested with scripts.
- The report of the output of each script is created.

Expected Results:

- The test result supports all use cases of the application.
- The report indicates the expected output of each script.

Severity: Critical
Date Tested: 19/05/23
Result: Pass 0, Fail 2
Notes: Automated test cases were not written.

Test ID: BR46

Test Type/Category: Non-Functional / Usability Testing

Title: Expert Review

Procedure of Expert Review Testing:

- Mobile application experts test the demo application.
- Experts give feedback about the usability of the application.
- They try to find a loophole in the application.
- They make a report about the application, including loopholes and feedback.

Expected Results:

- The application creates zero loopholes.
- The expert report does not contain any loopholes.
- Report can contain some improvement tips.

Severity: Major
Date Tested: 19/05/23
Result: Pass 0, Fail 3
Notes: Expert review was not completed.

Test ID: BR47

Test Type/Category: Non-Functional / User Interface Testing

Title: Evaluating the GUI

Procedure of User Interface Testing:

- Check if the user can use the application without any confusion.
- Check if the data is traversed correctly between pages.

Expected Results:

- GUI should be understandable and every user should be able to use the application easily.
- Data should be correct in every page.
- User interface should not annoy the user.
- User interface should be consistent for its look.

Severity: Major
Date Tested: 19/05/23
Result: Pass 4, Fail 0

Test ID: BR48

Test Type/Category: Non-Functional / Compatibility Testing

Title: Compatibility Testing of Mobile Devices

Procedure of Compatibility Testing:

- Check if the user can use the application with different mobile phones and different platforms such as Android and iOS.

Expected Results:

- The application should be compatible with different mobile devices platforms such as Android and iOS.

Severity: Critical
Date Tested: 19/05/23
Result: Pass 1, Fail 0

Test ID: BR49

Test Type/Category: Non-Functional / Documentation Testing

Title: Documentation Testing

Procedure of Documentation Testing:

- Check if the stated documents, read me files, release notes, user guides are provided properly.

Expected Results:

- User manuals and other documents should be provided properly.
- All of the documents should be clear and understandable.

Severity: Critical
Date Tested: 19/05/23
Result: Pass 2, Fail 0

Test ID: BR50

Test Type/Category: Non-Functional / Instability Testing

Title: Instability Testing

Procedure of Instability Testing:

- Check if the components of the applications are installed correctly on the mobile devices.
- Check if the application validates the sufficiency of the disc space.

Expected Results:

- Installation should be done without any missing component.
- The application should validate the insufficient disc space.

Severity: Major
Date Tested: 19/05/23
Result: Pass 2, Fail 0

6 Maintenance Plan and Details

6.1 Server Maintenance

Throughout the development and deployment of BilRide, we have meticulously maintained the server infrastructure to ensure optimal performance and availability. We have implemented a comprehensive server maintenance plan, which includes regular monitoring of the server's health and performance. This proactive approach allowed us to quickly identify and address any issues that arose, ensuring a smooth user experience. We have also prioritized server security by promptly applying security updates and patches to protect against potential vulnerabilities. By optimizing

server performance through techniques like caching and load balancing, we have optimized response times and enhanced the scalability of the app. Additionally, we have implemented backup and disaster recovery procedures, regularly backing up data and testing the restoration process to mitigate the risk of data loss. To ensure zero downtime, we implemented health check mechanisms inside each API server, our mechanism constantly checks the availability of the servers. If any of the available servers seem unhealthy, our mechanism automatically updates the Nginx load balancer we put in front of the API cluster. This way, we're not exposing our servers to the public internet. Our servers are currently located in one region, but we plan to distribute them to multi-region multi-zone architecture. Also, we are using a hardened version of Ubuntu 20.04 to minimize the risks in terms of unauthorized access and lateral movement. Only 2 people have direct SSH access to our API cluster, and our private keys are encrypted in a way that is only decrypted using biometric authentication (Yubikey).

6.2 Database Maintenance

As part of the development process for BilRide, we have given utmost importance to the maintenance of our database. Regular backups have been performed to ensure data integrity and availability. These backups have been securely stored offsite, minimizing the risk of data loss. We have focused on optimizing the database performance by fine-tuning queries, managing indexes, and refining the database schema. By implementing these optimization techniques, we have improved the efficiency of data retrieval and storage, resulting in a smooth and responsive user experience. We have also implemented monitoring and alert mechanisms to detect and address any performance issues or anomalies. Furthermore, we have prioritized data security by implementing access controls, encryption, and auditing mechanisms to safeguard user data. Also, our database cluster is inside the auto-scaling group.

6.3 Application Maintenance

Having successfully completed the development and deployment of BilRide, we understand the importance of ongoing application maintenance. Our maintenance plan includes continuous bug tracking and resolution to address any reported issues promptly. By utilizing bug-tracking and ticket-management software, we have

efficiently tracked and resolved bugs to ensure the stability and reliability of the app. We are also using a version control system to manage the source code, enabling easy tracking of changes and facilitating collaboration among team members. Through thorough testing and a well-defined deployment strategy, we have ensured that updates and releases are seamless and free of bugs. We have actively sought user feedback, providing channels for users to report issues and suggestions. This feedback has been invaluable in improving the app and enhancing the user experience. Additionally, we have provided timely support to users, addressing their queries and concerns promptly. We have also prioritized compatibility by staying up-to-date with the latest technologies and releasing updates to ensure BilRide remains compatible with different devices and operating system versions. To ensure our users consistently enjoy the best possible user experience, we have implemented an update enforcement screen within BilRide. This screen is designed to ensure that all users stay up-to-date with the latest version of the app, as updates often include important bug fixes, performance enhancements, and new features. By implementing this enforcement screen, we aim to maintain a cohesive user base that benefits from the most recent improvements and optimizations. It allows us to provide a seamless and secure app experience by ensuring that all users are on a consistent and up-to-date version of BilRide. This approach not only enhances the user experience but also helps us deliver the best support and address any reported issues more effectively, as we can focus on supporting the most recent version of the app. We believe that this update enforcement screen is crucial for keeping BilRide at its highest standard and providing our users with a reliable and enjoyable mobile app experience.

7 Other Project Elements

7.1 Consideration of Various Factors in Engineering Design

7.1.1 Environmental Factors

Our project addresses environmental factors such as environmental sustainability and benefits. In our analysis, we took what is best in terms of environmental benefit into consideration.

7.1.2 Public Safety Factors

Safety is a highly important factor for BilRide. In our thinking and consideration phase, we highly considered protecting users' personal data, which are protected by “Kişisel Verilerin Korunumu Kanunu” (KVKK). Our project must be highly secure because the user data is confidential and protected by data protection regulations. Also, we considered situations that may put our users’ safety at risk, like careless drivers, users performing unacceptable behaviors when participating in carpooling, and users with serious disciplinary actions. We came up with solutions to these kinds of situations when analyzing and designing our project.

7.1.3 Economical Factors

From an economic point of view, we need to address how much a ride will possibly cost and a recommended price to share between rider and passengers. Our application will be free to use, and we provided our business model with an optional payment between riders and passengers.

7.1.4 Social Factors

We also took social factors into consideration. In analysis and requirements, we developed features that can simulate a social platform like sharing playlists, commenting and rating, achievements, chat rooms, and forming teams.

7.1.5 Public Health Factors

Public health is also a factor that affects our thinking. Since we are affected by a pandemic, we may put more importance on public health if there are situations that put users at risk later, considering the situation of the pandemic.

7.1.6 Global Factors

Since our project focused on the transportation of Bilkent University specifically, global factors are not as important as other factors discussed.

	Effect level	Effect
Public health	8	Users should be notified when there is a concern of health.
Public safety	10	Users need to be notified and protected in any case that disrupts safety that may be caused by the driver or passenger.
Public welfare	4	Public welfare is connected with public health and public safety regarding our topic.
Global factors	1	Our scope is Bilkent University and its individuals.
Cultural factors	1	Our users consist of only Bilkent University users, but if we experience any cultural factor, we must consider it can affect our analysis and design later on.
Social factors	9	There are many features that will boost the sociality of the application, as discussed above.
Economical factors	7	Payment between drivers and passengers will be optional, and the amount will be decided by users.
Environmental factors	10	Main focus is to do what is best for the environment.

Table 1: Factors that can affect analysis and design.

7.2 Ethics and Professional Responsibilities

During the analysis phase, we considered two potential ethical and professional issues regarding our application. First one is the privacy of users which we see as both a professional and ethical responsibility of ours. Any private user data should be safe and secure to data leakages. This confidential data that is protected by “KVKK” should be safe and secure in any case of data leakage. Also, there may be some cases in which users participate in a ride from or to their houses. To reduce any problems caused by this inevitable sharing of private information, we decided to provide a feedback and complaint system in order to detect problematic users. In addition, we decided to get information from Bilkent University database if we can to detect if any user has a complaint or disciplinary action in the university database in order to prevent any harm to our users. Our second responsibility is providing users to choose which users they want to travel with. We find it useful to show some information about users, especially ratings and complaints. Additionally, we are planning to put a gender option because users can have some preferences related to the person who they will share a ride with.

7.3 Teamwork Details

7.3.1 Contributing and functioning effectively on the team

The most important factor of proper teamwork is communication. As a team, we scheduled face-to-face meetings. Also, we scheduled zoom meetings when we could not meet in the physical environment. Responsibilities were distributed to each team member considering the workload and previous experiences. However, the responsibilities increased, and they were changed before the end of the first semester due to the progress of the project and the change in plans and schedule. When we worked on the project, some plans changed. Nevertheless, it kept everything the same in terms of the contribution and effectiveness of the members. We have worked together before, so it is easier to decide and communicate based on our previous experiences.

In addition, we created a schedule for each sub-work in the project to avoid trying to complete the project on the last day. Though some sub-works changed in the first semester, we mostly did not extend the due date of tasks.

7.3.2 Helping create a collaborative and inclusive environment

The project's works are distributed among members, but this does not mean we work only on the tasks given. When one of the members has a problem or needs assistance with the task, one of the other team members helps with the problem. If it is a big problem, all of the members are here to help. Additionally, after work is done, it is reviewed by other members using Github. Also, we used some common test cases which we defined at the beginning of the first semester, and each member used these test cases before pushing the changes to Github. Thus, we made sure that there were minimum mistakes in the code. Even if there is a mistake, other members can solve the mistakes.

7.3.3 Taking the lead role and sharing leadership on the team

In the analysis report, we introduced the work packages in which each member is involved in the leadership of each package. However, there are some changes in the distribution of work packages. Even if there is a change in work packages, the distribution of leadership among team members is still equal. The

work packages and their distribution among team members can be found in Table 2.

WP#	Work package title	Leader	Members involved
WP1	Project Specification Report	Doğukan	Everyone
WP2	Analysis Report	Turgut	Everyone
WP3	Backend Implementation	Turgut	Everyone
WP4	Permission for the Data of Bilkent Members	İdil	Everyone
WP5	Frontend Implementation	Doğukan	Everyone
WP6	Mid Testing	Funda	Everyone
WP7	Demo Presentation	İdil	Everyone
WP8	Implementation of Shuttle Schedule	Dilay	Everyone
WP9	Detailed Design Report	Dilay	Everyone
WP10	Final Testing	Turgut	Everyone
WP11	Final Report	Funda	Everyone
WP12	Final Presentation	Doğukan	Everyone

Table 2. List of Work Packages

7.3.4 Meeting Objectives

The goals outlined in previous reports have been effectively achieved. However, certain modifications have been made to certain features and backend development. In the backend development, Flask was preferred instead of Django in the backend development, because Flask is easier to use, has a lighter structure, and can also have better performance. Additionally, Since collecting data is not easy, it was decided to utilize the Google Maps API, which assists in creating location-based application and website content.

For example, when Django was first used, it was noticed that there were modules that took up unnecessary space for the project due to the large codebase size. Since our project is not only web-based, Flask's small codebase size, flexible and scalable features provided a great advantage in terms of backend development.

In addition, It would not be easy for the team to calculate the route hours for each ring and collect data accordingly, and the accuracy of the collected data would not give precise information. Since the route of each ring is clear, it

took less time to use Estimated Time using Google Maps API instead of collecting data, as drivers can determine routes at the same time. This decision allowed us to integrate various features provided by Google Maps, including static and dynamic maps, high-quality directions, and routes generated with real-time traffic updates, as well as the ability to leverage rich location data.

7.4 New Knowledge Acquired and Applied

As BilRide, we aimed to design an app for the members of Bilkent University, including students, academics, and employees, with a focus on sustainability and effectiveness. The most crucial factor of the project was to accurately calculate and predict the estimated travel times for drivers and passengers, reducing transportation-related difficulties to a minimum through a user-friendly app. Therefore, we studied examples from different applications, created assets, and aimed for user satisfaction.

Another significant factor was to research the bus services passing through the stops within Bilkent and their schedules. We needed to determine the locations of the stops within Bilkent, identify the bus services passing through these stops, and know their routes. We also gathered the opinions of users who hitchhike, observed where they most commonly hitchhike within the school, and which routes they prefer. Taking into account the most preferred stops within the campus and the areas where hitchhiking is possible, we ensured the progress of the project. Additionally, considering one of the problems that drivers face during their journey, which is the price of gasoline/diesel, we attempted to introduce a tipping system that could benefit passengers. This way, the driver's financial burden is reduced while they continue on their route, and passengers can overcome transportation issues.

Since Django, which we decided to use in the backend of the application, occupies a lot of space and there are unused modules in it, we decided to use Flask, which takes up less space. Since data collection to be used to determine estimated time for each route will take a lot of time and doing this for each ring

will not give an accurate result, we decided to use the API of Google Maps and optimized the routes in our project in this way.

8 Conclusion and Future Work

In the end of the project, we created a proper app to solve the transportation problems in Bilkent University by completing the main functionality and main goals of the project. With the help of Flutter, as we planned, BilRide can run on both Android and IOS devices. For Bilkent students and alumni, we can propose a new application that solves the transportation problem in Bilkent University.

Although we have completed the main functionality, the application may face with some bugs. Also, deployment of the application on the mobile platforms is not cheap, Therefore, due to these reasons, we could not publish our application. But, even if we could not solve all the bugs, we can publish the application in the future. To resolve all the bugs in the app, we continue working on this project in the future. Also, the application always open to improve further since the world does not stand still. We can always improve our application to meet the requirements of the future world.

During implementation, we encounter many challenges. The most hard challenge was time management. However, we overcome this issue as a team by planning the actions, deadlines and continuous improvement. We learned a lot during the project and improved our coding skills. Besides, we improved our social skills as well, such as teamwork, leadership and time management. Even if we can think negatively from time to time, we can always find a positive side to overcome the issues and thoughts.

9 Glossary

- Ring shuttle/ring: The buses that take Bilkent students to the city center or neighborhoods.
- EGO: The official bus service of the Municipality of Ankara.
- Hitchhike: to travel by securing free rides from passing vehicles.
- Carpool: an arrangement in which a group of people commutes together by car.
- Dockerize: to pack, deploy, and run applications using Docker containers.
- GDPR: General Data Protection Regulation.
- KVKK: Kişisel Verileri Koruma Kanunu (Personal Data Protection Law).
- DoS Attack: denial of service attack meant shutting down a machine or network, making it inaccessible to its intended users.
- backend: the part of a computer system or application that is not directly accessed by the user, typically responsible for storing and manipulating data.
- API: Application Programming Interface.
- UI: User Interface.
- FAQ: Frequently Asked Questions.
- WAF: Web Application Firewall.
- IEEE: Institute of Electrical and Electronics Engineers.

10 References

[1] “BlaBlaCar Hakkında”. https://support.blablacar.com/hc/tr/categories/360002754379-BlaBlaCar-hakk%C4%B1nda. [Accessed: May 19, 2023]

[2] “Personal Data Protection Law”. https://www.kvkk.gov.tr/Icerik/6649/Personal-Data-Protection-Law. [Accessed: May 19, 2023]