Bilkent University
Department of Computer Engineering

Senior Design Project
T2320
BilRide

Detailed Design 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

2023-02-18

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 detailed design report includes the introduction of the system, which provides for the system and design goals. Also, it contains competitors, current and alternative solutions, proposed software architecture consisting of subsystem decomposition, persistent data management and access control and security, details of subsystem services, test cases divided as functional and non-functional, various factors in engineering design, and teamwork details.

1.1 Purpose of the system

The primary purpose of BilRide is to create a mobile app to solve the issues caused by the lack of ring shuttles at Bilkent University and the lack of hitchhiking culture. It also aims to transform the transportation environment of Bilkent and its individuals. It is willing to create an opportunity for people who are cautious about hitchhiking, enabling people to socialize with others. In the meantime, the driver will not suffer from high gas prices, and the passengers will be able to arrive at the campus on time and in a much more comfortable way. Also, the carbon emissions due to private car usage will decrease. In addition to carpooling, all users can track the ring shuttles and see the estimated location with the ring tracking feature, which we determine which station the rings are at with the data we will collect. Thus, when the passenger who wants to do carpooling cannot find a car and/or misses it, s/he can choose to get on the ring thanks to the BilRide app.

1.2 Design goals

1.2.1 Usability

The application's main aim is to solve the overall transportation problem in Bilkent by reaching every student and alumni at the university. It 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 should be easy for novice users to learn. The user should feel satisfied while using the application with intuitiveness. Also, the provided manual of the application increases usability.

1.2.2 Supportability

The application should be able to run on both Android (10.0 and above) and iOS (11.0 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.

1.2.3 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.

1.2.4 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.

1.2.5 Security

The application will be going to store and process some personal information. We should take the necessary actions to guarantee our users that their data is safe. Such as applying the security by design principle. Our application is also tightly connected to local and global regulators. The application should prevent leaking any kind of user data. Therefore the application’s security system will use a zero-trust approach.

1.2.6 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 Dockerizing the API to enhance security and scalability.

1.3 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. 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 Current Software Architecture

2.1 Competitors

Though there is not precisely the same application as ours in Turkey, some applications use nearly the same logic as our application worldwide. However, residents of Turkey do not use these applications for now, so our competitors are limited in Turkey. Though, if we can expand the app outside of Turkey, they can be a competitor for us. On the other hand, there is BlaBlaCar in Turkey. Passenger opens the app and lists the upcoming rides according to date, time, and place. Then, he or she enrolls in the ride. After the ride, he or she pays the price of the ride. This logic is valid for all carpooling applications, including our project [1].

2.2 Current and Alternative Solutions

These two competitors target general users nationwide and do not target any specific community. So, they currently are the solutions for the carpooling problem. However, BlaBlaCar specifies mostly city-to-city travel, and both of the applications have high prices compared to our proposed price. It is proposed that the only price is for the gas, so naturally, it is lower than the other apps since it works with the take-and-drop logic.

In our project, the target group is Bilkent University students and alumni. It is aimed that the project is the solution to the transportation problem of Bilkent University. The project also focuses on the ring problem by allowing users to track the ring times and estimated location of the rings, and since the only alternative solution for the rings currently, which is EGO buses, is not commonly used and it becomes expensive compared to rings, our application could be the solution to this problem.

3 Proposed Software Architecture

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

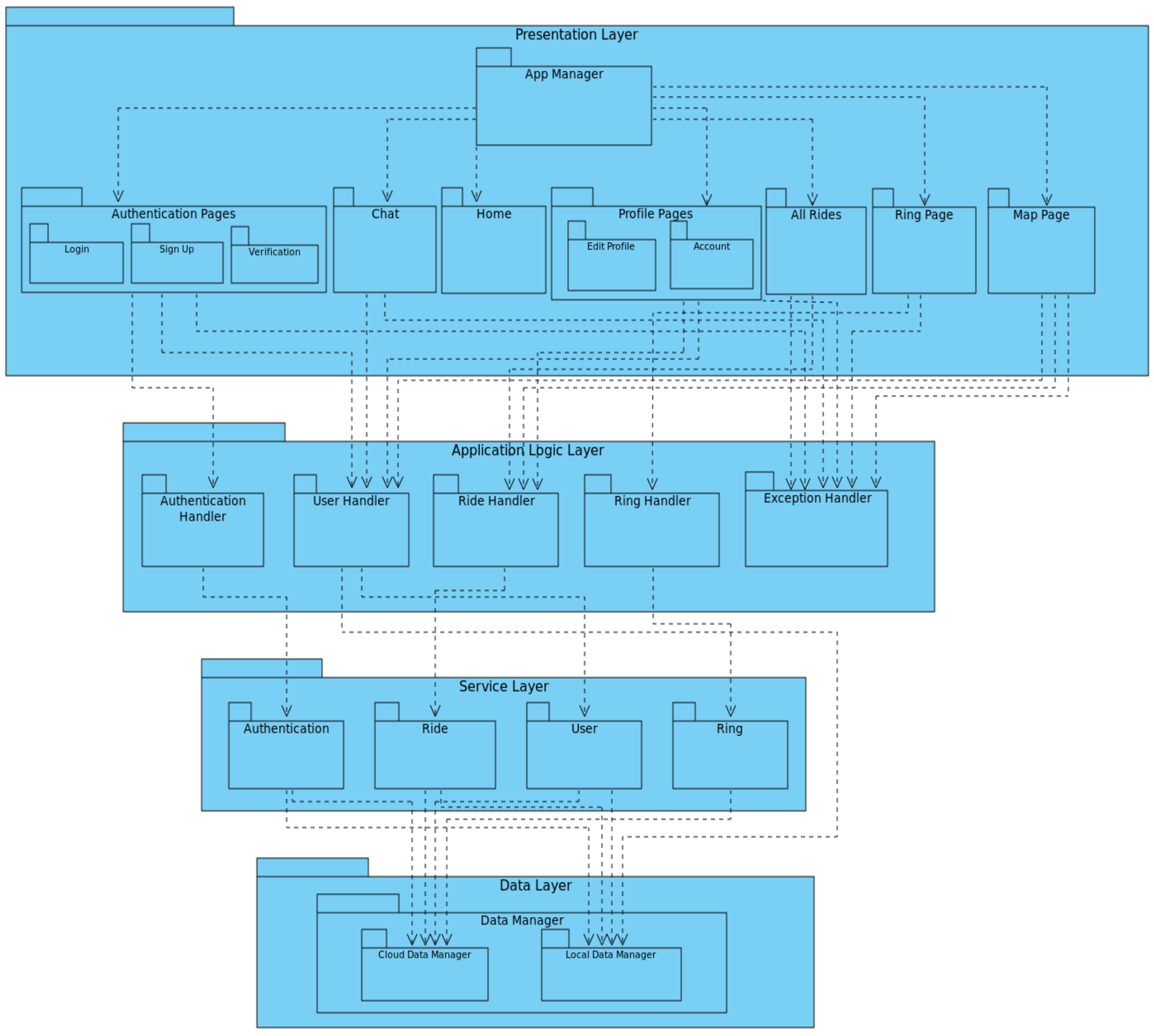

Figure 1: *Subsystem Decomposition Diagram*

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. 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.

4 Subsystem Services

In this section, figure 1 is explained in detail.

4.1 Presentation Layer

Presentation layer is the first layer of the application that allows users to communicate with the application via pages. It creates the UI of the application.

4.1.1 App Manager

It is the top level of the presentation layer, which organizes the mechanics of the application interface. It mainly manages two mechanics of the application, which are navigation and state. In addition to them, it helps the handlers and services to create a convenient UI. The app manager creates and manages states to run the application efficiently. It also uses local storage via Shared Preferences to store authentication cookies and responses from the cloud server, and the map chunks will be stored in local storage to optimize and speed up the application. Besides, it manages the navigation between pages to create a better application experience for the users and protect the app's logic. With the app manager, the application becomes faster and more efficient.

4.1.2 Home Page

This page includes the map that shows the available rides nearby the user's location. Also, users can filter the available rides by specifying the date, time, start, and end location using the find pool option on the page. Besides, users can offer a pool with date, time, start, and end location using the offer pool option on the page. This page allows users to go to the chat, ring, and profile pages with a navigation bar and to go to the all rides page to see all available rides with a button that is at the bottom right of the page.

4.1.3 Chat Page

This page allows users to communicate with each other. In the first place, there is a page that uses a list view to list all user chats. Then, the user can select one of the chats and go to the chat room. In the chat room, the users can communicate with each other via typing. As in the home page, this page also allows users to go to other pages via the navigation bar.

4.1.4 Ring Page

This page allows users to see the timetable of shuttles as a list view at the bottom of the page and the estimated locations of the buses on the map. Selecting upcoming rides from the list, the users can filter to see the route and estimated location of the bus on the map. As in the home page, this page also allows users to go to other pages via the navigation bar.

4.1.5 All Rides Page

This page has a list view that lists all available rides to the user with name, capacity, start location, end location, date, and time. The users can select one of these rides to see the route on the map and the details of the ride, such as the driver and other passengers. As in the home page, this page also allows users to go to other pages via the navigation bar.

4.1.6 Profile Pages

This part includes the pages that form a profile page together.

4.1.6.1 Account Page

This page includes the profile picture, name, surname, and activation status of the user. Also, it includes a list view that allows users to see their detailed profile page and their vehicle if they are drivers, manage their saved addresses, see their previous trips, see their teams, see the FAQ page, and log out from the application using the list items. As in the home page, this page also allows users to go to other pages via the navigation bar.

4.1.6.2 Edit Profile Page

This page includes editable text fields, which allow users to change their names, emails, phone numbers, a short bio, and their profile picture. Also, they can see the account creation date and the user type. Users can go back to the profile page using the back button in left most corner of the page.

4.1.7 Auth Pages

This part includes the pages that form an authentication page together.

4.1.7.1 Landing Page

This page introduces the application to the new user. The users can see the next page with the next button at the bottom of the page and skip to the last tab using the skip button at the top right of the page. In the last tab, there is a lets started button that navigates users to the login page.

4.1.7.2 Login Page

This page includes text fields that allow users to enter their emails and passwords. Also, there is a signin button that handles the login function and navigates users to the home page if the credentials are correct, and a register button that navigates users to the register page.

4.1.7.3 Register Page

This page includes text fields that allow users to enter their necessary information to sign up for the application, such as name, surname, email address, phone number, password, and bio. After entering the information, the signup button navigates to the verification page if the email address and phone number are unique. The email is verified on the verification page, and after this page, the users see the login page again.

4.2 Application Logic Layer

This layer handles the data transportation work between UI and services. The data entered in UI will be transferred to the services using the handlers in this layer.

4.2.1 Exception Handler

This handler handles all errors that users may face. The errors can be categorized into two types, response-related errors, and application-related errors. Response-related errors are the errors that are due to the response codes, such as unauthorized, same unique value, and not found. These errors can be expected, and they do not cause any runtime errors in the application; they just lead to warning and error screens in the UI. On the other hand, application-related errors are due to the mismanagement of the app manager in the presentation layer. These errors are not expected and lead to the view of the not found page.

4.2.2 Chat Handler

This component manages the data flow of the users' chat between UI and chat service. This handler reflects the data from the service to UI and from UI to the service. Therefore, the chat between users is possible, and the app manager renders UI whenever new data from the service or the user comes to the handler.

4.2.3 User Handler

This handler handles the user data flow between UI and service. With this handler, when the change in the user information occurs, the changes can be reflected in the service, and the app manager is triggered. Hence, the UI is changed after the change of user information occurs.

4.2.4 Ring Handler

This handler manages the ring data coming from the ring service and reflects the data to the UI using the app manager. Also, when the data is changed, it triggers the app manager, and the UI is changed.

4.2.5 Ride Handler

This handler manages the ride data coming from the ride service. Then, it helps the app manager to reflect the ride data in the necessary components of UI.

4.2.6 Auth Handler

This component handles the authentication-related jobs in the application, such as login and sign-up functionalities.

4.3 Service Layer

This layer is the first step of the response to the requests from the server. Services are responsible for making requests and organizing the response to be used in the presentation layer.

4.3.1 Chat Service

This service is responsible for the chat logic of the application. It provides information about the recent chat messages to the server. Then, it gets new messages from the server and sends these to the handler.

4.3.2 User Service

Since the application is user-related, all of the jobs in the UI needs user service. However, to reduce the workflow and speed up the application, user service is split into 5 services, which are chat, ring, ride, and auth. In this service, main user-related requests, such as getting, editing, and deleting user, are done, and the response from these requests is edited.

4.3.3 Ring Service

This service handles the requests about the rings, such as their time information.

4.3.4 Ride Service

This service handles the requests about the rides, and requests, such as creating, editing, and deleting rides, and creating, approving, and deleting requests.

4.3.5 Auth Service

This service handles authentication-related requests, which are login and sign-up. Also, it handles the authentication needed requests since the protected routes of the server need an authorization header to complete the request properly. When the user logs in, the authorization header part of each request is handled by this service since it provides this header.

4.4 Data Layer

This layer manages the data on the server.

4.4.1 Data Manager

This manager manages the data stored in the cloud server and local device. It forms the overall backend structure of the application. Since it manages the data in the server, each service needs to communicate with this manager.

4.4.1.1 Cloud Data Manager

It manages the data in the cloud server. Also, it manages CRUD operations of the server and provides protection for some operations.

4.4.1.2 Local Data Manager

It manages the local storage to store some information from the local device, such as user preferences and JWT tokens.

5 Test Cases

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:
Result:

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, and a confirmation message is displayed on the screen.

Severity: Critical
Date Tested:
Result:

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:
Result:

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:
Result:

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 the user enters invalid credentials or if there are any errors in loading the page.

Severity: Major
Date Tested:
Result:

Test ID: BR06

Test Type: Functional

Summary: Verify that the user can successfully offer a pool

Procedure:

- Launch the carpooling app.
- 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 “All Rides”.

Severity: Major
Date Tested:
Result:

Test ID: BR07

Test Type: Functional

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

Procedure:

- Launch the carpooling app.
- 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:
Result:

Test ID: BR08

Test Type: Functional

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

Procedure:

- Launch the carpooling app.
- Log in to the system using valid credentials
- Clicking the “All Rides” button from the main page.

Expected Results:

- The user should be able to log in to the application successfully.
- The "See All 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:
Result:

Test ID: BR09

Test Type: Functional

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

Procedure:

- Launch the carpooling app.
- 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:
Result:

Test ID: BR10

Test Type: Functional

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

Procedure:

- Launch the carpooling app.
- 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:
Result:

Test ID: BR11

Test Type: Functional

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

Procedure:

- Launch the carpooling app.
- 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:
Result:

Test ID: BR12

Test Type: Functional

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

Procedure:

- Launch the carpooling app.
- 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 a request.

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:
Result:

Test ID: BR13

Test Type: Functional

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

Procedure:

- Launch the carpooling app.
- 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:
Result:

Test ID: BR14

Test Type: Functional

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

Procedure:

- Launch the carpooling app.
- 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:
Result:

Test ID: BR15

Test Type: Functional

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

Procedure:

- Launch the carpooling app.
- 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:
Result:

Test ID: BR16

Test Type: Functional

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

Procedure:

- Launch the carpooling app.
- Log in to the system using valid credentials
- Click on the “Ring” button from the bottom menu
- Click on the "FAQ" button from the list 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:
Result:

Test ID: BR17

Test Type: Functional

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

Procedure:

- Launch the carpooling app.
- 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:
Result:

Test ID: BR18

Test Type: Functional

Summary: Verify that users can edit their profile information.

Procedure:

- Launch the carpooling app.
- 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:
Result:

Test ID: BR19

Test Type: Functional

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

Procedure:

- Launch the carpooling app.
- Login with valid credentials.
- Click on the “Profile” button from the bottom menu
- Click on the user name.
- Click on the "Edit Vehicle" 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:
Result:

Test ID: BR20

Test Type: Functional

Summary: Verify that users can edit their saved addresses.

Procedure:

- Launch the carpooling app.
- 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:
Result:

Test ID: BR21

Test Type: Functional

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

Procedure:

- Launch the carpooling app.
- 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:
Result:

Test ID: BR22

Test Type: Functional

Summary: Verify if the user can edit teams.

Procedure:

- Launch the carpooling app.
- 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:
Result:

Test ID: BR23

Test Type: Functional

Summary: Verify if the user can create teams

Procedure:

- Launch the carpooling app.
- 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:
Result:

Test ID: BR24

Test Type: Functional

Summary: Verify if the user sends a request for joining a team

Procedure:

- Launch the carpooling app.
- 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 “Join a Team” button.
- Browse from the teams list
- Click “Join” for a team from the team's list.
- Verify the request is created and sent successfully and that the recipient users (team owners) are notified.

Expected Results:

- Request is created and sent successfully.
- The recipient users (team owners) are notified.

Severity: Minor
Date Tested:
Result:

Test ID: BR25

Test Type: Functional

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

Procedure:

- Launch the carpooling app.
- 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:
Result:

Test ID: BR26

Test Type: Functional

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

Procedure:

- Launch the carpooling app.
- 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:
Result:

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 the carpooling app.
- Login with valid credentials.
- Click on the “Profile” button from the bottom menu
- Click on the "FAQ" button from the list menu.
- Verify that the information is displayed correctly.
- Validate the “FAQ” page has relevant and useful information.

Expected Results:

- The user should be able to access the FAQ section.

Severity: Minor
Date Tested:
Result:

Test ID: BR28

Test Type: Functional

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

Procedure:

- Launch the carpooling app.
- 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:
Result:

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:
Result:

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:
Result:

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:
Result:

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:
Result:

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 10000 users simultaneously.
- Database load should remain as lower as possible to prevent any unexpected work of the application.

Severity: Major
Date Tested:
Result:

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:
Result:

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:
Result:

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:
Result:

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:
Result:

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:
Result:

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:
Result:

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:
Result:

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:
Result:

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:
Result:

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:
Result:

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:
Result:

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:
Result:

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:
Result:

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:
Result:

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:
Result:

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:
Result:

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 validates the insufficient disc space.

Severity: Major
Date Tested:
Result:

6 Consideration of Various Factors in Engineering Design

6.1 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.

6.2 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.

6.3 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.

6.4 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.

6.5 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.

6.6 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 Teamwork Details

7.1 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.

7.2 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 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

8 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.

9 References

[1] “BlaBlaCar Hakkında”.
https://support.blablacar.com/hc/tr/categories/360002754379-BlaBlaCar-hakk%C4%B1nda. [Accessed: Oct 16, 2022]