**HumSawar Carpool**

Logo, company name

Description automatically generated

***By***

**Maviya Akram 191176**

**Taimoor Hassan 191098**

***Supervisor*Ms. Warda**

***Department of Computer Science***

**Air University, Islamabad**

***(2019-2023)***

**HumSawar Carpool**

Logo, company name

Description automatically generated

***By***

**Maviya Akram 191176**

**Taimoor Hassan 191098**

***Supervisor*Ms. Warna**

***Bachelor of Science in Computer Science / Information Technology***

**Air University, Islamabad**

***(2019-2023)***

**DECLARATION**

We hereby declare that this software, neither whole nor as a part has been copied out from any source. It is further declared that we have developed this software and accompanied report entirely on the basis of our personal efforts. If any part of this project is proved to be copied out from any source or found to be reproduction of some other, we will stand by the consequences. No Portion of the work presented has been submitted of any application for any other degree or qualification of this or any other university or institute of learning.

Maviya Akram Taimoor Hassan

--------------------------- --------------------

**CERTIFICATE OF APPROVAL**

It is to certify that the final year project of BS (CS) “HumSawar” was developed by   
**Maviya Akram (191176)** and **Taimoor Hassan (191098)** under the supervision of “Ms. Warda”and that in her opinion; it is fully adequate, in scope and quality for the degree of Bachelors of Science in Computer Sciences / Information Technology.

---------------------------------------

**Supervisor**

---------------------------------------

**Head of Department**

**(Department of Computer Science)**

**Executive Summary**

The HumSawar Carpool Application is a mobile application built on React Native, incorporating Firebase, Google Maps APIs, and various other technologies. Its primary objective is to provide a convenient, safe, and efficient carpooling system for commuters. With its unique features and advanced functionalities, HumSawar aims to revolutionize the way people travel, promoting sustainability, reducing traffic congestion, and enhancing social interactions.

HumSawar stands out from other carpooling applications by introducing the "Pink" feature, which allows female drivers to match exclusively with female passengers. This feature addresses the specific needs and preferences of female users, providing a sense of security and comfort during their shared rides.

The application leverages the power of Google Maps APIs to enable users to easily search for carpooling opportunities based on their location, desired destination, and preferred routes. Real-time traffic information and navigation assistance ensure efficient and seamless travel experiences for both drivers and passengers.

HumSawar incorporates a live messaging system, facilitating communication between drivers and passengers for smooth coordination and any necessary updates during the carpooling journey. This feature enhances convenience, flexibility, and reliability for all users.

The application utilizes Firebase for secure authentication, real-time data synchronization, and reliable storage. User profiles, trip details, and preferences are securely stored and easily accessible, ensuring a seamless and personalized experience for all users.

HumSawar is designed with a user-friendly interface, intuitive navigation, and a visually appealing design. It offers various additional features commonly found in carpooling applications, such as fare splitting, ratings and reviews, trip history, and notifications.

The underlying architecture of HumSawar is robust and scalable, allowing for potential future enhancements and expansions. Security and data privacy are prioritized, ensuring the protection of user information and fostering a trusted and secure environment for carpooling.

In conclusion, HumSawar Carpool Application provides a comprehensive and innovative solution for commuters seeking a reliable, sustainable, and efficient mode of transportation. By leveraging the power of technology, HumSawar aims to reshape the carpooling landscape, fostering a sense of community, reducing traffic congestion, and promoting eco-friendly travel.

**Acknowledgement**

All praise is to Almighty Allah who bestowed upon us a minute portion of His boundless knowledge by virtue of which we were able to accomplish this challenging task.

We are greatly indebted to our project supervisor “\_\_\_\_\_\_\_” and our Co-Supervisor “\_\_\_\_\_”. Without their personal supervision, advice and valuable guidance, completion of this project would have been doubtful. We are grateful to them for their encouragement and continual help during this work.

And we are also thankful to our parents and family who have been a constant source of encouragement for us and brought us with the values of honesty & hard work.

Student Name 1 Student Name 2

--------------------------- ---------------------------

**Abbreviations**

| **SRS** | Software Require Specification |
| --- | --- |
| **PC** | Personal Computer |
|  |  |
|  |  |
|  |  |

**Table of Contents**

[1](#_30j0zll) Introduction 11

[1.1](#_1fob9te) Vision Statement 11

[1.2](#_3znysh7) Related System Analysis/Literature Review 11

[1.3](#_2et92p0) Project Deliverables 12

[1.4](#_tyjcwt) System Limitations/Constraints 12

[1.5](#_3dy6vkm) Tools and Technologies 12

[1.6](#_1t3h5sf) Relevance to Course Modules 12

[2](#_2s8eyo1) Problem Definition 13

[2.1](#_17dp8vu) Problem Statement 13

[2.2](#_26in1rg) Problem Solution 13

[2.3](#_lnxbz9) Objectives of the Proposed System 13

[2.4](#_35nkun2) Scope 14

[2.5](#_1ksv4uv) Modules 14

[2.5.1](#_44sinio) Module 1: Module Name 14

[3](#_3j2qqm3) Requirement Analysis 15

[3.1](#_1y810tw) User classes and characteristics 15

[3.2](#_4i7ojhp) Requirement Identifying Technique 15

[3.3](#_2xcytpi) Functional Requirements 15

[3.3.1](#_1ci93xb) Functional Requirement X 15

[3.4](#_3whwml4) Non-Functional Requirements 16

[3.4.1](#_2bn6wsx) Reliability 16

[3.4.2](#_qsh70q) Usability 17

[3.4.3](#_3as4poj) Performance 17

[3.4.4](#_1pxezwc) Security 17

[3.5](#_49x2ik5) External Interface Requirements 17

[3.5.1](#_2p2csry) User Interfaces Requirements 17

[3.5.2](#_147n2zr) Software interfaces 18

[3.5.3](#_3o7alnk) Hardware interfaces 18

[3.5.4](#_23ckvvd) Communications interfaces 18

[4](#_ihv636) Design and Architecture 19

[4.1](#_1hmsyys) Architectural Design 19

[4.2](#_41mghml) Design Models 19

[4.3](#_2grqrue) Data Design 20

[4.3.1](#_vx1227) Data Dictionary 20

[4.4](#_3fwokq0) Human Interface Design 20

[4.4.1](#_4f1mdlm) Screen Images 20

[4.4.2](#_2u6wntf) Screen Objects and Actions 20

[5](#_19c6y18) Implementation 21

[5.1](#_3tbugp1) Algorithm 21

[5.2](#_28h4qwu) External APIs/SDKs 23

[5.3](#_nmf14n) User Interface 23

[5.3.1](#_37m2jsg) Login Screen 5.3.2 Home Screen 24

[5.3.3](#_46r0co2) Assignee Dashboard 25

[5.3.4](#_111kx3o) New Complaint 25

[5.4](#_3l18frh) Deployment 26

[6](#_206ipza) Testing and Evaluation 27

[6.1](#_2zbgiuw) Unit Testing 27

[6.2](#_1egqt2p) Functional Testing 27

[6.3](#_3ygebqi) Business Rules Testing 28

[6.4](#_2dlolyb) Integration Testing 28

[7](#_sqyw64) Conclusion and Future Work 29

[7.1](#_3cqmetx) Conclusion 29

[7.2](#_1rvwp1q) Future Work 29

[8.](#_4bvk7pj) References 30

**List of Figures**

[Figure 1 Home Screen](about:blank) 24

[Figure 2 Login Screen 24](#_1mrcu09)

[Figure 3 Assignee Dashboard 25](#_2lwamvv)

[Figure 4 New Complaint](about:blank) 25

# Introduction

In the following sections, we will delve into the vision statement, related system analysis or literature review, project deliverables, system limitations or constraints, tools and technologies utilized, and the relevance of the project to the course modules. Each section aims to provide a comprehensive understanding of the project and its significance.

## Vision Statement

For carpool commuters who seek a reliable and convenient transportation solution, HumSawar is a mobile application built on React Native with Firebase integration and Google Maps APIs. It is a comprehensive carpooling platform that enables users to efficiently connect with fellow travelers, reducing traffic congestion and promoting sustainability. Unlike traditional carpooling methods, HumSawar offers a unique feature called "Pink" that ensures female drivers and passengers can match exclusively with other female passengers, enhancing safety and comfort during the commute. With its live messaging system and a range of essential carpooling features, HumSawar revolutionizes the way people share rides, fostering a sense of community and optimizing travel experiences for all users.

## Related System Analysis/Literature Review

**Write** about the existing/similar systems related to your proposed project.

Don’t use more than 4 sentences for explaining a single system/application.

Briefly provide an analysis of the related system which may help you to specify the contribution of the proposed project.

**Table 1 Related System Analysis with proposed project solution**

| **Application Name** | **Weakness** | **Proposed Project Solution** |
| --- | --- | --- |
|  | Limited features and user interface design | HumSawar addresses the limitations by providing a gender-based matching for female users, and an intuitive user interface for a seamless carpooling experience. |
|  | Lack of Scheduled Rides | Without the ability to schedule rides in advance, users may face challenges in planning their carpooling activities. |

## Project Deliverables

The project deliverables for the "HumSawar" carpool application are as follows:

1. Mobile Application:

- Fully functional and user-friendly mobile application built on React Native.

- Compatible with both Android and iOS platforms.

- Includes all the necessary screens and features for carpooling, such as registration/login, profile management, ride search and booking, ride sharing, messaging, and notifications.

2. Database Integration:

- Integration of Firebase as the backend database for storing and managing user data, ride details, and messaging history.

- Implementation of secure user authentication and data storage.

3. Google Maps Integration:

- Integration of Google Maps APIs to provide mapping and location-based services.

- Implementation of features like real-time tracking of rides, route optimization, and distance calculations.

4. Pink Feature:

- Implementation of the unique "Pink" feature that allows female drivers to match exclusively with female passengers, ensuring a safer and more comfortable carpooling experience for women.

5. Live Messaging System:

- Development of a real-time messaging system within the application to facilitate communication between drivers and passengers.

6. User Documentation:

- Creation of comprehensive user documentation, including user guides and manuals, to assist users in understanding and utilizing the application effectively.

7. Testing and Quality Assurance:

- Thorough testing of the application to ensure functionality, performance, and security.

- Identification and resolution of any bugs or issues encountered during testing.

8. Project Documentation:

- Compilation of detailed project documentation, including requirements analysis, system design, implementation details, and testing results.

9. Presentation and Final Report:

- Preparation of a final presentation to showcase the project's objectives, features, and outcomes.

- Creation of a comprehensive final report summarizing the entire project journey, including the challenges faced, solutions implemented, and future recommendations.

## System Limitations/Constraints

SL-1: Limited Coverage Area: The "HumSawar" carpool application will be limited to a specific geographical area or region, restricting its usage and availability outside of that area.

SL-2: Device Compatibility: The application may have compatibility constraints with certain older or less common mobile devices, potentially limiting access for users who do not have compatible devices.

SL-3: Internet Connectivity: The application heavily relies on internet connectivity for real-time features such as ride searching, messaging, and tracking. Users without a stable internet connection may experience limited functionality or interruptions in service.

SL-4: User Participation: The success and effectiveness of the carpooling system depend on the active participation of users as both drivers and passengers. Low user participation may limit the availability of rides and potential matches.

SL-5: Trust and Security: Building trust among users and ensuring the security of personal information shared within the application are ongoing challenges. The system should implement robust security measures and establish mechanisms to verify the identity and reliability of users.

SL-6: Regulatory and Legal Constraints: The project must comply with local regulations, laws, and data protection policies related to carpooling services, privacy, and personal data handling. Adhering to these constraints may impact certain functionalities or require additional implementation efforts.

SL-7: Resource Constraints: The availability of resources, such as server capacity and bandwidth, may impose limitations on the scalability and performance of the system, especially during peak usage periods.

SL-8: User Adoption: Encouraging users to adopt and actively engage with the "HumSawar" application may be a challenge, especially if there are existing competing carpooling platforms or alternative transportation options already established in the target region.

SL-9: Integration Dependencies: The successful implementation of the project relies on the integration of third-party services, such as Firebase and Google Maps APIs. Any changes or limitations in these external services may impact the functionality and performance of the application.

SL-10: Development Time and Resources: The project's completion within the given time frame and resource constraints may pose challenges, including managing development tasks, coordinating team efforts, and ensuring timely delivery of all required features and functionalities.

## Tools and Technologies

| Tools and technologies | Version | Rational |
| --- | --- | --- |
| React Native |  | Cross-platform framework for mobile app development |
| Firebase |  | Backend-as-a-Service (BaaS) for authentication, database, and storage |
| Google Maps API |  | Integration for maps, geolocation, and routing functionalities |
| JavaScript |  | Programming language for frontend development |
| Visual Studio Code |  | Integrated development environment (IDE) for code editing |
| Git |  | Version control system for managing source code |

APIs:

1. Firebase Authentication API - for user authentication and authorization.

2. Firebase Realtime Database API - for real-time data synchronization and storage.

3. Google Maps JavaScript API - for map visualization, geolocation, and routing services.

4. Google Places API - for place search and details.

SDKs:

1. React Native SDK - for building native mobile apps using React Native framework.

2. Firebase SDK - for integrating Firebase services into the mobile app.

3. Google Maps SDK - for integrating Google Maps services into the mobile app.

Languages:

1. JavaScript - for frontend and backend development.

2. JSX - JavaScript syntax extension for writing React components.

The selected tools, technologies, APIs, and SDKs are chosen based on their popularity, community support, and their suitability for developing a cross-platform carpool application with real-time features, data storage, and mapping capabilities.

## Relevance to Course Modules

The "HumSawar" carpool application project is relevant to various courses studied during BCS. Here's a brief explanation of the relevance to different course modules:

1. Software Engineering: The project involves the complete software development life cycle, including requirements analysis, design, implementation, testing, and maintenance. It also requires applying software engineering principles and practices to ensure the quality and reliability of the application.

2. Mobile Application Development: The project focuses on developing a mobile application using React Native, a popular framework for building cross-platform apps. It involves designing the user interface, implementing features specific to mobile devices, and optimizing the app for performance and usability on both Android and iOS platforms.

3. Database Management Systems: The project utilizes Firebase Realtime Database as the backend for storing and retrieving data related to user profiles, ride details, and messaging. Understanding concepts of data modeling, querying, and database management is important for efficiently managing the app's data.

4. Web Technologies: The project involves integrating various web technologies, such as Firebase Authentication, Google Maps API, and RESTful APIs for communication with the backend server. Knowledge of web development concepts, HTML, CSS, and JavaScript, is necessary for implementing these integrations.

5. Object-Oriented Programming: The project utilizes object-oriented programming principles and concepts for structuring the codebase, defining classes and objects, and implementing features using reusable and modular code. Understanding concepts like inheritance, encapsulation, and polymorphism is beneficial for designing and implementing the application's architecture.

6. Software Testing and Quality Assurance: The project requires performing testing at different levels, including unit testing, integration testing, and functional testing, to ensure the reliability, functionality, and performance of the application. Applying testing methodologies, writing test cases, and ensuring proper error handling are essential for delivering a high-quality product.

7. Human-Computer Interaction: The project emphasizes the design and implementation of a user-friendly interface for the carpool application. Considering usability principles, conducting user research, and applying UI/UX design techniques contribute to creating an intuitive and visually appealing user experience.

8. Software Project Management: The project involves managing the development process, setting milestones, assigning tasks, and ensuring timely delivery. Understanding project management methodologies, risk assessment, and team collaboration are crucial for successful project execution.

Overall, the "HumSawar" carpool application project encompasses various concepts and skills learned throughout the BCS program, demonstrating the practical application of theoretical knowledge in real-world software development.

# Problem Definition

This chapter focuses on defining the problem statement, proposing a solution, and outlining the objectives of the proposed system. Additionally, the scope and modules of the project will be discussed, providing clarity on the project's boundaries and functionalities.

## Problem Statement

Carpool commuters in today's busy world often face challenges in finding reliable and convenient transportation solutions. The existing problem is that traditional carpooling methods lack a centralized and efficient platform that connects commuters with compatible travel partners. This leads to time-consuming and inefficient processes of searching for rides, coordinating schedules, and ensuring safety during the commute.

To address these challenges, the HumSawar software system is being developed. The purpose of developing this application is to provide a comprehensive solution that revolutionizes the carpooling experience for commuters. The software system aims to simplify the process of finding and connecting with fellow travelers, enabling users to share rides conveniently and effectively.

The problem that HumSawar aims to solve is the lack of a reliable and user-friendly platform for carpool commuters. The software system will offer a range of features, including real-time ride matching, secure messaging, and the unique "Pink" feature that ensures the safety and comfort of female drivers and passengers. By leveraging technologies such as React Native, Firebase, and Google Maps APIs, HumSawar will streamline the process of finding compatible travel partners, reducing traffic congestion, and promoting sustainability.

## Problem Solution

The HumSawar software application aims to provide a comprehensive solution to the challenges faced by carpool commuters, as identified in the problem statement. The objectives and goals of the application are as follows:

1. Enable Convenient Ride Matching: The application will allow users to easily find and connect with compatible travel partners based on their preferences, locations, and schedules. The goal is to simplify the process of ride matching and make carpooling a convenient option for commuters.

2. Enhance Efficiency and Time-Saving: By utilizing the HumSawar application, commuters will be able to save time by finding rides efficiently and avoiding the need for extensive manual searches and coordination. The objective is to streamline the carpooling process and reduce the time spent on organizing shared rides.

3. Reduce Traffic Congestion: Through increased carpooling and efficient ride matching, the application aims to contribute to the reduction of traffic congestion. By encouraging shared rides, HumSawar seeks to optimize the utilization of vehicles, leading to fewer cars on the road and a more sustainable transportation system.

4. Promote Sustainability: By promoting shared rides and reducing the number of vehicles on the road, HumSawar aims to contribute to a greener environment and reduce carbon emissions. The application encourages sustainable transportation practices and aligns with the goals of environmental conservation.

5. Ensure Safety and Comfort: The unique "Pink" feature of HumSawar provides an additional layer of safety and comfort for female drivers and passengers. By allowing female users to exclusively match with other female passengers, the application addresses concerns regarding safety during carpooling.

6. Foster Community and Social Interaction: HumSawar aims to create a sense of community among carpool commuters by providing a platform for communication and interaction. The application's live messaging system enables users to connect, coordinate, and build relationships with fellow travelers, fostering a supportive and engaging carpooling community.

7. Optimize Travel Experiences: By facilitating efficient and reliable carpooling, HumSawar seeks to enhance the overall travel experiences of users. The application aims to make carpooling a preferred choice for commuters, providing a comfortable and enjoyable journey.

8. Provide a User-Friendly Interface: HumSawar will prioritize a user-friendly interface that is intuitive and easy to navigate. The objective is to ensure that users, regardless of their technical expertise, can easily access and utilize the features of the application.

9. Seamless Integration with External Services: The application will integrate with external services such as Firebase for real-time data synchronization and Google Maps APIs for location-based services. The goal is to provide a seamless experience for users and leverage the capabilities of these services to enhance the functionality of HumSawar.

10. Continuous Improvement and Updates: The development team will strive for continuous improvement of the application based on user feedback and emerging technologies. The goal is to regularly update and enhance HumSawar to meet the evolving needs and expectations of carpool commuters.

In summary, the HumSawar software application aims to address the identified problems by providing a user-friendly and efficient platform for carpool commuters. The objectives include convenient ride matching, time-saving, traffic congestion reduction, sustainability promotion, safety and comfort assurance, community building, optimized travel experiences, user-friendly interface, seamless integration with external services, and continuous improvement. By achieving these objectives, HumSawar seeks to revolutionize the carpooling experience and provide a comprehensive solution to the challenges faced by commuters.

## Objectives of the Proposed System

**Objectives of the HumSawar Carpool Application:**

BO-1: Increase carpooling adoption: The objective is to promote carpooling as an alternative mode of transportation by increasing the number of users who actively participate in the carpooling system. The aim is to achieve a 20% increase in carpooling adoption among commuters.

BO-2: Reduce traffic congestion: The objective is to alleviate traffic congestion on roads by encouraging shared rides through the HumSawar application. The goal is to reduce the number of vehicles on the road by 15%, resulting in smoother traffic flow and reduced commute times.

BO-3: Enhance resource utilization: The objective is to optimize the utilization of vehicles by maximizing the number of occupied seats during commuting. The aim is to achieve an average occupancy rate of at least 2.5 passengers per vehicle, resulting in reduced fuel consumption and environmental impact.

BO-4: Improve commute experience: The objective is to enhance the overall commute experience for users by providing a reliable and convenient carpooling platform. The goal is to increase user satisfaction ratings by 20% through features such as live messaging, accurate ride matching, and user-friendly interface.

BO-5: Ensure safety and security: The objective is to prioritize the safety and security of users within the carpooling system. The aim is to implement robust safety measures, including verification of user profiles and the Pink feature for female drivers and passengers, resulting in a 25% increase in user confidence in the system.

BO-6: Foster a sense of community: The objective is to create a sense of community among carpool commuters by facilitating communication and interaction. The aim is to increase user engagement through community-building features, such as ratings and reviews, shared interests, and event coordination.

BO-7: Optimize travel cost: The objective is to reduce individual travel costs for commuters by sharing the expenses of transportation. The aim is to achieve an average cost reduction of 30% per user compared to individual commuting expenses.

BO-8: Support sustainable transportation: The objective is to promote sustainable transportation practices by reducing carbon emissions and environmental impact. The aim is to decrease carbon emissions by 20% through increased carpooling and reduced vehicle miles traveled.

BO-9: Continual system improvement: The objective is to continuously enhance the HumSawar application based on user feedback and emerging technologies. The aim is to release regular updates and feature enhancements to improve the functionality, usability, and user experience of the system.

BO-10: Expand user base: The objective is to expand the user base of the HumSawar application beyond the initial target audience by targeting new demographics and geographical regions. The aim is to achieve a 25% increase in the number of registered users within the first year of operation.

In summary, the objectives of the HumSawar carpool application include increasing carpooling adoption, reducing traffic congestion, enhancing resource utilization, improving the commute experience, ensuring safety and security, fostering a sense of community, optimizing travel costs, supporting sustainable transportation, facilitating continual system improvement, and expanding the user base. Achieving these objectives will contribute to a more efficient and sustainable transportation system while providing a positive and convenient carpooling experience for users.

## Scope

The scope of HumSawar is to develop a carpooling platform that provides a safe and reliable transportation solution for commuters. The proposed project will provide a user-friendly mobile application built on React Native and integrated with Firebase and Google Maps APIs. The application's main functionalities include creating a user profile, searching for and joining carpooling groups, posting and searching for ride requests, and communicating with other users through a live messaging system.

HumSawar will have a range of features that enhance user experience, including the ability to filter searches based on various criteria such as time, distance, and gender. The Pink feature will ensure female drivers and passengers can match exclusively with other female passengers, enhancing safety and comfort during the commute. Additionally, the app will allow users to rate each other after rides, providing feedback that improves the overall experience for all users.

## Modules

### Module 1: User Registration

- Allow users to create an account with their personal information.

- Validate and authenticate user registration details.

### Module 2: User Profile Management

- Enable users to create and manage their profiles.

- Provide options to add profile pictures, contact information, and preferences.

### Module 3: Ride Search and Booking

- Allow users to search for available rides based on their location, destination, and time preferences.

- Display a list of suitable ride options with relevant details such as driver information, vehicle details, and ratings.

- Enable users to book rides and receive confirmation notifications.

### Module 4: Ride Posting

- Allow users to post ride offers, specifying their location, destination, departure time, and available seats. - Provide options to set preferences such as gender preferences for matching passengers.

- In-app Messaging:

- Implement a real-time messaging system to facilitate communication between drivers and passengers.

- Allow users to send and receive messages related to ride details, pickup locations, and other relevant information.

### Module 5: User Management

- Allow the admin to manage user accounts, including registration approval and user profile verification.

- Provide options to block or suspend users for violating platform guidelines.

### Module 6: Real-time Chat

- Implement a real-time chat feature that allows users to communicate with each other within the app.

- Enable users to send text messages, emojis, and attachments to other users.

.

# Requirement Analysis

The requirement analysis chapter is a crucial phase in software development that focuses on understanding and documenting the needs and expectations of the stakeholders. It involves gathering, analyzing, and documenting the functional and non-functional requirements of the proposed system. This chapter provides an overview of the requirement analysis process and outlines the specific sections that will be covered in detail.

## User classes and characteristics

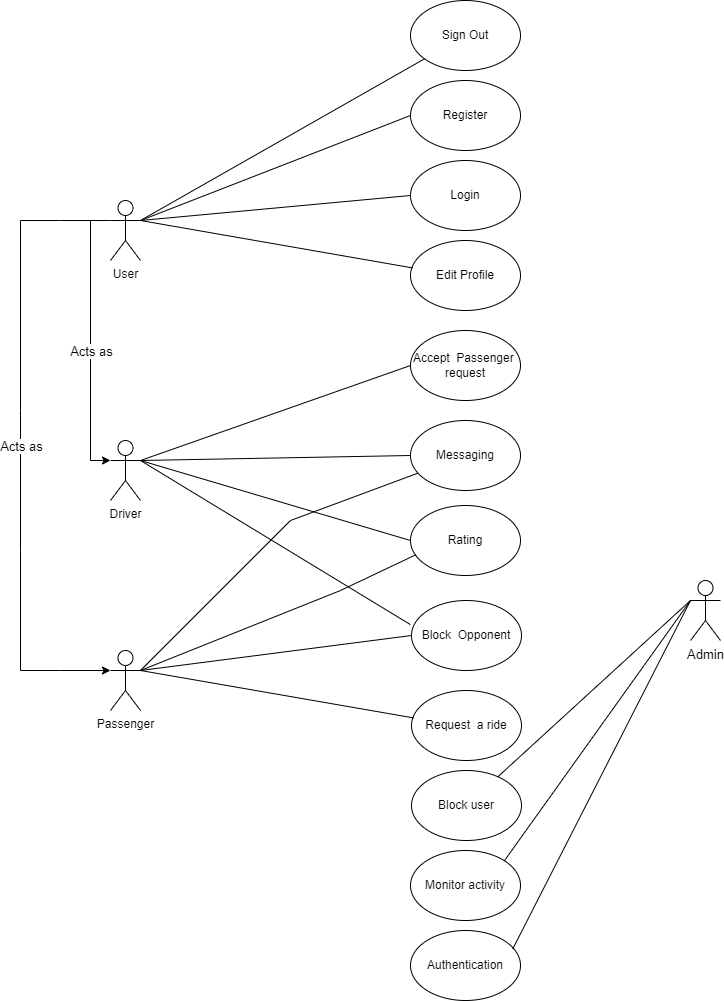
User Classes and Characteristics:

**Table A-1 Shows user classes and character**

| **User class** | **Description** |
| --- | --- |
| **Carpool Commuter** | A Carpool Commuter is a user who wants to share rides with fellow travelers using the HumSawar carpooling platform.  - Characteristics:  - Can be anyone who regularly commutes to work or other destinations.  - Interested in finding reliable and convenient transportation solutions.  - Seeks to reduce travel costs, save time, and contribute to sustainability by sharing rides.  - May have preferences regarding ride sharing arrangements such as gender preferences or preferences for specific routes. |
| **Driver** | A Driver is a user who offers their vehicle for carpooling through the HumSawar platform.  - Characteristics:  - Has a private vehicle and is willing to share rides with others.  - Interested in reducing travel costs, optimizing vehicle usage, and contributing to sustainability.  - Can set their own preferences for the number of passengers, pick-up and drop-off locations, and schedule. |
| **Admin/Platform Manager** | The Menu Manager is a cafeteria employee who establishes and maintains daily menus of the food items available from the cafeteria. Some menu items may not be available for delivery. The Menu Manager will also define the cafeteria’s daily specials. The Menu Manager will need to edit existing menus periodically. |

## Requirement Identifying Technique

**Use Case Digram:**



**3.7.1 Detail Use-case**

### Login In

| Use Case ID | UC1 |
| --- | --- |
| Actor(s) | User |
| Description | User Log In |
| Preconditions | The user should be signed up. |
| Postconditions | User will be navigated to profile. |
| Precedence | Mandatory |
| Normal flow of event | 1. The user Opens the Hum-Sawar App.  2.. The user enters their phone number and otp   1. The user clicks the log-in button |

### Sign Out

| Use Case ID | UC2 |
| --- | --- |
| Actor(s) | User |
| Description | User Log Out |
| Preconditions | The user shall be able to log in to the system. |
| Postconditions | User will be able to leave the system. |
| Precedence | is Not mandatory |
| Normal flow of event | 1. The user presses the log-out button. 2. The Application’s main page will be loaded. |

### Add Ride

| Use Case ID | UC3 |
| --- | --- |
| Actor(s) | User |
| Description | User shall be able to add a route from the map. |
| Preconditions | The user shall be able to sign in to the system. |
| Postconditions | User shall retrieve transportation requests from the other users. |
| Precedence | Mandatory |
| Normal flow of event | 1. The user shall enter her or his profile page. 2. User shall press add new route button. 3. Add route page will be loaded. 4. User enters departure time, available seats, and iteration. 5. User draws a route on the map panel. |

### 

### Search Ride

| Use Case ID | UC4 |
| --- | --- |
| Actor(s) | User |
| Description | Usr shall be able to search the route. |
| Preconditions | The user shall sign in to the system. |
| Postconditions | User will be able to select the route from the available route list. |
| Precedence | No mandatory |
| Normal flow of event | 1. User fills “from” input field. 2. User fills “to” input field. 3. User presses the search button. |
| Alternative Flow(s) | Flow 1:   1. User forgets to fill “from” or “to” input field. 2. The related warning message is shown to the user to fill the input fields properly.   Flow 2:   1. User fills the input fields properly. 2. The available routes will be listed. |

**Request Route**

| Use Case ID | UC5 |
| --- | --- |
| Actor(s) | User |
| Description | User shall be able to request a route. |
| Preconditions | The user shall search the route. |
| Postconditions | User will be able to contact the driver who owns the route. |
| Precedence | No mandatory |
| Normal flow of event | 1. The user selects the Request Route option. 2. The system sends the driver an e-mail and SMS notification. |

### 

### 

### Send Message

| Use Case ID | UC8 |
| --- | --- |
| Actor(s) | User |
| Description | User shall be able to send messages through the system. |
| Preconditions | The user shall sign in to the system. |
| Postconditions | Users will be able to communicate with each other. |
| Precedence | No mandatory |
| Normal flow of event | 1. The user enters the profile page of the user who is intended to be communicated. 2. User presses the send message button. 3. The message page will be loaded. 4. User types the content of the message. 5. User presses the send button to send the message content. 6. The message content will be stored and viewed in the message panel. |

## Functional Requirements

### Functional Requirement 1

### Login

Users will have to log into the system in order to use all the feature of application. When user prove his/her login credentials, Application will vcopare it with excisting credentials in the database and

allow user to login if user have registered before. If User’s credentials does not exist application will display an error saying “You are not registered.Please SignIn first”.

| **Identifier** | FR-1 |
| --- | --- |
| **Title** | Login |
| **Requirement** | In order to use the application’s features, both Driver and passengers must login by typing username and password.  If written in a **user perspective**  The users will be navigated to the home screen if provided credentials match with the registered credentials.  If written in a **system perspective**  The provided credentials will be validated by comparing them with existing credentials. |
| **Rationale** | This is necessary to validate users if they are registered or not. |
| **Dependencies** |  |
| **Priority** | High |

### Functional Requirement 2

### Modify Profile

After the user has logged in, they can modify their user profile. For example, if they want to change profile picture, password etc..When user changes his/her information in the profile, application will update this information in database.

| **Identifier** | FR-2 |
| --- | --- |
| **Title** | Modify Profile Information |
| **Requirement** | If written in a **user perspective**  Users shall be able to edit their personal information such as username, phone number, email etc..  This is required by the drivers in order to fill information of their vehicle’s details such as plate number, color, model and condition.  If written in a **system perspective**  Information will be stored in a database link to each user respectively. |
| **Rationale** | To store and display information in the user's profile. |
| **Dependencies** | FR-1 |
| **Priority** | High |

### Functional Requirement 3

### Create Rides

This is one of the main functionalities of the application where user can create a ride or we can say make a cluster of people who want to carpool in specified location, fare and priorities.

| **Identifier** | FR-3 |
| --- | --- |
| **Title** | Create new rides |
| **Requirement** | If written in a **user perspective**  The driver will create a new trip to be displayed when passengers search for trips. This requires all information about origin, destination, numbers of available seats etc…  The driver posts this information in order to attract passengers. A user can promote his newly constructed journey on social media after creating it in order to locate passengers to drive with.  If written in a **system perspective**  When a driver posts this information, Application will make it live for all users having the same origin and destination. |
| **Rationale** | This is one of the main functionalities of the application. |
| **Dependencies** | The following requirement (ID # FR-1) |
| **Priority** | High |

### Functional Requirement 4

### Search Rides

Users who do not own a vehicle will search for a ride by providing origin and destination location along with priorities of the ride and rides with same descriptions will be displayed and passenger can choose ride according to their requirements or priorities.

| **Identifier** | FR-4 |
| --- | --- |
| **Title** | Search for rides |
| **Requirement** | If written in a **user perspective**  When a passenger needs to find a driver for a specific trip, he or she can  utilize a search form that asks for the destination, origin, departure date/time, and other information. Additionally, he can define his travel preferences. When he locates a route that suits him, he may quickly reserve a seat by selecting the traveling partner, which will alert the driver that a passenger has reserved.  If written from a **system perspective**  When a passenger asks for a trip the application fetches information about drivers with asked preference and displays it on user’s screen. |
| **Rationale** | To allow users to create and select rides for their trip. |
| **Dependencies** | FR-1 |
| **Priority** | High |

### Functional Requirement 5

### Send Request

When user finds the ride according to starting and destination location, request can be sent from the user towards the driver.

| **Identifier** | FR-5 |
| --- | --- |
| **Title** | Request for ride |
| **Requirement** | If written in a **user perspective**  User need to send request to the driver who can either accept or decline hte request  If written from a **system perspective**  A request will be pushed into driver’s notification screen by using driver’s uid present in the ride fetched from the database |
| **Rationale** | To allow user request and initiate ride |
| **Dependencies** | FR-4 |
| **Priority** | High |

### Functional Requirement 6

### Accept Request

When driver receives the ride request, he can accept or decline the request.

| **Identifier** | FR-6 |
| --- | --- |
| **Title** | Accept ride requset |
| **Requirement** | If written in a **user perspective**  Driver will need to accept ride request in order to allow user share ride with him.  If written from a **system perspective**  A new ride node will be created in database containing id’s of both user and driver. |
| **Rationale** | To allow user request and initiate ride |
| **Dependencies** | FR-5 |
| **Priority** | High |

### 

### Functional Requirement 7

### Decline Request

When driver receives the ride request, he can accept or decline the request.

| **Identifier** | FR-7 |
| --- | --- |
| **Title** | Accept ride requset |
| **Requirement** | If written in a **user perspective**  Driver can decline the request if he find it unsuitable to match.  If written from a **system perspective**  Sender will get notification that your request has been declined |
| **Rationale** | To allow user request and initiate ride |
| **Dependencies** | FR-5 |
| **Priority** | High |

### Functional Requirement 8

### ChatBox

When driver has accepted passenger’s request, a chat box will be available for them to discuss about time of arrival and to negotiate accordingly.

| **Identifier** | FR-8 |
| --- | --- |
| **Title** | Chat between drivers and passengers. |
| **Requirement** | If written in a **user perspective**  Users can utilize the chat option in the application in order to decide meeting point, and priorities about the trip.  If written in a **system perspective** |
| **Rationale** | To achieve Communication between Driver and passenger. |
| **Dependencies** | N/A |
| **Priority** | Medium |

### 

## Non-Functional Requirements

### Reliability

The application should be available to users at all times, with minimal downtime for maintenance or upgrades. It should have a high uptime percentage, aiming for at least 99.9% availability to ensure users can access the carpooling services.

### Usability

The application will have a very easy-to-use interface for the users. They can select and search for routes using google maps and communication between the passenger and driver will add easiness to using the application. Users can see their previous rides and reselect the previousl used locations

### Performance

The application should be able to handle a large number of users and concurrent requests without significant performance degradation. It should be designed to scale and optimize resource usage to maintain responsiveness even during peak usage periods.

### Security

The personal information of all the users will be protected. Furthermore, the admin will validate the driver’s documents . Only admin has access to manipulate the database so that’s why user data is secure. Moreover, the pink feature in application will help female users to be more secure and comfortable.

## External Interface Requirements

This section provides information to ensure that the system will communicate properly with users and with external hardware or software elements. A complex system with multiple subcomponents should create a separate interface specification or system architecture specification. The interface documentation could incorporate material from other documents by reference. For instance, it could point to a hardware device manual that lists the error codes that the device could send to the software.

### User Interfaces Requirements

Describe the logical characteristics of each user interface that the system needs. Some possible items to include are

* References to GUI standards or product family style guides that are to be followed.
* Standards for fonts, icons, button labels, images, color schemes, field tabbing sequences, commonly used controls, and the like.
* Screen layout or resolution constraints.
* Standard buttons, functions, or navigation links that will appear on every screen, such as a help button.
* Shortcut keys.
* Message display conventions.
* Layout standards to facilitate software localization.
* Accommodations for visually impaired users.

Document the user interface design details, such as specific dialog box layouts, in a separate user interface specification, not in the SRS. Including screen mock-ups in the SRS to communicate another view of the requirements is helpful but make it clear that the mock-ups are not the committed screen designs. If the SRS is specifying an enhancement to an existing system, it sometimes makes sense to include screen displays exactly as they are to be implemented. The developers are already constrained by the current reality of the existing system, so it's possible to know up front just what the modified, and perhaps the new, screens should look like.

### Software interfaces

Connections between HumSawar Carpool Application and other software components:

SI-1: Firebase Realtime Database

SI-1.1: The HumSawar Carpool Application shall integrate with Firebase Realtime Database to store and retrieve user data, carpooling requests, driver/passenger preferences, and other relevant information.

SI-2: Google Maps APIs

SI-2.1: The HumSawar Carpool Application shall utilize Google Maps APIs to provide mapping and geolocation functionalities, including real-time traffic information, route calculation, and navigation assistance for drivers and passengers.

SI-3: React Native Framework

SI-3.1: The HumSawar Carpool Application shall be developed using React Native as the primary framework for cross-platform mobile app development.

SI-4: Firebase Authentication

SI-4.1: The HumSawar Carpool Application shall integrate with Firebase Authentication to provide secure user authentication and authorization features, allowing users to create accounts, log in, and access their personal profiles.

### Hardware interfaces

Describe the characteristics of each interface between the software components and hardware components, if any, of the system. This description might include the supported device types, the data and control interactions between the software and the hardware, and the communication protocols to be used. For more about specifying requirements for systems containing hardware, see Chapter 26, “Embedded and other real-time systems projects.”

### Communications interfaces

State the requirements for any communication functions the product will use, including email, web browser, network protocols, and electronic forms.

Example:

*CI-1: The COS shall send an email or text message (based on user account settings) to the Patron to confirm acceptance of an order, price, and delivery instructions.*

# Design and Architecture

The following parts of Software Design Description (SDD) report should be included in this chapter.

## Architectural Design

Develop a modular program structure and explain the relationships between the modules to achieve the complete functionality of the system. This is a high-level overview of how the system’s modules collaborate with each other in order to achieve the desired functionality.

Don’t go into too much detail about the individual subsystems. The main purpose is to gain a general understanding of how and why the system was decomposed, and how the individual parts work together.

Provide a diagram showing the major subsystems and their connections.

* In initial design stage create Box and Line Diagram for simpler representation of the systems
* After finalizing architecture style/pattern diagram (MVC, Client-Server, Layered, Multi-tiered) create a detailed mapping modules/components to each part of the architecture

To view example of box and line diagram and architecture styles, see Appendix B.

## Design Models

Create design models as are applicable to your system. Provide detailed descriptions with each of the models that you add. Also ensure visibility of all diagrams.

***Design Models for Object Oriented Development Approach***

The applicable models for the project using object oriented development approach may include:

* Activity Diagram
* Class Diagram
* Sequence Diagram
* State Transition Diagram (for the projects which include event handling and backend processes)

***Design Models for Procedural Approach***

The applicable models for the project using procedural approach may include:

* Activity Diagram
* Data Flow Diagram (data flow diagram should be extended to 2-3 levels. It should clearly list all processes, their sources/sinks and data stores.)
* State Transition Diagram (for the projects which include event handling and backend processes)

To view examples of all above models, see Appendix C

## Data Design

The following are the major data entities and their organization:

1. chat: This collection stores chat messages between users. Each message is identified by a unique ID. The data structure includes fields like "\_id" (message ID), "text" (message content), and "user" (user details).

2. rides: This collection stores information about rides offered by users. Each ride is identified by a unique ride ID. The data structure includes fields like "date" (ride date), "from" (origin location), "fromLat" and "fromLng" (latitude and longitude of the origin), "id" (user ID), "price" (ride price), "seat" (available seats), "to" (destination location), "toLat" and "toLng" (latitude and longitude of the destination), and "token" (security token).

3. users: This collection stores user information. Each user is identified by a unique user ID. The data structure includes fields like "age" (user age), "cnic" (user CNIC number), "email" (user email address), "gender" (user gender), and "name" (user name).

### Data Dictionary

Alphabetically list the system entities or major data along with their types and descriptions. If you provided a functional description, list all the functions and function parameters. If you provided an OO description, list the objects and its attributes, methods and method parameters.

## Human Interface Design

### Screen Images

Display screenshots showing the interface from the user’s perspective. These can be hand-drawn, or you can use an automated drawing tool. Just make them as accurate as possible. (Graph paper works well.)

### Screen Objects and Actions

A discussion of screen objects and actions associated with those objects.

# Implementation

This chapter will discuss implementation details of the project. You will not put your source code here, however, are required to write the core modules functionalities in pseudocode form (Following sections are required in this chapter).

Note: You are required to follow proper coding standard to write your source code. For guidelines, **General Coding Standards & Guidelines** are provided in Appendix D.

## Algorithm

Mention the algorithm(s) used in your project to get the work done with regards to major modules. Provide a pseudocode explanation regarding the functioning of the core features. Be sure to use the correct syntax and semantics for algorithm representations. Following are few examples of algorithms/pseudocode:

**Table 4 Example of Algorithm**

| **Algorithm 1 MHCF co-authorsBasedClustering** | | |
| --- | --- | --- |
| **Input:** n groups Gn where each group has set of papers (pr) | | |
| **Output:** Set of system generated clusters/groups Gn | | |
| 1: merge ← true  2: Flag ← false  3: **While**(merge==’true’) **do:**  4: merge ← false  5: **for** i in range (0: len(G)-1):  6: **for** j in range (i+1: len(G)):  7: **if** (similarCoauthors(GiLco-authors, GjLco-authors) == true) **then**  8: Flag ← true  9: **Else** (checkNameFragments(GiLco-authors, GjLco-authors) == true) **then**  10: Flag ← true  11: **if** (Flag == true) **then**  12: Gi ← Gi U Gj  13: G ← G.pop(j)  14: merge ← true  15: **end if**  16: **end for**  17: **end for**  18: **end while** | | |
| **Algorithm 1.1 checkNameFragments** | | |
| **Input:** two lists GiLco-authors, GjLco-authors where each list has co-authors | | |
| **Output:** Boolean value | | |
| 1: Flag ← false  2: Count ← 0  3: **foreach co-author1 in GiLco-authors:**  4: coauthor1Fragments ← co-author1.split(‘ ’)  5: **foreach co-author2 in GjLco-authors:**  6: coauthor2Fragments ← co-author2.split(‘ ’) //split author name into name fragments  7: **if** (len(coauthor1Fragments) == 3 **and** len(coauthor2Fragments) == 3) **then** //both authors have three name fragments  8: **if** (len(coauthor1Fragments[0]) ≥ 3 and len(coauthor1Fragments[0]) ≥ 3) **then**  9: //both authors first name have more than three characters  10: **if**(coauthor1Fragments[0] == coauthor2Fragments[0]) and (coauthor1Fragments[2] == coauthor1Fragments[2])) **then**  11: //both authors have same full first and last name  12: **if** ((coauthor1Fragments[1] == coauthor2Fragments[1]) or (coauthor1Fragments[1][0] == coauthor2Fragments[1][0])) **then**  13: //both authors have same middle full name or same first character of middle name  14: Count++  15: **end if**  16: **elseif** ((coauthor1Fragments[0][0] == coauthor2Fragments[0][0] and coauthor1Fragments[0][1] == coauthor2Fragments[0][1] and coauthor1Fragments[0][2] == coauthor2Fragments[0][2]) and (coauthor1Fragments[2] == coauthor1Fragments[2])) **then**  17: //both authors have same first three characters of first name and full last name  18: **if** ((coauthor1Fragments[1] == coauthor2Fragments[1]) or (coauthor1Fragments[1][0] == coauthor2Fragments[1][0])) **then**  19: //both authors have same middle full name or same first character of middle name  20: Count++  21: **endif**  22: **end elseif**  23: **elseif** (len(coauthor1Fragments) > 3 and len(coauthor2Fragments) > 3)) **then** //both have more than three name fragments  24: **if** ((coauthor1Fragments[0][0] == coauthor2Fragments[0][0] and coauthor1Fragments[0][1] == coauthor2Fragments[0][1] and coauthor1Fragments[0][2] == coauthor2Fragments[0][2]) and ((coauthor1Fragments[len(coauthor1Fragments)-1] == coauthor2Fragments[len(coauthor2Fragments)-1]) or (coauthor1Fragments[len(coauthor1Fragments)-1][0] == coauthor2Fragments[len(coauthor2Fragments)-1][0]))) **then**  //both have same first three characters of first name and either full last name or first character of last name  25: **if** (coauthor1Fragments[1][0] == coauthor2Fragments[1][0]) **then** //both have same first character of their second name  26: count++  27: **end if**  28: **end if**  29: **end elseif**  30: **end foreach**  31: **end foreach**  32: **if** (count ≥ 1) **then //**number ofsimilar co-authors excluding author in question  33: Flag ← true  34: **endif**  35: **return** Flag | | |
| **Algorithm 2 MHCF titleBasedClustering** | | **Algorithm 2.1 similarTitles** |
| **Input:** n groups Gn where each group has set of papers pr | | **Input: string literals of paper title, threshold\_value** |
| **Output:** Set of system generated clusters/groups **Gn** | | **Output: Boolean value** |
| 1: merge ← true, threshold-title ← threshold ← 0.85  2: **While** (merge ==’true’) **do:**  3: merge ← false  4: **for** i in range (0: len(G)-1):  5: **for** j in range (i+1: len(G)):  6: coathr-count = check-coauthors(Gi , Gj)  7: if coathr-count ≥ 1 **then**  8: threshold-title ← threshold – 0.35  9: **if** similarTitles(Gi\_title, Gj\_title, threshold-title) **then**  10: Gi ← Gi U Gj  11: G ← G.pop(j)  12: merge ← true  13: **end if**  14: **end for**  15: **end for**  16: **end while** | | 7.1: vectori, vectorj ← Research2Vec (titlei, titlej)  7.2: score = 1 – spatial.distance.cosine(vectori, vectorj)  7.3: **if** score ≥ threshold:  7.4: return true  7.5: **else**  7.6: return false  7.7: **end if** |
| **Algorithm 2.2 check-coauthors** | | |
| **Input:** Two groups co-authors list Gi and Gj  **Output:** Count of common co-authors (Number Value)  6.1: common = 0  6.2: if **set** (Gi) & **set** (Gj) **then** //if there exists a common co-author.  6.3: common ← **set** (Gi) & **set** (Gj)  6.4: return len(common) | | |
| **Algorithm 3 MHCF author\_affiliationBasedClustering** | **Algorithm 3.1 cosinesimilarity** | |
| **Input:** n groups Gn where each group has set of papers pr | **Input: List of affiliations/emails within the G, threshold** | |
| **Output:** Set of system generated clusters/groups Gn | **Output: Boolean value** | |
| 1: merge ← true  2: **While**(merge==’true’) **do:**  3: merge ← false  4: **for** i in range (0: len(G)-1):  5: **for** j in range (i+1: len(G)):  6: **if** cosinesimilarity (GiLaffiliation, GjLaffiliation) **then**  7: Gi ← Gi U Gj  8: G ← G.pop(j)  9: merge ← true  10: **end if**  11: **end for**  12: **end for**  13: **end while** | 6.1: **for** m in range (0: len(GiLaff) - 1):  6.2 vectori ← cosineSim (GiLaff[m])  6.3: **for** k in range (m+1: len(GjLaff)):  6.4: vectorj ← cosineSim (GjLaff[k])  6.5: score = 1 – spatial.distance.cosine(vectori, vectorj)  6.6: **if** score ≥ threshold:  6.7: return true  6.8: **end if**  6.9: **end for**  6.10: **end for**  6.11: return false | |

## External APIs/SDKs

Table: Third-Party APIs/SDKs Used in the Project

| **Name of API/SDK** | **Description** | **Purpose of Usage** | **Example Usage in the Project** |
| --- | --- | --- | --- |
| Firebase Authentication API | User authentication and authorization | Managing user sign-in and sign-up | `onAuthStateChanged` for user authentication |
| Firebase Realtime Database API | Real-time data synchronization and storage | Storing and retrieving application data | `getDoc`, `doc` for retrieving user data |
| Google Maps JavaScript API | Map visualization, geolocation, routing | Displaying maps and location services | Used for map rendering and route calculation |
| Google Places API | Place search and details | Retrieving place information | Not specified in the provided code |
| React Native SDK | Framework for building native apps | Mobile app development | React Native components and features |
| Firebase SDK | Integration with Firebase services | Integrating Firebase into the app | Firebase authentication and database operations |
| Google Maps SDK | Integration with Google Maps services | Integrating Google Maps into the app | Map rendering and geolocation services in the app |

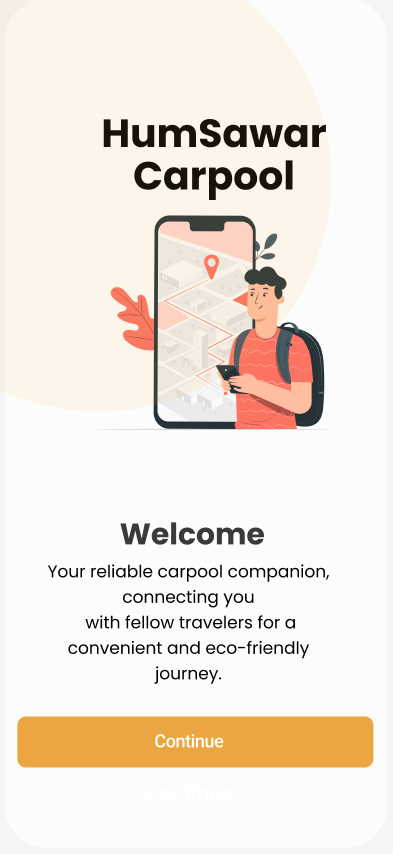
## User Interface

Details about user interface with descriptions. Provide the User Interface for each sub-system (such as Mobile App, Web App, Client App, Admin App). Provide description of each User Interface explaining the details.

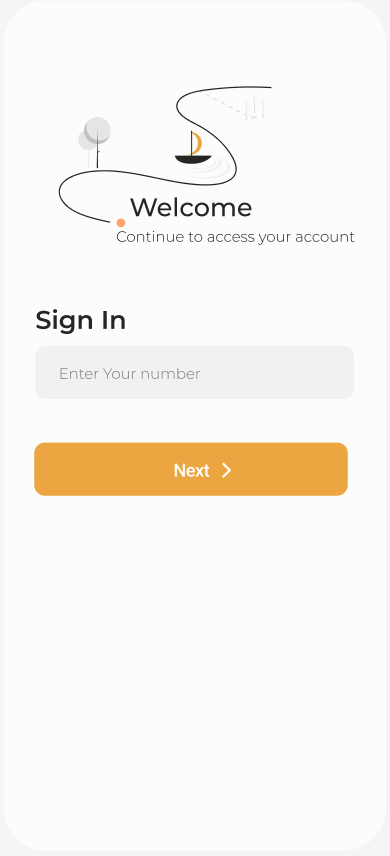
When inserting User Interfaces, use appropriate size of the image, for example, for mobile app, 2-4 screens can be placed on a single page.

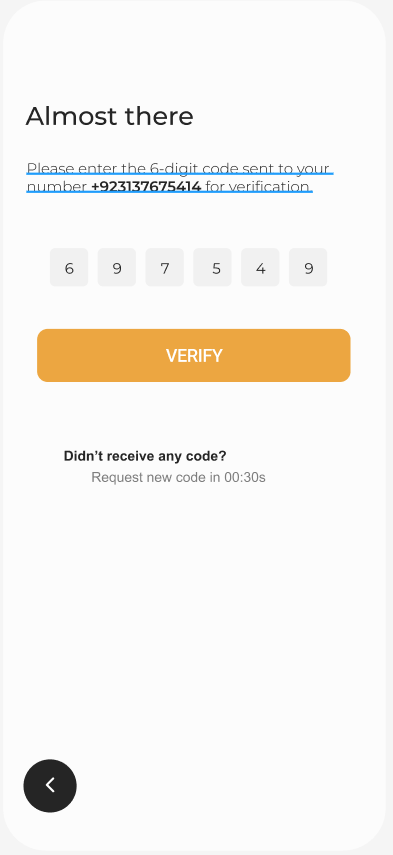
Following are few examples of User Interfaces:

### Welcome Screen

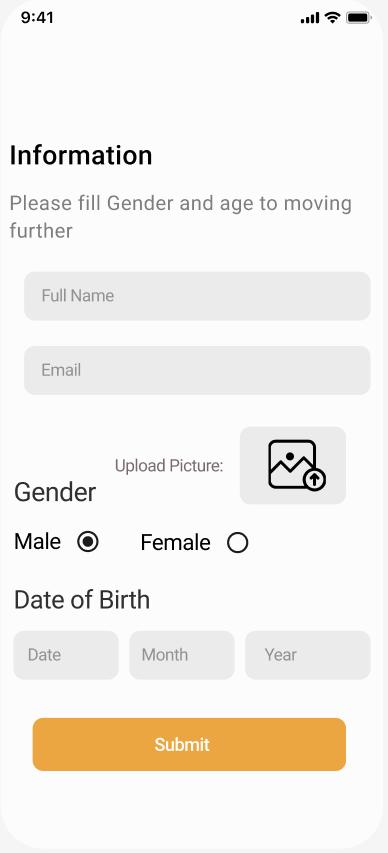


**1.3.2 Login Screen**

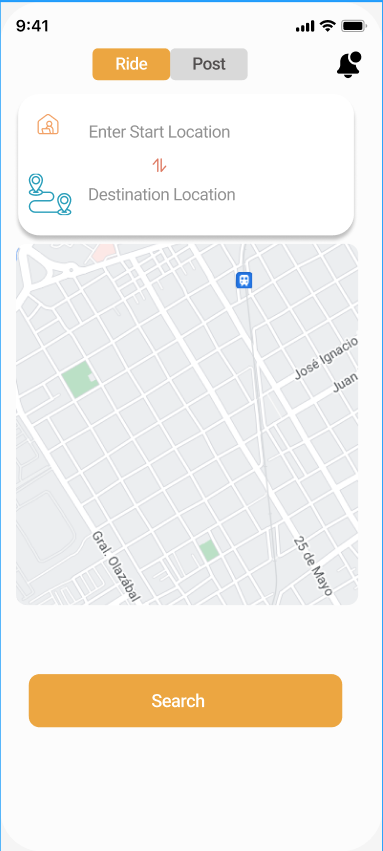
**1.3.3 Verification Screen**

****

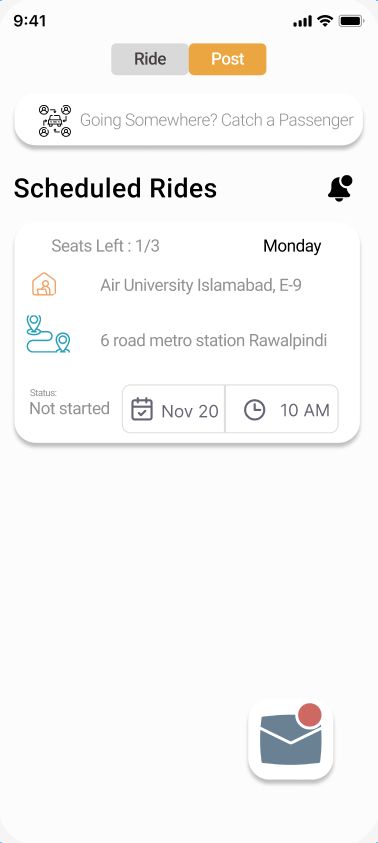
**1.3.4 Basic Information**

****

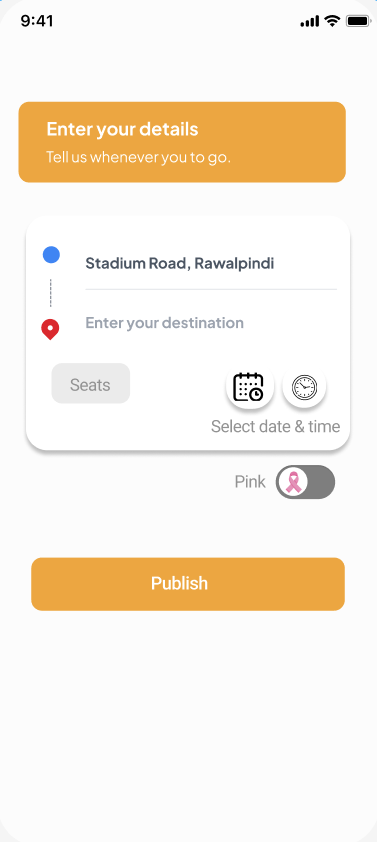
**1.3.4 Ride (user side):**

****

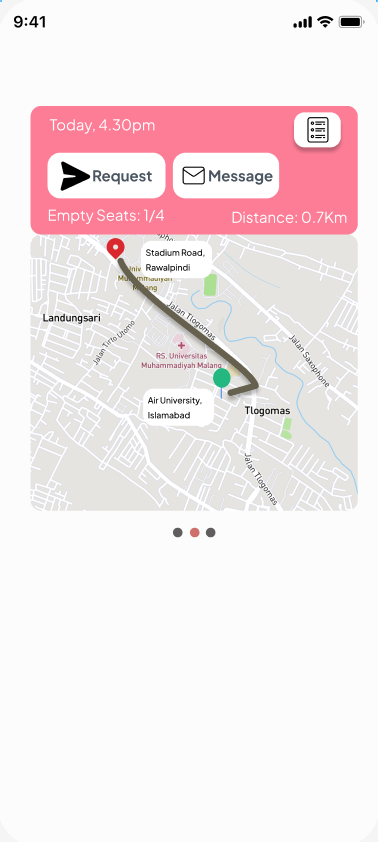
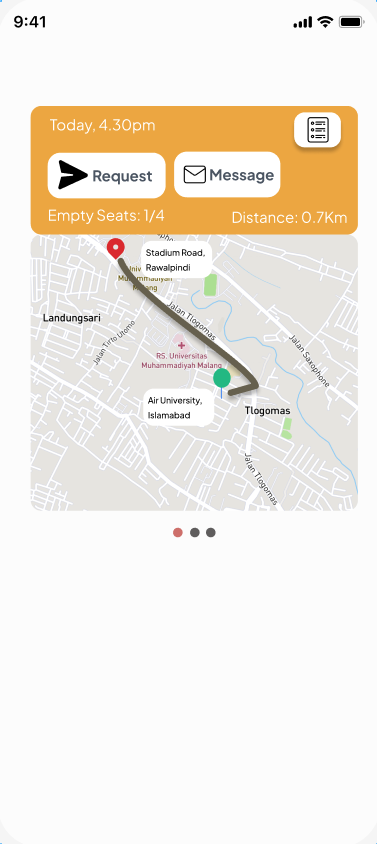
**1.3.4 Post (Driver Side):**

****

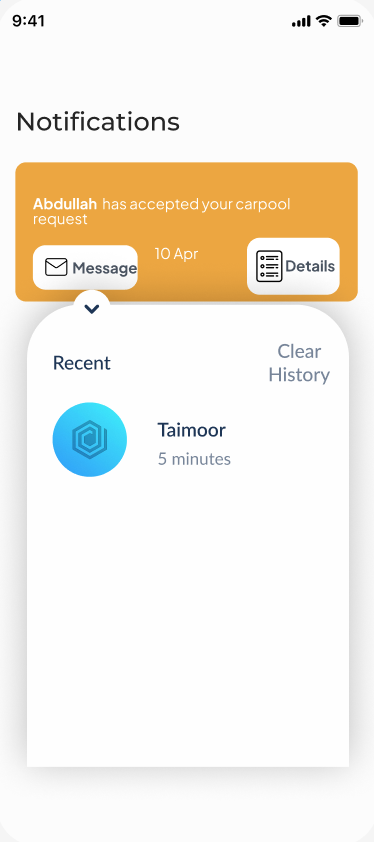
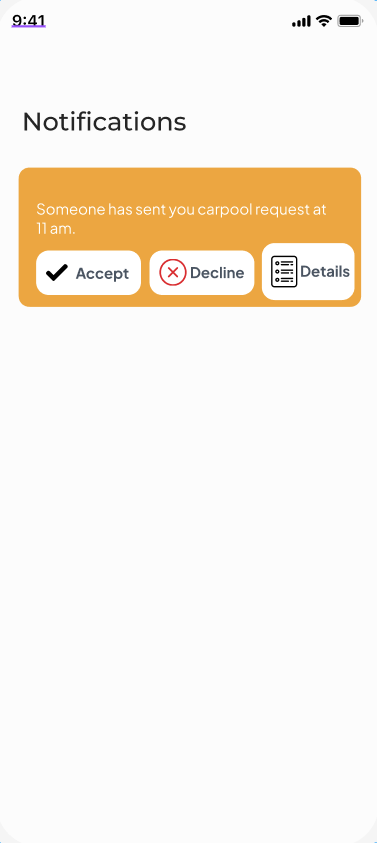
**1.3.4 Create Ride screen:**

****

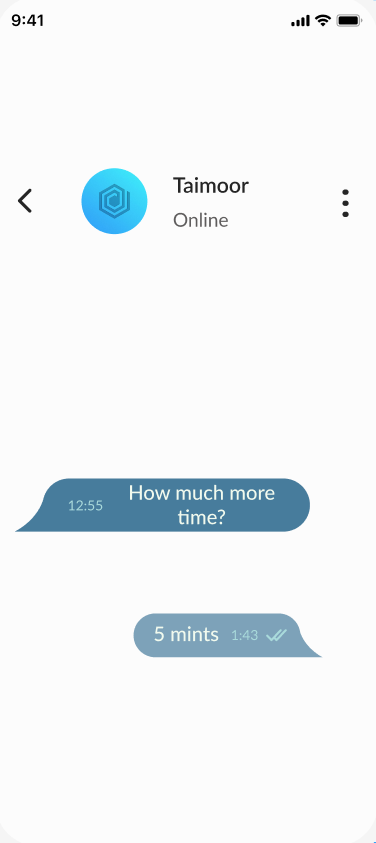
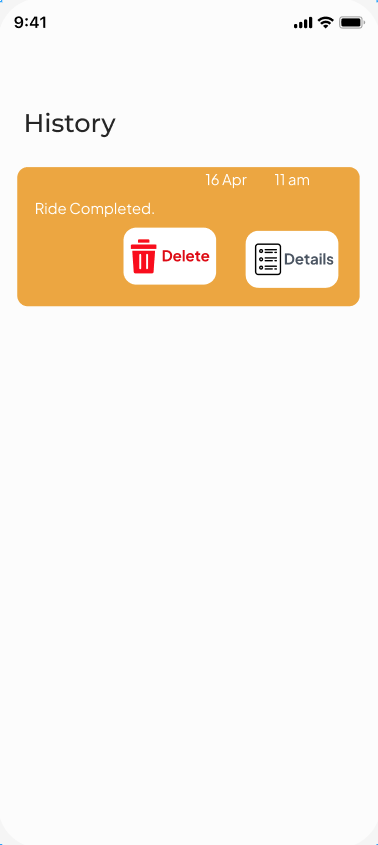
**1.3.5 Available rides: 1.3.6 Available pink rides:**

****

**1.3.7 Request Notification: 1.3.8 Request accept notification:**

****

**1.3.9 History: 1.3.10 Chat Screen**

****

# Testing and Evaluation

Once the system has been successfully developed, testing has to be performed to ensure that the system working as intended. This is also to check that the system meets the requirements stated earlier. Besides that, system testing will help in finding the errors that may be hidden from the user. The testing must be completed before it is deployed for use.

There are few types of testing which includes the unit testing, functional testing and integration testing.

You are *required* to perform each of these in-depth to ensure system quality.

## Unit Testing

It’s a level of software testing where individual units of a software/component are tested. The purpose is to validate that each unit of the software performs as designed.

**Unit Testing 1:** Login as Patient with valid and invalid credentials

**Testing Objective:** To ensure the login form is working correctly with valid and invalid credentials/inputs.

| **No.** | **Test case/Test script** | **Attribute and value** | **Expected result** | **Result** |
| --- | --- | --- | --- | --- |
| 1 | Check the email field of login to validate that it takes proper email | Email: [abc@gmail.com](mailto:abc@gmail.com) | Validates email address and moves cursor to next textbox | Pass |
| 2 | Check the email field of login to validate that it displays error message. | Email: [abc.gmail.com](mailto:abc@gmail.com) | Highlights field and displays error message | Pass |

## Functional Testing

The functional testing will take place after the unit testing. In this functional testing, the functionality of each of the module is tested. This is to ensure that the system produced meets the specifications and requirements.

**Functional Testing 1:** Login with different roles (Management, Patient, Doctor)

**Objective**: To ensure that the correct page with the correct navigation bar is loaded.

| **No.** | **Test case/Test script** | **Attribute and value** | **Expected result** | **Actual result** | **Result** |
| --- | --- | --- | --- | --- | --- |
| 1. | Login as a ‘Management’ member. | Username: (correct username M003)  Password:  (correct password 1234) | Main page for the Management is loaded with the Management navigation bar. | Logged in and redirected to management main page. | Pass |
| 2. | Login as a ‘Doctor’ member. | Username: D003  Password:  1234 | Main page for the Doctor is loaded with the doctor navigation bar. | Login failed – invalid credentials error | Fail |

## Business Rules Testing

Decision table based testing technique is used to test business rules. The business rules were defined in FRs and Use Cases

Decision based testing uses a systematic approach where input and outputs are provided in tabular form. It is a precise and compact way to model complicated logic. The table contains conditions and actions are used for test cases where conditions as inputs and actions as outputs.

Detailed example is as given in Appendix E.

## Integration Testing

Integration tests assess whether a set of classes that must work together do so without error. They

ensure that the interfaces and linkages between different parts of the system work properly. At this point, the classes have passed their individual unit tests, so the focus now is on the flow of control among the classes and on the data exchanged among them. Integration testing follows the same general procedures as unit testing: The tester develops a test plan that has a series of tests, which, in turn, have a test. Integration testing is often done by a set of programmers and/or systems analysts.

**Integration Testing 1:** Scheduling Patient Appointment

**Testing Objective:** To ensure the scheduling is being done correctly and *the* ***interface*** *between* module ‘Patient/Doctor Management’ and module ‘Appointment/Scheduling’ *is running correctly*.

| **No.** | **Test case/Test script** | **Attribute and value** | **Expected result** | **Actual result** | **Result** |
| --- | --- | --- | --- | --- | --- |
| 1. | Make Appointment | Doctor schedule, patient preferred date and time | Successfully create doctor-patient appointment showing date and time of appointment. | Appointment created successfully | Pass |
| 2. | Change Appointment booking | Select Date and time | Final date and time of appointment will be shown. | Appointment time changed successfully | Pass |

# Conclusion and Future Work

This chapter concludes the project and highlights future work.

## Conclusion

## Future Work

# References

List any documents or other resources to which this SRS refers, if any. These might include user interface style guides, standards, system requirements specifications, interface specifications, or the SRS for a related product. The following are a few examples of different resources.

**Book**

Author(s). Book title. Location: Publishing company, year, pp.

Example:

W.K. Chen. Linear Networks and Systems. Belmont, CA: Wadsworth, 1993, pp. 123-35.

**Article in a Journal**

Author(s). “Article title”. Journal title, vol., pp, date.

Example:

G. Pevere. “Infrared Nation.” The International Journal of Infrared Design, vol. 33, pp. 56-99, Jan. 1979.

**Articles from Conference Proceedings (published)**

Author(s). “Article title.” Conference proceedings, year, pp.

Example:

D.B. Payne and H.G. Gunhold. “Digital sundials and broadband technology,” in Proc. IOOC-ECOC, 1986, pp. 557-998.

**World Wide Web**

Author(s)\*. “Title.” Internet: complete URL, date updated\* [date accessed].

M. Duncan. “Engineering Concepts on Ice. Internet: www.iceengg.edu/staff.html, Oct. 25, 2000 [Nov. 29, 2003].

**Appendix A**

**Example: Context Diagram**

The Cafeteria Ordering System is a new software system that replaces the current manual and telephone processes for ordering and picking up meals in the Process Impact cafeteria. The context diagram in Figure 1 illustrates the external entities and system interfaces.

Diagram

Description automatically generated

**Figure A-5: Context diagram of the Cafeteria Ordering System.**

**Example: User Classes and Characteristics**

**Table A-1 Shows user classes and character**

| **User class** | **Description** |
| --- | --- |
| **Patron** | A Patron is a Process Impact employee who wants to order meals to be delivered from the company cafeteria. There are about 600 potential Patrons, of which 300 are expected to use the COS an average of 5 times per week each. Patrons will sometimes order multiple meals for group events or guests. An estimated 60 percent of orders will be placed using the corporate intranet, with 40 percent of orders being placed from home or by smartphone or tablet apps. |
| **Cafeteria Staff** | The Process Impact cafeteria employs about 20 Cafeteria Staff who will receive orders from the COS, prepare meals, package them for delivery, and request delivery. Most of the Cafeteria Staff will need training in the use of the hardware and software for the COS. |
| **Menu Manager** | The Menu Manager is a cafeteria employee who establishes and maintains daily menus of the food items available from the cafeteria. Some menu items may not be available for delivery. The Menu Manager will also define the cafeteria’s daily specials. The Menu Manager will need to edit existing menus periodically. |
| **Meal Deliverer** | As the Cafeteria Staff prepare orders for delivery, they will issue delivery requests to a Meal Deliverer’s smartphone. The Meal Deliverer will pick up the food and deliver it to the Patron. A Meal Deliverer's other interactions with the COS will be to confirm that a meal was (or was not) delivered. |

**Example: Use case Diagram**

Following use case diagram is of an appointment system in which all the use case diagram relationships are presented. In further in **Table A-2** to the detail of use case diagram syntax is provided.

Diagram

Description automatically generated

**Figure A-6: Use Case Diagram of an Appointment System**

**Syntax for Use case Diagram**

**Table A-2 Syntax for Use case Diagram**

| **An actor:**  ■ Is a person or system that derives benefit from and is external to the subject.  ■ Is depicted as either a stick figure (default) or, if a nonhuman actor is involved, a  rectangle with <<actor>> in it (alternative).  ■ Is labeled with its role.  ■ Can be associated with other actors using a specialization/superclass association,  denoted by an arrow with a hollow arrowhead.  ■ Is placed outside the subject boundary. |  |
| --- | --- |
| **A use case:**  ■ Represents a major piece of system functionality.  ■ Can extend another use case.  ■ Can include another use case.  ■ Is placed inside the system boundary.  ■ Is labeled with a descriptive verb–noun phrase. |  |
| **subject boundary:**  ■ Includes the name of the subject inside or on top.  ■ Represents the scope of the subject, e.g., a system or an individual  business process. |  |
| **An association relationship:**  ■ Links an actor with the use case(s) with which it interacts. |  |
| **An include relationship:**  ■ Represents the inclusion of the functionality of one use case within another.  ■ Has an arrow drawn from the base use case to the used use case. |  |
| **An extend relationship:**  ■ Represents the extension of the use case to include optional behavior.  ■ Has an arrow drawn from the extension use case to the base use case. |  |
| **A generalization relationship:**  ■ Represents a specialized use case to a more generalized one.  ■ Has an arrow drawn from the specialized use case to the base use case. |  |

**Detail Use Case Example**

The **Table A-3** below indicates a comprehensive use case template filled in with an example drawn from the Cafeteria ordering system (COS).

**Table A-3 Show the detail use case template and example**

| **Use Case ID:** | Enter a unique numeric identifier for the Use Case. e.g. UC-1 |
| --- | --- |
| **Use Case Name:** | Enter a short name for the Use Case using an active verb phrase. e.g.  Order a Meal |
| **Actors:** | [An actor is a person or other entity external to the software system being specified who interacts with the system and performs use cases to accomplish tasks.] e.g.   | Primary Actor: | Patron | Secondary Actors: | Cafeteria Inventory System | | --- | --- | --- | --- | |
| **Description:** | [Provide a brief description of the reason for and outcome of this use case.] e.g.  A Patron accesses the Cafeteria Ordering System from either the corporate intranet or external Internet, views the menu for a specific date, selects food items, and places an order for a meal to be picked up in the cafeteria or delivered to a specified location within a specified 15-minute time window. |
| **Trigger:** | [Identify the event that initiates the use case.]e.g.  A Patron indicates that he wants to order a meal. |
| **Preconditions:** | [List any activities that must take place, or any conditions that must be true, before the use case can be started.  PRE-1. Patron is logged into COS.  PRE-2. Patron is registered for meal payments by payroll deduction. |
| **Postconditions:** | [Describe the state of the system at the conclusion of the use case execution.  POST-1. Meal order is stored in COS with a status of “Accepted.”  POST-2. Inventory of available food items is updated to reflect items in this order.  POST-3. Remaining delivery capacity for the requested time window is updated. |
| **Normal Flow:** | [Provide a detailed description of the user actions and system responses that will take place during execution of the use case under normal, expected conditions.  1.0 Order a Single Meal  1. Patron asks to view menu for a specific date. (see 1.0. E1, 1.0.E2)  2. COS displays menu of available food items and the daily special.  3. Patron selects one or more food items from menu. (see 1.1)  4. Patron indicates that meal order is complete. (see 1.2)  5. COS displays ordered menu items, individual prices, and total price, including taxes and delivery charge.  6. Patron either confirms meal order (continue normal flow) or requests to modify meal order (return to step 2).  7. COS displays available delivery times for the delivery date.  8. Patron selects a delivery time and specifies the delivery location.  9. Patron specifies payment method.  10. COS confirms acceptance of the order.  11. COS sends Patron an email message confirming order details, price, and delivery instructions.  12. COS stores order, sends food item information to Cafeteria Inventory System, and updates available delivery times. |
| **Alternative Flows:** | [Document legitimate branches from the main flow to handle special conditions (also known as extensions). For each alternative flow reference the branching step number of the normal flow and the condition which must be true for this extension to be executed. e.g.  1.1 Order multiple identical meals  1. Patron requests a specified number of identical meals. (see 1.1. E1)  2. Return to step 4 of normal flow.  1.2 Order multiple meals  1. Patron asks to order another meal.  2. Return to step 1 of normal flow.  Note: Insert a new row for each distinctive alternative flow. ] |
| **Exceptions:** | 1.0. E1 Requested date is today and current time is after today’s order cutoff time  1. COS informs Patron that it’s too late to place an order for today.  2a. If Patron cancels the meal ordering process, then COS terminates use case.  2b. Else if Patron requests another date, then COS restarts use case.  1.0. E2 No delivery times left  1. COS informs Patron that no delivery times are available for the meal date.  2a. If Patron cancels the meal ordering process, then COS terminates use case.  2b. Else if Patron requests to pick the order up at the cafeteria, then continue with normal flow, but skip steps 7 and 8.  1.1. E1 Insufficient inventory to fulfill multiple meal order  1. COS informs Patron of the maximum number of identical meals he can order, based on current available inventory.  2a. If Patron modifies number of meals ordered, then return to step 4 of normal flow.  2b. Else if Patron cancels the meal ordering process, then COS terminates use case. |
| **Business Rules** | Use cases and business rules are intertwined. Some business rules constrain which roles can perform all or parts of a use case. Perhaps only users who have certain privilege levels can perform specific alternative flows. That is, the rule might impose preconditions that the system must test before letting the user proceed. Business rules can influence specific steps in the normal flow by defining valid input values or dictating how computations are to be performed e.g.  BR-1 Delivery time windows are 15 minutes, beginning on each quarter hour.  BR-2 Deliveries must be completed between 11:00 A.M. and 2:00 P.M. local time, inclusive.  Note: If you are maintaining the business rule in a separate table in SRS then only mention here their IDs. |
| **Assumptions:** | [List any assumptions.   1. e.g. Assume that 15 percent of Patrons will order the daily special (Source: previous 6 months of cafeteria data). |

**Event-Response Tables**

In order to develop Event Response table first it is required to identify all possible events. Following is an example of events list and event-response table of highway intersection system.

The highway intersection system described earlier has to deal with various events, including these:

* A sensor detects a car approaching in one of the through lanes.
* A sensor detects a car approaching in a left-turn lane.
* A pedestrian presses a button to request to cross a street.
* One of many timers counts down to zero.

**Table A-4** presents a fragment of what an event-response table might look like for such a system. Each expected system behavior consists of a combination of event, system state, and response.

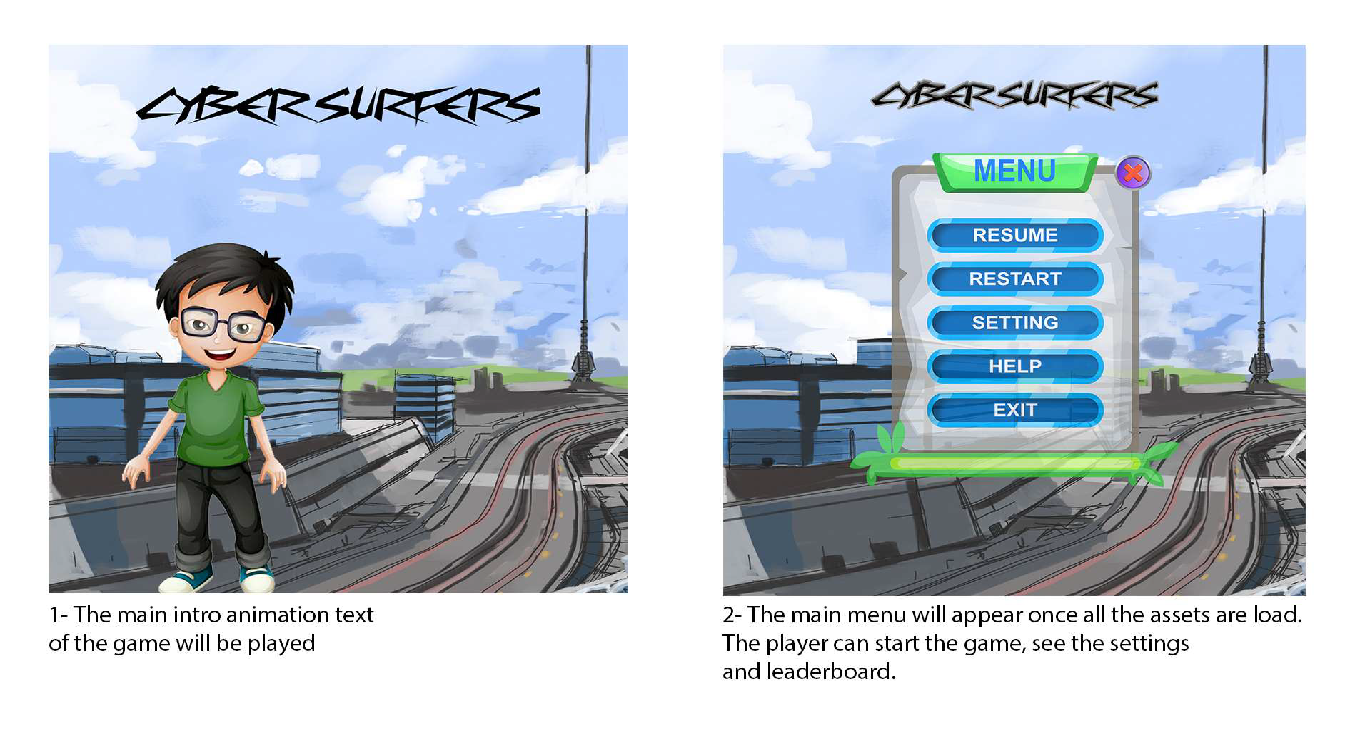
**Note:** You may add more information to event-response table in order to perform detail requirement analysis which includes:

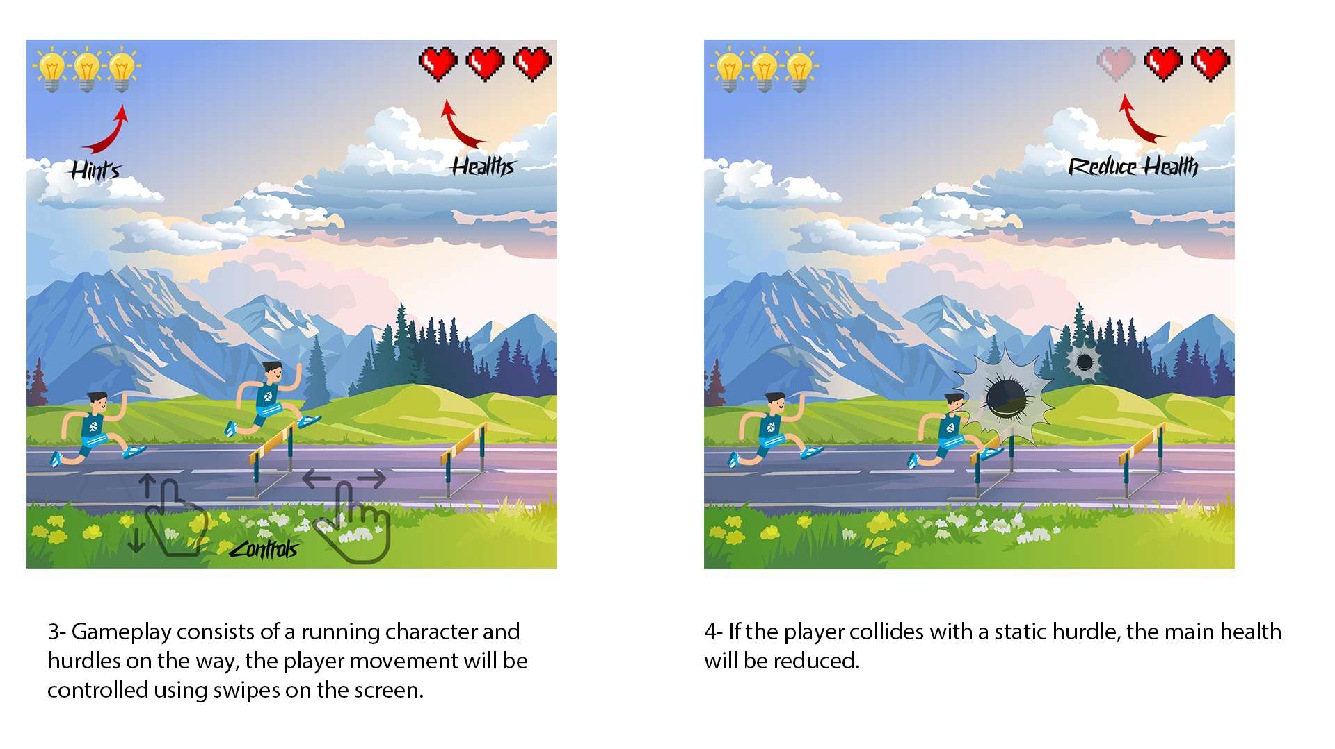
* The event frequency (how many times the event takes place in a given time period, or a limit to how many times it can occur).
* Data elements that are needed to process the event.
* The state of the system after the event responses is executed.

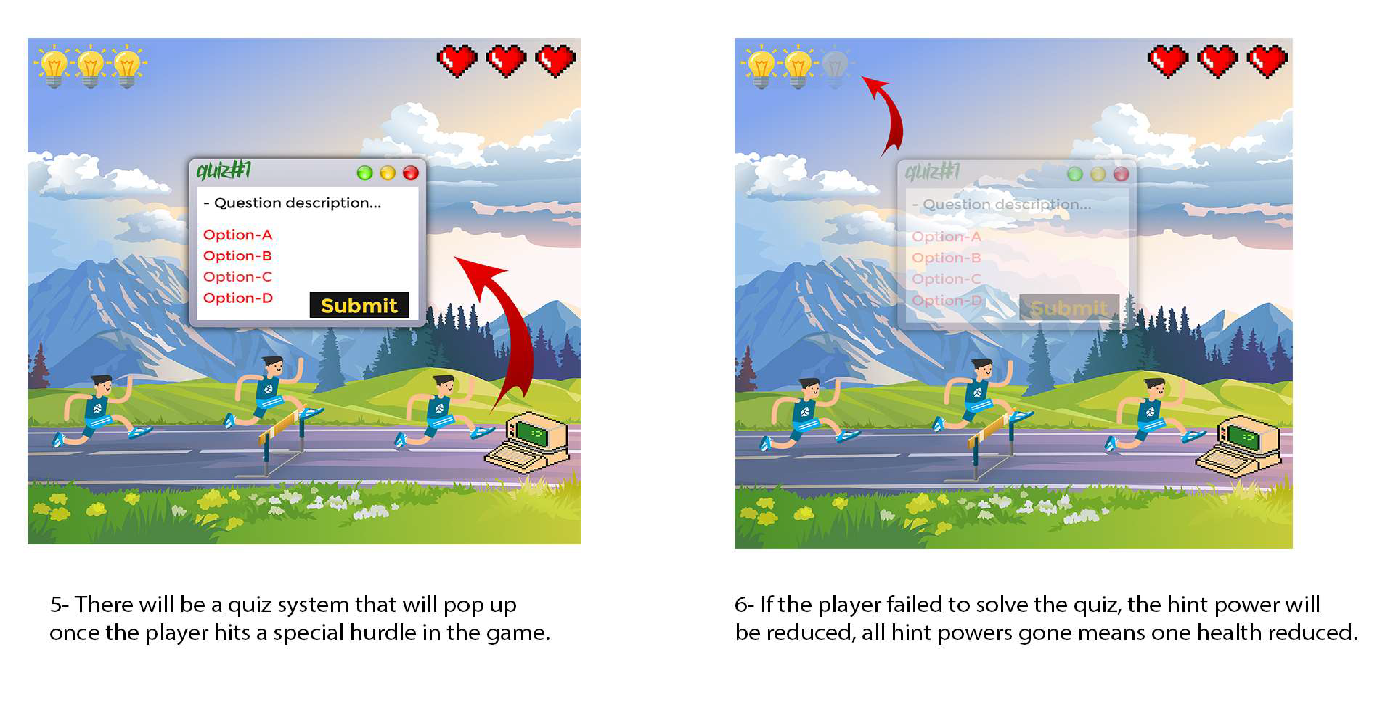
**Table A-4 Partial Event-Response Table for a Highway Intersection**

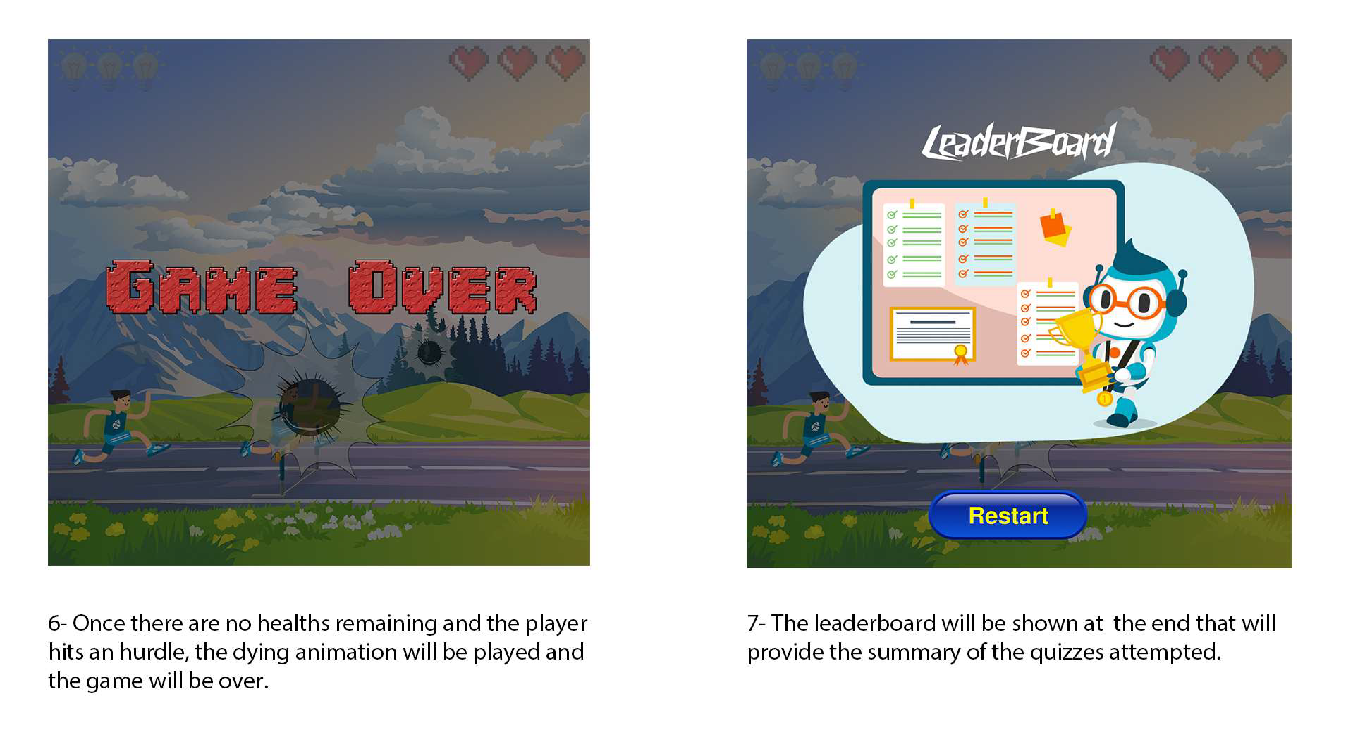
| **Event** | **System State** | **Response** |
| --- | --- | --- |
| Road sensor detects  vehicle entering left-turn  lane. | Left-turn signal is red. Cross-traffic signal is green. | Start green-to-amber  countdown timer for cross-traffic  signal. |
| Green-to-amber  countdown timer reaches  zero. | Cross-traffic signal is green. | 1. Turn cross-traffic signal amber.  2. Start amber-to-red countdown timer. |
| Amber-to-red  countdown timer reaches  zero. | Cross-traffic signal is amber. | 1. Turn cross-traffic signal red.  2. Wait 1 second.  3. Turn left-turn signal green.  4. Start left-turn-signal countdown timer. |
| Pedestrian presses a  specific walk-request  button. | Pedestrian sign is solid Don’t Walk.  Walk-request countdown timer is not activated. | Start walk-request countdown timer. |
| Pedestrian presses  walk-request button. | Pedestrian sign is solid Don’t Walk.  Walk-request countdown timer is activated. | Do nothing. |
| Walk-request  countdown timer reaches  zero plus the amber  display time. | Traffic signal in walk  direction is green. | Change all green traffic signals to amber. |
| Walk-request  countdown timer reaches  zero. | Traffic signal in walk  direction is amber. | 1. Change all amber traffic signals to red.  2. Wait 1 second.  3. Set pedestrian sign to Walk.  4. Start don’t-walk countdown timer. |

**Story Boarding**

Following is an example of story boarding which is use to identify and analyze graphically intensive applications.





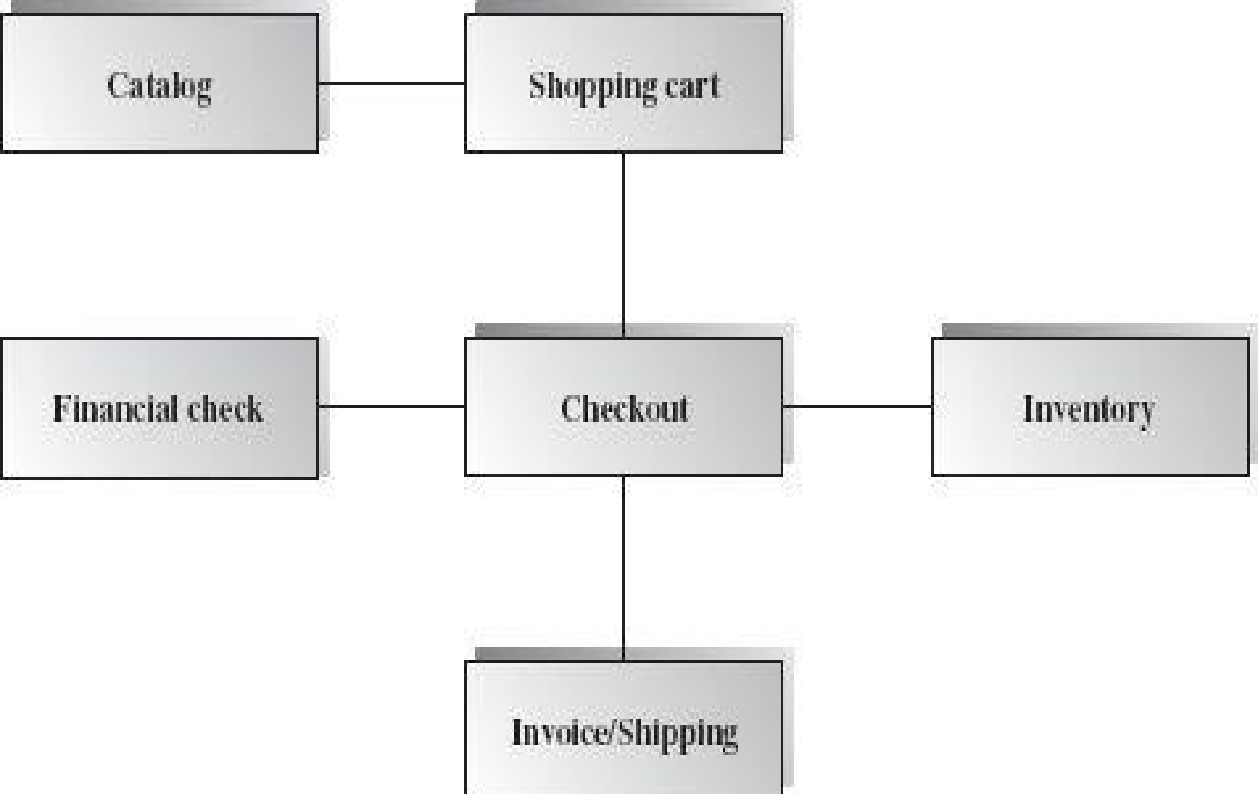


**Appendix B**

**Box-and-line diagram**

Box-and-line diagrams are often used to describe the business concepts and processes during the analysis phase of the software development lifecycle. These diagrams come with descriptions of components and connectors, as well as other descriptions that provide common inherent interpretations.

**Example:**





Lines in the box-and-line diagrams indicate the relationship among components

* The semantic of lines may refer dependency, control flow, data flow, etc
* Lines may be associated with arrows to indicate the process direction and sequence.
* A box-and-line diagram can be used as a business concept diagram describing its application domain and process concepts

**Example of Architecture Pattern:**

The **figure B-2** shows an example of the logical package organization of the layered architecture. The top level deals with user interface, the next level is for utilities, and the one below utility provides core services. Each layer gets support from its lower adjacent layer by an interface implementation and from the related classes in the same layer.

A simple software system may consist of two layers: an interaction layer and a processing layer:

* The interaction layer provides user interfaces to clients, takes requests, validates and forwards requests to the processing layer for processing, and responds to clients.
* The processing layer receives the forwarded requests and performs the business logic process, accesses the database, returns the results to its upper layer, and lets the upper layer respond to clients since the upper layer has the GUI interface responsibility.

Diagram

Description automatically generated

**Note:** The Architecture pattern shall be selected according to the targeted system’s requirements and quality attributes. Above example is provided to demonstrate that how the system architecture is required to be presented.

**Appendix C**

**Design Models**

**Activity Diagram**

Following activity diagram is of an appointment system presenting **make an appointment** process in which all diagram’s elements are presented. In further in **Table C-1** to the detail of activity diagram syntax is provided.

**Example**Diagram

Description automatically generated

**Activity Diagram Syntax**

**Table C-1** Activity Diagram Syntax

| **Term and definition** | **Symbol** |
| --- | --- |
| **An action:**   * Is a simple, non-decomposable piece of behavior. * Is labeled by its name. |  |
| **An activity:**   * Is used to represent a set of actions. * Is labeled by its name. |  |
| **An object node:**   * Is used to represent an object that is connected to a set of object flows. * Is labeled by its class name. |  |
| **A control flow:**   * Shows the sequence of execution. |  |
| **An object flow:**   * Shows the flow of an object from one activity (or action) to another activity (or action). |  |
| An initial node:   * Portrays the beginning of a set of actions or activities. |  |
| **A final-activity node:**   * Is used to stop all control flows and object flows in an activity (or action). |  |
| **A final-flow node:**   * Is used to stop a specific control flow or object flow. |  |
| **A decision node:**   * Is used to represent a test condition to ensure that the control flow or object flow only goes down one path. * Is labeled with the decision criteria to continue down the specific path. |  |
| **A merge node:**   * Is used to bring back together different decision paths that were created using a decision node. |  |
| **A fork node:**   * Is used to split behavior into a set of parallel or concurrent flows of activities (or action) |  |
| **A join node:**   * Is used to bring back together a set of parallel or concurrent flows of activities (or action) |  |
| **A swimlane:**   * Is used to break up an activity diagram into rows and columns to assign the individual activities (or actions) to the individuals or objects that are responsible for executing the activity (or action) * Is labeled with the name of the individual or object responsible |  |

**Class Diagram**

Following class diagram is of an appointment system in which all class diagrams elements are presented. In further in **Table C-2** to the detail of class diagram syntax is provided.

**Example**

Diagram

Description automatically generated



**Class Diagram Syntax**

**Table C- 2 Class Diagram Syntax**

| **Term and definition** | **Symbol** |
| --- | --- |
| **A class:**   * Has a name typed in bold and centered in its top compartment. * Has a list of attributes in its middle compartment. * Represents a kind of person, place, or thing about which the system will need to capture and store information. * Has a list of operations in its bottom compartment. * Does not explicitly show operations that are available to all classes. |  |
| **An attribute:**   * Represents properties that describe the state of an object. * Can be derived from other attributes, shown by placing a slash before the attribute’s name. | attribute name  /derived attribute name |
| **An operation:**   * Represents the actions or functions that a class can perform. * Can be classified as a constructor, query, or update operation. * Includes parentheses that may contain parameters or information needed to perform the operation. | operation name () |
| **An association:**   * Represents a relationship between multiple classes or a class and itself. * Is labeled using a verb phrase or a role name, whichever better represents the relationship. * Can exist between one or more classes. * Contains multiplicity symbols, which represent the minimum and maximum times a class instance can be associated with the related class instance. |  |
| **A generalization:**   * Represents a-kind-of relationship between multiple classes. |  |
| **An aggregation:**   * Represents a logical a-part-of relationship between multiple classes or a class and itself. * Is a special form of an association. |  |
| **A composition:**   * Represents a physical a-part-of relationship between multiple classes or a class and itself * Is a special form of an association. |  |

**Sequence Diagram**

Following example shows an instance sequence diagram that depicts the objects and messages for the Make Old Patient Appt use case, which describes the process by which an existing patient creates a new appointment or cancels or reschedules an appointment. In further in **Table C-3** to the detail of class diagram syntax is provided.

**Example**Diagram

Description automatically generated



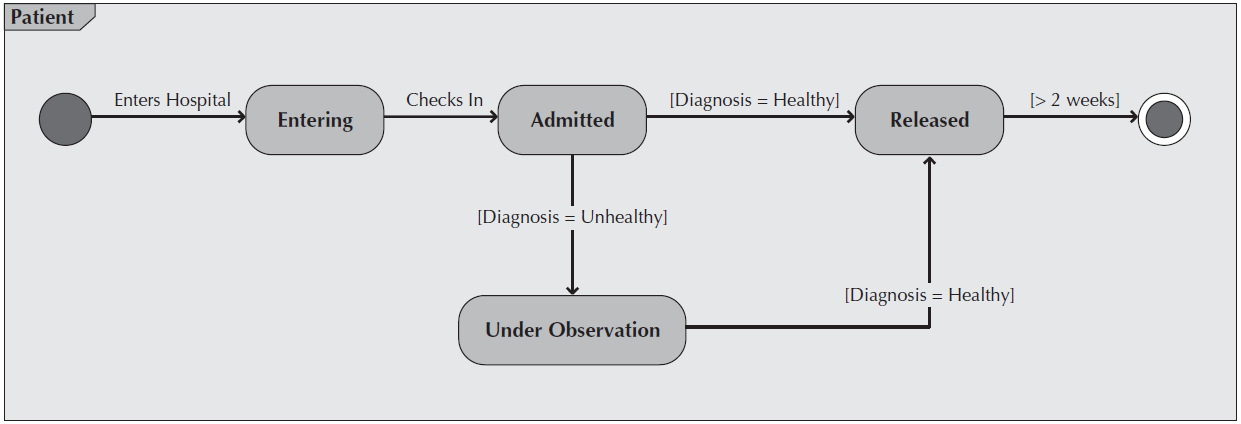
**Sequence diagram Syntax**

**Table C-3 Sequence Diagram Syntax**

| **Term and definition** | **Symbol** |
| --- | --- |
| **An actor:**   * Is a person or system that derives benefit from and is external to the system. * Participates in a sequence by sending and/or receiving messages. * Is placed across the top of the diagram. * Is depicted either as a stick figure (default) or, if a nonhuman actor is involved, as a rectangle with <<actor>> in it (alternative). |  |
| **An object:**   * Participates in a sequence by sending and/or receiving messages. * Is placed across the top of the diagram. |  |
| **A lifeline:**   * Denotes the life of an object during a sequence. * Contains an X at the point at which the class no longer interacts. |  |
| **An execution occurrence:**   * Is a long narrow rectangle placed atop a lifeline. * Denotes when an object is sending or receiving messages. |  |
| **A message:**   * Conveys information from one object to another one. * An operation call is labeled with the message being sent and a solid arrow, whereas a return is labeled with the value being returned and shown as a dashed arrow. |  |
| **A guard condition:**   * Represents a test that must be met for the message to be sent. |  |
| **For object destruction:**   * An X is placed at the end of an object’s lifeline to show that it is going out of existence. |  |
| **A frame:**   * Indicates the context of the sequence diagram. |  |

**Behavioral State Machine Diagram**

Following example of a behavioral state machine representing the patient class in the context of a hospital environment. From this diagram, we can tell that a patient enters a hospital and is admitted after checking in. If a doctor finds the patient to be healthy, he or she is released and is no longer considered a patient after two weeks elapse. If a patient is found to be unhealthy, he or she remains under observation until the diagnosis changes.

**Example**

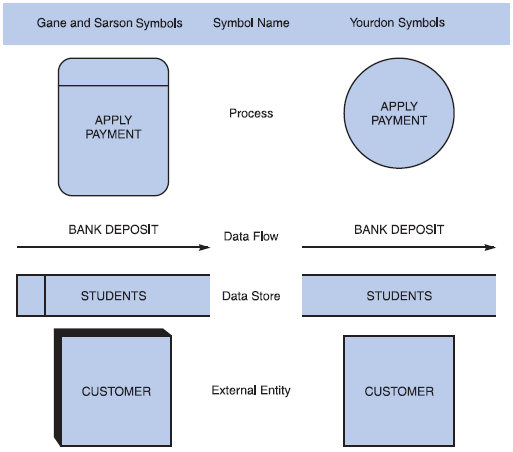


**Behavioral State Machine Diagram Syntax**

**Table C-4 Behavioral State Machine Diagram Syntax**

| **Term and definition** | **Symbol** |
| --- | --- |
| **A state:**   * Is shown as a rectangle with rounded corners. * Has a name that represents the state of an object. |  |
| **An initial state:**   * Is shown as a small, filled-in circle. * Represents the point at which an object begins to exist. |  |
| **A final state:**   * Is shown as a circle surrounding a small, filled-in circle (bull's-eye). * Represents the completion of activity. |  |
| **An event:**   * Is a noteworthy occurrence that triggers a change in state. * Can be a designated condition becoming true, the receipt of an explicit signal from one object to another, or the passage of a designated period of time. * Is used to label a transition. |  |
| **A transition:**   * Indicates that an object in the first state will enter the second state. * Is triggered by the occurrence of the event labeling the transition. * Is shown as a solid arrow from one state to another, labeled by the event name. |  |
| **A frame:**   * Indicates the context of the behavioral state machine. |  |

**Data Flow Diagram**

**Data flow diagram symbols, symbol names, and examples**

**Guidelines for Drawing DFDs**

**Step 1: Draw a Context Diagram**: The first step in constructing a set of DFDs is to draw a context diagram. A **context diagram** is a top-level view of an information system that shows the system’s boundaries and scope. Data stores are not shown in the context diagram because they are contained within the system and remain hidden until more detailed diagrams are created.

**Example**Diagram

Description automatically generated



**Step 2: Draw a Diagram 0 DFD:** To show the detail inside the black box, you create DFD diagram 0. **Diagram 0** zooms in on the system and shows major internal processes, data flows, and data stores. Diagram 0 also repeats the entities and data flows that appear in the context diagram. When you expand the context diagram into DFD diagram 0, you must retain all the connections that flow into and out of process 0.

**Example**Diagram

Description automatically generated

**Step 3: Draw the Lower-Level Diagrams:**

To create lower-level diagrams, you must use leveling and balancing techniques. **Leveling** is the process of drawing a series of increasingly detailed diagrams, until all functional primitives are identified.

**Leveling Example**Diagram

Description automatically generated

**Balancing** maintains consistency among a set of DFDs by ensuring that input and output data flows align properly.

**Balancing Example**

Order System Diagram 0 DFDDiagram

Description automatically generated

Order System Diagram 3 DFDDiagram

Description automatically generated

The order system diagram 0 is shown at the top of the figure and exploded diagram 3 DFD (for the APPLY PAYMENT process) is shown at the bottom. The two DFDs are balanced, because child diagram at the bottom has the same input and output flows as the parent process 3 shown at the top.

**Appendix D: General Coding Standards & Guidelines**

1. Follow a consistent variable naming convention throughout the code. E.g.
   * Snakecase (words are delimited by “\_” like: variable\_one)
   * Pascalcase (words are delimited by capital letters like: VariableOne)
   * Camelcase (words are delimited by capital letters except the initial word like: variableOne)
   * Hungarian Notation (describes the variable type or purpose at the start of the variable name like: arrDistributeGroup)
2. Use naming that visually describes scope like privateField, Const etc
3. Use read only/immutable when a field’s value should not be changed after initialization
4. Use only get, for properties that should not be updated from outside
5. Name functions according to their functionalities.
6. Insert appropriate comments to make the code understandable to any reader. Additionally follow a consistent style to do so. E.g.

/\* the below function will be used for the addition of two variables\*/

int Add(){

//logic of the function

}

Avoid commenting on obvious things

1. Make use of indentation for indicating the start and end of the control structures along with a clear specification of where the code is between them.
2. Follow consistent naming convention for files and folders.
3. Follow proper structure for classes
4. Group code entities logically into projects/packages/modules/folders
   1. Separate logical layers of application into different modules/services/utilities etc.
   2. User separate files for each class, struct, interface, enum etc. Name of the file and the enclosing entity must be same. E.g., class Employee in Employee.cs/Employee.java
5. Define and use everything within the minimum scope possible
6. Use proper access modifier for all code entities if required
7. Code entities should have maximum cohesion and least coupling possible.
8. Follow DRY law.
   1. Do not repeat code.
   2. A piece of knowledge should exist only in one place within the codebase/application
   3. Reuse code as much as possible
   4. Always write short methods
   5. Single method should not have too many logic, long conditional flow or too many parameters
9. Strictly follow Single Responsibility Principle (SRP) when writing methods, classes, modules, projects, packages, or any other code entities.
10. Write classes and other code entities that are easy to extend without modification.
11. Handle exceptions
12. Log exception and other significant event details
13. Follow a consistent convention for logging all over the application

**Appendix E**

**Business rules testing**

Methodology for creating decision table:

Table

Description automatically generated

**Example:**

The provided example is of a super store.

Table

Description automatically generated

**Table

Description automatically generated**

Table

Description automatically generated

Table, calendar

Description automatically generated

Table

Description automatically generated

* Now Combine rules where it is apparent that an alternative does not make a difference in the outcome

Calendar

Description automatically generated

This is the final table and now you have to create test cases on every rule. In above example there are 6 rules so there shall be 6 test cases.