Software
Engineering

University of Belize
CMPS4131 – Group 3

Report 3
Specs & Design

Azriel Baris Cuellar
Levi Coc
Daniel Eugenio Garcia
Victor Alexander Jr. Tillet
Floyd Jason Ack

Website Link

https://2019120154.wixsite.com/buscommutecompanion

Table of Contents

1. INDIVIDUAL CONTRIBUTIONS BREAKDOWN

Responsibility Level	**Team Member Names**				
	Azriel Cuellar	**Daniel Garcia**	**Victor Tillet**	**Levi Coc**	**Floyd Ack**
Sec. 1: Customer Statement	20%	20%	20%	20%	20%
Sec. 2: System Requirements	20%	20%	20%	20%	20%
Sec. 3: Functional Requirement Specification	20%	20%	20%	20%	20%
Sec. 4: Effort Estimation	20%	20%	20%	20%	20%
Sec. 5: Domain Analysis	20%	20%	20%	20%	20%
Sec. 6: Interaction Diagrams	20%	20%	20%	20%	20%
Sec. 7: Class Diagram & Interface Specifications	20%	20%	20%	20%	20%
Sec 8: System Architecture and Design	20%	20%	20%	20%	20%

Sec 9: Algorithms and Data Structure	20%	20%	20%	20%	20%
Sec 10: User Interface Design and Implementation	20%	20%	20%	20%	20%
Sec 11: Test Case Design	20%	20%	20%	20%	20%
Sec 12: History of work	20%	20%	20%	20%	20%
Sec 13: Project Management	20%	20%	20%	20%	20%

Customer Statement of Requirements

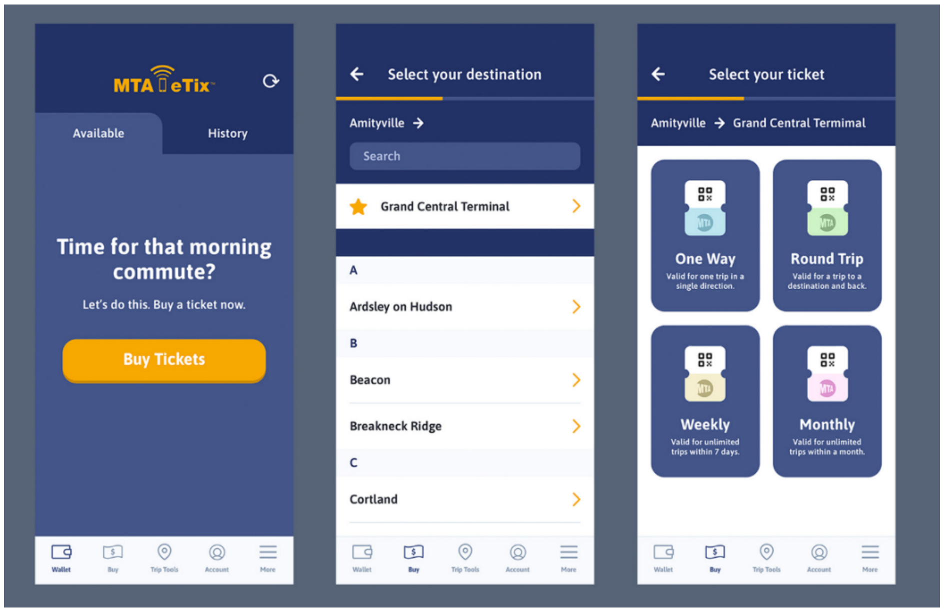
As a bus company, we understand the number of challenges passengers face when it comes to using bus services in Belize. One of the most significant issues is the lack of reliability in bus schedules at terminals. With so many factors that can impact the timeliness of buses, it can be difficult for passengers to plan their trips with confidence. Additionally, the current ticketing system is also prone to flaws, which can result in a frustrating experience for passengers.

For example, the issue of standees on buses is caused in part by a lack of available tickets or the over-selling of tickets at the terminal. In the past, passengers would simply have to line up and push their way onto the bus in order to secure a seat. This was not only dangerous in the event of an accident, but it also favored able-bodied individuals over those who were elderly, disabled, or accompanied by young children. To remedy this, our company began selling tickets for sale. However, this solution has brought with it a new set of problems. The ticket seller at the terminal is not always aware of the number of available seats on the bus, and sometimes a purchased ticket does not guarantee a seat because someone has already pushed their way onto the bus.

The current system is also not very efficient from a management perspective. Our company does not have an accurate way of tracking the number of passengers on a specific bus on any given day. This information can only be estimated based on revenue, but there is always the possibility that money goes missing and we are not able to detect it. The conductor of the bus calls the terminal manager to report the number of passengers on board and the number of available seats, but this process can be prone to inaccuracies or even deceit. The counting of tickets sold is also not a fool-proof method, as some passengers may not purchase tickets in advance, or the conductor may fail to distribute them along the way.

On occasion, drivers or conductors may choose to let more passengers onto the bus, even if it means standing, which can be highly unsafe in the event of an accident. This practice is also unfair to those who are elderly, disabled, or parents with young children. Additionally, some ticket holders may be unable to find a seat due to other passengers pushing ahead of them, rendering their ticket useless until the next bus arrives. While bus conductors may attempt to help, not all of them do, causing passenger frustration and potential loss of revenue for bus companies. The current system also poses a risk of financial loss or robbery, as passengers must pay at the terminal, which can potentially be a dangerous location for employees. Moreover, even with the current system in place, some passengers still attempt to pay on the bus, causing further confusion and potential conflicts over seat availability.

To improve the experience for passengers and address these challenges, we are looking for a software solution that can streamline and optimize the bus ticketing and management system. The proposed bus ticket management system will enable individuals to buy bus tickets using their mobile phones, and the purchasing process could be as straightforward as placing the phone on a surface or making the purchasing online, akin to Amazon. The model could be comparable to the MTA eTix system, which permits the buying of train tickets before boarding. The MTA eTix framework will be used as a reference for this bus ticket managing system.



Time for that morning commute?

Let's do this. Buy a ticket now.

Some of the key requirements that we would like to see in this solution include:

1. An accurate and real-time tracking system for bus schedules and arrivals, to provide passengers with up-to-date information on the status of their bus.
2. A secure and efficient ticketing system that eliminates the possibility of overselling tickets or providing an inaccurate count of available seats. This should also be easily accessible for all passengers, regardless of their physical ability.
3. A management system that allows our company to track the number of passengers on each bus on a real-time basis, as well as a clear and accurate record of ticket sales.
4. Integration with existing payment methods, such as mobile payment platforms, to allow for a seamless and convenient experience for passengers.
5. A user-friendly interface for both passengers and bus conductors, to make the process of buying and using tickets as simple and straightforward as possible.
6. Robust security measures to protect passenger information and prevent unauthorized access to the system.

We understand that this is a complex project, but we believe that a well-designed and implemented bus ticket management system has the potential to significantly improve the experience of passengers and the efficiency of bus companies. We are looking forward to hearing from you about how you can help us achieve these goals.

Glossary of Terms

Notes

- ***Terms defined in the lexicon are italicized when used in definitions within the lexicon.***
- ***When used in the lexicon, "entity" includes a natural person where the context requires.***

Term	**Definition**
Access Control	Means to ensure that access to assets is authorized and restricted based on business and security requirements. Source: ISO/IEC 27000:2018
Accountability	Property that ensures that the actions of n entity may be traced uniquely to that entity. Source: ISO/IEC 2382:2015
API	An Application Programming Interface is a set of protocols and tools used to build software and allow different software systems to interact and communicate with each other.
Asset	Something of either tangible or intangible value that is worth protecting, including people, information, infrastructure, finances, and reputation. Source: Adapted from NIST
Authentication	The process of verifying a user's identity, usually through a username and password, to ensure that only authorized users can assess the software system.
Authenticity	Property that an entity is what it claims to be. Source: ISO/IEC 27000:2018
Availability	Property of being accessible and usable on demand by an authorized entity. Source: ISO/IEC 27000:2018
Compromise	Violation of the security of an *information system*.

	Source: Adapted from ISO 21188:2018
Confidentiality	Property that information is neither made available nor disclosed to unauthorized individuals, entities, processes or systems. Source: Adapted from ISO/IEC 27000:2018
Course of Action	An action or actions taken to either prevent or respond to a *cyber incident*. It may describe technical, automatable responses but can also describe other actions such as employee training or policy changes. Source: Adapted from STIX
Cyber	Relating to, within, or through the medium of the interconnected information infrastructure of the interactions among persons, processes, data, and *information systems*. Source: Adapted from CPMI-IOSCO (Citing NICCS)
Cyber Alert	Notification that a specific cyber incident has occurred, or a *cyber threat* has been directed at an organization's *information systems*. Source: Adapted from NIST
Cyber Event	Any observable occurrence in an information system. *Cyber events sometimes provide indication that a cyber-incident is* occurring. Source: Adapted from NIST (definition of "Event")
Cyber Incident	A cyber event that: (i) Jeopardize the *cyber security* of an information system or the information the system processes, stores or transmits; or (ii) Violates the security policies, security procedures or acceptable use policies, whether resulting from malicious activity or not. Source: Adapted from NIST (definition of "Incident")
Cyber Security	Preservation of confidentiality, integrity, and availability of information and/or *information systems* through the *cyber* medium. In addition, other properties, such as authenticity, *accountability*, *non-repudiation, and reliability can also be* involved. Source: Adapted from ISO/IEC 27032:2012

Data Analytics	The process of collecting, analyzing, and interpreting data to gain insights and make informed decisions about the bus ticket system.
Encryption	A process of converting sensitive information into a code or cipher to protect it from unauthorized access and use.
Exploit	Defined way to breach the security of information systems through *vulnerability*. Source: ISO/IEC 27039:2015
FURPS	A requirements analysis framework that stands for Functionality, Usability, Reliability, Performance, and Supportability.
GPS Tracking	A technology that uses satellite signals to determine the exact location of a bus, which can be used to track the bus's progress along its route and provide real-time updates to users.
Incident	An unplanned interruption to a service, a reduction in the quality of a service or event that has not yet impacted the service to the customer or user. Source: ISO/IEC 20000-1
Information Systems	Set of applications, services, information technology *assets* or other information-handling components, which includes the operating environment. Source: Adapted from ISO/IEC 27000:2018
Mobile Application	A software application that runs on mobile devices, allowing users to interact with the software system on their mobile phones or tablets.
Multi-Factor Authentication	The use of two or more of the following factors to verify a user's identity. -- knowledge factor, "something an individual knows". -- possession factor, "something an individual has". -- biometric factor," something that is a biological and behavioral characteristic of an individual".

	Source: Adapted from ISO/IEC 27040:2015 and ISO/IEC 283237:2017 (definition of “biometric characteristic”)
Notification System	A feature of the software system that sends alerts to users about any updates or changes to their bus ticket, such as delays or cancellations.
Payment Gateway	A secure online service that processes payments from users for the purchase of bus tickets.
Protect (function)	Develop and implement the appropriate safeguards to ensure delivery of services and to limit or contain the impact of *cyber incidents*.
	Source: Adapted from NIST Framework
Real-time Updates	Immediate information about changes or delays to the bus schedule or route, provided to the passenger through the software system.
Recover (function)	Develop and implement the appropriate activities to maintain plans for *cyber resilience* and to restore any capabilities or services that were impaired due to a *cyber-incident*.
	Source: Adapted from NIST Framework
Refund Policy	The terms and conditions outlining the process for issuing refunds to users who have purchased a bus ticket but are unable to use it.
Reliability	Property of consistent intended behaviour and results.
	Source: ISO/IEC 27000:2018
Respond (function)	Develop and implement the appropriate activities to take action regarding a detected *cyber-event*.
	Source: Adapted from NIST Framework
Responsive Design	A design approach that ensures the software system is optimized for different devices and screen sizes, providing a seamless passenger experience regardless of the device being used.
Use Case Diagram	A graphical representation of the interactions between users and the system, showing the different use cases for the software system.

User Interface (UI)	The visual and interactive components of the software system that users interact with to complete tasks.
Verification	Confirmation, through the provision of objective evidence that can be exploited by one or more threats. Source: ISO/IEC 27042:2015
Vulnerability	A weakness, susceptibility or flaw of an asset or control that can be exploited by one or more threats. Source: Adapted from CPMI-IOSCO and ISO/IEC 27000:2018

System Requirements

Functional Requirements (REQ)

These requirements define the specific functions and features that the bus ticket management system must have in order to meet the needs of the bus company. They are the “what” of the system.

Identifier	PW	Requirement
REQ1	5	The system shall allow passengers to create and manage passenger profiles including passenger information and payment methods.
REQ2	3	The system shall enable passengers to purchase available tickets online.
REQ3	3	The system shall allow users to view the details of previous and present bookings. (History of Purchases)
REQ4	5	The system shall provide passengers with information about their trip, including departure and arrival time, bus route, and fare pricing (schedule).
REQ5	3	The system shall allow the bus companies to easily manage and update bus schedules to prevent delays.
REQ6	3	The system shall allow bus companies to track ticket sales, passenger information, and revenue.
REQ7	5	The system shall allow users to generate e-tickets or print their tickets with the system.
REQ8	4	The system shall generate reports on ticket sales, passenger traffic, and revenue, which can be used to track performance and make informed business decisions.
REQ9	2	The system shall allow users to be able to log in and log out.
REQ10	1	The system shall allow users to modify the time or conditions under which a notification is activated.

Identifier	PW	Requirement
REQ11	4	The system shall allow users to be able to cancel their ticket purchase.
REQ12	2	The system shall allow users to change the date and time their ticket is valid.
REQ13	3	The system shall allow users to offer feedback of their trip experience.

Non-Functional Requirements (NONREQ)

These requirements define the quality attributes and performance characteristics that the bus ticket management system must meet. They define the "how" of the system.

Identifier	PW	Requirement
NONREQ1	5	The system must have a user-friendly interface that is easy to use and navigate, providing a seamless experience for passengers.
NONREQ2	5	The system must have strong security measures to protect passenger information, including encryption for sensitive data, safely notify users of details regarding the ticket purchased and multi-factor authentication for passenger accounts.
NONREQ3	5	The system shall be scalable to accommodate future growth and increased passenger traffic.
NONREQ4	3	The system shall have a fast response time and be able to handle high volumes of traffic, ensuring that tickets can be purchased, and reservations can be managed in real-time.
NONREQ5	3	The system shall be compatible with various platforms and devices, including desktop computers, laptops, tablets, and mobile devices.
NONREQ6	5	The system shall be available and reliable for use at all times.
NONREQ7	5	The system shall accurately calculate fees based on routes, passengers, and trip dates.

NONREQ8	3	The system shall allow users to purchase a ticket without experiencing crashes or processing failures in it.
NONREQ9	2	The system shall apply a cut-off time and location limit beyond which it will not be possible to buy or cancel tickets.

On-Screen Appearance Requirements (ONSREQ)

These requirements refer to the visual and graphical design elements of the bus ticket management system that determines how it appears on the passenger's screen.

Identifier	**PW**	**Requirement**
ONSREQ1	5	The system shall have a clear display of available schedules and routes, along with real-time updates on any changes or delays.
ONSREQ2	5	The system shall give users the ability to select their desired route, departure time, and seat type quickly and easily, and view a summary of their ticket details before confirming the purchase.
ONSREQ3	3	The system shall have a secure payment gateway for users to securely enter their payment information and complete the transaction.
ONSREQ4	3	The system shall have a notification system to inform users of any updates or changes related to their ticket, such as schedule changes or delays.
ONSREQ5	3	The system shall give users the ability to view their ticket history and manage their account information, including updating their personal information, email address, and payment information.
ONSREQ6	5	The system shall have a responsive design that is optimized for different screen sizes and devices, including mobile devices, to provide a seamless experience for users.

Functional Requirements Specifications

Stakeholders

Stakeholders in the context of an application refer to individuals, groups, or organizations that have a direct or indirect interest in the development, implementation, and use of the application. The stakeholders of Bus Commute Companion, classified as primary, secondary, or tertiary stakeholders are:

1. Primary Stakeholders:
    a. Passengers: who will use Bus Commute Companion to purchase tickets and manage their trips.
    b. Bus Company: including ticket agents, customer service representatives, and operations managers who will use the system to manage passenger bookings and track bus schedules.

2. Secondary Stakeholders:
    a. Bus Company Management: who will oversee the system's implementation and use, as well as make decisions based on generated reports and data.
    b. Software Developers: responsible for designing, building, and maintaining the system.
    c. Payment Gateway Providers: who will facilitate online payments and transactions.
    d. Regulatory Authorities: who may need to approve the system's implementation and use and ensure compliance with local laws and regulations.

3. Tertiary Stakeholders:
    a. Third-Party Service Providers: who may provide additional services through the system, such as advertising, promotions, or travel insurance.
    b. Investors: who may have a financial stake in the bus company and the success of the system.

Actor And Goals

Originally, we had several actors within our system but they were never reflected in any form of documentation (diagrams). Based on Bus Commute Companion's stakeholders, the following are the actors and their goals:

1. Passengers

   Aim to register for an account, search, and view bus schedules, book a ticket, pay for a ticket, view booking details, cancel a booking, view bus and trip updates, and provide feedback and ratings through their smartphones.

2. Bus company management and staff

   Manage bus schedules, routes, availability, ticket sales, bookings, and passenger feedback through desktop or laptop computers.

Use Cases

Casual Description

1. **UC-1. Register:** This use case allows a passenger to register on the Bus Commute Companion website or application by providing their personal information, such as their name, email address, and contact number. After successful registration, the passenger can access the website's features that require login.

   - ➔ Functional Requirements: REQ1
   - ➔ Non Functional Requirements: NONREQ1, NONREQ2, NONREQ3, NONREQ4, NONREQ5, NONREQ6
   - ➔ On-Screen Appearance Requirements: ONSREQ6

2. **UC-2. Log in:** A registered passenger can log in to the website by entering their username and password. This use case verifies the user's credentials and allows them to access their account details and booking history.

   - ➔ Functional Requirements: REQ9
   - ➔ Non Functional Requirements: NONREQ1, NONREQ2, NONREQ4, NONREQ5, NONREQ6
   - ➔ On-Screen Appearance Requirements: ONSREQ6

3. **UC-3. View Profile:** This use case enables the passenger to view their profile details, such as their name, profile photo, email address, and contact number. The passenger can also view their booking history and other account information.

   - ➔ Functional Requirements: REQ1
   - ➔ Non Functional Requirements: NONREQ1, NONREQ4, NONREQ5, NONREQ6
   - ➔ On-Screen Appearance Requirements: ONSREQ5, ONSREQ6

4. **UC-4. Edit Profile:** This use case enables the passenger to edit their profile information, such as their name, profile photo, email address, and contact number. The passenger can update their personal details and save the changes to their profile.

   - ➔ Functional Requirements: REQ1
   - ➔ Non Functional Requirements: NONREQ1, NONREQ2, NONREQ4, NONREQ5, NONREQ6
   - ➔ On-Screen Appearance Requirements: ONSREQ5, ONSREQ6

5. **UC-5. Search Trip:** The use case enables the passenger to search for available bus trips by selecting the departure and arrival locations, dates, and other filters, such as the number of passengers and preferred travel class. The system displays the available options based on the search criteria.

- ➔ Functional Requirements: REQ4
- ➔ Non Functional Requirements: NONREQ1, NONREQ4, NONREQ5, NONREQ6
- ➔ On-Screen Appearance Requirements: ONSREQ1, ONSREQ6

6. **UC-6. Select Trip:** After searching for trips, the passenger can select a preferred option and view the trip details, including the route, schedule, and ticket price. Then the passenger begins the process of purchasing a ticket for the specific trip .

- ➔ Functional Requirements: REQ4
- ➔ Non Functional Requirements: NONREQ1, NONREQ4, NONREQ5, NONREQ6
- ➔ On-Screen Appearance Requirements: ONSREQ1, ONSREQ2, ONSREQ6

7. **UC-7. Select Seat:** After selecting a trip, the passenger can choose a seat from the available options on the bus and proceed to the payment page. This use case displays the seating layout and allows the passenger to select their preferred seat.

- ➔ Functional Requirements: REQ4
- ➔ Non Functional Requirements: NONREQ1, NONREQ4, NONREQ5, NONREQ6
- ➔ On-Screen Appearance Requirements: ONSREQ1, ONSREQ2, ONSREQ6

8. **UC-8. Select Payment:** This use case enables the passenger to pay for the selected trip using a secure online payment gateway. The passenger enters their payment details, and the system verifies and processes the payment.

- ➔ Functional Requirements: REQ2
- ➔ Non Functional Requirements: NONREQ1, NONREQ2, NONREQ4, NONREQ5, NONREQ6, NONREQ7 NONREQ8, NONREQ9
- ➔ On-Screen Appearance Requirements: ONSREQ1, ONSREQ2, ONSREQ3, ONSREQ6

9. **UC-9. View Payment History:** This use case enables the passenger to view their payment history, including all payments made for past and current tickets.

- ➔ Functional Requirements: REQ2

- ➔ Non Functional Requirements: NONREQ1, NONREQ2, NONREQ4, NONREQ5, NONREQ6, NONREQ7, NONREQ8
- ➔ On-Screen Appearance Requirements: ONSREQ1, ONSREQ2, ONSREQ6

10. **UC-10. View Booking History:** This use case enables a passenger to view their booking history, including all past and current tickets, cancellations, and modifications.

- ➔ Functional Requirements: REQ3, REQ4
- ➔ Non Functional Requirements:NONREQ1, NONREQ2, NONREQ4, NONREQ5, NONREQ6
- ➔ On-Screen Appearance Requirements: ONSREQ1, ONSREQ5, ONREQ6

11. **UC-11. Cancel Ticket:** This use case enables the passenger to cancel their current ticket purchase, which will cancel their ticket purchase and refund any applicable fare based on the cancellation policy.

- ➔ Functional Requirements: REQ4, REQ11
- ➔ Non Functional Requirements: NONREQ1, NONREQ4, NONREQ5, NONREQ6, NONREQ9
- ➔ On-Screen Appearance Requirements: ONSREQ1, ONSREQ5, ONSREQ6

12. **UC-12. Reschedule Ticket:** This use case enables a passenger to modify the date, time, and destination of their ticket, subject to availability and any applicable change fees.

- ➔ Functional Requirements: REQ4, REQ12
- ➔ Non Functional Requirements: NONREQ1, NONREQ2, NONREQ4, NONREQ5, NONREQ6
- ➔ On-Screen Appearance Requirements: ONSREQ1, ONSREQ4, ONSREQ5, ONSREQ6

13. **UC-13. Generate e-Ticket:** This use case enables a passenger to generate an e-ticket for their current ticket, which they can print or save as a digital copy to their device.

- ➔ Functional Requirements: REQ7
- ➔ Non Functional Requirements:NONREQ1, NONREQ4, NONREQ5, NONREQ6
- ➔ On-Screen Appearance Requirements: ONSREQ6

14. **UC-14. Provide Feedback:** This use case enables the passenger to provide feedback and rating for their trip experience, including the bus condition, driver

behaviour, and overall satisfaction. The passenger can rate various aspects of the trip and provide comments to help the bus company improve its services.

- ➔ Functional Requirements: REQ13
- ➔ Non Functional Requirements:NONREQ1, NONREQ4, NONREQ5
- ➔ On-Screen Appearance Requirements: ONSREQ6

15. UC-15 Change Notification Settings: This use case enables a passenger to set up notifications for their current ticket, including reminders of the trip date and time, and any changes or updates to the ticket.

- ➔ Functional Requirements: REQ10
- ➔ Non Functional Requirements:NONREQ1, NONREQ4, NONREQ5
- ➔ On-Screen Appearance Requirements: ONSREQ4

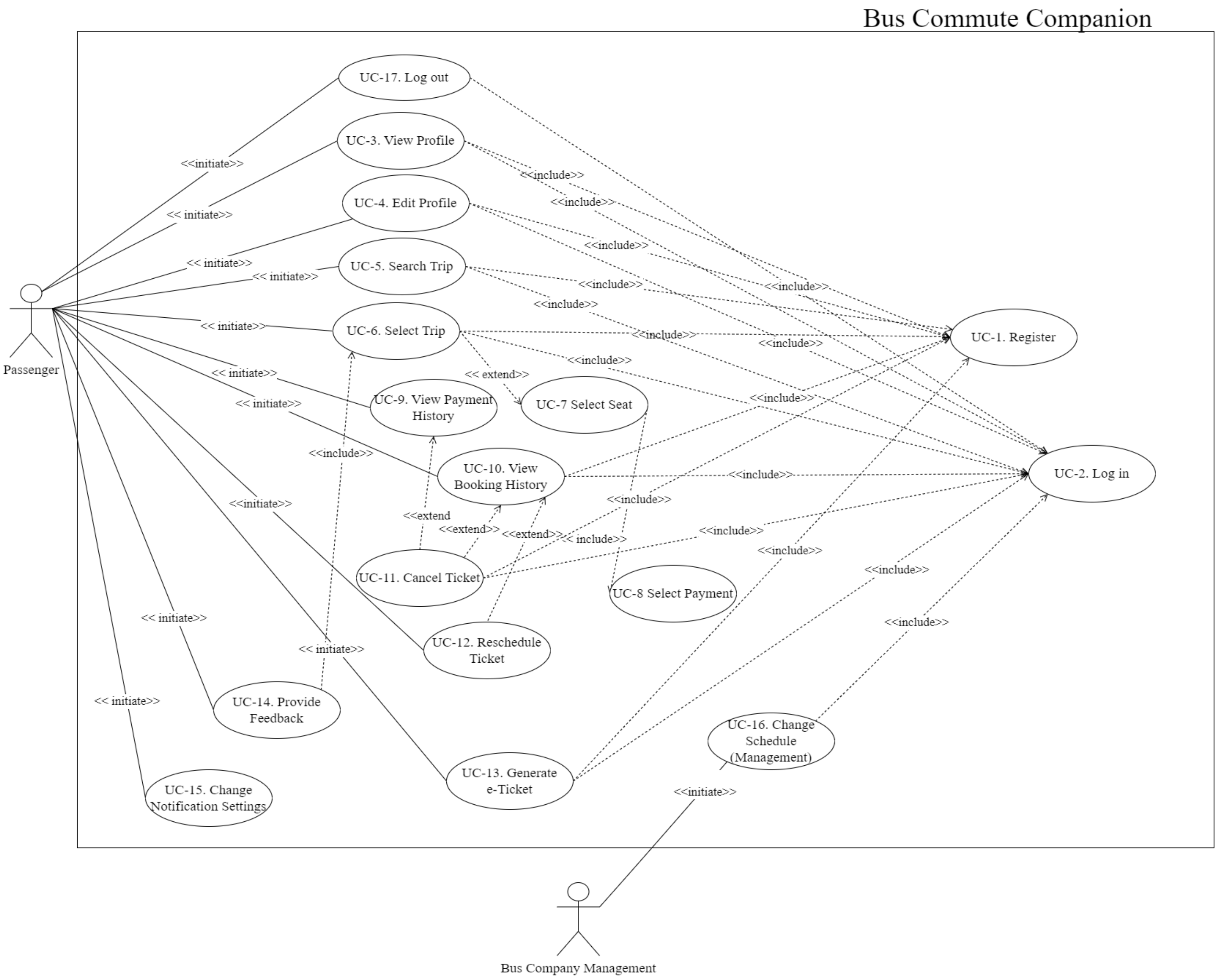
16. UC-16. Change Schedule (Management): This use case provides the bus company management to adjust the arrival time and departure time for the buses.

- ➔ Functional Requirements: REQ5
- ➔ Non Functional Requirements: NONREQ1, NONREQ2, NONREQ5, NONREQ6
- ➔ On-Screen Appearance Requirements:ONSREQ6

17. UC-17. Log out: This use case allows the passenger to log out of their account and end their session on the website or application. The passenger can log in again later to access their account and make new bookings.

- ➔ Functional Requirements: REQ9
- ➔ Non Functional Requirements: NONREQ1, NONREQ2, NONREQ4, NONREQ5, NONREQ6
- ➔ On-Screen Appearance Requirements: ONSREQ6

Case Diagram



Traceability Matrix

Requirements	**UC-1**	**UC-2**	**UC-3**	**UC-4**	**UC-5**	**UC-6**	**UC-7**	**UC-8**	**UC-9**	**UC-10**	**UC-11**	**UC-12**	**UC-13**	**UC-14**	**UC-15**	**UC-16**	**UC-17**
REQ1	5		5	5													
REQ2								3	3								
REQ3										3							
REQ4					5	5	5			5	5	5					
REQ5																3	
REQ6																	
REQ7													5				
REQ8																	
REQ9		2															2
REQ10															1		
REQ11											4						
REQ12												2					
REQ13														3			
REQ1																	
NONREQ1	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
NONREQ2	5			5				5	5	5		5	5			5	5
NONREQ3	5	5															
NONREQ4	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3		3
NONREQ5	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
NONREQ6	5	5	5	5	5	5	5	5	5	5	5	5	5			5	5
NONREQ7								4	4								
NONREQ8								3	3								
NONREQ9								2			2						
ONSREQ1					5	5	5	5	5	5	5	5					
ONSREQ2						5	5	5	5								
ONSREQ3								3									
ONSREQ4												3			3		
ONSREQ5			3	3	3					5	3	3					

ONSREQ6	5	5	5	5		5	5	5	5	5	5	5	5	5		5	5
Total Weight	**36**	**28**	**29**	**34**	**29**	**36**	**36**	**53**	**26**	**42**	**39**	**43**	**31**	**19**	**15**	**26**	**28**

Fully Dressed Description

The fully-dressed description was performed in the use cases that obtained a high priority weight in the traceability matrix. These consist of UC-1, UC-6, UC-9, UC-10, & UC-12. UC-7. UC-7 & UC-8 were not done as they are required for UC-6.

Use Case #1	**Register**
Initiating Actor	Passenger
Actor's Goal	Register to the system
Participating Actor	System & Database.
Preconditions	1. The website is running 2. The device is connected to the internet. 3. The passenger has the website open.
Post Conditions	1. The user is now registered to the system.
Flow of the events for main success scenario □ 1. The passenger clicks the “Register Now ” button. □ 2. The system displays the registration page. □ 3. The passenger (a)fills out the necessary requirements in the form and (b) selects “Register”. □ 4. The system (a) adds the user credentials in the database and (b) the passenger is now able to access the system. (c) The passenger is redirected to the login page.	
Events for extensions (Alternate Scenarios): 3a. Passenger fails to fill out required in formation □ System (a) detects missing information, and (b) signals to the passenger of missing information 4a. System fails to upload credentials. □System (a) signals error and (b) displays registration with note to try again. □ The passenger fills out the necessary requirements in the form and selects “Register”. □ The system (a) adds the user credentials in the database and (b) the passenger is now able to access the system. (c) The passenger is redirected to the login page.	

Use Case #6	**Select Trip**
Initiating Actor	Passenger
Actor's Goal	Select a trip to purchase a ticket.
Participating Actor	System, Database & Bank.
Preconditions	1. The website is running. 2. The device is connected to the internet. 3. The passenger is on the website. 4. The passenger is already registered. 5. The passenger is already logged in. 6. The passenger has determined which ticket to purchase.
Post Conditions	1. The user has purchased a ticket
Flow of the events for main success scenario □1. The passenger clicks the “ Select” button. □ 2. The system displays the “Select Seat “page. □ 3. The passenger clicks the preferred seat. □ 4. The system shows payment options available. □ 5. The passenger selects the credit card option. □ 6. The system prompts for credit card information. □ 7. The passengers fills in credit card information and select submit. □ 8. System passes card information and payment amount to the bank for authorization. □ 9. Bank processes transactions. □ 10. System (a) signals to the Passenger the purchase was complete, and (b) removes the seat from the list of available seats in the database.	
Flow of events for extensions (Alternate Scenarios): 4. Passenger selects Pay on Arrival □ 1. System displays the total cost of the ticket and instructions on how to pay on arrival. □ 2. Passenger closes prompt.	

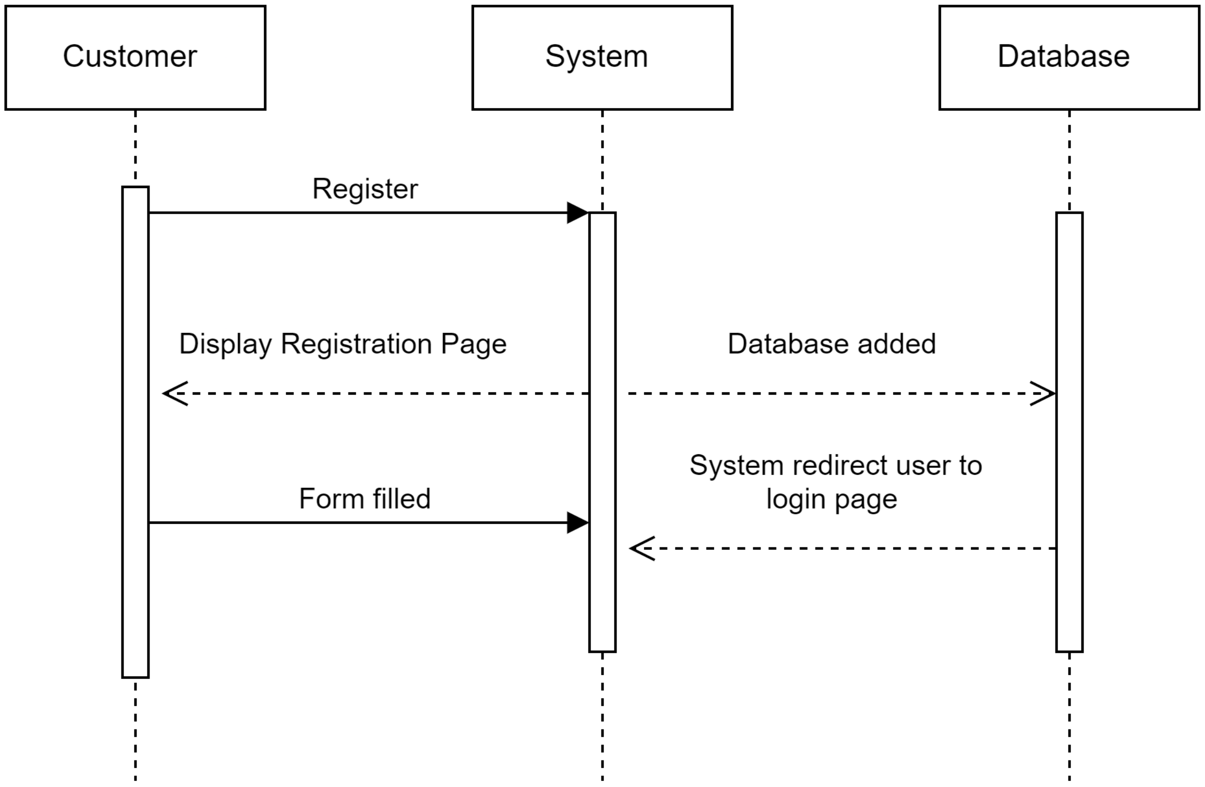
Use Case #9	**View Payment History**
Initiating Actor	Passenger
Actor's Goal	View previous and current payments made.
Participating Actor	System & Database
Preconditions	1. The website is running. 2. The device is connected to the internet. 3. The passenger is on the website. 4. The passenger is already registered. 5. The passenger is already logged in. 6. The passenger has purchased tickets previously
Post Conditions	1. The system shows payment history. 2. The system calculates total($) spent.
Flow of the events for main success scenario □ 1. The passenger selects the Account drop-down list and clicks “View History.” □ 2. The system displays the “View History” page that displays the passenger's transaction history(including the “View Booking History”).	
Flow of events for extensions (Alternate Scenarios): No alternate flow.	

Use Case #10	View Booking History
Initiating Actor	Passenger
Actor's Goal	View previous and current tickets purchased.
Participating Actor	System & Database
Preconditions	1. The website is running. 2. The device is connected to the internet. 3. The passenger is on the website. 4. The passenger is already registered. 5. The passenger is already logged in. 6. The passenger has purchased tickets before. 7. The passenger is before the time cutoff in which tickets are not purchasable
Post Conditions	1. The passenger is able to view their purchase history. 2. The system shall show the detail of present booking
Flow of the events for main success scenario □ 1. The passenger clicks the "View Booking History" button. □ 2. The system queries the database for the user's previous bookings and returns it. □ 3. The system displays the passenger Booking History (including "Cancel Ticket). □ 4. The passenger views the pass and current purchase, cancellations, and modifications.	
Events for extensions (Alternate Scenarios): 3. Passenger selects "Cancel Ticket".	

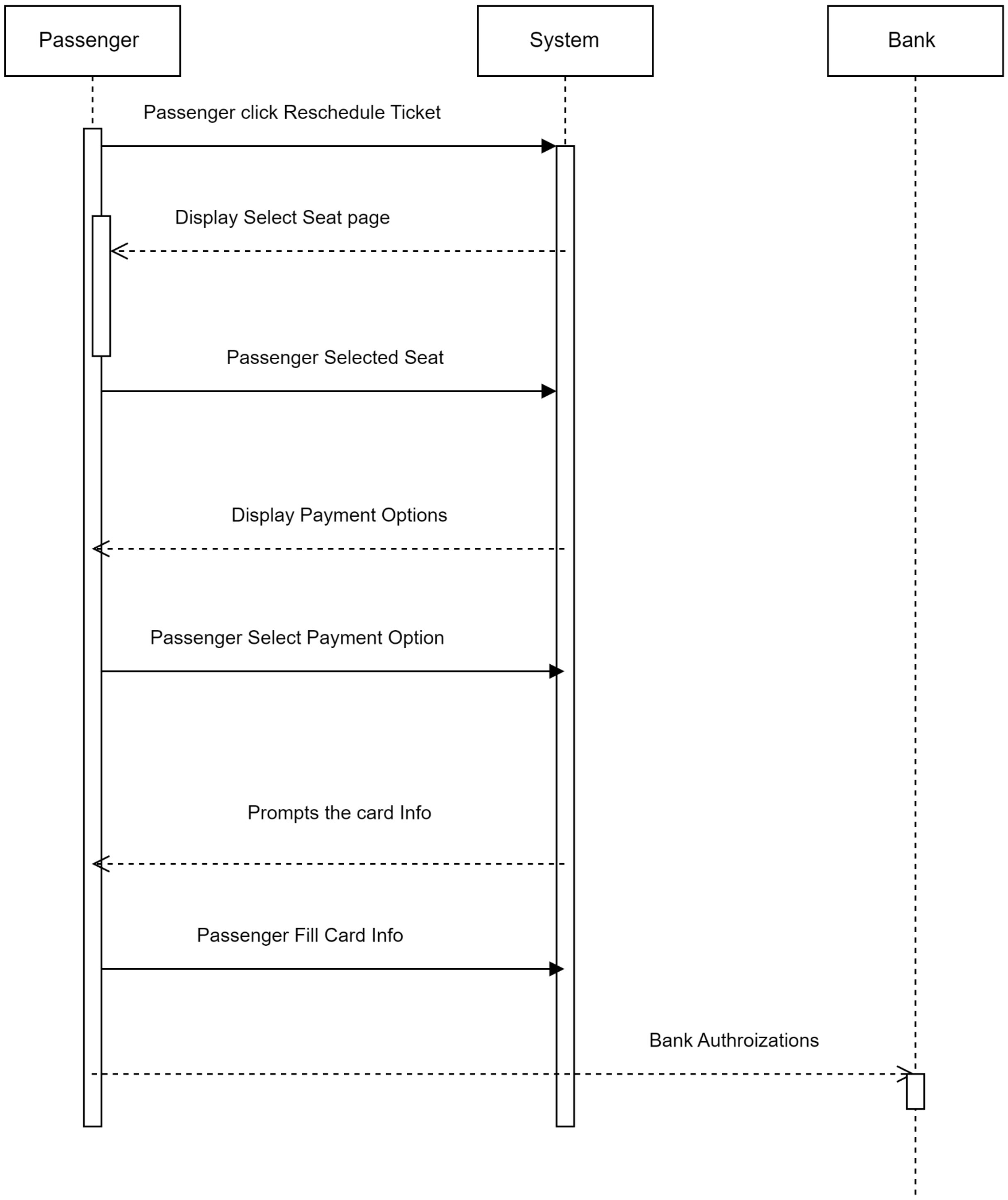
Use Case #12	**Reschedule Ticket**
Initiating Actor	Passenger
Actor's Goal	Reschedule Ticket
Participating Actor	System & Database
Preconditions	1. The website is running. 2. The device is connected to the internet. 3. The passenger is on the website. 4. The passenger is already registered. 5. The passenger is already logged in. 6. The passenger has purchased a ticket previously. 7. The passenger is before the time cutoff in which tickets are not purchasable 8. The passenger has decided on which ticket to reschedule.
Post Conditions	1. The user has successfully rescheduled the ticket. 2. The appropriate amount of funds have been added or refunded.
Flow of the events for main success scenario □ 1. The passenger clicks the “ Reschedule Ticket” button. □ 2. The system redirects to the “Select Trip” page. □ 3. The passenger clicks the “Select” button of the new ticket to be purchased. □ 4. The system redirects to “Select Seat” page □ 5. The passenger clicks the preferred seat. □ 6. The system shows two payment options available. □ 7. The passenger selects the credit card option. □ 8. The system prompts for credit card information. □ 9. The passengers fill in credit card information and selects submit. □ 10. System passes card information and payment amount to the bank for authorization. □ 11. Bank processes transactions and refunds the original ticket price. □ 12. System (a) signals to the Passenger the purchase was complete, (b) deletes the previous ticket purchase from the database, and (c) removes the seat from the list of available seats in the database.	
Events for extensions (Alternate Scenarios): □2a. The passenger selects the cancel option □ System displays confirm prompt □ Passenger confirms yes.	

System Sequence Diagrams

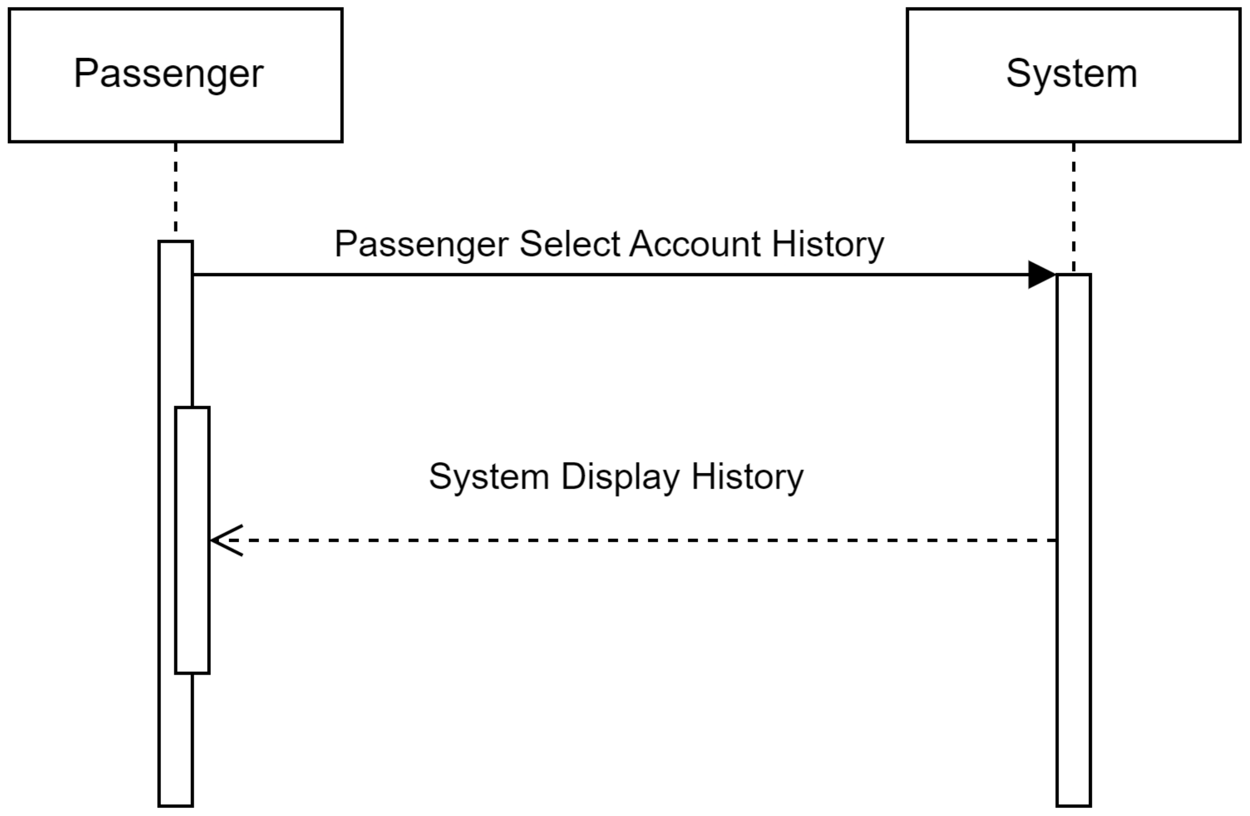
- Use case UC-1



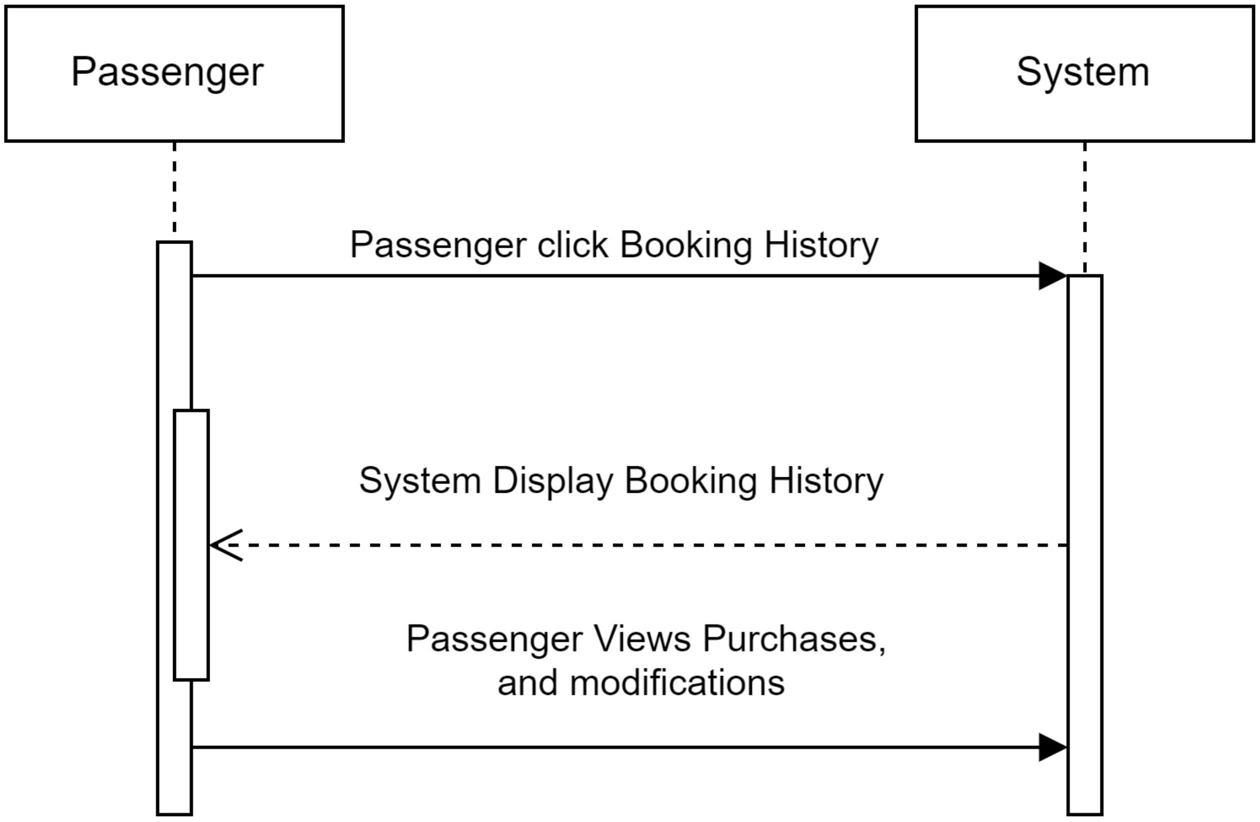
- Use Case UC-6


Passenger
System
Bank
Passenger click Reschedule Ticket
Display Select Seat page
Passenger Selected Seat
Display Payment Options
Passenger Select Payment Option
Prompts the card Info
Passenger Fill Card Info
Bank Authroizations

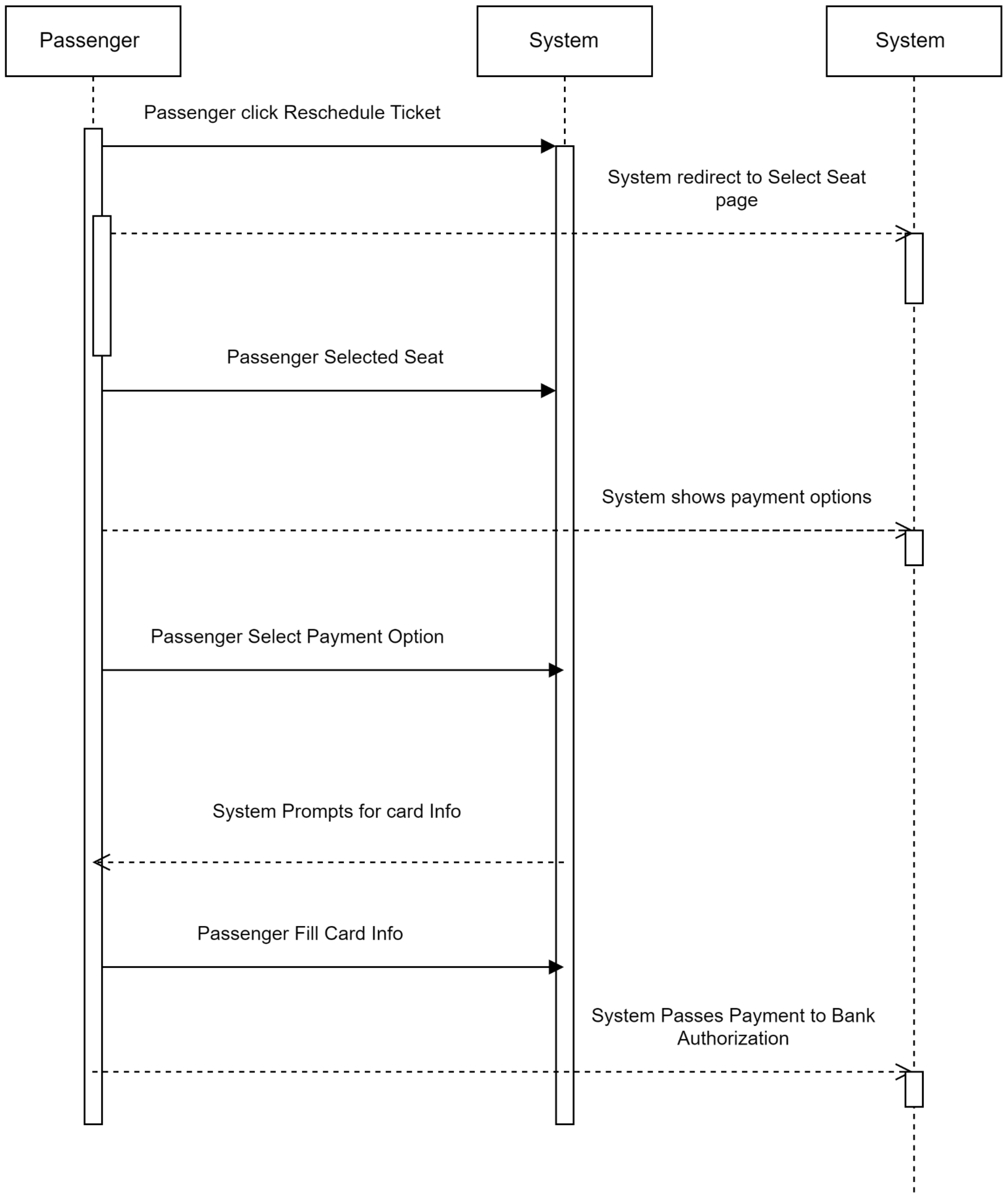
- Use Case UC-9



- Use Case UC -10


Passenger
System
Passenger click Booking History
System Display Booking History
Passenger Views Purchases, and modifications

- Use Case UC -12


Passenger
System
System
Passenger click Reschedule Ticket
System redirect to Select Seat page
Passenger Selected Seat
System shows payment options
Passenger Select Payment Option
System Prompts for card Info
Passenger Fill Card Info
System Passes Payment to Bank Authorization

Effort Estimation

Actor Type	Simple	Average	Complex
Number of Actors	*0*	*1*	*1*
Weightages	1	2	3
Unadjusted Actor Weights (UAW			5

Use Case Type	Simple	Average	Complex
Number of Use Cases	*3*	*2*	*2*
Weightages	5	10	15
Unadjusted Use Case Point (UUCW)			65
Unadjusted Case Point			70
Adjusted Use Case Points			73.1325
Staff Hours Per UCP			28
Effort Estimate			2047.71

Technical Factors	Weightage	Value
Distributed System	2	*3*
Performance Activities	1	*2*
End User Efficiency	1	*4*
Complex Processing	1	*4*
Reusable Code	1	*3*
Easy to Install	0.5	*5*
Easy to Use	0.5	*5*
Portable	2	*4*
Easy to Change	1	*2*
Concurrent	1	*5*
Security Features	1	*3*
Access to Third Parties	1	*2*
Special Training Required	1	*1*
TFactor		45
TCF		1.05

Enviromental Factors	Weightage	Value
Familiar with Development Process	1.5	*3*
Application Experience	0.5	*2*
Paradigm Experience	1	*3*
Lead Analyst Capability	0.5	*4*
Motivation	1	*1*

					Stable Requirements	2	*3*
					Part Time Workers	-1	*1*
					Difficult Programming Language	-1	*3*
					Enviromental Factor		13.5
					ECF		0.995

Domain Analysis

Concept Definitions

UC-1: Register

Responsibility Description	**Concept Name**
Coordinates the actions of all concepts associated with the use case and delegate the work to other concepts.	Controller
Renders the page that is specified after a request is made.	Page Maker
HTML document that displays to the actor what actions can be done and required fields to be filled out in the form.	Interface Page
Form specifying the data for the database upload request.	Upload Request
Verify if all fields of the form were filled out and if it meets the requirements.	Upload Checker
Prepare a database query that uploads the user's login credentials.	Database Connection

Table 1: Concepts for Register

UC-6: Select Trip

Responsibility Description	**Concept Name**
Coordinates the actions of all concepts associated with the use case and delegate the work to other concepts.	Controller
HTML document that displays to the actor what actions can be done and required fields to be filled out in the form.	Interface Page
Prepares a database update query to alter the seats table by changing the availability of a specific seat.	Database Connection
Renders the page that is specified after a request is made.	Page Maker
Retrieves the seat identification of the seat the user requests.	Capture Seat Request
Retrieves the credit card information from the form.	Capture Card Information
Retrieves the route identification to determine which route the ticket is being purchased for.	Capture Ticket Request

Informs the system whether or not the bank was successful with the transaction.	Notifier
Point of interaction between the system and the bank's internal system to conduct transactions (not a part of the system being built).	Bank Application Programming Interface

Table 2: Concepts for Select Trip

UC-9: View Payment History

Responsibility Description	**Concept Name**
Coordinates the actions of all concepts associated with the use case and delegate the work to other concepts.	Controller
Prepare a database query that locates the user's previous payment and booking.	Database Connection
Renders the page that is specified after a request is made.	Page Maker
HTML document that displays to the actor what actions can be done and required fields to be filled out in the form.	Interface Page
Information specifying the user's unique identifier.	Payment Request

Table 3: Concepts for Payment History

UC-10: View Booking History

Responsibility Description	**Concept Name**
Coordinates the actions of all concepts associated with the use case and delegate the work to other concepts.	Controller
Prepare a database query that locates the user's previous booking and retrieves it.	Database Connection
Renders the page that is specified after a request is made.	Page Maker
HTML document that displays to the actor what actions can be done and required fields to be filled out in the form.	Interface Page
Information specifying the user's unique identifier to access the user's record.	Booking Request

Table 10: Concepts for Booking History

UC-12: Reschedule Ticket

Responsibility Description	**Concept Name**
Coordinates the actions of all concepts associated with the use case and delegate the work to other concepts.	Controller
HTML document that displays to the actor what actions can be done and required fields to be filled out in the form.	Interface Page
Renders the page that is specified after a request is made.	Page Maker
Retrieves the seat identification of the seat the user requests.	Capture Seat Request
Retrieves the credit card information from the form.	Capture Card Information
Retrieves the route identification to determine which route the ticket is being purchased for.	Capture Ticket Request
Informs the system whether or not the bank was successful with the transaction.	Notifier
Point of interaction between the system and the bank's internal system to conduct transactions (not a part of the system being built).	Bank Application Programming Interface
Prepares database query able to delete a database record, which best fulfils the user's request and executes it.	Delete Database Record
Prepares database query able to add a database record and executes it.	Add Database Record

Table 12: Concepts for Reschedule Ticket

Association Definitions

UC-1: Register

Concept Pair	**Association Description**	**Association Name**
Page Maker ↔ Interface Page	Page Maker prepares the Interface Page to be displayed	prepares
Controller ↔ Page Maker	Controller passes the specific page request to Page Maker	convey request
Controller ↔ Upload Checker	Controller passes the form information to Upload Checker to check if the information supplied meets the requirements.	convey request
Controller ↔ Interface Page	Controller updates the Interface Page with the requested web page.	posts
Upload Request ↔ Controller	Upload Request provides Controller with the form information	receives
Upload Checker ↔ Database Connection	Upload Checker uploads the checked information to the database.	saves data to

Table 6: Associations for Register

UC-6: Select Trip

Concept Pair	**Association Description**	**Association Name**
Controller ↔ Interface Page	Controller updates the Interface Page with the requested web page.	posts
Controller ↔ Capture Seat Request	Capture Seat Request provides the Controller with the seat identification number.	receives
Controller ↔ Capture Card Information	Capture Card Information provides the Controller with the form information.	receives
Controller ↔ Capture Ticket Request	Capture Ticket Request provides the Controller with the route identification number.	receives

Controller ↔ Database Connection	Controller provides the Database connection with the seat identification	convey-request
Controller ↔ BankAPI	Controller provides the BankAPI with the credit card information of the user.	forward information
Controller ↔ Page Maker	Controller passes the specific page request to Page Maker and receives back page prepared for displaying.	convey-request
Page Maker ↔ Interface Page	Page Maker prepares the Interface Page	prepares
Notifier ↔ Controller	Notifier informs the Controller whether or not the transaction was completed.	notifies
BankAPI ↔ Notifier	BankAPI passes Notifier an indication if the transaction was successful or not.	forwards

Table 7: Associations for Select Trip

<u>UC-9: View Payment History</u>

Concept Pair	**Association Description**	**Association Name**
Page Maker ↔ Interface Page	Page Maker prepares the Interface Page to be displayed.	prepares
Controller ↔ Payment Request	Payment Request provides the Controller with the necessary identifier to locate the user’s payment history.	receives
Controller ↔ Interface Page	Controller updates the Interface Page with the requested web page.	posts
Controller ↔ Database Connection	Controller passes Payment Request query to Database Connection	convey-request

Controller ↔ Page Maker	Controller passes requests to Page Maker and receives back pages prepared for displaying.	convey-request
Database Connection ↔ Page Maker	Database Connection passes the retrieved data to Page Maker to render them for display.	provides data

Table 8: Associations for Payment History

<u>UC-10: View Booking History</u>

Concept Pair	**Association Description**	**Association Name**
Page Maker ↔ Interface Page	Page Maker prepares the Interface Page to be displayed.	prepares
Controller ↔ Booking Request	Payment Request provides the Controller with the necessary identifier to locate the user's booking history.	receives
Controller ↔ Interface Page	Controller updates the Interface Page with the requested web page.	posts
Controller ↔ Database Connection	Controller passes Payment Request query to Database Connection	convey-request
Controller ↔ Page Maker	Controller passes requests to Page Maker and receives back pages prepared for displaying.	convey-request
Database Connection ↔ Page Maker	Database Connection passes the retrieved data to Page Maker to render them for display.	provides data

Table 9: Associations for Booking History

<u>UC-12: Reschedule Ticket</u>

Concept Pair	**Association Description**	**Association Name**
Controller ↔ Interface Page	Controller updates the Interface Page with the requested web page.	posts

Controller ↔ Capture Seat Request	Capture Seat Request provides the Controller with the seat identification number.	receives
Controller ↔ Capture Card Information	Capture Card Information provides the Controller with the form identification.	receives
Controller ↔ Capture Ticket Request	Capture Ticket Request provides the Controller with the route identification number.	receives
Controller ↔ Retrieve Previous Order	Retrieve Previous Order provides the Controller with the ticket order identification number.	receives
Controller ↔ Delete Database Record	Controller passes delete request to Delete Database Record	convey-request
Controller ↔ Add Database Record	Controller passes add request to Add Database Record	convey-request
BankAPI ↔ Notifier	BankAPI passes Notifier an indication if the transaction was successful or not.	forwards

Table 10: Associations for Reschedule Ticket

Attribute Definitions

UC-1: Register

Concept	**Attributes**	**Attribute Description**
Upload Request	user information	First name, last name, email, address, phone number, and password.

Table 11: Attributes for Register

UC-6: Select Trip

Concept	**Attributes**	**Attribute Description**
Capture Card Information	credit card information	Account Number
Capture Card Request	seat's identification	It is used to identify the seat the user would like to buy a ticket for.
Capture Ticket Request	route's identification	It is used to identify the route the user would like to buy a ticket for.

Table 12: Attributes for Select Trip

UC-9: View Payment History

Concept	**Attributes**	**Attribute Description**
Payment Request	user’s identification	It is used to determine the user’s previous purchases.

Table 13: Attributes for Payment History

UC-10: View Booking History

Concept	**Attributes**	**Attribute Description**
Booking Request	user’s identification	It is used to determine the user’s previous bookings.

Table 14: Attributes for Booking History

UC-12: Reschedule Ticket

Concept	**Attributes**	**Attribute Description**
Capture Card Information	credit card information	Account Number

Capture Seat Request	seat's identification	It is used to identify the seat the user would like to buy a ticket for.
Capture Ticket Request	route’s identification	It is used to identify the route the user would like to buy a ticket for.
Retrieve Previous Order	ticket order’s identification	It is used to identify the ticket order the user would like to request.

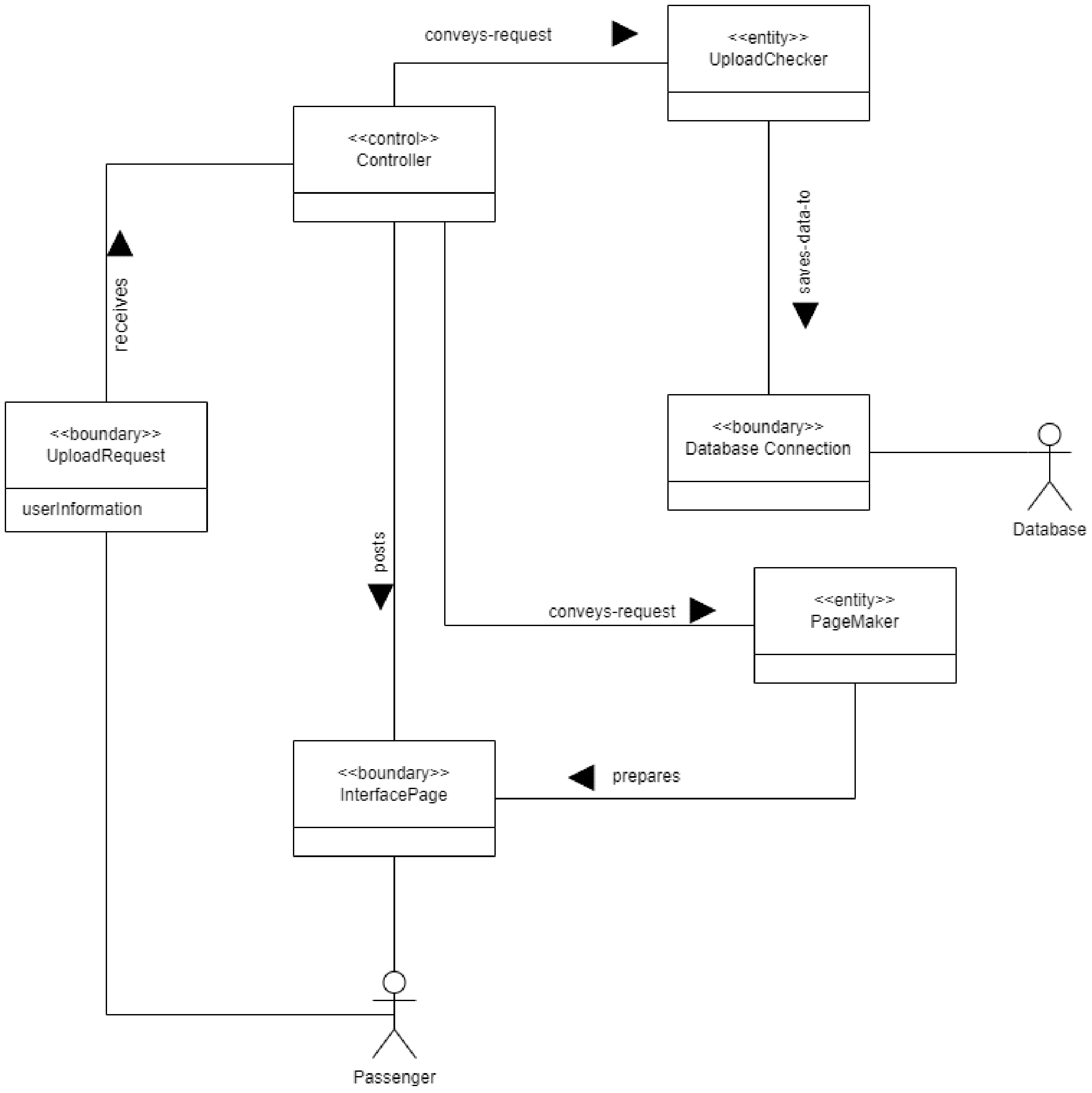
Table 15: Attributes for Reschedule Ticket

Traceability Matrix

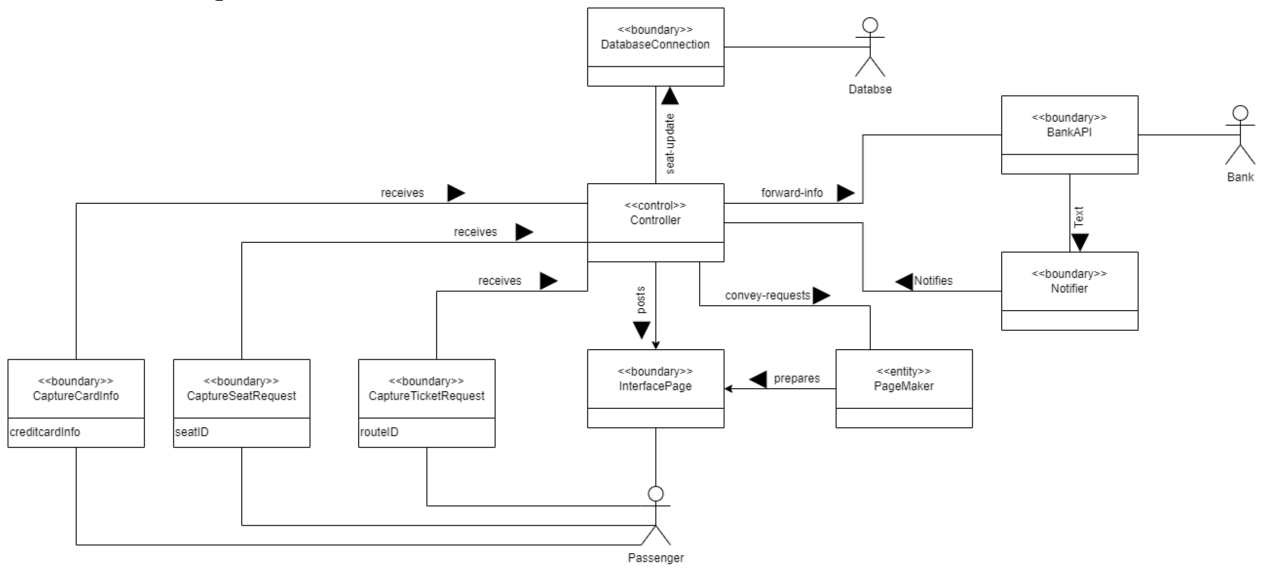
		Domain Concepts														
Use Cases	PW	Controller	Page Maker	Interface Page	Upload Request	Upload Checker	Database Connection	Capture Seat Request	Capture Card Information	Notifier	BankAPI	Payment Request	Booking Request	Capture Ticket Request	Add Database Record	Delete Database Record
UC-1	36	X	X	X	X	X	X									
UC-2	28	X	X	X		X	X									
UC-3	29	X	X	X			X									
UC-4	34	X	X	X	X		X									
UC-5	29	X	X	X			X									
UC-6	36	X	X	X			X	X	X	X	X					
UC-7	36	X	X	X			X	X								
UC-8	53	X	X	X			X		X	X	X					
UC-9	26	X	X	X			X					X				
UC-10	42	X	X	X			X						X			
UC-11	37	X	X	X			X				X					
UC-12	45	X	X	X			X	X	X	X	X			X	X	X
UC-13	31	X	X	X			X	X					X			
UC-14	21	X	X	X												
UC-15	15	X	X	X												
UC-16	26	X	X	X												
UC-17	28	X	X	X												

Domain Models

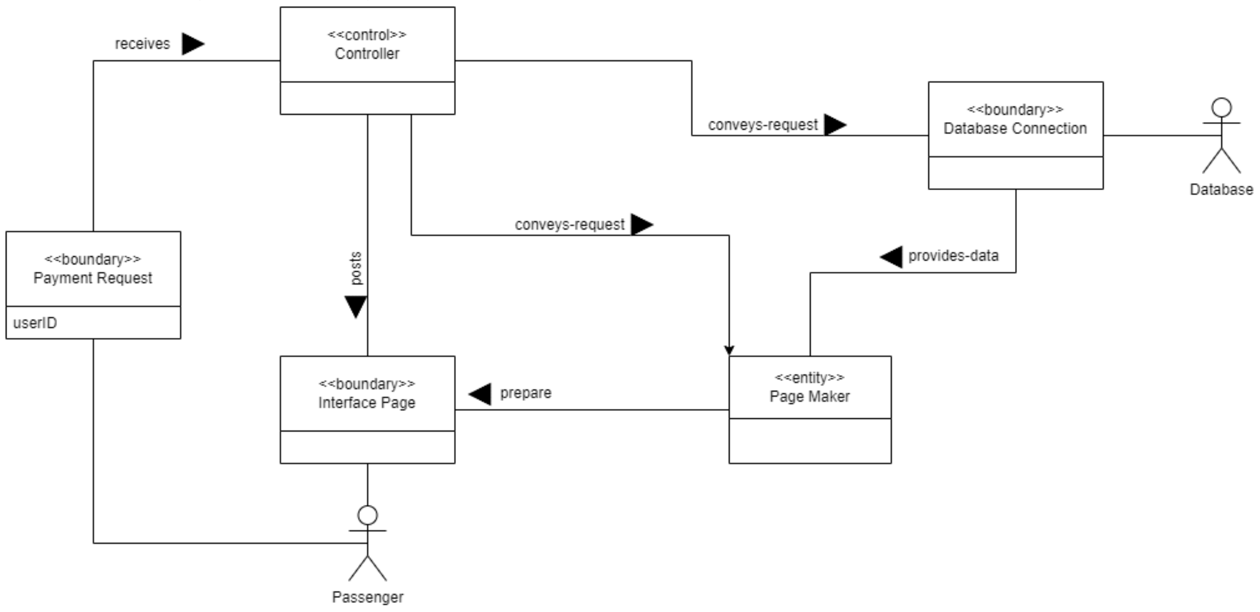
UC-1: Register



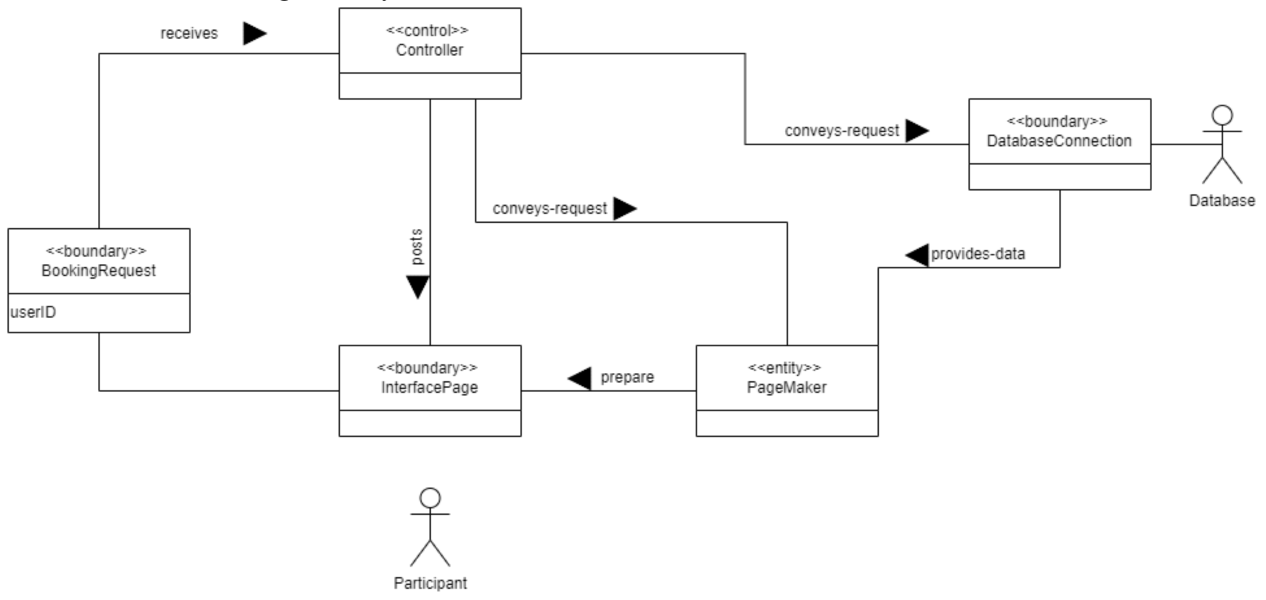
UC-6: Select Trip



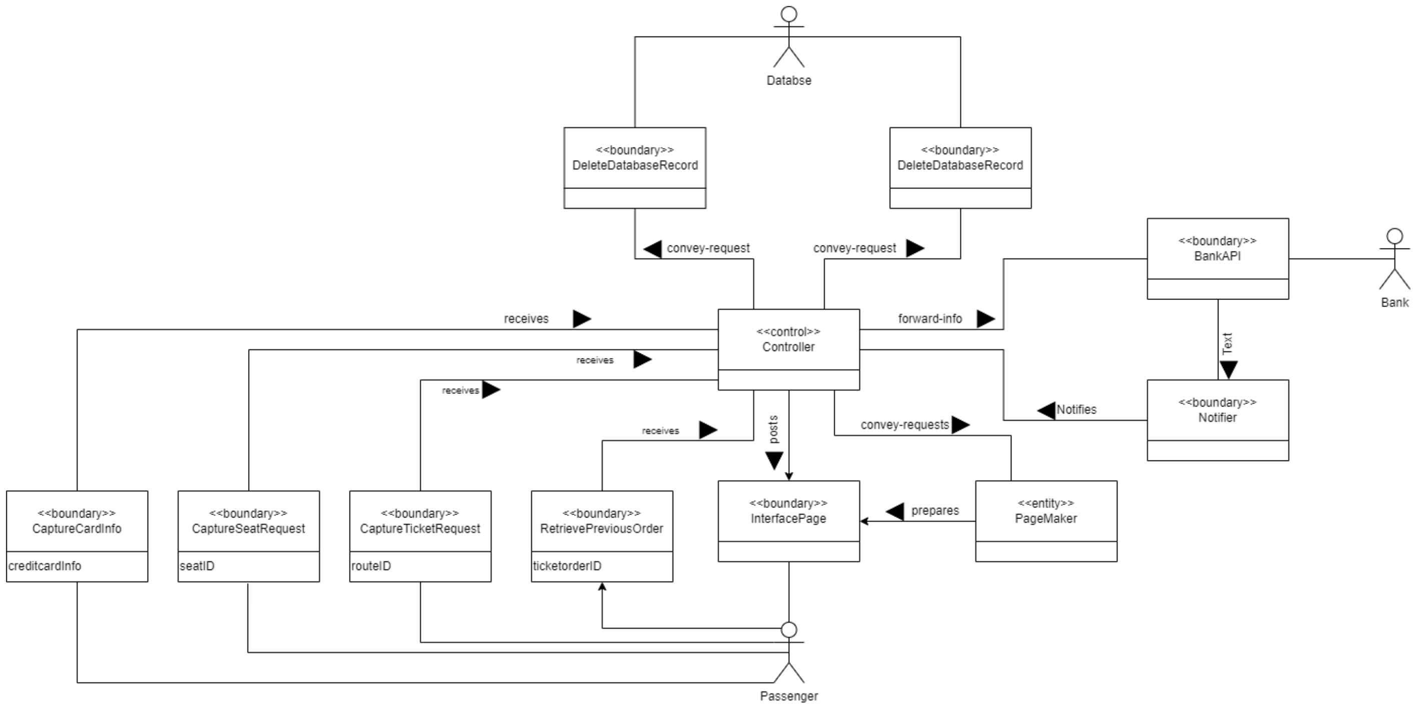
UC-9: View Payment History



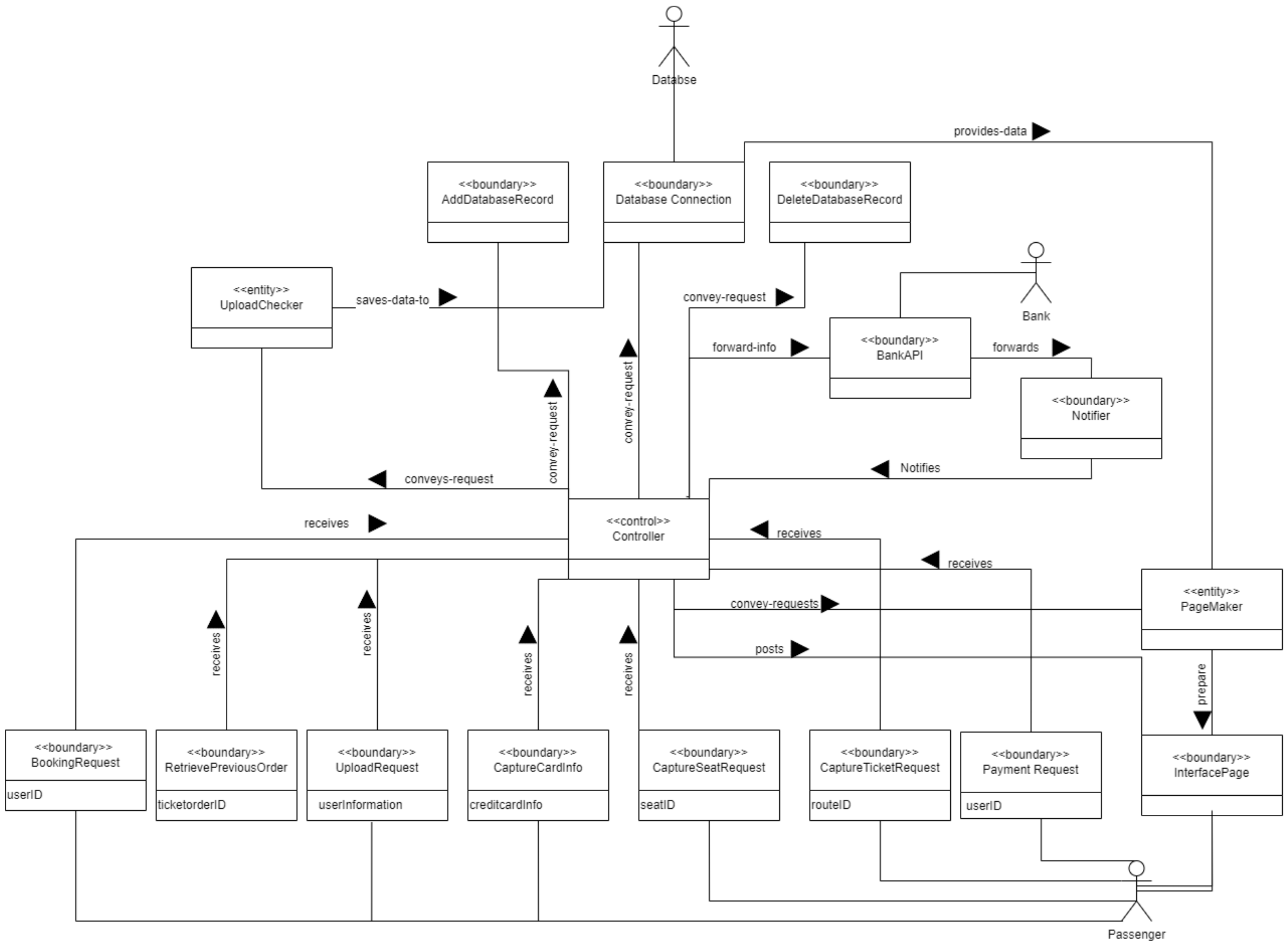
UC-10: View Booking History



UC-12: Reschedule Ticket



Domain Model of the Entire System



System Operation Contracts

Table 16: System Operation Contract for Register

Contract Name	updateUserInfo(fname, lname, email, phonenum, password)
Cross Reference	Use Case: Register
Responsibility	Add user's information into the system
Type	System
Precondition	Internet is available and the user hasn't registers with the same email before.
Postcondition	updateUserInfo is completed and fname, name, email, phonenum, and password are successful entered into the database and the user is now registered.

Table 17: System Operation Contract for Register & Reschedule Ticket

Contract Name	checkSeat(seat identification)
Cross Reference	Use Case: Select Trip & Reschedule Ticket
Responsibility	Check if the seat is available to be selected.
Type	System
Precondition	User has selected a ticket to purchase.
Postcondition	The user is informed whether or not the seat selection has been purchased before the transaction was completed.

Table 18: System Operation Contract for Payment History

Contract Name	requestPaymentHistory(user identification)
Cross Reference	Use Case: Payment History
Responsibility	Locate all records for the previous purchases the user has completed.
Type	System
Precondition	User is registered to the system and has internet connection.
Postcondition	Request is completed and records are returned.

Contract Name	displayPaymentHistory(user identification)
Cross Reference	Use Case: Payment History
Responsibility	Display the Payment History onto the interface as specified.
Type	System
Precondition	Payment history is known.
Postcondition	displayPaymentHistory returns user's information and the handler utilizes the information to display the payment information.

Table 19: System Operation Contract for Reschedule Ticket

Contract Name	requestPreviousOrder(user identification, order identification)
Cross Reference	Use Case: Reschedule Ticket
Responsibility	Locate a previous purchase completed by the user.
Type	System

Precondition	The user has purchased a ticket before.
Postcondition	Request is completed.

Contract Name	removePreviousOrder(user identification, order identification)
Cross Reference	Use Case: Reschedule Ticket
Responsibility	Delete the previous record purchased by the user.
Type	System
Precondition	Request was completed.
Postcondition	The user’s previous purchase was removed.

Interaction Diagrams

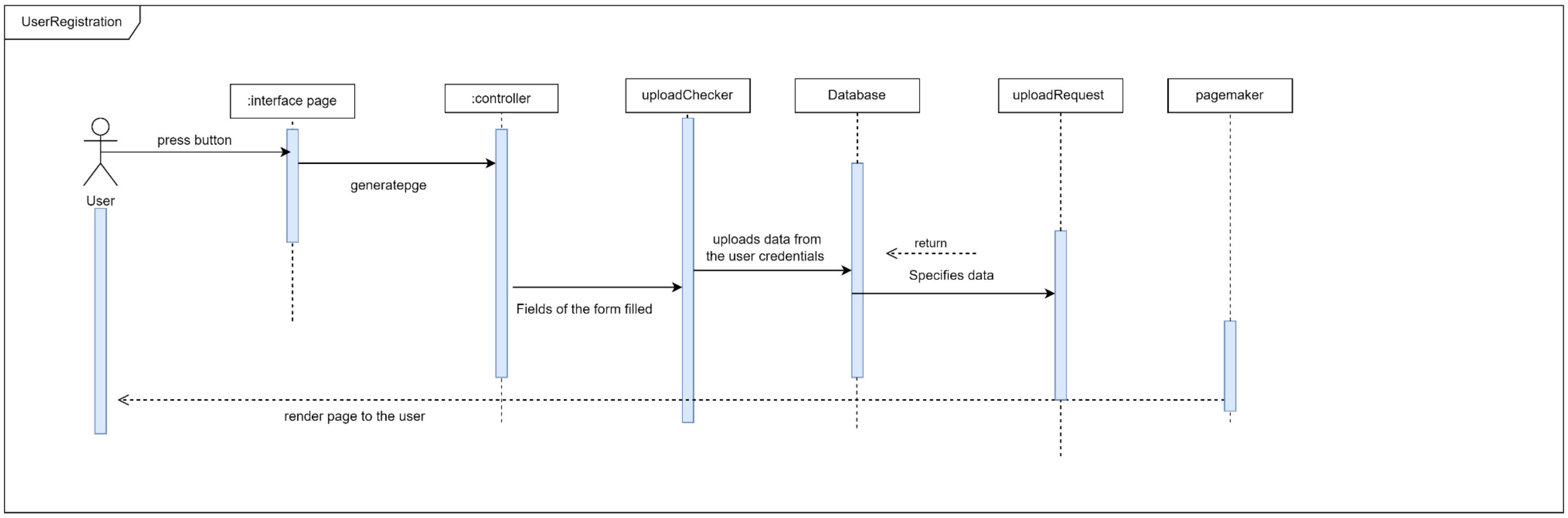


Figure 1: Sequence Diagram for Registration

The archive and subsequent notification of a successful register are the responsibility of the registration or register. The CONTROLLER will coordinate the action of all concepts when the user presses the registration button. After that, PAGE MAKER will display the page that has been requested. After that, the required fields to be filled out and the actor's actions will be displayed on the interface page. After that, the data for the database will be specified by the UPLOAD REQUEST, and the UPLOAD CHECKER will check to see if all of the form's fields were filled in or met the requirements. In conclusion the Data set Associations will set up the data set question that will transfer the client to the login certifications.

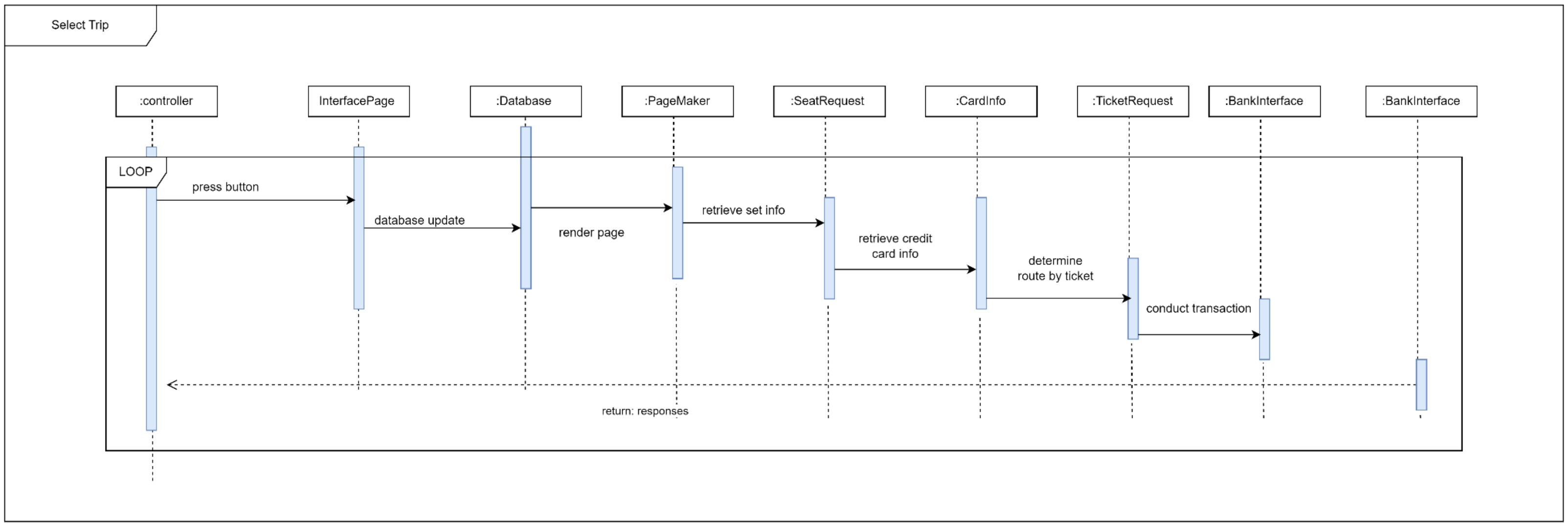


Figure 2 : Sequence Diagram for Select trip

The select trip prerequisite is for the traveler to figure out which pass to buy. The Regulator facilitates the activity, everything being equal. A web page called the INTERFACE PAGE will show the actor's action, which is the system and banking. After that, it will go to the DATABASE CONNECTION and prepare the database query for a specific set of available data. When the data set is done then the PAGE MARKER will deliver the page to the predetermined solicitation that is made following the Catch SEAT Solicitation, this will recovers the seat information that the client mentioned, and the following system will be the Catch CARD Data where by this will recover the Mastercard data from the structure. The BANK Connection point is essentially the place of association between the framework and the banking interior framework. Last but not least is the NOTIFIER, which will inform the system of the bank's success or failure with the transaction.

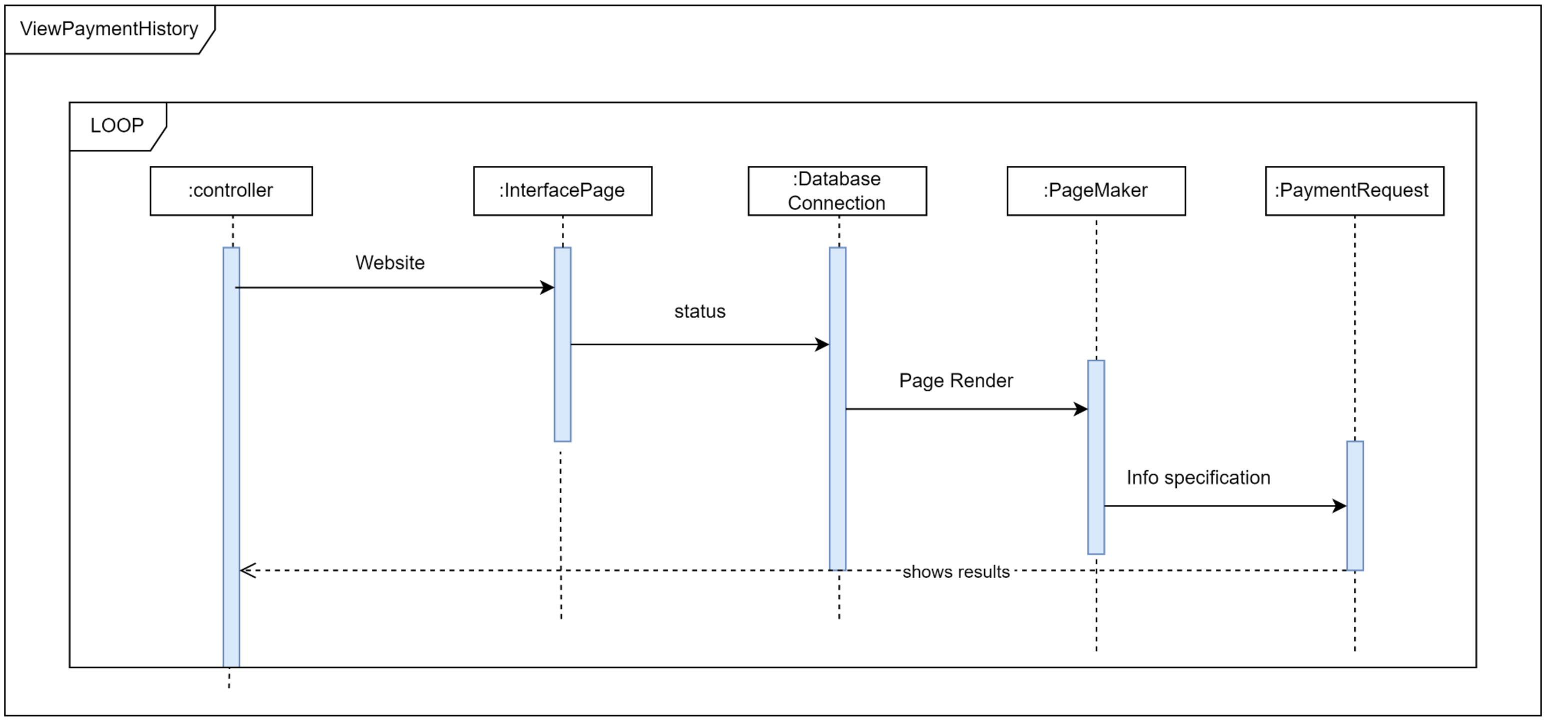


Figure 3 : Sequence diagram of View payment History

The purpose of View payment history is to show the passenger's past transactions. The Regulator will organize the activity of all ideas and will take the ongoing solicitation. It moves on to the DATABASE CONNECTION, where it creates the database query that looks up the user's previous booking and payment history. After that, it will return to the PAGE MAKER, where the request was made, the INTERFACE PAGE, which displays the actor's actions, and the PAYMENT REQUEST, in which the user's unique identifier is specified.

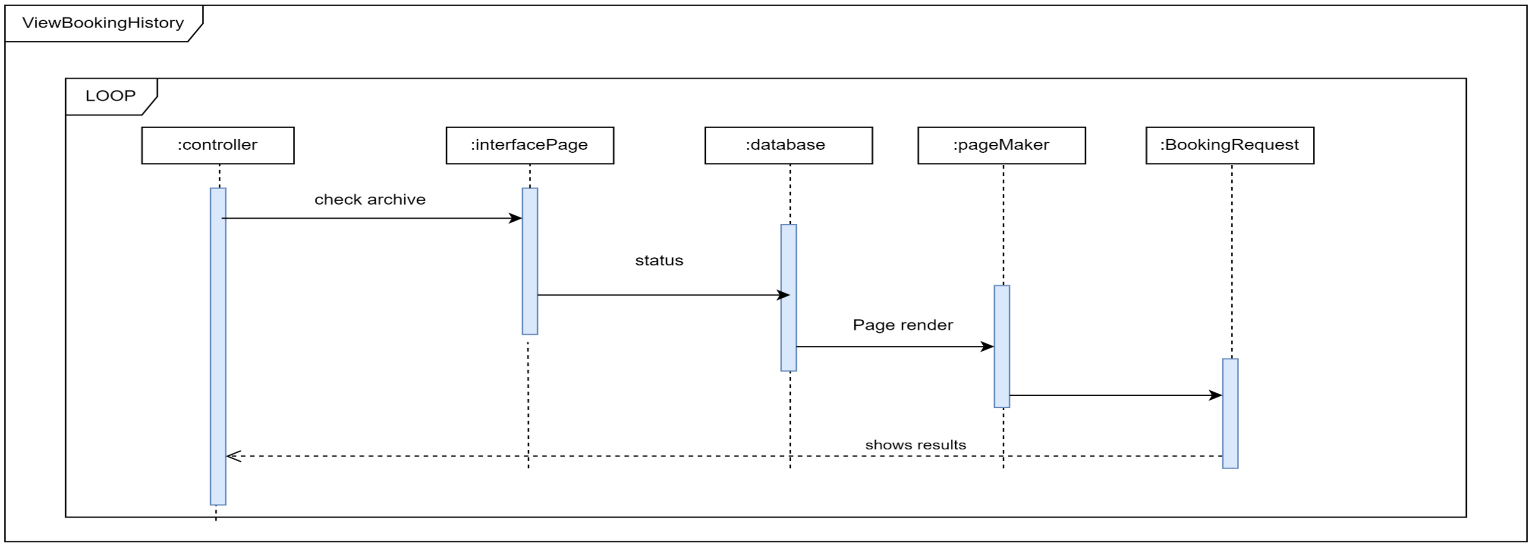


Figure 4: Sequence Diagram od View Booking History

This diagram allows passengers to view both previous and current purchases. First it goes to the Regulator which will organize all activities and from the ongoing solicitation the Data set Association will set up the information base and the area the client's recently reserved. After that, the PAGE MAKER will display the page specified in the request, and the INTERFACE PAGE will explain what the user can do to prompt them to complete the form. What's more, finally the BOOKING Solicitation will determine the client's identifier to get to their records.

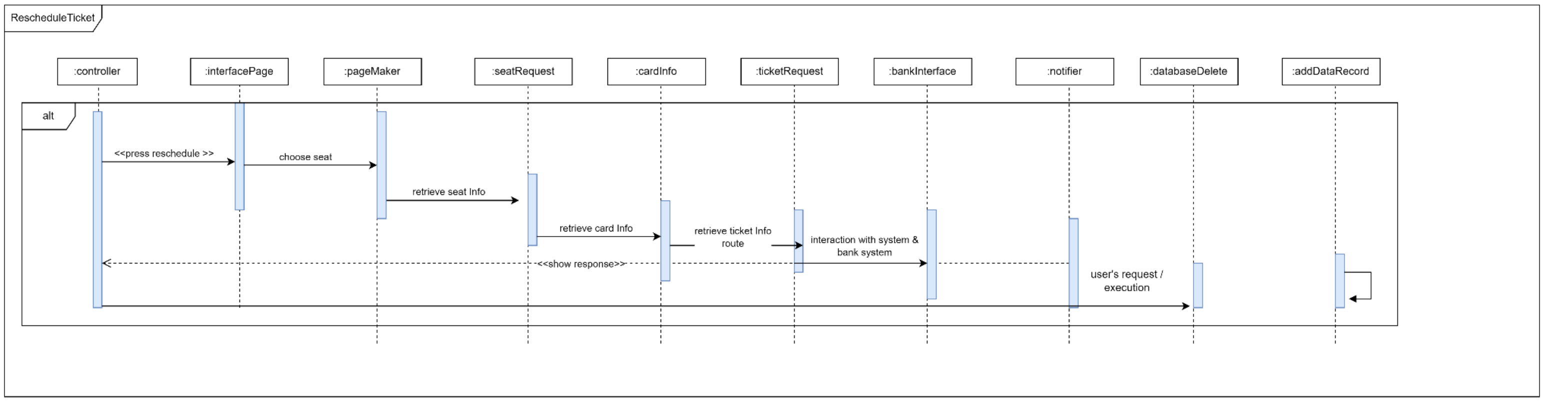
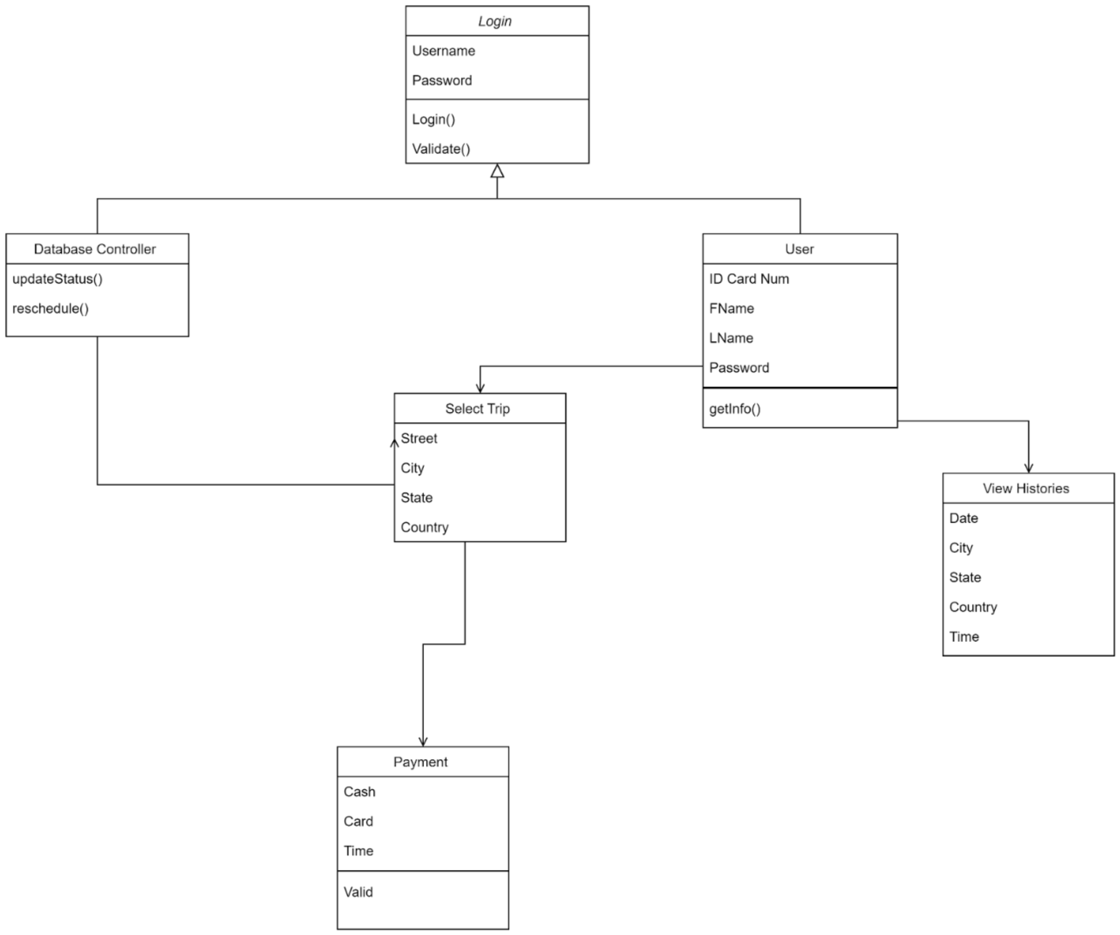


Figure 5: Sequence Diagram of Reschedule Ticket

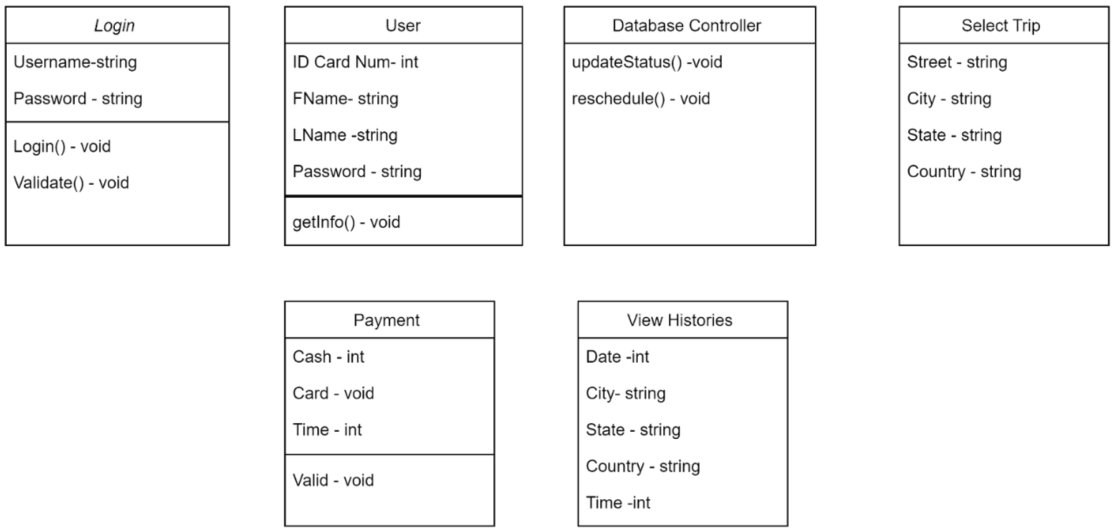
The user can choose which ticket to reschedule using the Reschedule Ticket. After the CONTROLLER coordinates the user's current request, the user will be presented with an interface page to complete a web form. at the point when the structure is finished the page will be delivered by the PAGE Creator and the CAPTURE SEAT will recover the seat distinguishing proof of the seat the client mentioned, and the card data will be recovered by the CAPTURE CARD Data, and the CAPTURE TICKET will recover the defeat data to figure out which course the ticket is being bought for. The NOTIFIER will tell the system whether or not the payment went through when the process is finished. The DELETE DATABASE RECORD command then instructs the database to delete all records that best satisfy the user's request. Last but not least, the ADD DATABASE RECORD option instructs the database query to add a new record and carry it out.

Class Diagram & Interface Specification

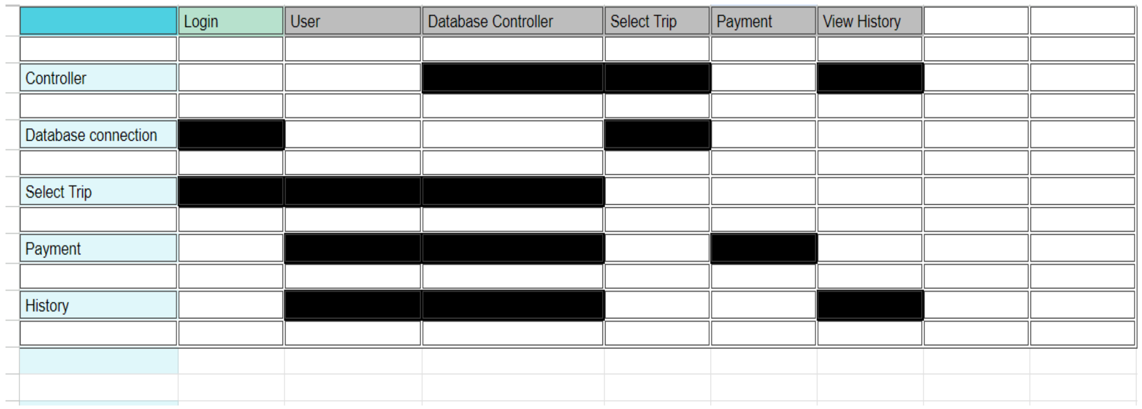
Class Diagram



6.2 Data Types and Operation Signature



6.3 Traceability Matrix



	Login	User	Database Controller	Select Trip	Payment	View History
Controller			■	■		■
Database connection	■			■		
Select Trip	■	■	■			
Payment		■	■		■	
History		■	■			■

Figure showing the Traceability Matrix

Login:

The login enables the user to access the mobile application and communicates with the database connection to determine whether the username and password are valid. It likewise utilizes the select excursion connection point to provoke the client to sign in and select where he/she needs to go.

User:

The User is the main branch of the domain models.

Database Controller:

Throughout the system's lifespan, it is in charge of keeping in touch with the database and making attempts to reconnect if the connection is lost. Additionally, it must respond to each function call requesting database access. The connection serves as the channel for all queries.

Select Trip:

Evolved from a controller that dealt with specific controller locations or users.

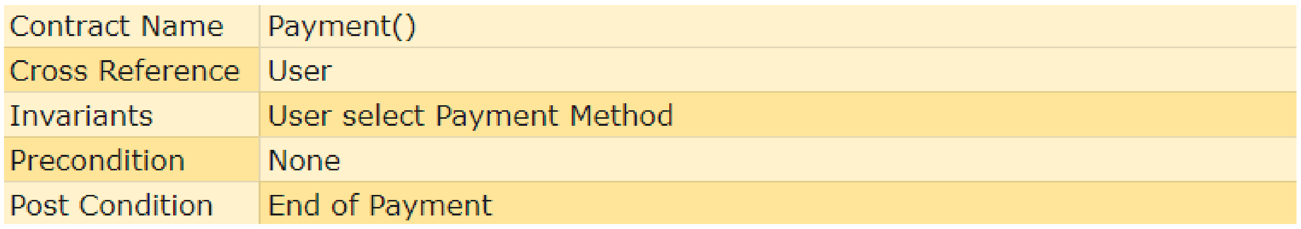
Payment:

The user's method of payment, whether by cash or credit card, is the responsibility of the Payment.

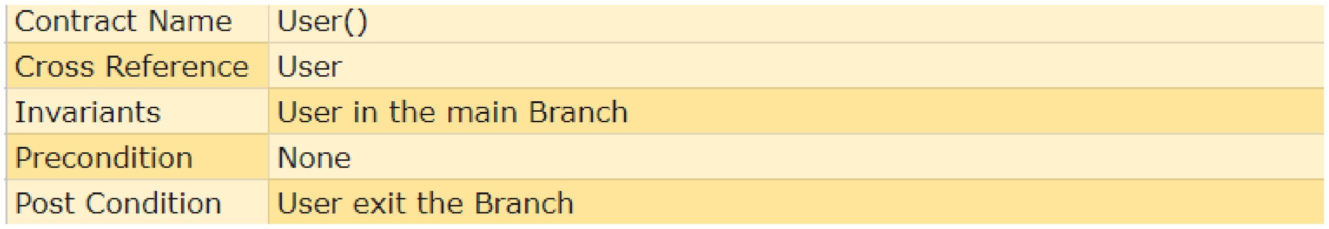
History:

Users can view past and current payments made and tickets purchased using the user's method of history.

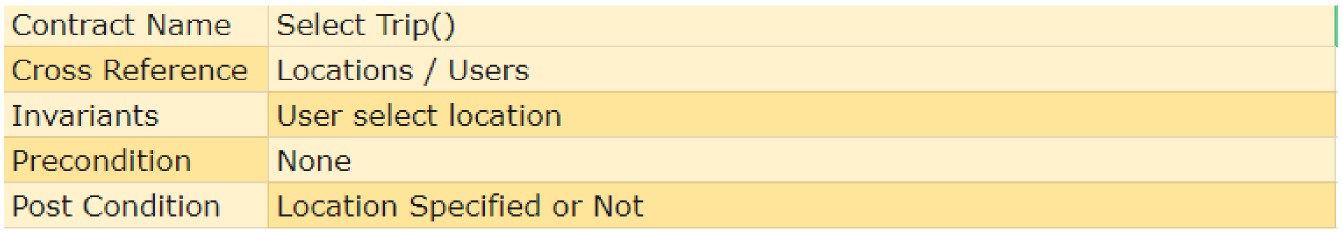
Object Constraint Language (OCL) Contract



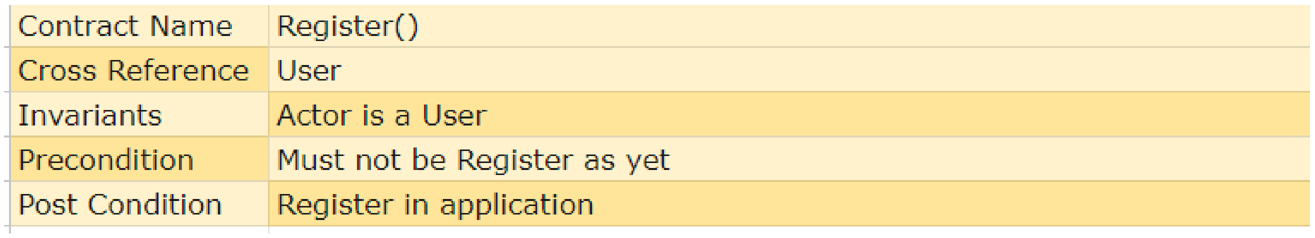
Contract Name	Payment()
Cross Reference	User
Invariants	User select Payment Method
Precondition	None
Post Condition	End of Payment



Contract Name	User()
Cross Reference	User
Invariants	User in the main Branch
Precondition	None
Post Condition	User exit the Branch



Contract Name	Select Trip()
Cross Reference	Locations / Users
Invariants	User select location
Precondition	None
Post Condition	Location Specified or Not



Contract Name	Register()
Cross Reference	User
Invariants	Actor is a User
Precondition	Must not be Register as yet
Post Condition	Register in application

System Architecture & System Design

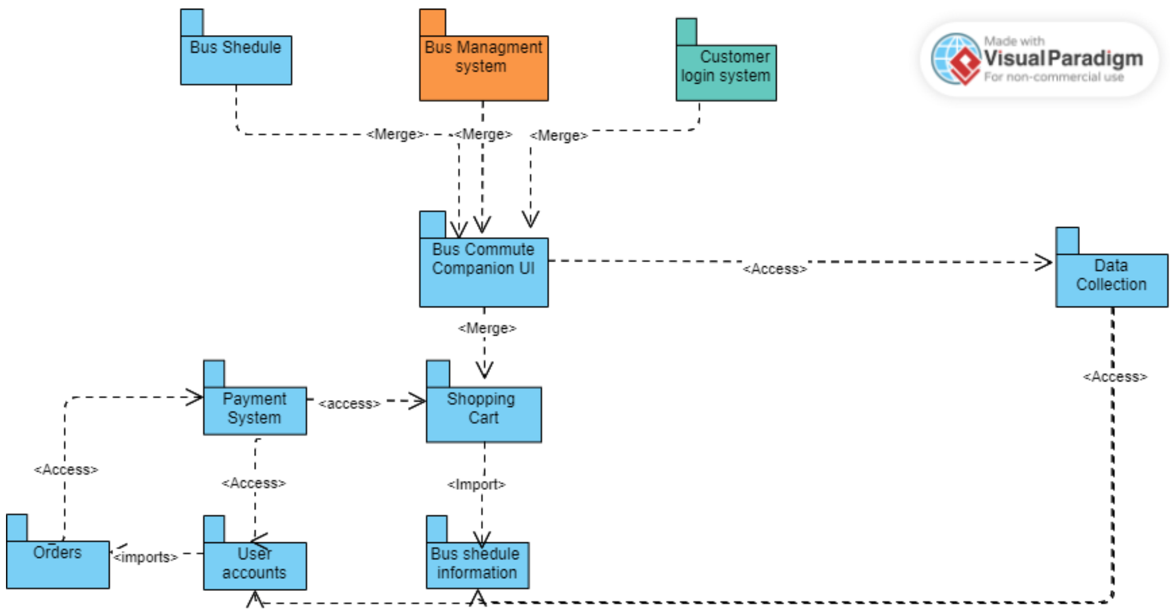
Architecture Styles

This bus ticketing system will implement three architecture styles: Software AS A Service, Real-time Computing, and Data-centric architectures.

Software As A Service(SAAS): This system will be hosted online so the passengers can use the web-based application on their devices. The server for the system will be hosted online. The users will only need an internet connection, and they will be able to use the ticketing system.

Data-Centric and real-time computing (RTC): This ticketing system is significantly data-driven, and all data will be saved on a database. Every detail will be saved on the database, and the database will be updated in real-time. There will not be any delays. It is crucial to the system that the database is updated as the user makes a change so that no clashes happen. This system needs to have high reliability and availability.

Identifying Subsystems



Mapping Subsystems to Hardware

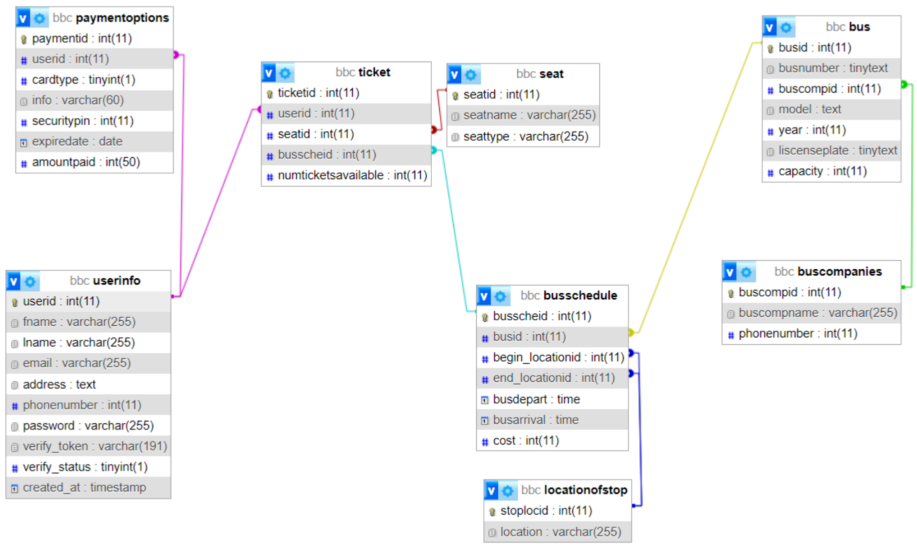
The server will handle tickets that will be issued to the customers.

The server will also handle the flow of payments for the respected bus companies.

The server will also help gather Customer data(the routes they take, What times they usually take the bus) this will help the traffic department and by extension the government when planning new routes and the addition of new bus companies.

Persistent Data Storage

Database:



This database represents a schema for persistent data storage. It includes several tables with their respective structures and relationships. Here's an explanation of the tables and their purpose:

bus: This table stores information about buses.

Columns: busid, busnumber, buscompid, model, year, liscenseplate, capacity.

Relationship: The buscompid column references the buscompid column in the buscompanies table.

buscompanies: This table stores information about bus companies.

Columns: buscompid, buscompname, phonenumber.

busschedule: This table stores information about bus schedules.

Columns: busscheid, busid, begin_locationid, end_locationid, busdepart, busarrival, cost.

Relationships: The busid column references the busid column in the bus table. The begin_locationid and end_locationid columns reference the stoplocid column in the locationofstop table.

locationofstop: This table stores information about the locations where buses make stops.

Columns: stoplocid, location.

paymentoptions: This table stores information about payment options.

Columns: paymentid, userid, cardtype, info, securitypin, expiredate, amountpaid.

Relationship: The userid column references the userid column in the userinfo table.

seat: This table stores information about seats on buses.

Columns: seatid, seatname, seattype.

ticket: This table stores information about tickets.

Columns: ticketid, userid, seatid, busscheid, numticketsavailable.

Relationships: The userid column references the userid column in the userinfo table. The seatid column references the seatid column in the seat table. The busscheid column references the busscheid column in the busschedule table.

userinfo: This table stores information about user profiles.

Columns: userid, fname, lname, email, address, phonenumber, password, verify_token, verify_status, created_at.

The tables are connected through foreign key constraints, which ensure data integrity and maintain relationships between related entities.

Overall, this database schema is designed to store information related to buses, bus companies, bus schedules, locations, payment options, seats, tickets, and user profiles.

Connectors and Network Protocols

HTTPS: This application will be hosted on the web. Since our web application will be processing transactions and be a place where users log in and input their personal information, this website must be hosted on a secured website.

PHP: It will be used to establish the website and server communication and when someone is connecting via Computer/Phone via Browser.

AJAX PHP: When establishing the communication between the app and the server. When someone is connecting via a smartphone.

TCP: This service will be using the TCP protocol to have a reliable transmission of data over the internet when communicating between the server and the web application/ smartphone application.

Global Control Flow

Execution Orderness: For this bus ticketing system, the execution orderliness is event-driven. This will allow the users to use and execute any of the subsystems on the system, be it reserving a seat or canceling a seat. For this to work, the server will be in a loop, constantly waiting for the events to be executed by the users.

Time dependency: For this system, the time dependence is real-time dependency. As stated above, the server will always be waiting for any updates. This is the most important part of the system because the available seats must be updated as they become accessible, and this must be shown on the application.

Hardware Requirements

Mobile Requirements:

OS: Android 4.0 or greater ; Apple IOS 6.0 or greater

Storage : 2GB

Screen Resolution: Minimum Screen Resolution of 640 X 480

Network Bandwidth: 60 kbs

PC Requirements:

OS: Any os that can run the following software below

Web Browsers: Chrome, Version 4 or later
Firefox, Version 4 or later
Safari, Version 4 or later
Opera, Version 4 or later

Network: Broadband Internet connection is required

DATA STRUCTURES

The Bus Commute Companion application will utilize several data structures, namely arrays, linked lists, hash tables, and JavaScript Object Notation (JSON). The following section provides a detailed explanation of each of these data structures.

1. Arrays

In the Bus Commute Companion application, arrays will be used to store and retrieve information such as the list of available bus routes, bus schedules, and ticket prices. This data structure is useful because it allows a single variable to store multiple values of the same type, making it easier to manage and manipulate the data. The PHP language will be used as a back end to interact with the database and retrieve the data, such as the list of available routes or schedules, which will be stored in arrays. When a user searches for a bus route, the array will be queried, and the matching route details will be displayed to the user.

2. Linked Lists

Linked Lists will be used in Bus Commute Companion to manage the data of passengers, trips, and bookings. For example, a linked list of passengers can be used to store the information of all passengers on a particular trip. Each node in the linked list will contain information such as the passenger's name, age, and contact information. Linked list will also manage the queue of passengers waiting to purchase a ticket and track the history of bookings and trips for each passenger. Each passenger will have their own linked list containing their booking history, which will be updated when they make a new booking or modify existing ones.

3. Hash Tables

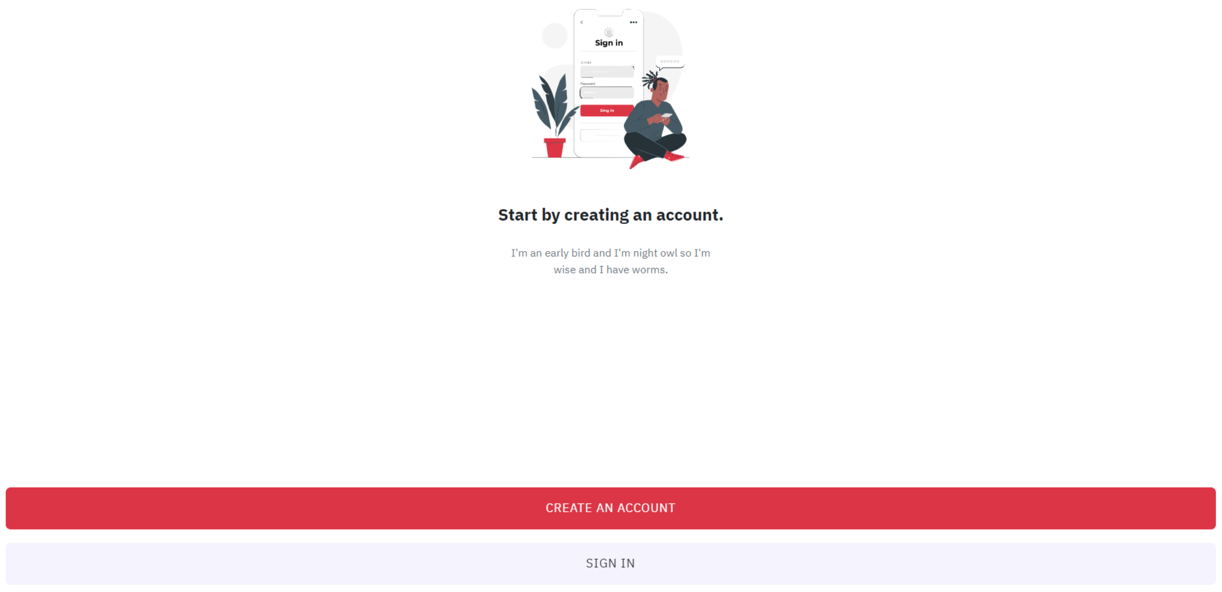
In the Bus Commute Companion application, hash tables will be useful for storing and accessing data related to routes and schedules, such as the departure time, arrival time, and stop locations. Since hash tables can search for data quickly based on the key, it will improve the performance and efficiency of the application, making it more user-friendly for customers.

Criteria Used To Choose The Data Structure

The criteria used to choose these data structures for Bus Commute Companion include:

1. Efficiency and speed of data retrieval: Since the application is targeting availability and reliability, search queries should yield almost immediate results. As a result, arrays will be used because they are fast and easy to traverse.
2. Security: To ensure that user data is secure, we will use hash tables to store sensitive information such as passwords. Hash tables are more secure than arrays as they use a hashing function to store data in a way that makes it difficult to reverse engineer.
3. Scalability: To ensure the app can handle a large number of users and data, arrays and hash tables can be easily scaled up by adding more memory or processing power, while linked lists can be optimized for specific use cases to improve performance.

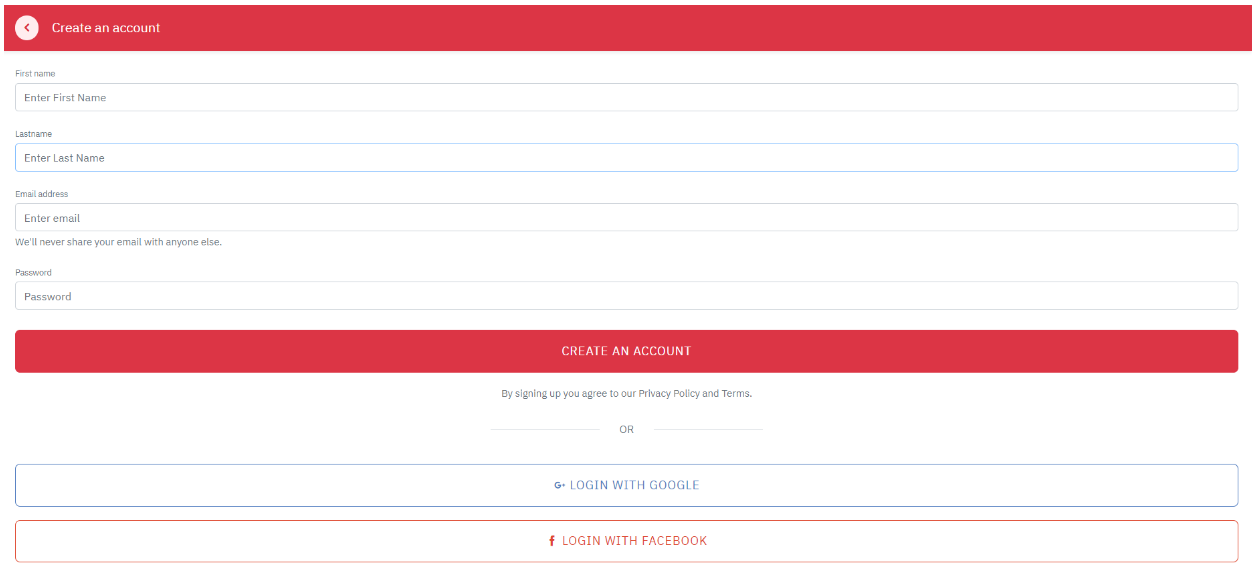
User Interface Design & Implementation



Start by creating an account.

I'm an early bird and I'm night owl so I'm wise and I have worms.

Welcome to the startup boot screen for Belize Bus Companion As the message tell you The customer that you can enter the website by either creating a new Account or By signing in.


Create an account

First name

Enter First Name

Lastname

Enter Last Name

Email address

Enter email

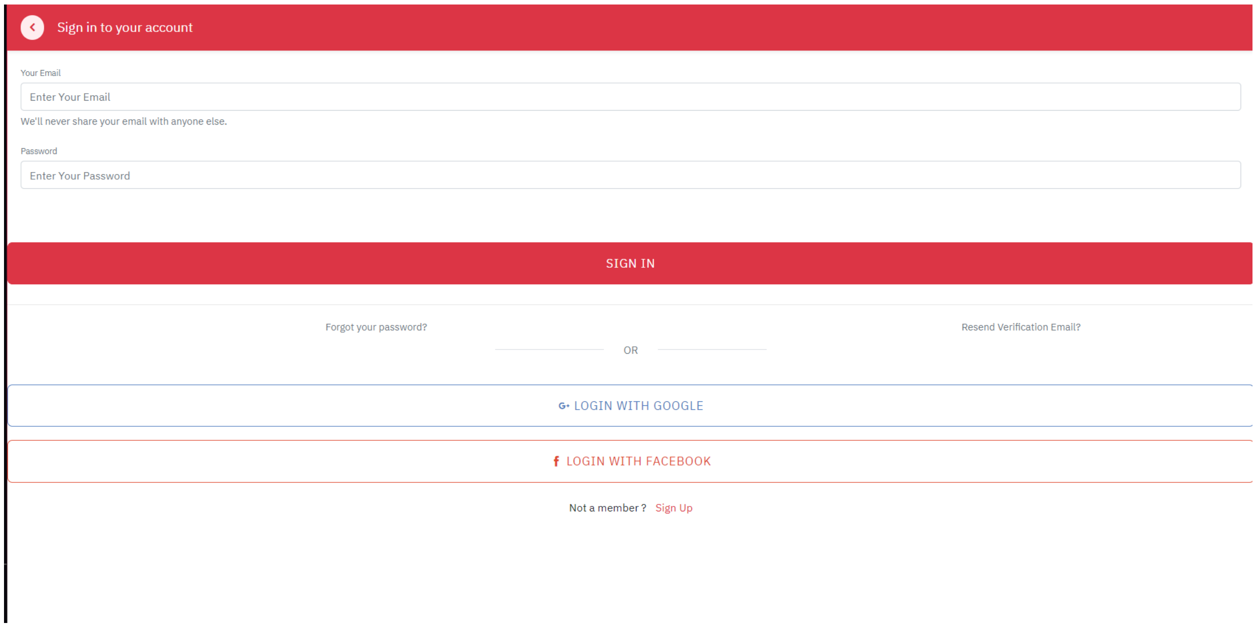
We'll never share your email with anyone else.

Password

Password

By signing up you agree to our Privacy Policy and Terms.

For those who do not have an account you should start by signing up using your email address with Your First name, Last name and Password. This will prompt you to get an email verification message that will authenticate you automatically before Logging into our Website.


Sign in to your account

Your Email

Enter Your Email

We'll never share your email with anyone else.

Password

Enter Your Password

SIGN IN

Forgot your password?

Resend Verification Email?

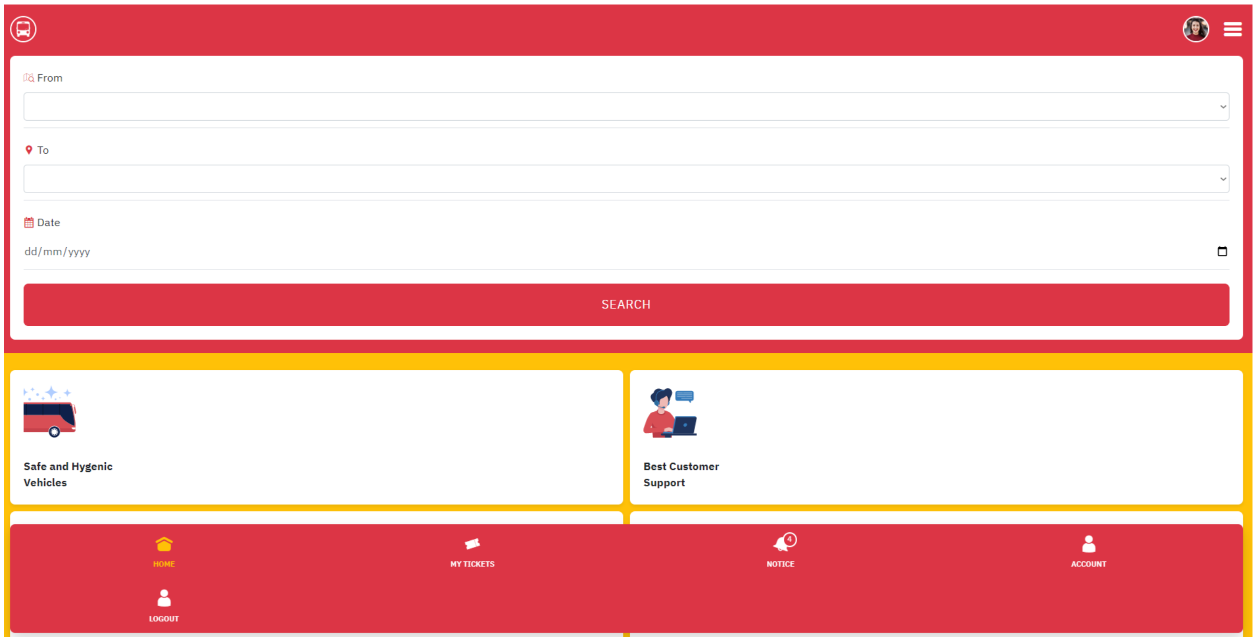
OR

LOGIN WITH GOOGLE

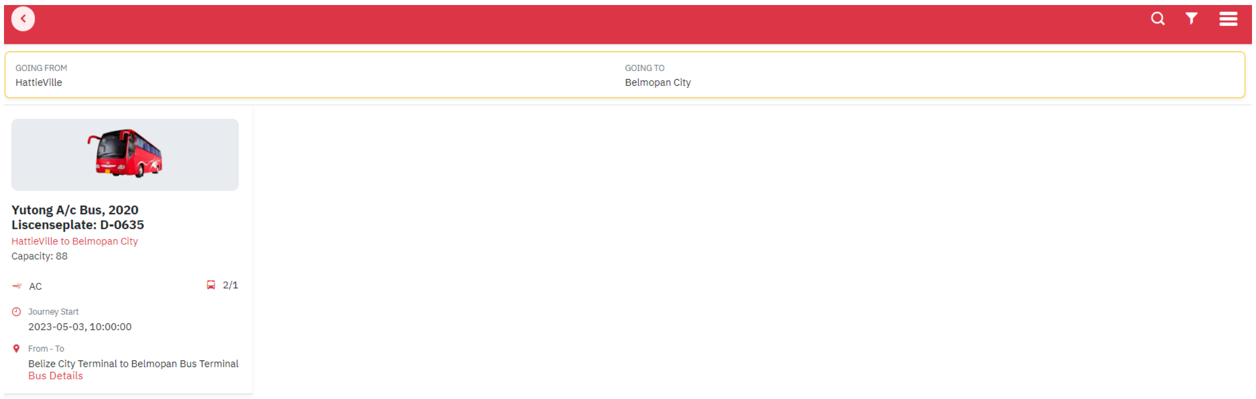
LOGIN WITH FACEBOOK

Not a member ? Sign Up

For those that Do have an account with use I encourage you to Login with your verified email address and password. Doing so will give you the customer access to our program



Once logged in you will be greeted by our home screen with the search bar immediately available to you. You are to select your desired pickup point and your destination of your drop off point along with the date that you want to travel.



GOING FROM
HattieVille

GOING TO
Belmopan City

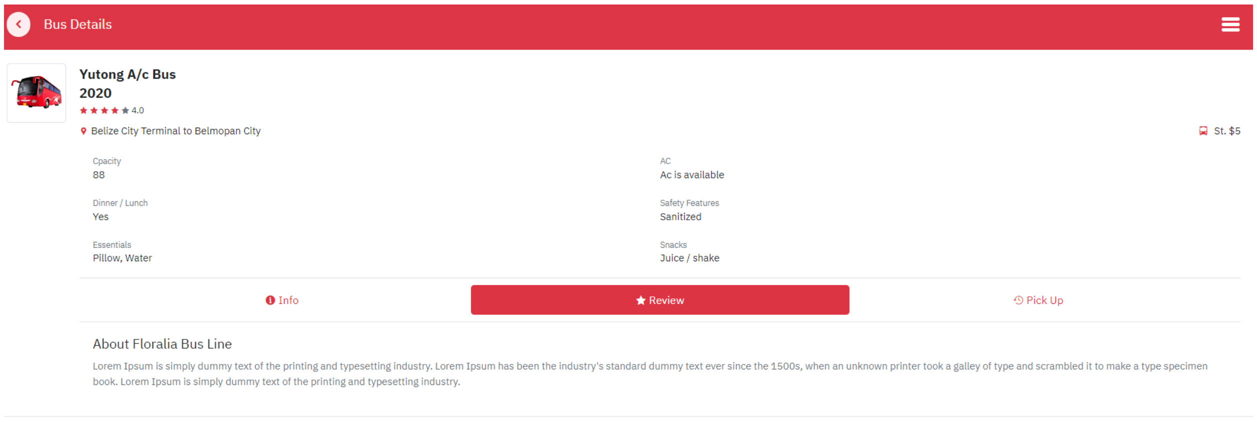
Yutong A/c Bus, 2020
Liscenseplate: D-0635
HattieVille to Belmopan City
Capacity: 88

AC 2/1

Journey Start
2023-05-03, 10:00:00

From - To
Belize City Terminal to Belmopan Bus Terminal
Bus Details

Once the search is active you will see the desired bus that is right for you along with the desired information clicking on one of the busses with send you to the bus detail



Bus Details

Yutong A/c Bus
2020
★★★★★ 4.0

Belize City Terminal to Belmopan City

St. $5

Cpacity 88	AC Ac is available
Dinner / Lunch Yes	Safety Features Sanitized
Essentials Pillow, Water	Snacks Juice / shake

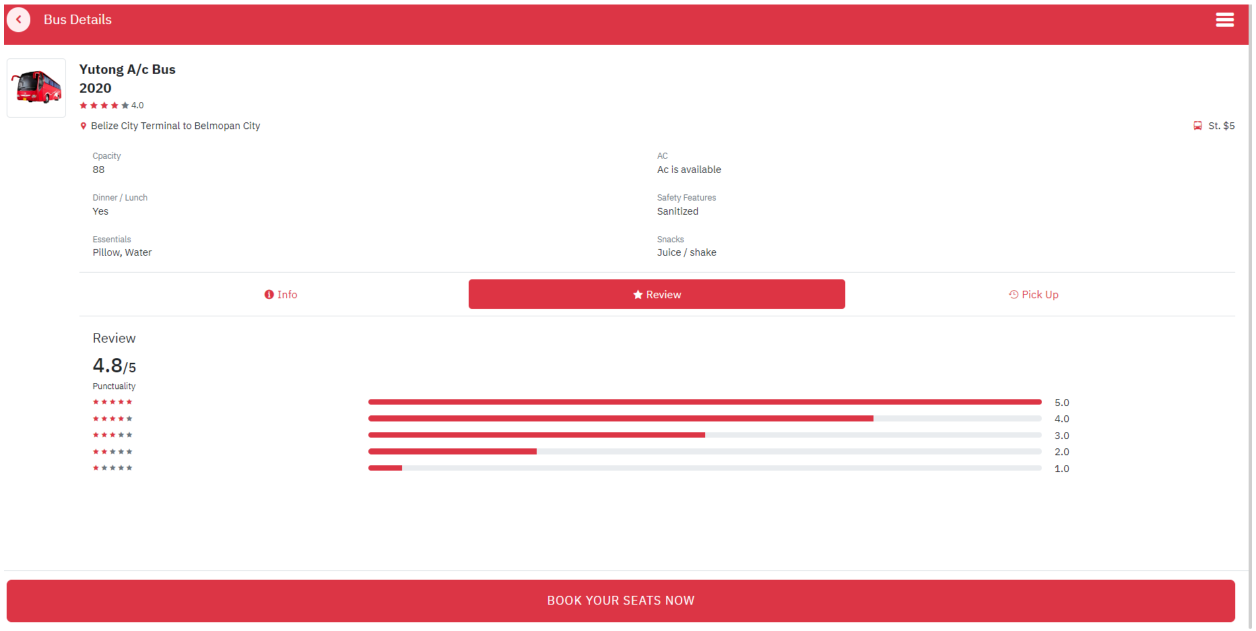
Info | Review | Pick Up

About Floralia Bus Line

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. Lorem Ipsum is simply dummy text of the printing and typesetting industry.

Here It will give you more information about the bus along with information about pick up

The program will then prompt you by a red button saying book seat now



Bus Details

Yutong A/c Bus
2020
★★★★★ 4.0

Belize City Terminal to Belmopan City

St. $5

Cpacity 88	AC Ac is available
Dinner / Lunch Yes	Safety Features Sanitized
Essentials Pillow, Water	Snacks Juice / shake

Info | Review | Pick Up

Review

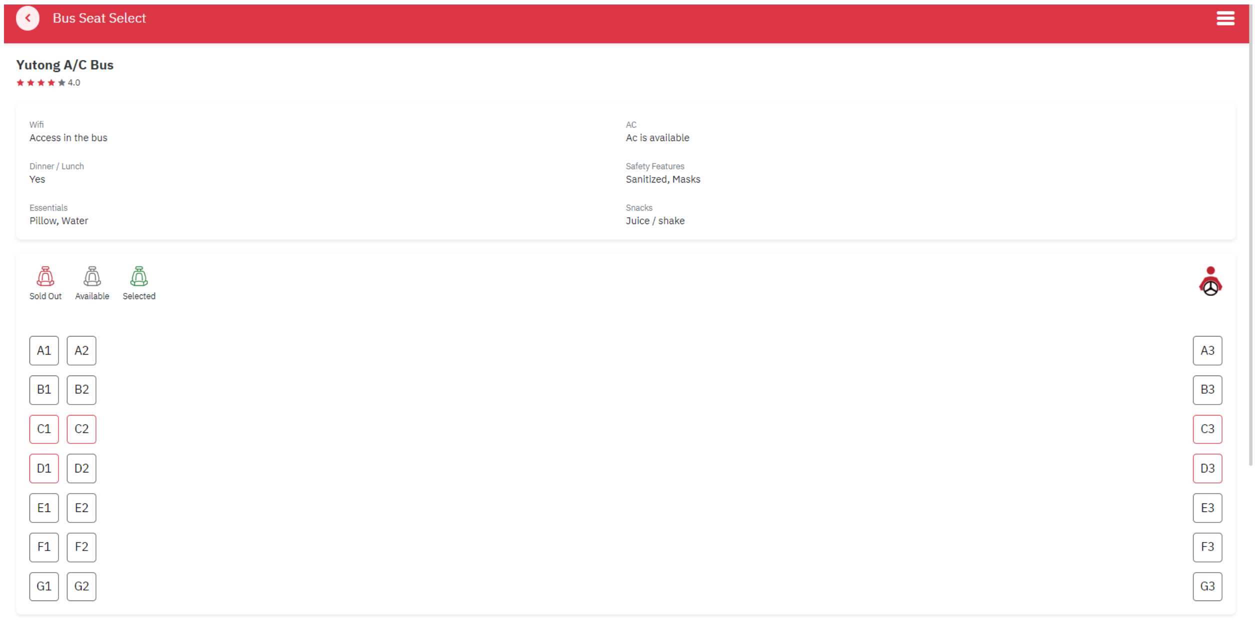
4.8/5

Punctuality

★★★★★	5.0
★★★★★	4.0
★★★★★	3.0
★★★★★	2.0
★★★★★	1.0

BOOK YOUR SEATS NOW

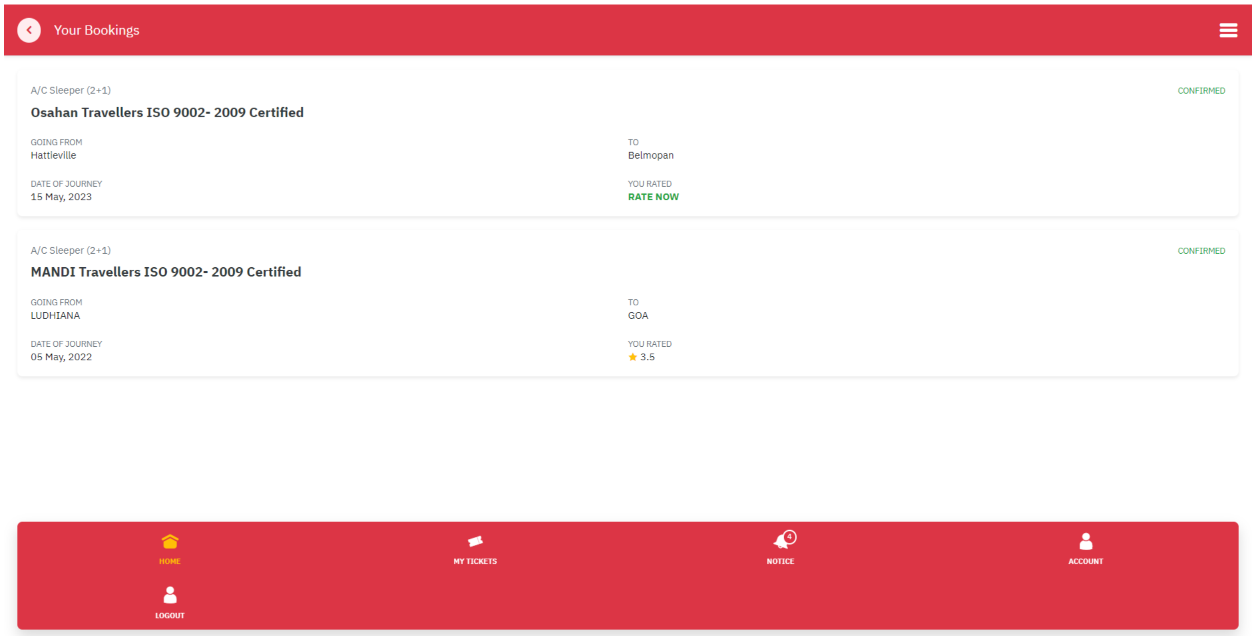
This action will take you to reserve your seat on that specific bus


Bus Seat Select

Yutong A/C Bus
★★★★★ 4.0

Wifi Access in the bus	AC Ac is available
Dinner / Lunch Yes	Safety Features Sanitized, Masks
Essentials Pillow, Water	Snacks Juice / shake

After reserving your seat on the bus it will give you the ticket information along with a prompt that say buy Now After you bought your ticket you are then sent back to the home screen but now if you go to my ticket you can now see all of the ticketing information


Your Bookings

A/C Sleeper (2+1) CONFIRMED
Osahan Travellers ISO 9002- 2009 Certified

GOING FROM Hattieville	TO Belmopan
DATE OF JOURNEY 15 May, 2023	YOU RATED RATE NOW

A/C Sleeper (2+1) CONFIRMED
MANDI Travellers ISO 9002- 2009 Certified

GOING FROM LUDHIANA	TO GOA
DATE OF JOURNEY 05 May, 2022	YOU RATED ★ 3.5

Over all we wanted to make sure that the experience a customer might have buying a ticket is simple and easy and we also wanted to create a design that was easy to use and can be implemented later on in a phone app where we could later add features like bus tracking and connections to your favorite mobile cash apps like Digi wallet, Belize Bank Mobile ect.

Design of Tests

In our final implementation of our code, we did not utilize any test cases so the information below is from a previous version of the software.

We will be testing our five Main Use-Cases. We need these tests so we can evaluate what the user will input. Overall, we cannot test everything that a user might want to do, so the following will be what we anticipate the user might try.

UC #10 – View Booking History

Test Case: Test for a Valid User ID

Description: This test case checks if the function displays the correct booking history for a valid user ID with existing bookings.

Test Steps:

a. Set a valid user ID that exists in the database with a booking history.
b. Call a function with a valid user ID.
c. Capture the output of the function using output buffering.
d. Compare the expected output with the actual output of the function.
e. Assert that the desired output and actual output are equal.

Test Case Name: Test for an invalid User ID

Description: This test case checks if the function displays the correct message when an invalid user ID that does not exist in the database is provided.

Test Steps:

a. Set an invalid user ID that does not exist in the database.
b. Call a function with a invalid user ID.
c. Capture the output of the function using output buffering.
d. Compare the expected output with the actual output of the function.
e. Assert that the desired output and actual output are equal.

Test Case Name: Test for an empty string

Description: This test case checks if the function displays the correct message when an empty string is provided as the user ID.

Test Steps:

a. Set an empty string as the user ID.

b. Call the function with the empty string user ID.
c. Capture the output of the function using output buffering.
d. Compare the expected output with the actual output of the function.
e. Assert that the desired output and actual output are equal.

Test Case Function	Inputs	Actual Result	Expected Result
testValidUserIdWithHistory()	$userid = “1”;	True	False
testInvalidUserIdNonExistent()	$userid = “5678”;	False	False
testInvalidUserIdEmptyString()	$userid = “”	False	False

All of these will be using the PHP unit function assertEqual(). In order for the test case to pass, the actual result and the expected result must be the same. We made sure the expected results and actual results are the same which creates an overall result where everything passes.

UC #9 – View Payment History

Test Case: Test for a Valid User ID

Description: This test case checks the function’s behaviour when a valid user ID is provided. It expects the function to return a string containing payment information for the user with ID 1.

Test Case Name: Test for an invalid User ID

Description: This test case checks the function’s behaviour when an invalid user ID (i.e., a non-existent user ID) is provided. It expects the function to return a string indicating that no payment information was found for the user.

Test Case Name: Empty Payment Information

Description: This test case checks the function’s behaviour when a valid user ID is provided, but no payment information is associated with the user. It expects the function to return a string indicating that no payment information was found for the user.

Test Case Name: Null user-id

Description: This test case checks the function’s behaviour when a null user ID is provided. It expects the function to return a string indicating that no payment information was found for the user.

Test Case Function	Inputs	Actual Result	Expected Result
testValidUserId()	paymentinfo(1)	True	True
testInvalidUserId()	paymentinfo(1000)	False	False
testNoPaymentForUser()	paymentinfo(2)	False	False
testNullUserId()	paymentinfo(0)	False	False

All of these will be using the PHP unit function assertEqual(). In order for the test case to pass, the actual result and the expected result must be the same. We made sure the expected results and actual results are the same which creates an overall result where everything passes.

UC #4 – Edit User Profile

Test Case: Update User Info with Valid Input

Description: This test case checks if the function "edithprofile" updates the user's information with valid inputs (i.e., user ID, first name, last name, phone number, and address) and returns true, indicating that the update was successful.

Test Case Name: Update User Info with Empty First Name

Description: This test case checks if the function "edithprofile" returns false when the first name is empty, indicating that the update was unsuccessful.

Test Case Name: Update User Info with Invalid User Id

Description: This test case checks if the function "edithprofile" returns false when an invalid user ID is provided, indicating that the update was unsuccessful.

Test Case Name: Update User Info with Invalid Phone Number

Description: This test case checks if the function “edithprofile” returns false when an invalid phone number is provided, indicating that the update was unsuccessful.

Test Case Function	Inputs	Actual Result	Expected Result
testUpdateUserInfoWithValidInput()	$userid = 1; $fname = ‘walter’; $lname = ‘white’; $phone = ‘4443333844’;	True	True

	$address = '123 Aroura Lane';		
testUpdateUserInfoWithEmptyFirstName()	$userid = 1; $fname = ''; $lname = 'doe'; $phone = '123456789'; $address = '123 Main St';	False	False
testUpdateUserInfoWithInvalidUserId()	$userid = invalid; $fname = 'John'; $lname = 'doe'; $phone = '1234567890'; $address = '123 Main St';	False	False
testUpdateUserInfoWithInvalidPhoneNumber()	$userid = null; $fname = 'John'; $lname = 'doe'; $phone = 'Invalid'; $address = '123 Main St';	False	False

Each test case includes input values, expected results, and actual results. The PHPUnit framework is used to perform the assertions, which compare the expected and actual results to ensure that the function "edithprofile" is working as intended passes.

UC #6 – Search Trip

Test Case: No trips found

Description: This test case checks if the searchTrips function returns the "No trips found" message when no trips are found for the given input parameters.

Test Case Name: Invalid location IDs

Description: This test case checks if the searchTrips function returns the "No trips found" message when invalid location IDs are given as input parameters.

Test Case Name: Invalid time format

Description: This test case checks if the searchTrips function returns the "No trips found" message when invalid time formats are given as input parameters.

Test Case Name: Empty parameters

Description: This test case checks if the searchTrips function returns the “No trips found” message when empty parameters are given as input.

Test Case Function	Inputs	Actual Result	Expected Result
testValidSearch()	searchTrips(“08:00:00”, “08:55:00”, 1, 2);	True	True
testNoTripsFound()	searchTrips(“09:00:00”, “11:00:00”, 2, 3);	False	False
testInvalidLocationId()	searchTrips(“09:00:00”, “11:00:00”, “invalid”,”location”2);	False	False
testInvalidTimeForm()	searchTrips(“9:00”, “11:00”, 1, 2);	False	False
testEmptyParameters()	searchTrips(“”, “”, “”, ””);	False	False

These test cases cover various scenarios that could occur when using the searchTrips function, and they will help ensure that the function works correctly and reliably.

UC #12 – Reschedule Ticket

Test Case: Valid Input

Description: This test case ensures the update trip function returns true with valid input. In this case, valid unput is defined as a non-empty ticket and bus schedule id.

Test Case Name: Invalid Bus Schedule Id

Description: This test case ensures the update trip function returns false when an empty bus schedule id is provided. This simulates the scenario where the user forgets to enter a bus schedule id value, and the function is called with an empty string.

Test Case Name: Invalid Id

Description: This test case ensures the update trip function returns false when an empty ticket id is provided. This simulates the scenario where the user forgets to enter a ticket id value, and the function is called with an empty string.

Test Case Function	Inputs	Actual Result	Expected Result
testValidInput()	$ticketid = 1; $busscheduleid = 1;	True	True
testInvalidBusScheduleId()	$ticketid = 1; $busscheduleid = “”;	False	False
testInvalidId()	$ticketid = “”; $busscheduleid = 5;	False	False

These test cases aim to validate the correct behaviour of the updatetrip function in different scenarios and ensure that the function handles invalid input gracefully by returning false instead of causing errors or exceptions.

Strategy to Conduct Unit Testing

My strategy for conducting integration testing would involve the following steps:

a. Identify the components: The first step is to identify the different components of the system that need to be tested.

b. Define integration points: Once the components have been identified, the next step is to define the integration points, which are the places where the members interact with each other.

c. Develop integration test cases: Based on the integration points, develop a set of integration test cases that verify the interaction between the components. The test cases should be designed to cover all possible scenarios during the interaction between the members.

d. Prioritize test cases: Prioritize the integration test cases based on their importance and risk level. Test cases that cover critical functionality or high-risk areas of the system should be given higher priority.

e. Conduct tests: Conduct the integration tests, starting with the highest-priority test cases first. The tests should be conducted in a controlled environment to ensure that the results are reliable and accurate.

f. Debug and fix issues: If any issues are identified during the integration tests, they should be logged, investigated, and fixed before proceeding with the next set of tests.

g. Report results: After completing the integration tests, the results should be documented, and a report should be generated. The information should include the tests conducted, any issues identified, and their resolution.

History of Work

As our group went through the process of creating our system, there was a continuous change in deadlines due to issues in the development, rearrangement of assignments, and various issues. This project was managed using a GitHub Repository for our code & our final documents and a shared Google Drive Folder, which held all necessary documentation before being published to the main. All changes made to the documents and code were immediately posted to those mediums. We originally stated that debugging and group discussions would occur over a group call, but this was not utilized. We ended up doing most of our discussions over Whatsapp, with the occasional group meetings to discuss the following deliverable. The primary issue we had in development was implementing the search feature. Currently, there are a lot of bugs within the system, and some use cases have not been fully implemented due to limited time.

The milestones in our project were:

- Determining the Use Cases
- Establish the UI design
- Incorporating the SQL Database
- Finalizing our Class Diagrams

As for the future, we would like to implement the software as a mobile application. This would be a key focus for any future development as this will significantly improve our software.

References

Aweda, Z. I. (2022). How to Test PHP Code With PHPUnit. *freeCodeCamp.org*. https://www.freecodecamp.org/news/test-php-code-with-phpunit/

draw.io - free flowchart maker and diagrams online. (n.d.). https://draw.io/

Søren Spangsberg Jørgensen. (2021, March 7). *Unit Testing with PHP Unit* [Video]. YouTube. https://www.youtube.com/watch?v=a5ZKCFINUkU

Hello Joe Weaver - JustRide App. (n.d.). https://hellojoeweaver.com/justride