Cover Page

Table of Contents

[Table of Contents 2](#_Toc198997247)

[1 Introduction 4](#_Toc198997248)

[1.1 Purpose 4](#_Toc198997249)

[1.2 Scope 4](#_Toc198997250)

[1.3 Product Overview 4](#_Toc198997251)

[1.3.1 Product Perspective 5](#_Toc198997252)

[1.3.1.1 Goal of the system 6](#_Toc198997253)

[1.3.2 Product Functions 7](#_Toc198997254)

[1.3.3 User Characteristics 9](#_Toc198997255)

[1.3.4 Limitations 9](#_Toc198997256)

[1.4 Definitions 11](#_Toc198997257)

[2 References 13](#_Toc198997258)

[3 Requirements 14](#_Toc198997259)

[3.1 Functions 14](#_Toc198997260)

[3.2 Performance Requirements 40](#_Toc198997261)

[3.3 Usability Requirements 40](#_Toc198997262)

[3.4 Interface Requirements 40](#_Toc198997263)

[3.4.1 System Interfaces 40](#_Toc198997264)

[3.4.2 User Interfaces 40](#_Toc198997265)

[3.4.3 Hardware Interfaces 41](#_Toc198997266)

[3.4.4 Software Interfaces 41](#_Toc198997267)

[3.4.5 Communication Interfaces 42](#_Toc198997268)

[3.5 Logical Database Requirements 42](#_Toc198997269)

[3.6 Design Constraints 48](#_Toc198997270)

[3.7 Software System Attributes 50](#_Toc198997271)

[4 Verification 53](#_Toc198997272)

[4.1 Verification Approach 53](#_Toc198997273)

[4.1.1 Compatibility testing 53](#_Toc198997274)

[4.1.4 Usability Testing 55](#_Toc198997275)

[4.1.5 Database Testing 56](#_Toc198997276)

[4.1.6 API Testing 57](#_Toc198997277)

[5 Appendices 58](#_Toc198997278)

[5.1 Assumptions and Dependencies 58](#_Toc198997279)

[5.2 Acronyms and Abbreviations 58](#_Toc198997280)

1 Introduction

1.1 Purpose

The purpose of the Campus Ride-Sharing Platform with Parking System Integration is to create a secure and efficient application for the university community—including students, faculty, and staff—to coordinate carpooling and access real-time parking availability on campus. This system aims to alleviate campus parking congestion, promote sustainable transportation practices, and enhance user safety through integration with the university’s identity verification and parking management systems. The application will allow users to offer or request rides, track rides in real-time, and view up-to-date parking availability data. Additionally, administrators will be able to monitor user and vehicle details to ensure compliance with campus policies.

1.2 Scope

The Campus Ride-Sharing Platform integrates ride-sharing capabilities with the existing campus parking system and MMU Identity Management (IDM) system. It is designed to enable authenticated users to request or offer rides, plan commutes in advance, and receive real-time updates on parking lot occupancy. The platform is mobile-first, supporting Android and iOS, with responsive design for web browsers. Administrators can access dashboards to view user profiles and vehicle data. This system supports user-to-user communication, real-time ride tracking using GPS, and secure logins using university digital ID credentials. The system operates exclusively within the university’s ecosystem, and its features are restricted to registered university affiliates.

1.3 Product Overview

The Campus Ride-Sharing Platform with Parking System Integration is a hybrid mobile and web application solution developed to enhance transportation efficiency across campus. It interacts with three key external systems: the Campus Parking Management System (for parking availability), the MMU Identity Management System (for user authentication and role assignment), and external GPS/Map APIs (for location-based services and ride navigation). Users can request or offer rides, schedule and track rides, view parking availability, and communicate via in-app messaging. MMU IDM handles user authentication during sign-up and login. The Parking System provides real-time parking data to the platform. Admins access the system to view user and vehicle information and manage login/logout. This setup ensures secure, efficient coordination of campus transportation and parking resources.

A diagram of a system

AI-generated content may be incorrect.

Figure 1.3.1 An overview of the system context

1.3.1 Product Perspective

This system acts as a middleware between end-users and the university's infrastructure. Users—including students, faculty, and staff—access the platform through a mobile or web interface after authenticating via MMU’s Identity Management (IDM) system. They provide inputs such as ride requests, ride offers, vehicle details, and scheduling preferences. In return, they receive real-time ride matches, tracking updates via GPS, in-app messages, and parking availability data (Pohl, 2010).

Administrators log in through a dedicated admin portal, authenticated via IDM, to manage user and vehicle data, monitor ride activity, and push notifications. Their inputs include queries and commands that retrieve user summaries, ride logs, and system statistics.

The platform integrates with the university’s Campus Parking Management System to fetch real-time data on space availability, which is then visualized in the user interface. MMU IDM provides secure login tokens and role-based access, ensuring only authorized users interact with the platform’s services.

This ecosystem ensures secure, efficient, and real-time coordination of transportation and campus parking.

A diagram of a system

AI-generated content may be incorrect.

Figure 1.3.2 System context diagram

1.3.1.1 Goal of the system

Table 1.3.1.1: Goals of the System – Campus Ride-Sharing Platform with Parking System Integration

|  |  |
| --- | --- |
| Requirement ID | Goals |
| REQ\_G001 | The system shall reduce parking demand by enabling coordinated ride-sharing among users |
| REQ\_G002 | The system shall allow students, faculty, and staff to request, offer, and schedule rides. |
| REQ\_G003 | The system shall provide real-time parking availability using campus parking system data. |
| REQ\_G004 | The system shall display visual parking data through an intuitive user interface. |
| REQ\_G005 | The system shall track rides in real-time using GPS and display ride location to users. |
| REQ\_G006 | The system shall allow users to log in securely using MMU Digital ID credentials. |
| REQ\_G007 | The system shall enable administrators to monitor user and vehicle data securely. |
| REQ\_G008 | The system shall provide a responsive mobile/web platform accessible across devices. |
| REQ\_G009 | The system shall allow communication between matched riders and drivers via in-app messaging |
| REQ\_G0010 | The system shall facilitate scheduling of rides in advance and send timely notifications |

1.3.2 Product Functions

Table 1.3.2.1 Product Functions of the system

|  |  |  |  |
| --- | --- | --- | --- |
| Function ID | Function Name | Description | Actors |
| REQ\_F-001 | User Sign Up | Allow users (students, faculty, staff) to register using their MMU Digital ID for authentication. | User |
| REQ\_F-002 | Login Authentication | Verify user credentials via MMU IDM system and grant access based on user roles. | User, Admin |
| REQ\_F-003 | Logout | Enable users to securely end their session from the platform. | User, Admin |
| REQ\_F-004 | Request Ride | Allow users to post a request for a ride, specifying time, destination, and preferences. | User |
| REQ\_F-005 | Offer Ride | Enable users with vehicles to offer rides, detailing route, availability, and time. | User |
| REQ\_F-006 | Schedule Ride in Advance | Provide users with the ability to schedule future rides based on their weekly or daily schedules. | User |
| REQ\_F-007 | View Ride Details | Allow users to view detailed information about offered or requested rides. | User |
| REQ\_F-008 | Track Real-Time Ride Location | Enable real-time GPS-based tracking of ongoing rides for both drivers and passengers. | User |
| REQ\_F-009 | In-App Messaging | Allow users to communicate via text-based messaging specific to a ride. | User |
| REQ\_F-010 | Notification Management | Send real-time updates and alerts related to ride statuses, confirmations, and platform announcements. | User |
| REQ\_F-011 | View Real-Time Parking Availability | Provide live data on available parking lots using integration with the campus parking system. | User |
| REQ\_F-012 | Admin Login | Allow admin access to the backend dashboard for platform management. | Admin |
| REQ\_F-013 | Admin Logout | Securely log out admin users from the dashboard. | Admin |
| REQ\_F-014 | View User Information and Vehicle Details | Allow admins to access user profiles, ride history, and registered vehicle information. | Admin |

1.3.3 User Characteristics

Table 1.3.3.1 User characteristics of the system

|  |  |  |
| --- | --- | --- |
| User Role | Description | Technical Skill Level |
| Students | Primary users who request or offer rides; use the app for navigation | Basic computer and mobile literacy |
| Faculty/Staff | Use the system for regular commuting; may offer recurring rides | Moderate computer proficiency |
| Admin | Responsible for monitoring data and ensuring compliance | Advanced IT/admin dashboard usage |

The table outlines the roles, descriptions, and expected technical skill levels of the users interacting with the Campus Ride-Sharing Platform. Students serve as the primary users, utilizing the system to request or offer rides, view parking availability, and navigate routes. They are expected to have basic computer and mobile literacy, sufficient for using the mobile app or web interface effectively. Faculty and staff also use the platform for their daily commute and may offer recurring ride opportunities. Their role requires moderate computer proficiency, allowing them to manage ride schedules and interact with features like messaging or parking data visualization. Administrators are responsible for overseeing system activity, ensuring user compliance, and managing ride and vehicle data. They access the platform through a dedicated admin dashboard and are expected to possess advanced IT knowledge and familiarity with administrative tools, data monitoring, and system reporting functionalities. This distribution of roles ensures that the platform accommodates users with varying levels of technical expertise while maintaining security and operational integrity. Also, all users are expected to have internet access and MMU-issued credentials to use the application.

1.3.4 Limitations

When developing a platform that integrates a campus ride-sharing application with a university parking system and identity management system, several limitations must be considered to ensure operational efficiency, legal compliance, and system reliability. These limitations include:

a) Regulatory Requirements and Policies

Data Privacy Laws: Compliance with data protection regulations such as Malaysia's PDPA (Personal Data Protection Act) is mandatory to protect user identities, ride histories, and vehicle data.

University Policies: The platform must adhere to institutional policies on transportation, digital identity use, and data management.

b) Hardware Limitations

Mobile Device Constraints: The system depends on end-users’ devices (smartphones or tablets), which may vary in processing power, GPS accuracy, and battery performance.

Server Capacity: Backend performance may be constrained during peak usage periods such as semester start or exam weeks.

c) Interfaces with Other Applications

Integration with University Systems: Smooth communication with the MMU IDM system and Parking Management System is essential. API limitations or downtime may hinder operations.

GPS/Map APIs: Dependence on external mapping services (e.g., Google Maps) introduces risks due to rate limits, downtime, or API changes.

d) Parallel Operation

System Availability: Ensuring uptime during maintenance or deployment phases is critical, especially during campus rush hours.

Load Balancing: The system must efficiently manage concurrent user sessions, especially during peak class times.

e) Audit Functions

Logging: The platform must record ride transactions, user activities, and admin access for future audits or incident investigations.

Compliance Reporting: Logs must be exportable and formatted to demonstrate adherence to MMU IT policies and national laws.

f) Quality Requirements

Reliability: The platform must maintain continuous service availability for real-time features such as ride tracking and parking updates.

Scalability: The system should handle growth in users and usage without degradation in performance.

g) Criticality of the Application

Campus Utility: Since the system is intended to reduce parking congestion and improve mobility, its failure may cause widespread disruption in transportation planning and access.

h) Safety and Security Considerations

Data Encryption: All communications and data at rest must be encrypted using secure protocols (e.g., HTTPS, AES).

Authentication: Secure access through MMU Digital ID must include role-based authorization and multi-factor authentication if applicable.

Physical / Mental Considerations

User Interface Design: The application must be intuitive and accessible to users with varying levels of technical expertise.

Accessibility: The platform should support accessible design principles to accommodate users with disabilities.

j) Limitations from Other Systems

Real-time Synchronization: Parking availability and user location tracking depend on external systems, which may introduce delays or data inconsistencies.

Dependency Management: Upgrades or API changes from third-party systems (e.g., GPS APIs, MMU IDM) can affect system behaviour.

k) User Privacy Concerns

Data Transparency: Users must be clearly informed about how their ride data, identity, and vehicle details are collected, used, and stored.

Control Options: Privacy controls (e.g., opt-in for notifications, messaging visibility) should be incorporated to enhance trust.

l) Geographic and Network Constraints

On-Campus Limitation: The system is intended for use within campus boundaries, and GPS resolution may suffer in indoor areas.

Internet Dependency: A stable data connection is required for real-time ride tracking and parking updates, which may not be consistently available to all users.

1.4 Definitions

Table 1.4.1 Definitions of the key terms used in the SRS

|  |  |
| --- | --- |
| Term | Definition |
| MMU IDM | Multimedia University’s Identity Management system used for authentication |
| Carpool | A shared ride between two or more users commuting in the same direction |
| Parking Management System | University system for tracking parking lot occupancy and access control |
| Digital ID | Official MMU-issued credential for identity verification |
| GPS | Global Positioning System used for real-time ride tracking and directions |
| Ride Offer | A user posting availability of vehicle seats for a shared commute |
| Ride Request | A user seeking a ride posted by another user |
| Admin Dashboard | Backend interface for managing users and data |
| Push Notification | A message sent directly to a user's device for updates or alerts |
| API | Application Programming Interface, used for system integrations |
| Ride-Sharing Module | The core logic engine of the platform responsible for processing ride offers, requests, and matching users |
| In-App Messaging | Communication feature allowing matched users (drivers/riders) to chat within the application |
| Parking Data Visualization | A front-end feature that displays current parking lot occupancy data in graphical or map form |
| Authentication Token | A secure, temporary credential issued by MMU IDM used to validate a user's login session |
| Role-Based Access | A security mechanism that grants different system privileges based on the user’s role (e.g., student, admin) |
| Real-Time Tracking | A GPS-enabled feature that allows users to view the live location of their ongoing ride |
| Mobile App/Web Interface | The user-facing platform through which all interactions with the system occur, including ride booking and tracking |

2 References

DataGaps. (n.d.). Database testing concepts.<https://www.datagaps.com/data-testing-concepts/database-testing/>

GeeksforGeeks. (n.d.). Stress testing in software testing.  <https://www.geeksforgeeks.org/stress-testing-software-testing/>

IEEE. (2018). \*ISO/IEC/IEEE 29148:2018 Systems and software engineering—Life cycle processes—Requirements engineering\*.<https://www.iso.org/standard/72089.html>

Interaction Design Foundation. (n.d.). Usability testing. <https://www.interaction-design.org/literature/topics/usability-testing>

Pohl, K. (2010). Requirements engineering: Fundamentals, principles, and techniques. Springer. [Pohl, K. (2010). Requirements Engineering Fundamentals, Principles, and Techniques (p. 813). Heidelberg Springer. - References - Scientific Research Publishing](https://www.scirp.org/reference/referencespapers?referenceid=2211357)

QA Touch. (n.d.). Component testing. <https://www.qatouch.com/blog/component-testing/>

ResearchGate. (2011). Effective and scalable software compatibility testing.<https://www.researchgate.net/publication/220854681_Effective_and_scalable_software_compatibility_testing>

TechTarget. (n.d.). API testing.<https://www.techtarget.com/searchapparchitecture/definition/API-testing>

W3Schools. (n.d.). SQL Constraints. <https://www.w3schools.com/sql/sql_constraints.asp>

3 Requirements

3.1 Functions

3.1.1 Use Case

Use Case Table

|  |  |
| --- | --- |
| Actor | Use Case |
| Driver | Driver Offer Ride |
| Passenger | Passenger Request Ride |
| Passenger Schedule Ride in Advance |
| User (Driver, Passenger) | User View Ride Details |
| User Track Real-Time Ride Location |
| User Send and Receive Messages per Ride |
| User View Notifications |
| User View Real-Time Parking Availability |
| User Sign Up Using MMU Digital ID |
| User Log In |
| User Log Out |
| Admin | Admin View User Information and Vehicle Details |

Use Case Diagram

A diagram of a system

AI-generated content may be incorrect.

3.1.1.2 Use Case 1: Driver Offer Ride

|  |  |
| --- | --- |
| Use Case Name | Driver Offer Ride |
| Use Case ID | UC-01 |
| Purpose | Allow drivers to offer available seats in a ride. |
| Primary Actor | Driver |
| Secondary Actor | Passenger |
| Preconditions | - Drivers must be logged into the system. |
| Postconditions | - Ride offer is available for system matching. |
| Main Flow | 1. Drivers navigate to “Offer Ride”.  2. Enter the seats available, pickup, and destination.  3. Confirm offer.  4. System matches with passengers’ requests.  5. System notify the user for match success.  6. System display ride details. |
| Alternate Flow | Alternate Flow 1:  1. Drivers navigate to “Offer Ride”.  2. Enter the number of seats available, destination.  3. Confirm offer.  4. System matches with passengers’ requests.  5. Drivers cancel the ride.  6. Enter cancel reason.  7. Confirm cancel.  Alternate Flow 2:  1. Drivers navigate to “Offer Ride”.  2. Enter the number of seats available, destination.  3. Confirm offer.  4. System matches with passengers’ requests.  5. If the passenger cancels the ride, display the cancel reason.  6. System continues to match with passengers’ requests. |
| Special Requirements | - Driver’s account must be verified via MMU ID.  - Drivers must add vehicle details. |

3.1.1.1 Use Case 2: Passenger Request Ride

|  |  |
| --- | --- |
| Use Case Name | Passenger Request Ride |
| Use Case ID | UC-02 |
| Purpose | Allow passengers to request a ride to a specific destination. |
| Primary Actor | Passenger |
| Secondary Actor | Driver |
| Preconditions | - Passengers must be logged into the system. |
| Postconditions | - Ride request is submitted and pending acceptance. |
| Main Flow | 1. Passengers navigate to “Request Ride”.  2. Enter pickup and destination.  3. Confirm request.  4. System matches with the available driver.  5. Send a request to the driver.  6. System notify the passenger for match success.  7. System display ride details. |
| Alternate Flow | Alternate Flow 1:  1. Passengers navigate to “Request Ride”.  2. Enter destination.  3. Confirm request.  4. If there is no available driver, notify the user to try again.  Alternate Flow 2:  1. Passengers navigate to “Request Ride”.  2. Enter pickup and destination.  3. Confirm request.  4. System matches with the available driver.  5. Send a request to the driver.  6. If the driver cancels the ride, display the cancel reason.  7. System notify the passenger to try again.  Alternate Flow 3:  1. Passengers navigate to “Request Ride”.  2. Enter pickup and destination.  3. Confirm request.  4. System matches with the available driver.  5. Send a request to the driver.  6. System notify the passenger for match success.  7. System display ride details.  8. Passengers cancel the ride.  9. Enter cancel reason.  10. Confirm cancel. |
| Special Requirements | - Passenger’s account must be verified via MMU ID. |

3.1.1.3 Use Case 3: Passenger Schedule Ride in Advance

|  |  |
| --- | --- |
| Use Case Name | Passenger Schedule Ride in Advance |
| Use Case ID | UC-03 |
| Purpose | Allow passengers to schedule ride requests in advance. |
| Primary Actor | Passenger |
| Secondary Actor | - |
| Preconditions | - Passenger must be logged into the system. |
| Postconditions | - Ride request is scheduled for future matching. |
| Main Flow | 1. Passengers navigate to “Request Ride > Schedule”.  2. Enter pickup, destination and date/time.  3. Confirm schedule.  4. System matches with the available driver on a specific date/time. |
| Alternate Flow | Alternate Flow 1:  1. Passengers navigate to “Request Ride > Schedule”.  2. Enter pickup, destination and date/time.  3. Confirm schedule.  4. System matches with the available driver on a specific date/time.  5. Passengers cancel the ride schedule.  6. Enter cancel reason.  7. Confirm cancel. |
| Special Requirements | - Passenger’s account must be verified via MMU ID. |

3.1.1.4 Use Case 4: User View Ride Details

|  |  |
| --- | --- |
| Use Case Name | User View Ride Details |
| Use Case ID | UC-04 |
| Purpose | Allow users to view active ride session details like destination, date/time, user and vehicle info etc. |
| Primary Actor | User |
| Secondary Actor | - |
| Preconditions | - Users must be logged into the system.  - Users have upcoming or past rides. |
| Postconditions | - Ride details are displayed. |
| Main Flow | 1. Users navigate to “My Rides”.  2. System loads all ride sessions if any.  3. Users select a ride.  4. System displays ride details like date/time, destination etc. |
| Alternate Flow | Alternate Flow 1:  1. Users navigate to “My Rides”.  2. If there’s no upcoming or past ride, show “no available details”. |
| Special Requirements | - |

3.1.1.5 Use Case 5: User Track Real-Time Ride Location

|  |  |
| --- | --- |
| Use Case Name | User Track Real-Time Ride Location |
| Use Case ID | UC-05 |
| Purpose | Allow users to track the location of the current active ride session. |
| Primary Actor | User |
| Secondary Actor | - |
| Preconditions | - Users must be logged into the system.  - Users must be in an active ride session. |
| Postconditions | - Location is continuously updated. |
| Main Flow | 1. Users navigate to “My Rides”.  2. System displays a list of ride sessions if any.  3. Users select the current active ride.  4. System displays ride details.  5. Users click “Track Ride”.  6. System displays the live location on the map. |
| Alternate Flow | Alternate Flow 1:  1. Users navigate to “My Rides”.  2. If there’s no upcoming or past ride, show “no available details”.  Alternate Flow 2:  1. Users navigate to “My Rides”.  2. System displays a list of ride sessions if any.  3. Users select the current active ride.  4. System displays ride details.  5. Users click “Track Ride”.  6. If the location service is off, display the last known location.  7. Display message “Turn on location to track ride”. |
| Special Requirements | - User’s device must turn on location service. |

3.1.1.6 Use Case 6: User Send and Receive Messages per Ride

|  |  |
| --- | --- |
| Use Case Name | User Send and Receive Messages per Ride |
| Use Case ID | UC-06 |
| Purpose | Allow drivers and passengers in a shared ride to communicate. Once ride session ends, |
| Primary Actor | User |
| Secondary Actor | - |
| Preconditions | - Users must be logged into the system.  - Users must be in an active ride session.  - Both users (driver, passenger) must be matched for the same ride. |
| Postconditions | - Messages are exchanged and stored temporarily for each ride session. |
| Main Flow | 1. Users navigate to “My Rides”.  2. System displays a list of ride sessions if any.  3. Users select the current active ride.  4. System displays ride details.  5. Users click the chat button.  6. System loads the message if any.  7. Users type and send a message.  8. Recipients view and respond to the message.  9. Conversation continues until the ride ends or the user closes the session. |
| Alternate Flow | Alternate Flow 1:  1. Users navigate to “My Rides”.  2. If there’s no upcoming or past ride, show “no available details”.  Alternate Flow 2:  1. Users navigate to “My Rides”.  2. System displays a list of ride sessions if any.  3. Users select the current active ride.  4. System displays ride details.  5. Users click the chat button.  6. If the ride session is canceled or ends, system disables the messaging functionality.  7. System loads the message if any. |
| Special Requirements | - Messaging is only accessible during an active ride session.  - Messages are automatically archived once the ride ends and deleted after 7 days. Users are informed within the interface. |

3.1.1.7 Use Case 7: User View Notifications

|  |  |
| --- | --- |
| Use Case Name | User View Notifications |
| Use Case ID | UC-07 |
| Purpose | Allow users to view ride and system-related notifications. |
| Primary Actor | User |
| Secondary Actor | - |
| Preconditions | - Users must be logged into the system. |
| Postconditions | - Notifications are read. |
| Main Flow | 1. Users navigate to “Notification”.  2. System displays a list of notifications if any.  3. Users select a notification.  4. System displays the notification. |
| Alternate Flow | Alternate Flow 1:  1. Users navigate to “Notification”.  2. If there’s no notification, show “no notifications”. |
| Special Requirements | - Notifications must be time-stamped and clearly categorized (e.g., Ride, Parking, System). |

3.1.1.8 Use Case 8: User View Real-Time Parking Availability

|  |  |
| --- | --- |
| Use Case Name | User View Real-Time Parking Availability |
| Use Case ID | UC-08 |
| Purpose | Show users available parking spots in real time. |
| Primary Actor | User |
| Secondary Actor | Parking Management System |
| Preconditions | - Users must be logged into the system.  - Parking availability data is available in the system. |
| Postconditions | - Users view updated parking availability. |
| Main Flow | 1. Users navigate to “Parking”.  2. System requests real-time parking data.  3. Parking Management System provides the latest data.  4. System loads the campus interactive map and displays available parking locations to the user. |
| Alternate Flow | Alternate Flow 1:  1. Users navigate to “Parking”.  2. System requests real-time parking data.  3. If the Parking Management System fails to respond, show message "parking data is currently unavailable". |
| Special Requirements | - |

3.1.1.9 Use Case 9: User Sign Up Using MMU Digital ID

|  |  |
| --- | --- |
| Use Case Name | User Sign Up Using MMU Digital ID |
| Use Case ID | UC-09 |
| Purpose | Allows users to sign up with their MMU Digital ID. |
| Primary Actor | User |
| Secondary Actor | MMU Digital ID Database |
| Preconditions | - User/Driver must have a valid MMU Digital ID.  - MMU Digital ID Database must be accessible. |
| Postconditions | - New account is created. |
| Main Flow | 1. User navigates to the sign-up screen.  2. User enters their details and credentials.  3. System validates the fields and credentials with the MMU Digital ID Database.  4. A new account is created. |
| Alternate Flow | Alternate Flow 1:  1. User navigates to the sign-up screen.  2. User enters their details and credentials.  3. System validates the fields and credentials with the MMU Digital ID Database.  4. The validation process failed. An error message is shown. |
| Special Requirements | - |

3.1.1.10 Use Case 10: User Log In with MMU Digital ID

|  |  |
| --- | --- |
| Use Case Name | User Log In with MMU Digital ID |
| Use Case ID | UC-10 |
| Purpose | Allow users to log into the system with their MMU Digital ID. |
| Primary Actor | User |
| Secondary Actor | MMU Digital ID Database |
| Preconditions | - User/Driver must have a valid MMU Digital ID  - MMU Digital ID Database must be accessible |
| Postconditions | - User is logged in. |
| Main Flow | 1. User navigates to the login page.  2. User enters their credentials.  3. System validates the credentials.  4. User is logged in. |
| Alternate Flow | Alternate Flow 1:  1. User navigates to the log-in page.  2. User enters the wrong credentials.  3. System validates the credentials.  4. System shows an error message. |
| Special Requirements | - |

3.1.1.11 Use Case 11: User Log Out

|  |  |
| --- | --- |
| Use Case Name | User Log Out |
| Use Case ID | UC-11 |
| Purpose | Allow users to log out of the session. |
| Primary Actor | User |
| Secondary Actor | - |
| Preconditions | - Users must be logged into the system. |
| Postconditions | - Session is deleted. |
| Main Flow | 1. User clicks the “Logout” button.  2. System logs the user out and deletes session objects related to the user. |
| Alternate Flow | - |
| Special Requirements | - |

3.1.1.12 Use Case 12: Admin View User Information and Vehicle Details

|  |  |
| --- | --- |
| Use Case Name | Admin View User Information and Vehicle Details |
| Use Case ID | UC-12 |
| Purpose | Allows admin to view users’ personal details and vehicle details |
| Primary Actor | Admin |
| Secondary Actor | - |
| Preconditions | - Admin must be logged into the system. |
| Postconditions | - System shows user information and vehicle details. |
| Main Flow | 1. Admin navigates to the “View user information and vehicle details” page.  2. Admin enters the user’s ID or car plate number to filter the search result.  3. Admin views the user information or vehicle details. |
| Alternate Flow | Alternate Flow 1:  1. Admin navigates to the “View user information and vehicle details” page.  2. Admin enters the user’s ID or car plate number to filter the search result.  3. If there is no result, system will show a “no result” message. |
| Special Requirements | - |

3.1.2 Activity Diagram

Use case 1

A diagram of a system

AI-generated content may be incorrect.

Use Case 2:

A diagram of a flowchart

AI-generated content may be incorrect.

Use case 3:

A diagram of a system

AI-generated content may be incorrect.

Use Case 4:

A diagram of a system

AI-generated content may be incorrect.

Use case 5:

A diagram of a system

AI-generated content may be incorrect.

Use Case 6:

A screenshot of a diagram

AI-generated content may be incorrect.

Use case 7:

A diagram of a system

AI-generated content may be incorrect.

Use Case 8:

A diagram of parking management system

AI-generated content may be incorrect.

Use case 9:

A diagram of a system

AI-generated content may be incorrect.

Use Case 10:

A diagram of a system

AI-generated content may be incorrect.

Use case 11:

A diagram of a user login

AI-generated content may be incorrect.

Use case 12:

A diagram of a flowchart

AI-generated content may be incorrect.

3.2 Performance Requirements

* The product shall take initial load time depending on internet connection strength which also depends on the media from which the product is run but is capped at a maximum of 2 seconds.
* The users shall have their ride request accepted within a timeframe depending on the number of active drivers at the time but capped at a maximum of 15 seconds.
* The maximum acceptable real-time parking availability update period shall be capped at 5 seconds.

3.3 Usability Requirements

* The product shall include helpful, clear, comprehensible and accessible help menus for the functions the product provides.
* The product shall include helpful, clean, clear and comprehensible error messages that are shown when errors occur.
* The product should be accessible on a mobile platform in the form of an application on both android and iOS.

3.4 Interface Requirements

3.4.1 System Interfaces

**Web browser compatibility**

Functionality: User interface will be rendered on compatible web browsers.

Interface description: The product shall be compatible with the latest stable version of major browsers like Google Chrome, Microsoft Edge, Mozilla Firefox and Brave Browser.

**Platform compatibility**

Functionality: Product shall run on all major mobile platforms and desktop operating systems.

Interface description: The product shall be compatible with mobile platforms like Android (10 +), iOS(13 +) and available on desktop operating systems like Windows (10/11) and macOS (Big Sur 11.0+).

3.4.2 User Interfaces

**Utilisation of icons and toolbars**

Logical characteristics:

* The product shall provide the use of icons and toolbars for ease of navigation.
* Interactable elements shall have icons and tooltips to provide clarity on their functions.

**Theme switching**

Logical characteristics:

* The product shall allow users to switch between light and dark themes.
* The default theme shall follow the device preference of the users.

**Layout structure**

Logical characteristics:

* The product shall have a well-structured and consistent layout where the menus and buttons are organised and intuitively placed.
* Buttons with related functionalities shall be placed together.
* Users shall be able to use the product with minimal help.

3.4.3 Hardware Interfaces

**Hardware requirements for hosting:**

Logical characteristics:

* Have NodeJS runtime environment (v18 +) installed.
* Minimally 1GB of free hardware space.
* Minimally 2GB of RAM

**Hardware requirement for end-user devices:**

Logical characteristics:

* with Pentium 4 processor or higher
* with minimally 256MB of RAM.
* with minimally 128MB of free space
* has an active internet connection.

3.4.4 Software Interfaces

**MMU Digital ID Database**

The product shall communicate with the MMU Digital ID Database for authentication.

**Parking Sensor Management System**

The product shall communicate with the parking sensor management system for parking availability.

**User Management System**

The product shall communicate with the user management system to get, update or delete the details of users and their vehicles.

**E-hailing System**

The product shall communicate with the e-hailing system to submit and get ride requests.

**GPS System**

The product shall communicate with the GPS system to get real-time driver’s location.

**Notification System**

The product shall communicate with the notification system to submit and get notifications.

3.4.5 Communication Interfaces

**HTTPS protocol**

Functionality: To encrypt all communication over the internet

Description:

* TLS 1.2+ required.
* All API calls must use HTTPS.

**Network Requirement**

Functionality: Ensure smooth data transfer and good user experience.

Description:

* A minimum of 2MBps of internet speed shall be required.

3.5 Logical Database Requirements

A diagram of a computer

AI-generated content may be incorrect.

Below are the attributes exist in each entity:

|  |  |
| --- | --- |
| ENTITY | ATTRIBUTES |
| USER | MMU\_ID (PK)  NAME  EMAIL  PHONE\_NUMBER |
| RIDE | RIDE\_ID (PK)  VEHICLE\_ID (FK)  DRIVER\_ID (FK reference USER.MMU\_ID)  DESTINATION (FK reference LOCATION.LOCATION\_ID)  PICKUP (FK reference LOCATION.LOCATION\_ID)  DATE  TIME  AVAILABLE\_SEATS  STATUS  CANCEL\_REASON |
| REQUEST | REQUEST\_ID (PK)  PASSENGER\_ID (FK reference USER.MMU\_ID)  DESTINATION (FK reference LOCATION.LOCATION\_ID)  PICKUP (FK reference LOCATION.LOCATION\_ID)  DATE  TIME  STATUS  CANCEL\_REASON  RIDE\_ID (FK) |
| VEHICLE | VEHICLE\_ID (PK)  MMU\_ID (FK)  PLATE\_NUMBER  BRAND  MODEL  COLOUR |
| LOCATION | LOC\_ID (PK)  LOC\_LONGITUDE  LOC\_LATITUDE  LOC\_ADDRESS  AREA |
| NOTIFICATION | NOTIFICATION\_ID (PK)  MMU\_ID (FK)  SENT\_AT |
| CHAT | CHAT\_ID (PK)  RIDE\_ID (FK)  DRIVER\_ID (FK reference USER.MMU\_ID)  PASSENGER\_ID (FK reference USER.MMU\_ID) |

|  |  |
| --- | --- |
| MESSAGE | MESSAGE\_ID (PK)  CHAT\_ID (FK)  SENDER\_ID (FK reference USER.MMU\_ID)  CONTENT  SENT\_AT |

**Entity Relationship**

|  |  |  |  |
| --- | --- | --- | --- |
| USER | — | RIDE | 1:M |
| USER | — | REQUEST | 1:M |
| USER | — | VEHICLE | 1:M |
| USER | — | NOTIFICATION | 1:M |
| USER | — | CHAT | 1:M |
| USER | — | MESSAGE | 1:M |
| RIDE | — | VEHICLE | 1:1 |
| RIDE | — | REQUEST | 1:M |
| RIDE | — | CHAT | 1:1 |
| RIDE | — | LOCATION | 1:1 |
| REQUEST | — | LOCATION | 1:1 |
| CHAT | — | MESSAGE | 1:M |

**Entity Creation**

* When a user signup, a USER entity is being created.
* A VEHICLE entity is created after the user adds the vehicle information into the system.
* When driver offers a ride, a RIDE entity will be created, status will be PENDING. Status will change to MATCHED after match success.
* When passenger requests a ride, a REQUEST entity will be created, date/time will be filled automatically, status will be PENDING. After match success, status will change to MATCHED, RIDE\_ID (FK) will be filled, and a CHAT entity will be created.
* When passenger schedules a ride, a REQUEST entity will be created, date/time need to be filled manually, status will be PENDING. System will start matching at the scheduled date/time. After match success, status will change to MATCHED, RIDE\_ID (FK) will be filled, and a CHAT entity will be created.
* A MESSAGE entity is created for each message sent in a chat.
* Notification is generated dynamically using existing data in the database. A NOTIFICATION entity is created whenever a user receives a notification.

**Frequency of Use**

|  |  |  |
| --- | --- | --- |
| Entity | Frequency | Justification |
| RIDE | High | Created for every offered ride, frequently queried for matching and tracking. |
| REQUEST | High | Created for every passenger ride request or scheduled ride, status updated dynamically. |
| MESSAGE | High | Created every time a user sends a message during a ride. |
| NOTIFICATION | High | Generated dynamically, created frequently upon new system events. |
| CHAT | Medium | Created after a match, updated during communication. |
| USER | Medium | Created once per signup, referenced frequently during login, requests, and rides. |
| LOCATION | Medium | Referenced frequently for pickup/destination coordinates. |
| VEHICLE | Low | Typically created once by each driver, only updated when vehicle info changes. |

**Accessing Capabilities**

|  |  |  |
| --- | --- | --- |
| Role | Accessible Entities | Access Type |
| Passenger | USER, REQUEST, RIDE, CHAT, MESSAGE, NOTIFICATION, LOCATION | View, Create, Cancel |
| Driver | USER, VEHICLE, RIDE, CHAT, MESSAGE, NOTIFICATION, LOCATION | View, Create, Cancel, Update |
| System | All entities | Full read and write access for automation and matching. |
| Admin | All entities | View only for audit and reporting. |

* Access to CHAT and MESSAGE is restricted to matched users only.
* Only the system can link REQUEST to RIDE after matching.

**Integrity Constraints**

|  |  |
| --- | --- |
| Constraints Type | Explanation |
| NOT NULL | Ensures that a column cannot have a NULL value. DESTINATION must be filled in every time passengers request a ride; thus, the system is able to match with available ride offers. CANCEL\_REASON is NOT NULL if STATUS = ‘CANCELED’. |
| UNIQUE | Ensures that all values in a column are different. The record of each PLATE\_NUMBER must be unique in order to identify and track vehicles precisely. |
| PRIMARY KEY | A combination of NOT NULL and UNIQUE. Uniquely identifies each row in a table. Each entity has a primary key (eg. USER.MMU\_ID, RIDE.RIDE\_ID, REQUEST.REQUEST\_ID). |
| FOREIGN KEY | Prevents actions that would destroy links between tables. All foreign key are enforced (eg. RIDE.DRIVER\_ID reference USER.MMU\_ID, DESTINATION reference LOCATION.LOCATION\_ID). |
| CHECK | Ensures that the values in a column satisfies a specific condition. Only REQUESTs with status = ‘MATCHED’ may be assigned a RIDE\_ID. CANCEL\_REASON is NOT NULL if STATUS = ‘CANCELED’. |
| DEFAULT | Sets a default value for a column if no value is specified. When a RIDE or REQUEST entity is created, the default status is ‘PENDING’. |

**Security**

|  |  |
| --- | --- |
| Security Area | Measures Implementation |
| Authentication | Users log in using their verified MMU Digital ID. |
| Authorization | Role-based access ensures users can only view/edit their own data. |
| Message Access | CHAT and MESSAGE access is restricted to users in that ride session only. |
| Data Encryption | Sensitive data (e.g., user credentials, messages) are encrypted in transit and at rest. |
| Audit Logging | System maintains logs of login/logout and key ride operations (matching, cancellations). |

**Data Retention Requirements**

|  |  |
| --- | --- |
| Entity | Retention Requirements |
| USER | Retained indefinitely unless account deletion is requested. |
| VEHICLE | Retained until user removes vehicle or account is deleted. |
| RIDE | Retained for 1 year for audit and usage analysis. |
| REQUEST | Retained for 1 year for audit and usage analysis. |
| LOCATION | Permanent record as it is static reference data. |
| CHAT | Retained for 7 days after ride completion. |
| MESSAGE | Retained for 7 days, after which it is automatically deleted. |
| NOTIFICATION | Retained for 30 days or until manually cleared by the user. |

3.6 Design Constraints

3.6.1 External Constraints

**Business Constraints**

* University Platform Scope: The system must only be accessible to Multimedia University (MMU) students and staff. Users must authenticate using MMU Digital ID to gain access.
* Campus-Only Operations: Ride-sharing and parking services are strictly limited to MMU campus community members.

**Organizational Constraints**

* University Branding Requirements: The user interface must comply with MMU’s branding guidelines, including university-approved colors, logos, fonts, and layout components.
* Approved Internal Services Only: Integration is restricted to authorized MMU internal systems (e.g., MMU Digital ID, Parking Management). Third-party APIs are not allowed unless pre-approved by the university.

3.6.2 Regulatory Constraints

**Data Protection and Privacy**

* PDPA Compliance (Malaysia Personal Data Protection Act 2010): All personal data collected and processed by the system must comply with the PDPA. Consent must be obtained for all data transactions, and data should only be used for specified educational or operational purposes.
* User Consent and Transparency: The system must inform users of data usage and require their consent before storing or processing identifiable personal data (e.g., MMU ID, name, email, contact number).
* Data Protection & Privacy: Purpose Limitation: User personal data may only be processed for functions directly related to the ride-sharing service. Secondary uses (e.g., marketing or fundraising) are prohibited unless separate consent is obtained.
* Data Retention Limitation: Personal data must be retained only for as long as necessary to fulfill the ride-sharing service purpose. Beyond that, the system must support anonymization or deletion mechanisms.
* Right to Access and Correction: Users must be able to request access to their personal data, and submit correction or withdrawal requests, which must be processed within a reasonable time.
* Confidential Disclosure: User data must only be shared with authorized parties (e.g., MMU internal departments or systems) and cannot be disclosed to external parties without consent, unless required by law.
* Data Protection Mechanisms: The system must implement encryption (e.g., SSL/TLS) for secure transmission and storage, along with access control mechanisms to prevent unauthorized use, alteration, or disclosure.

3.6.3 Technical Constraints

These are system-related constraints arising from design choices, platform requirements, or functional behaviour.

**Platform Constraints**

* Mobile Compatibility: The system must be fully responsive for use on mobile devices such as smartphones and tablets, ensuring consistent access across platforms.

**Feature Behaviour Constraints**

* System-Controlled Matchmaking: Users are not allowed to browse or select rides manually. All ride matching must be performed automatically by the system based on time, pickup/destination, and available seats.
* Chat Lifecycle Control: The chat feature is only enabled for matched passengers and drivers during an active ride. It is disabled and auto-deleted post-ride.

**Data Retention Constraints**

* Message and Chat Expiry: Messages and chat records must be auto-deleted 7 days after ride completion to comply with MMU’s short-term communication policy and storage optimization.
* Notification Expiry: Notifications are retained for a maximum of 30 days and then auto-cleared.

3.7 Software System Attributes

Reliability

* The system shall successfully process at least 99.5% of valid user ride requests and offers without failure during normal operations.
* If the system encounters a failure during ride requests or offers, it shall provide a retry option and store the partially completed form for a maximum of 10 minutes.
* The system shall log all failed transactions with timestamps and error codes for later review by administrators.
* Ride cancellation messages shall be delivered to the opposite user interface within 5 seconds whenever driver or passenger cancel the ride.
* The system shall be able to handle at least 1,000 users at the same time.

Availability

* The system shall be operational and accessible to users 99.9% of the time per calendar month, excluding scheduled maintenance.
* System downtime shall not exceed x minutes per month, including unexpected outages.
* Scheduled maintenance shall occur during non-peak hours (2:00 AM – 4:00 AM local time) and shall be announced at least 24 hours in advance to all users.
* The system shall create a checkpoint of all in-progress ride and ride request data every 5 minutes, enabling recovery from the last known good state in case of failure.
* Following an unplanned service outage, the system shall automatically resume operation from the latest checkpoint with no user action required.
* The system shall store uptime statistics per service, and generate monthly uptime reports for internal auditing.

Security

**Utilize Certain Cryptographic Techniques**

* All sensitive data transmitted between the client and server (eg. ride details, personal info) shall be protected using HTTPS (TLS).

**Keep Specific Log or History Data Sets**

* The system shall maintain secure audit logs for all admin actions, such as user management and system configuration changes.
* All user actions related to ride requests, cancellations, and confirmations shall be logged with timestamps and user IDs for accountability.

**Assign certain Functions to different Modules**

* The system shall assign different functions to separate modules based on different users:

Admin: Manage user accounts, view system-wide statistics, handle abuse reports, and monitor system logs.

User (Driver, Passenger): Request rides, accept rides, view ride status, and manage personal profiles.

* These functions shall be enforced using role-based access control (RBAC) to prevent unauthorized access.

**Restrict Communications between some areas of the Programme**

* The system shall prevent direct communication between user-side clients and admin-only backend services.
* Admin APIs shall be protected behind authentication and authorization layers and not exposed to normal users.
* The frontend shall load only the components that correspond to the authenticated user’s role.

**Check Data Integrity for Critical Variables**

* The system shall perform integrity checks on all incoming data of  ride request and offer to ensure it has not been tampered with.
* Server-side validation shall ensure that required fields (eg. pickup location, time, vehicle ID) are complete and within acceptable limits.
* Database constraints shall enforce the integrity of ride schedules, payment amounts, user roles, and relational links between rides and users.
* To prevent abuse, the system shall implement rate limiting mechanisms to block users from spamming ride requests or offers within a short time frame.
* Malicious or excessive submissions (eg. automated spam) shall trigger temporary account suspension, CAPTCHA challenges, or administrative review.

**Assure Data Privacy**

* The system shall ensure that users can only view and edit their own personal data (eg. name, user info, vehicle details).
* Admins shall access user data only for administrative purposes, and access shall be recorded in audit logs.
* The system shall comply with Malaysia’s Personal Data Protection Act, protecting all personal information collected from users.

Maintainability

* The system shall use a modular design where key components (user management, ride-sharing, parking, administration) are separated, allowing isolated maintenance and updates.
* Each module shall expose only well-defined interfaces, avoiding direct dependency on internal logic or data of other modules.
* Code complexity shall be limited through short, following single responsibility principle and consistent naming conventions, making the code easier to understand and update.
* Changes or bug fixes shall be localized to individual modules, minimizing impact on other parts of the system.
* The system shall be structured so that future enhancements or modifications can be made without requiring large-scale changes to the codebase.

Portability

* The system shall be designed to run on major operating systems (Windows, macOS, Linux) with no modification to the source code.
* At least 95% of the codebase shall remain host-independent, minimizing reliance on system-specific paths, commands, or APIs.
* All platform-dependent configurations (e.g., file paths, environment variables) shall be abstracted and handled through environment-specific configuration files.

4 Verification

4.1 Verification Approach

4.1.1 Compatibility testing

Software compatibility testing is a quality assurance task aimed at ensuring that multi-component-based systems build and/or execute correctly across all their versions' combinations, or configurations. It is essential in our product to ensure that the software can run on specified platforms, environments and configurations.

**Testing goals:**

To ensure that the software can run on specified platforms, environments and configurations without any errors.

**Testing methods:**

The software is installed on different web browsers platforms, and environments of different versions that are specified as requirements above. Once installed, the software is tested by following basic workflows, ensuring all features function as expected without errors. Any error occurred during testing is documented.

**Test cases:**

|  |  |  |
| --- | --- | --- |
| Test Aspect | Test Scenario | Expected outcome |
| Web browser compatibility | The software is installed on the latest stable version of  Google Chrome. | The software runs without any errors. |
| The software is installed on the latest stable version of  Microsoft Edge. | The software runs without any errors. |
| The software is installed on the latest stable version of Mozilla Firefox. | The software runs without any errors. |
| The software is installed on the latest stable version of  Brave Browser. | The software runs without any errors. |
| Mobile Platform Compatibility | The software is installed on Android v10 and above. | The software runs without any errors. |
| The software is installed on iOS v13 and above. | The software runs without any errors. |
| The software is installed on Windows 10/11. | The software runs without any errors. |
| The software is installed on macOS v11 and above. | The software runs without any errors. |
| Runtime environment compatibility | The software is hosted in Node.js runtime environment. | The software backend can be hosted in Node.js environment and provides backend services without any errors. |
| Processor compatibility | The software is run on a Pentium 4 processor. | The software runs without any errors or crashing. |

4.1.4 Usability Testing

Usability testing is the practice of testing how easy a design is to use with a group of representative users. It usually involves observing users as they attempt to complete tasks and can be done for different types of designs. It is often conducted repeatedly, from early development until a product’s release.

**Testing goals:**

To understand how users navigate and use the application, ensuring that it is user-friendly and meets the needs and expectations of end users. It is also to gain users’ feedback on the user interface of the software.

**Testing methods:**

The different usability requirements are listed down. 50 participants of different age groups are then chosen to conduct the test remotely, with each participant given a task and is timed to measure the time taken to complete the tasks. The participants are then asked to give feedback on the overall usability and potential improvements of the software’s user interface.

**Test cases:**

|  |  |
| --- | --- |
| Tasks | Expected timeframe |
| Sign up | < 10 minutes |
| Login | < 20 seconds |
| Schedule ride in advance | < 10 minutes |
| Request a ride | < 5 minutes |
| View ride details | < 15 seconds |
| View ride requests | < 15 seconds |
| Track active driver’s location | < 20 seconds |
| View real-time parking availability | < 15 seconds |
| Send messages | < 30 seconds |
| View user and vehicle details | < 1 minute |

4.1.5 Database Testing

Database testing, also known as db testing, is the process of assessing database systems to ensure data accuracy, reliability, and performance. It involves the use of database testing tools to guarantee the consistency, validity, and the ability to manipulate data for business requirements.

**Testing goals:**  
To ensure the database is accurate, reliable and performant, and if the constraints are obeyed.

**Testing methods:**

Several test scenarios are identified, and the database is initially seeded. Then, several test cases for each table are created based on the defined scenarios, and the test outcome is documented.

**Test cases:**

|  |  |  |
| --- | --- | --- |
| Test aspect | Test scenario | Expected outcome |
| Null/Not Null constraint | Each table is inserted a number of rows equivalent to the number of null-constraint columns of the database, with each test data having empty value for the null-constraint columns | The insert operation is successful. |
| Each table is inserted a number of rows equivalent to the number of not-null-constraint columns of the database, with each test data having empty value for the null-constraint columns | The insert operation fails. |
| For any existing row in every table, update the null-constraint column to null | The update operation is successful. |
| For any existing row in every table, update the not-null-constraint column to null | The update operation fails. |
| Data format | For every data type of each table, insert a row that has correct data types. | The insert operation is successful. |
| For every data type of each table, insert a row that has incorrect data types. | The insert operation fails. |
| For every data type of each table, update a column of an existing row with a value of the correct data type | The update operation is successful. |
| For every data type of each table, update a column of an existing row with a value of the incorrect data type | The update operation fails. |
| Foreign key constraint | For each foreign key column of each table, insert a new row with the foreign key columns referencing existing data | The insert operation is successful. |
| For each foreign key column of each table, insert a new row with the foreign key columns referencing non-existing data | The insert operation fails. |
| For each foreign key column of each table, update the columns referencing existing data | The update operation is successful. |
| For each foreign key column of each table, update the columns referencing non-existing data | The update operation fails. |

4.1.6 API Testing

API testing is a type of software testing that analyzes an application programming interface (API) to verify that it fulfils its expected functionality, security, performance and reliability.

**Testing goals:**

To ensure that the software API only accepts HTTPS traffic with version TLS 1.2 and above.

**Testing methods:**

All endpoints are tested using the Postman API client using HTTP and HTTPS of different versions to ensure that only the correct protocol of compatible versions are accepted.

**Test cases:**

|  |  |  |
| --- | --- | --- |
| Test Aspect | Test scenario | Expected outcome |
| Application layer protocol | All API endpoints are tested using HTTPS traffic. | The endpoints accept the traffic and return the correct responses. |
| All API endpoints are called using HTTP traffic. | The endpoints respond with error 400 bad request. |
| Different version of HTTPS | All API endpoints are tested using HTTPS traffic of versions TLS 1.2 and above. | The endpoints accept the traffic and return the correct responses. |
| All API endpoints are tested using HTTPS traffic of versions TLS 1.1 and below. | The endpoints respond with error 400 bad request. |

5 Appendices

tx

5.1 Assumptions and Dependencies

tx

5.2 Acronyms and Abbreviations

tx