Software Requirements Specification

for

<Antigonish Rideshare>

Version <1.0>

Prepared by

Group Name: <*Group C*>

<Yuxuan Liang>	**<202004058>**	**<x2020fdf@stfx.ca>**
<Francis Jose>	**<202106946>**	**<x2021grz@stfx.ca>**
<Owen Shay>	**<202006664>**	**<x2020gft@stfx.ca>**
<Michael Taylor>	**<202003075>**	**<x2020cqx@stfx.ca>**

Instructor: **<*Othman Soufan*>**

Course: **<CSCI485>**

Date: **<February 6th, 2023>**

Contents

Revisions

Version	Primary Author(s)	Description of Version	Date Completed
0.1	The whole team	Filled out first revision of all sections	02/06/2023

1. Introduction

The goal of this project is to design and develop an application that provides a ride-sharing service between Antigonish and Halifax. This introduction section of the document will provide context as to what readers can expect to see within this document, as well as a summary of our planned approach in developing the software to accomplish our goals.

1.1. Document Purpose

The purpose of this document is to provide an overview of the ride-sharing software application we will be developing for our software design course's group project. The scope of the project is to deliver a ride-sharing service between Antigonish and Halifax through a web application. This document will describe all components required for the system that the rideshare application will be built upon. The main interacting systems that will be outlined in this document are the application's frontend, backend, and required database.

1.2. Product Scope

The purpose of our software is to give students an alternative method of travelling between Antigonish and Halifax. Demand for means of travel between these locations is high at the beginning and end of semesters as students need to get from the Halifax Airport to the Town of Antigonish. Our application would also provide new opportunities to earn money for students who had planned on making that drive already, on the condition they have extra space they are willing to share. Our service would also have more room for flexibility in scheduling in comparison to the already existing competitor, the Maritime Bus, as a ride could theoretically be scheduled for any time.

1.3. Intended Audience and Document Overview

The intended audience of this document is the professor of this course, who is also the "client" we are developing this application for. This SRS contains details outlining our planned approach to implement all required components of the system that will provide our ride sharing service. It will describe each components purpose, what technologies will be used to develop the component, and interactions between components. It is best to refer to the section titles when navigating this document to find relevant information

1.4. Definitions, Acronyms and Abbreviations

- UI: User interface
- UX: User experience
- API: Application programming interface
- AWS: Amazon Web Services
- OTP: One time password

1.5. Document Conventions

In general, this document follows the IEEE formatting requirements. Italics are used for comments. Document text should be single spaced and maintain 1" margins.

1.6. References and Acknowledgments

2. Overall Description

2.1. Product Overview

Our ride-sharing application will be a new, self-contained product. With our service, users will be able to schedule ride requests as well as post ride offerings. Ride postings and requests will be editable using our application as well. Our product will be composed of three key systems or components: the frontend, the backend, and the required database. The front end is the component that users will be interacting with. It will provide the required inputs users will need to make use of our service.

The frontend will also be responsible for sending/receiving data to/from the backend in an appropriate format. It will be built using React.js, a well-regarded library for frontend JavaScript applications. It will provide an intuitive interface for users to access their desired services. The design will be kept simple and strict in how it allows users to input their data. The purpose of this decision is to minimize user error and enforce data type requirements.

The second component of our application will be the backend. The backend will handle inserting and retrieving user data into our application's database, as well as performing necessary operations on the data. The backend will also be responsible for the sending/receiving/formatting of data between itself and both the frontend and database. The backend will be built with Python using the Django framework. It will also handle user payments and ensure drivers don't accidentally end up providing free rides.

The last major component of our product is the database, which will store and connect all necessary data to provide our ride-sharing service. It will be built using PostgreSQL and will interact with solely the backend of our system. It will be responsible for relating drivers to riders and creating data entities to represent scheduled rides. It will also need to respond accordingly to user requests to modify previously scheduled rides, and whatever impacts those changes may have on related data.

2.2. Product Functionality

Key Functions:

- Submit ride requests / book rides
- Post ride offerings
- Edit or cancel scheduled rides
- Payment handling

2.3. Design and Implementation Constraints

Our team does not have the budget to implement paid software, so we have relied on free to use libraries, APIs, and tools to build the application. Our team is comprised of full-

time students, leaving limited time to develop the application to deploy before the deadline in April while trying to balance all other responsibilities. The application is comprised of a PostgreSQL database, a Python/Django backend, and a JavaScript/React frontend, making the serialization and transportation of the data within the different levels essential for the functionality of the application. Support for both mouse and keyboard, as well as touch screen must be provided. The application requires an internet connection to connect with other users and book rides. No text translations are currently available within our application, so the application is only provided in English. Our ride-sharing service, since it is provided in Nova Scotia, must abide by Nova Scotia's laws, requirements, and regulations for ride-sharing, data protection and safety.

The use of the UML Modeling language is required within this documentation.

2.4. Assumptions and Dependencies (Owen)

Django, React.js: Django was the framework used for development of the software, and React.js was used for the UI/UX of the application, so we are dependent on the various other software's and platforms (such as AWS or Heroku) to provide support for this framework and library.

Jira: Our project team relied on Jira in the development stages of our application to keep us on track and document our progress through various sprints.

Slack: Our project team was dependent on Slack to communicate/document communications between team members for this application to stay organized.

Web browsers, Internet connection: Our application is heavily dependent on the assumption of a stable internet connection, as well as access to a web browser, such as Chrome, Firefox, Safari, etc.

Figma: Mock-ups of the various web pages are essential to the design and organization of the application, in which we relied on Figma.

PostgreSQL: Our application is heavily dependent on data, such as user log-in information and payment information, in order to run properly so it of high importance that our database, which we chose to use PostgreSQL, functions properly at all times.

Payment API: Trust is placed in the privacy and protection of end-user's payment information within the software and database of a third-party payment API.

3. Specific Requirements

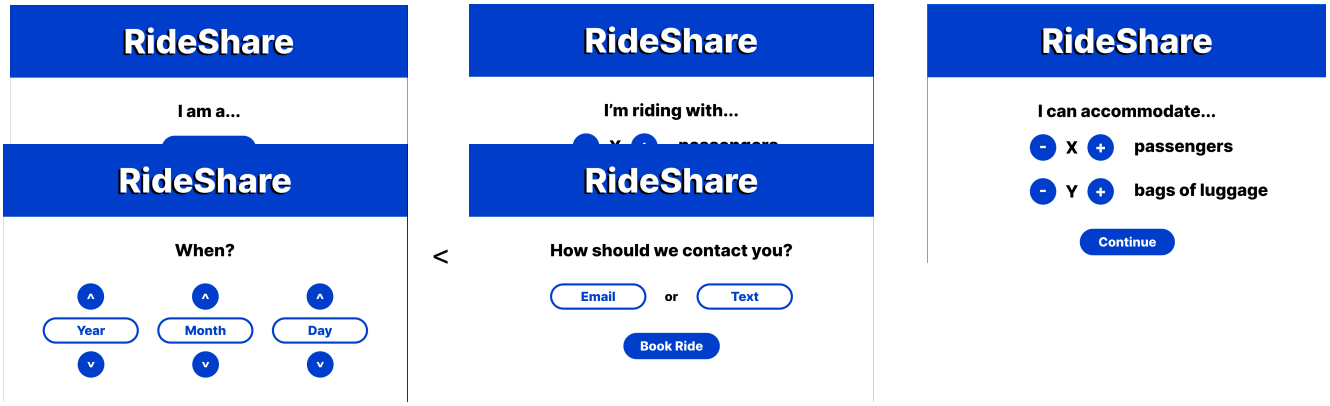
3.1. External Interface Requirements

3.1.1. User Interfaces (Owen & Michael)

We designed rough UI mock-ups of the streamlined UX we hope to achieve within this application. Figma was used in the UI design for ease of use and to aid in the

implementation as it provides the CSS code attached to each design. We are still in early development and intend on updating these designs in the coming future.

Landing page **Passenger landing page** **Driver landing page**



3.1.2. Hardware Interfaces

Any device that can run a web browser.
Internet Connection.
Personal/Payment information.

3.1.3. Software Interfaces

Three main components of our software:

- PostgreSQL
- Django
- React.js

All used for the functionality of the application and to display the application within a web browser (Chrome, Firefox, Safari, etc.)

3.2. Functional Requirements

3.2.1. Home

When the application is opened, there will be a signup page for riders and/or drivers. This allows the user of the application to register with their data.

3.2.2. Signup

New users will set a username and password for their account, which will be linked to their email address and used to log in.

3.2.2.1. Riders

To use the application the riders are required to enter their details. The required details include the user's first name, last name, email address, phone number. These details will be registered on the rider database. Once the details are put into the system, riders must accept the terms and conditions to continue with registration. To confirm the registration an OTP will be sent to the given phone number. After entering the OTP on the prompted window, the registration will be completed. This application incorporates a more streamlined registration process using the google account.

3.2.2.2. Drivers

The home page provides the option for the driver signup. Customers intending to register as a driver have to provide the following details: first name, last name, home address, phone number, email address, license details, tax details, vehicle details (VIN number, model, make, year, insurance details), criminal record check, demerit point details, road offense details, and accident history. The driver registration process will take longer as it must meet certain requirements. After filling in the details, the driver must accept the terms and conditions of the application. To confirm the registration, a OTP will be sent to the registered phone number for verification. Once OTP verification is completed, the driver account will be created and will be stored in the database. The account will be activated for use once the user's credentials are verified. An activation link will be sent to the customer (driver) account to activate the account.

3.2.3. Login

For the registered riders and drivers to access features they must first login to the application with their registered username and password. Once the credentials entered are verified within the database, the login page will render the ride booking page. Otherwise, redirect to the login page.

3.2.4. Reset Password

A reset password feature is included in the login page, which allows drivers and/or riders to reset their password in the case of a user forgetting their password, or other reasons. If our software only provides Google or Apple account login, the password reset function is not required. When an email or mobile phone login is provided, we will send the user a link to reset the password and send the user verification information to verify whether the reset password account matches the acceptance verification information. If the information matches, the user reset password is accepted and the new user-password pair is entered into the database.

Ride booking

After a successful login, the rider will be redirected to the ride booking page. Riders must enter the desired pick-up and drop off locations. After entering the locations, the rider has the following options,

- schedule the ride for a later time (booking in advance up to 7 days)
- choose the type of car economy, premium, premium luxury. Cost of ride will be displayed on the side of each type.
- You can choose multiple stops, with an extra waiting charge, depending on how long the waiting time is.
- Can choose the rideshare option.

Once the ride is selected, the next step is to choose the payment method. If the payment method is not given while registering, the user can provide it while booking.

Payment Method

Once the customer chooses add payment method three options are available,

- Using wallet
- Credit/debit card
- Uber gift card
- Pay-pal

Once the user adds any of the above-mentioned payment methods, it will be set as the default and will be used for future rides as well. The customer has the option to add and delete new card details and add or withdraw money from wallet.

3.2.5. Request ride

After adding a payment method, the user can request a ride. This request will be sent to all the drivers who are in Antigonish and registered with the application. Once a driver accepts the current request, the request will be locked. This window will be prompted with an ETA (Estimated Time of Arrival), distance from pick up to drop off location, total cost of the ride, driver details (name, phone number), car details (plate number, make, model) and time for the driver to reach the pick-up location.

3.2.6. Accept ride request (Driver)

This interface is only for those who registered as drivers. Once a request comes to this interface, the driver will have the option to accept or decline it. This window will prompt the distance from the driver to the pick-up location. The drop off location of the customer will be masked.

3.2.7. Ride-pairing (pair riders with drivers)

This interface provides drivers and passengers who wish to find a match a “hub” to match. The driver provides his departure location and time, the destination of arrival, and the characteristics of passengers he prefers. Passengers provide the number of passengers, the place and time of departure, the destination of arrival, as well as the demand for vehicles and drivers. When users enter this interface, they can choose the matching object they want and communicate. If a consensus is reached between a rider and driver, they will be matched, and the pairing is successful. When it is full or 15 minutes before departure, find pairing information will not be visible.

3.2.8. Modify and notify pre-existing rides

This interface is used to cancel or modify successful pairings. When the driver or passenger sends a change to the itinerary, the modified information will be released to the paired object through this page. The recipient of the message chooses to accept or cancel the pairing once prompted with new information. This page can provide simple information reservation, but the communication requires the driver and passengers to communicate directly by phone. When the pairing is canceled, the driver and passenger would need to re-publish their information to the pairing page to find a new pairing.

3.2.9. Update account information

This interface is used to change/update a user's information. The information that passengers can change includes password, gender, phone number, etc.; the information that the driver can change includes password, gender, phone number, car model, etc. When the user changes the information, the new information will be entered into the database and what can be seen by the users who need to be paired will be updated with the new information.

3.2.10. Browse available rides

This interface is used to view existing itinerary information. These messages can include drivers looking for passengers or passengers looking for drivers. The itinerary for the driver to find passengers will provide his departure location and time, the destination of arrival, and the characteristics of passengers he prefers. The itinerary for passengers to find the driver will provide the number of passengers, the place and time of departure, and the destination of arrival, as well as the demand for vehicles and drivers.

3.2.11. Feedback and rating

This interface includes feedback and a rating page for rider and driver.

3.2.11.1. Rider feedback

The driver will have an option to give feedback about the passenger. This includes whether the passenger gives a tip, their behavior towards the driver, etc. They will assign a rating for the passenger based on these various factors.

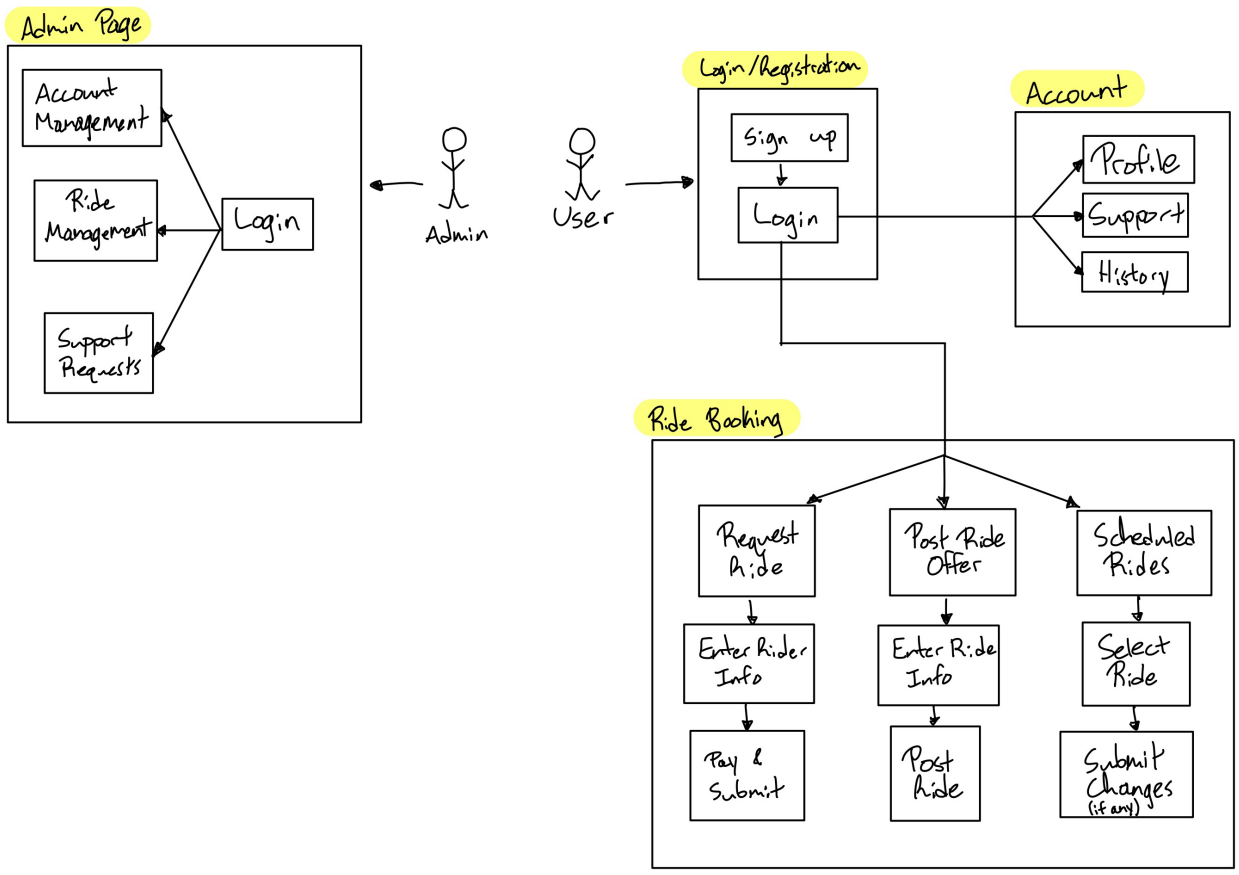
3.2.11.2. Driver feedback and rating

The rider will have the option to give feedback about the driver. This includes their behavior, competence driving, car cleanliness, etc. They will assign a rating for the driver based on these various factors.

3.2.12. User Support

This interface is to help users if they have any issues with the application. This can be done by filling out a ticket or contacting the customer care support team.

3.3. Use Case Model



3.3.1. Use Case #1 (New user registration, NU)

Author – Michael Taylor

Purpose – To allow a new customer to sign up for our service.

Requirements Traceability – 3.2.2.1

Priority - High

Preconditions – An internet connection

Post conditions – Passenger account will be created

Actors – Human (new user)

Flow of Events

1. Basic Flow – User arrives at homepage and enters relevant information (See section 3.2.2.1 for required passenger information), clicks register and creates account
2. Exceptions – Invalid information, make user review information and resubmit

3.3.2. Use Case #2

Author – Owen Shay

Purpose – To allow a passenger to book a ride

Requirements Traceability – 3.2.3, 3.2.5, 3.2.6, 3.2.7, 3.2.9, 3.2.12

Priority - High

Preconditions – Existing user with valid credentials to the application

Post conditions – The user books a ride

Actors – Human

Flow of Events

1. User logs into the application with valid credentials.
2. User selects how much luggage and how many people they intend to bring and selects the date they wish to depart.
3. User provides desired method of contact and submits ride request.

Includes - None

Notes/Issues - User will receive error message if log in credentials are invalid.

Users cannot book two rides in the same day.

3.3.3. Use Case #3

Author – Owen Shay

Purpose – To allow a new driver to sign up for our service

Requirements Traceability – 3.2.2.2

Priority - High

Preconditions – An internet connection

Post conditions – Driver account will be created

Actors – Human

Flow of Events

1. User arrives at homepage and enters all relevant information (See section 3.2.2.2 for required driver information), clicks register and creates account

Includes - None

Notes/Issues - User will receive error message if sign up credentials are invalid in the current field and be asked to review the given information and resubmit. (ex. 12345 entered in the email field)

3.3.4. Use Case #4

Author – Michael Taylor

Purpose – For a customer to request support

Requirements Traceability – 3.2.3, 3.2.14

Priority - Medium

Preconditions – An existing account and a reason for needing app-related support

Post conditions – A support ticket will be submitted

Actors – Human

Flow of Events

1. User arrives at support page, provides information detailing the nature of their issue, submits support ticket.

3.3.5. Use Case #5

Author – Owen Shay

Purpose – For a driver to accept ride request from their posting

Requirements Traceability – 3.2.8, 3.2.10

Priority - High

Preconditions – An existing driver account with valid credentials

Post conditions – A passenger and a driver are paired together

Actors – Human

Includes – Use case #3

Flow of Events

1. Driver logs into the application with valid credentials.
2. Driver posts their availability to offer ride-sharing service.
3. Once a passenger requests to ride with the driver, the driver will be prompted to accept or decline the passenger.

4. Other Non-functional Requirements

4.1. Performance Requirements

As long as wait times are reasonable, this shouldn't be an issue.
P1: An account can only have one identity at the same time, which can be a driver or a passenger, but cannot have two identities at the same time. Our software also provides identity change.
P2: Our software will put the itineraries close to the departure time in the front to facilitate users to find the nearest itinerary (if our software can choose to search for itineraries by time or location, it can also be used as a performance indicator).

P3: When users make data changes or search and view, and encounter pages that need to be loaded, our software should provide visual feedback on the loading process.

4.2. Safety and Security Requirements

4.2.1. Keep user data safe / secure
4.2.2. Filtering text fields in a secure manner
4.2.3. There should be a corresponding user reputation scoring mechanism. When the user complains too much, the software should limit the user's usage. Likewise, there are grievance mechanisms. When the complained user feels that it is unreasonable, they can submit an appeal application to the app staff, and the final decision will be made by the staff. The reputation score can be restored when enough trips have gone through without further complaints.
4.2.4. The driver should provide necessary information such as vehicle driving license, driver's license, insurance, etc. The driver also needs to provide a photo of the vehicle for verification and identification.

4.3. Software Quality Attributes

4.3.1. Performance

Performance refers to how quickly a software system, or a specific component of it, reacts to certain user inputs while handling a specific workload. Given the current user base, this statistic often indicates how long a user must wait before the goal activity occurs (the website renders, a transaction is executed, etc.). But it isn't always the case. Performance specifications could list unnoticed by users' background tasks like backup. Let's instead concentrate on user-centric performance.

4.3.2. Scalability

Scalability measures the greatest workloads that the system can handle while still delivering the required levels of performance. When workloads increase, your system can grow in two different ways: horizontal scaling and vertical scaling.

4.3.3. Portability

How a system or an element may be launched in one environment, or another is determined by portability. Typically, it contains details on the hardware, software, or other use platform. Simply put, it determines how smoothly activities carried out on one platform are executed on another. Additionally, it specifies how easily two separate environments may access and communicate with various system components.

4.3.4. Compatibility

The ability of a system or its component to be launched in one environment or another is known as portability. Hardware, software, or other use platform specs are typically included. In other words, it determines how successfully activities carried out on one platform are executed on another. Additionally, it specifies how easily system components may be accessible and may interact from two various settings.

4.3.5. Reliability

The reliability of a system or component indicates the likelihood that it will function well for a predetermined amount of time under specific circumstances. This likelihood is often stated as a percentage. For instance, if a system has an 85 percent dependability rate for a month, it indicates that throughout that month, there is an 85 percent likelihood that it won't suffer a major failure under typical usage conditions.

4.3.6. Maintainability

Maintainability is the length of time needed to solve a problem, make changes to a solution or its component to improve performance or other characteristics, or adapt to a changing environment. It may be stated as a likelihood of repair over time, just as dependability. A component has a 75% probability of being corrected within 24 hours, for instance, if its maintainability is 75% during that period. A statistic called MTTRS (the mean time to restore the system) is frequently used to assess maintainability.

4.3.7. Availability

Availability refers to the likelihood that a user will be able to use the system at a specific moment. You can define it as a percentage of the time the system is available for operation within a certain time period, while it can alternatively be represented as an expected proportion of requests that are successful. For instance, throughout a month, the system may be accessible 98% of the time. The most business-critical criterion is probably availability, but in order to describe it, you also need to have estimates for reliability and maintainability.

4.3.8. Security

Security is a non-functional criterion that guarantees that every piece of data inside the system will be safe from virus assaults and unauthorised access. There's a catch, though. The majority of security criteria that are not functional may be transformed into actual functional counterparts. To prevent unauthorised access to the admin panel, you would specify the login process and various user roles as system behaviour or user actions.

4.3.9. Localization

The localization characteristic determines how effectively a system or an individual component fits into the overall framework of the target local market. Local languages, laws, currencies, cultures, spellings, and other factors are all part of the background. A product should be more successful with a certain target market the longer it remains with it.

4.3.10. Usability

Usability is yet another classical non-functional requirement that addresses a simple question: How hard is it to use the product? Defining these requirements isn’t as easy as it seems.

Appendix A – Data Dictionary

Appendix B - Group Log

<Please include here all the minutes from your group meetings, your group activities, and any other relevant information that will assist in determining the effort put forth to produce this document> You need to provide snapshots from JIRA showing roles of the users and created tasks.