Software Engineering
Software Requirements Specification (SRS) Document

GoChariot
https://github.com/logashton/gochariot

May 1, 2024

Final Version

Logan P., Hasan Hashim, Madelyn Good

LP, HH, MG

Table of Contents

1. Introduction

1.1. Purpose

The purpose of the GoChariot project is to develop a comprehensive web application to enhance the existing Spartan Chariot tracking system at UNCG. By incorporating features like ride requests, driver reviews, viewing routes/stops, and real-time location tracking, the project aims to improve safety, convenience, and community engagement for students and drivers.

1.2. Document Conventions

This SRD outlines the requirements and specifications for GoChariot, focusing on user-centric design and developer requirements. It ensures alignment with the functional and non-functional requirements detailed in subsequent sections.

1.3. Definitions, Acronyms, and Abbreviations

Web Application	A software application that runs on a web server, accessible via a web browser.
UNCG	University Of North Carolina At Greensboro
GC	GoChariot, the project name.
API	Application Programming Interface, used for communication between different software components.
MVC	Model-View-Controller, an architectural pattern used for developing user interfaces.
HTML/CSS/JS:	Standard technologies for creating web pages and applications. Backend Framework: The server-side framework used to implement the application logic and database management.
Backend Framework	The server-side framework used to implement the application logic and database management.

1.4. Intended Audience

This document is intended for multiple audiences, including the group, stakeholders at UNCG, and potential end-users such as students and drivers. Each section of the document targets the needs and understanding of these varied groups.

1.5. Project Scope

GoChariot aims to align closely with UNCG's goals of enhancing campus safety and improving student services. The software will provide a comprehensive solution for managing the Spartan Chariot system, offering benefits like:

- Improved safety through real-time tracking and safety alerts.
- Enhanced user experience for students and drivers.
- Efficient management of ride requests.
- Building a community through driver reviews and interactions.

1.6. Technology Challenges

Real-time Data Processing: Implementing efficient real-time updates for Spartan Chariot locations.
Cross-Platform Compatibility: Ensuring functionality across diverse devices and browsers.
Scalability: Adapting to growing user numbers and data.
Data Security: Protecting sensitive user data.
User Interface Design: Creating an intuitive, responsive interface.
System Integration: Integrating with existing UNCG systems.
New Technologies: Incorporating advanced mapping and geolocation services.

1.7. References

Alred, G. J., Brusaw, C. T., & Oliu, W. E. (2003). Handbook of Technical Writing (7th ed.). Boston: Bedford/St. Martin’s.
Sommerville, I. (2015). Software Engineering (10th ed.). Boston: Pearson.
Gamma, E., et al. (1994). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley.
Fielding, R. T. (2000). Architectural Styles and the Design of Network-based Software Architectures. [Doctoral dissertation].
Tidwell, J. (2010). Designing Interfaces: Patterns for Effective Interaction Design (2nd ed.). O'Reilly Media.
Nielsen, J., & Loranger, H. (2006). Prioritizing Web Usability. New Riders.
UNCG website and current Spartan Chariot system users’ feedback.

2. General Description

2.1. Product Features

- **Account Management**: Registration and login for users, drivers, and admins.
- **Bus Route Visualization**: Real-time tracking and display of bus routes and stops.
- **Ride Request System**: For students to request rides and for drivers to accept/decline them.
- **Driver Review Mechanism**: To share and view feedback.
- **Safety Alerts**: Critical updates and notifications.
- **Admin Controls**: Manage users, verify registered drivers, and system alerts.

2.2. User Class and Characteristics

The application caters to students, drivers, and admins at UNCG, designed for ease of use without requiring advanced technical skills.

2.3. Operating Environment

GoChariot is a web-based application accessible via various internet-connected devices, including smartphones, tablets, and computers.

2.4. Constraints

Key constraints include internet connectivity, cross-platform compatibility, and integration with existing UNCG systems.

2.5. Assumptions and Dependencies

The project assumes the availability of essential software components like Bootstrap, Jquery, Spring Boot, and relies on external APIs for location tracking.

This refined approach ensures that the introduction and general description sections are in harmony with the detailed functional and non-functional requirements you've outlined, providing a cohesive and comprehensive overview of the GoChariot project.

3. Functional Requirements

The system first has a sign up and login feature for each respective user. The system will respond to bus routes the driver inputs and user requests. The admin will also have the ability to manage all users, verify registered drivers, and send alerts.

3.1. Primary

The system doesn't consist of design, graphics, or operating system requirements or constraints

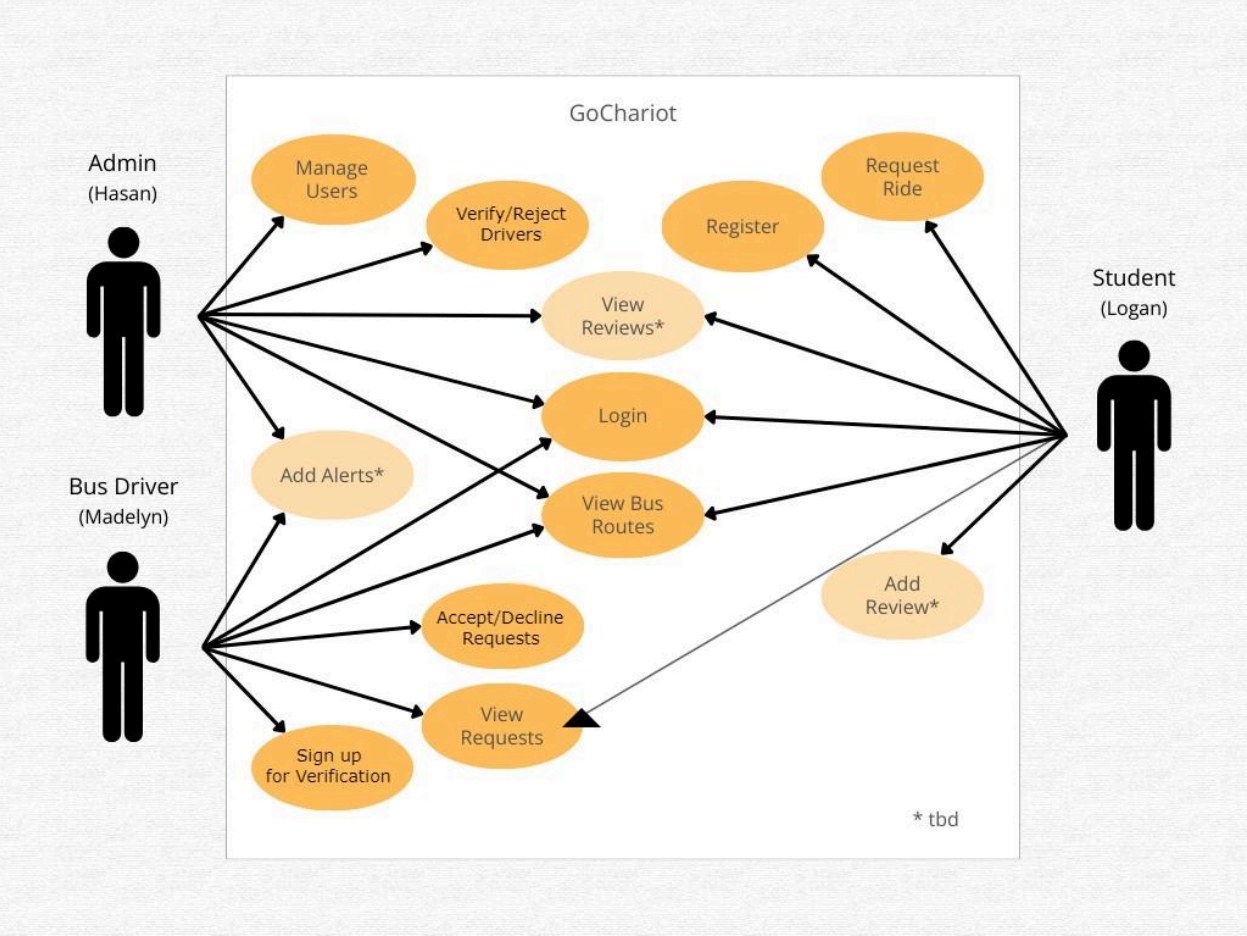
- FR0: The system will allow the user, driver, and admin to view all UNCG bus routes
- FR1: The system will allow the user to register an account
- FR2: The system will allow the user to request a ride
- FR3: The system will allow the user to add and view reviews
- FR4: The system will allow the driver to view other tracked buses, bus routes and stops.
- FR5: The system will allow the driver to view ride requests and respond to them
- FR6: The system will allow the driver and admin to add alerts
- FR7: The system will allow the admin to manage users
- FR8: The system will allow the admin to verify registered drivers

3.2. Secondary

- Location services for all users
- TransitStat/Passiogo API for chariot tracking

3.3. Use-Case Model

3.3.1. Us



e-Case Model Diagram

3.3.2. Use-Case Model Descriptions

3.3.2.1. Actor: Student (Logan)

- Register: make an account
- Login: use credentials to login
- View Buses: view live map of all available and running buses, their routes, and bus stops
- Request Ride: submit ride request from point A to point B for a specific driver and route
- Add Review: submit evaluation of bus ride experience
- View reviews: view reviews that other students have submitted and search by driver
- View requests: view request history and the status of current requests

3.3.2.2. Actor: Driver (Madelyn)

- View Bus Routes: view live map of all available and running bus routes
- View Requests: view ride requests and accept/decline them
- Add Alert: create and post announcement/notification

- Register: register and wait for manual verification

3.3.2.3. Actor: Admin (Hasan)

- View Bus Routes: view map of all available and running UNCG bus routes
- Manage Users: ability to view and edit users
- Add Alert: create and post announcement/notification
- View Reviews: view evaluations of bus ride experiences
- Verify drivers: view registered drivers, contact them for manual verification, and then verify or reject their sign up to unlock their account or keep it locked.

3.3.3. Use-Case Model Scenarios

3.3.3.1. Actor: Student (Logan, Madelyn, Hasan)

- **Use-Case Name**: View Bus Routes
  - **Initial Assumption**: The user is logged in
  - **Normal**: The user will see a live map of all available and running UNCG buses and their routes/stops
  - **What Can Go Wrong:** The map is not visible
  - **Other Activities**: Logout, request ride, add review, view reviews
  - **System State on Completion**: View bus routes live map
- **Use-Case Name**: Register/Login
  - **Initial Assumption**: The user has access to the web app
  - **Normal**: The user will have successfully logged in and will see a live map of all available and running UNCG bus routes
  - **What Can Go Wrong**: Invalid credentials
  - **Other Activities**: n/a
  - **System State on Completion**: Full access to web app
- **Use-Case Name**: View reviews
  - **Initial Assumption**: The user has access to the web app
  - **Normal**: The user navigates to the reviews page and can go through reviews or search by driver.
  - **What Can Go Wrong**: The controller fails to fetch the reviews.
  - **Other Activities**: view dashboard, view live tracking, view requests, write a review
  - **System State on Completion**: The reviews are loaded onto the page
- **Use-Case Name**: Add review
  - **Initial Assumption**: The user has access to the web app
  - **Normal**: The user navigates to the add reviews section and decides they want to write a review. They then select a driver, give a rating 1-5, and write a text portion of their review and submit it.
  - **What Can Go Wrong**: The service fails to write the review to the database.

        - **Other Activities**: view dashboard, view live tracking, view requests, view reviews
        - **System State on Completion**: The review is added to the database.
    - **Use-Case Name**: View requests
        - **Initial Assumption**: The user has access to the web app
        - **Normal**: The user navigates to the requests page. There they can see their request history and the status of their current requests.
        - **What Can Go Wrong**: The controller fails to fetch the requests.
        - **Other Activities**: view dashboard, view live tracking, view reviews
        - **System State on Completion**: The reviews are loaded onto the page
    - **Use-Case Name**: Request ride
        - **Initial Assumption**: The user has access to the web app
        - **Normal**: The user navigates to the tracking page, clicks a bus, and requests a ride from that driver for the bus.
        - **What Can Go Wrong**: The controller fails to add the request to the database.
        - **Other Activities**: view dashboard, view live tracking, view reviews
        - **System State on Completion**: The request is added to the database

3.3.3.2. Actor: Driver (Madelyn)

- **Use-Case Name**: View ride requests
    - **Initial Assumption**: The driver is logged in and in route
    - **Normal**: The driver will see ride requests that students have sent to them. There will be the status of the request and the option to accept/decline
    - **What Can Go Wrong**: The endpoint for fetching requests fails
    - **Other Activities**: Logout, add alert, view reviews, view dashboard
    - **System State on Completion**: View table of requests
- **Use-Case Name**: Add alert
    - **Initial Assumption**: The driver is logged in
    - **Normal**: The driver feels the need to make an alert and goes to the alert page, writes an alert, and submits it
    - **What Can Go Wrong**: The service fails to write the review to the database
    - **Other Activities**: Logout, view requests, view reviews, view dashboard
    - **System State on Completion**: Alert is added to database

- **Use-Case Name**: Accept/decline requests
    - **Initial Assumption**: The driver is logged in
    - **Normal**: The driver sees a pending ride request. The driver then chooses to accept or decline the request.
    - **What Can Go Wrong**: The service fails to update the status of the request.
    - **Other Activities**: Logout, add alerts, view reviews, view dashboard
    - **System State on Completion**: The status of the request is updated to reflect the driver's choice
- **Use-Case Name**: Sign up for verification
    - **Initial Assumption**: The driver is not signed up and wants to create an account
    - **Normal**: The driver goes to the sign up page for drivers and enters the necessary information to sign up. They then await manual verification
    - **What Can Go Wrong**: The driver fails to enter data properly or the service fails to save the user to the database.
    - **Other Activities**: View reviews, live tracking
      System State on Completion: The account is added to the database with the status locked and awaits verification.

3.3.3.3. Actor: Admin (Hasan)

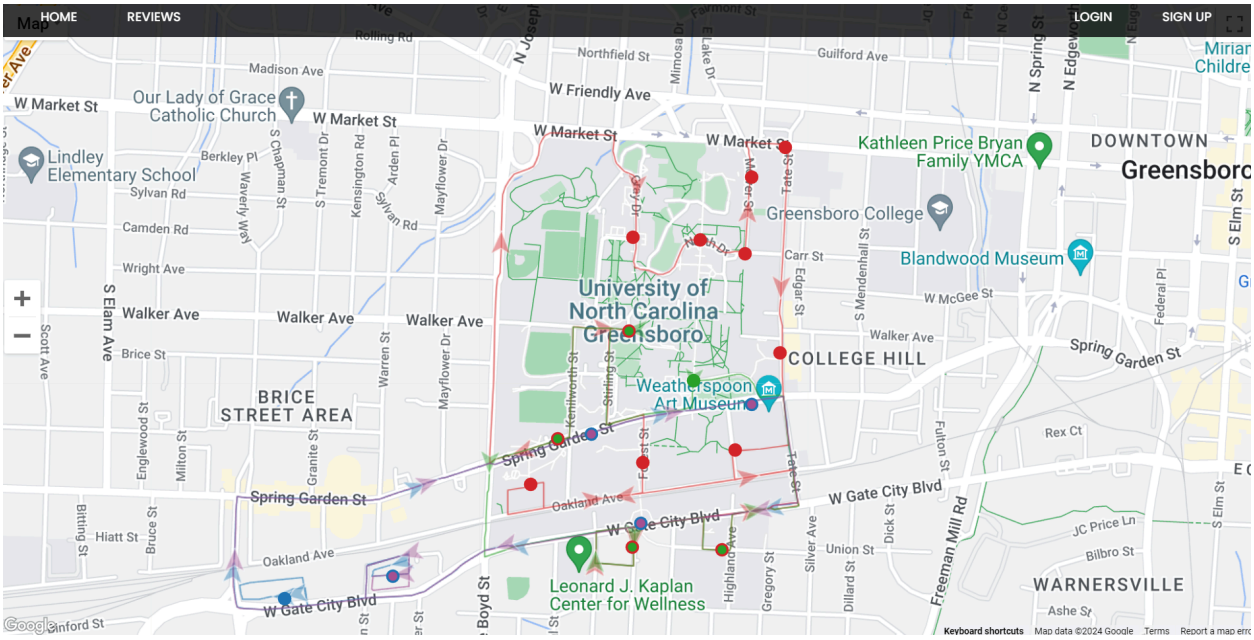
- **Use-Case Name**: Verify reject/drivers
    - **Initial Assumption**: The admin is logged in
    - **Normal**: The admin navigates to the verify drivers page, contacts a driver to verify their identity, then makes a decision to verify the driver's account to unlock the account.
    - **What Can Go Wrong**: The controller fails to set the driver's account status to unlocked
    - **Other Activities**: Logout, manage users, view bus routes, view reviews
    - **System State on Completion**: Driver's account is unlocked or they are removed from the driver table in database.
- **Use-Case Name**: Manage Users
    - **Initial Assumption**: The admin is logged in
    - **Normal**: The admin can edit users
    - **What Can Go Wrong**: Invalid account info
    - **Other Activities**: Logout, edit bus routes, view bus routes
    - **System State on Completion**: Updated user accounts

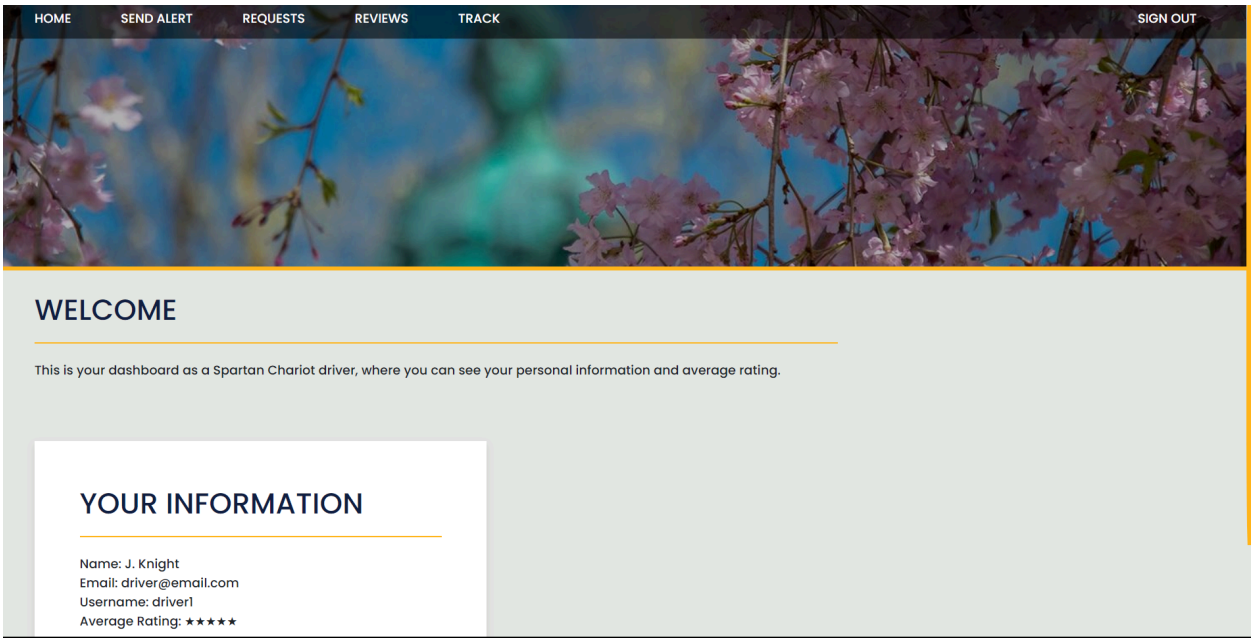
4. Technical Requirements

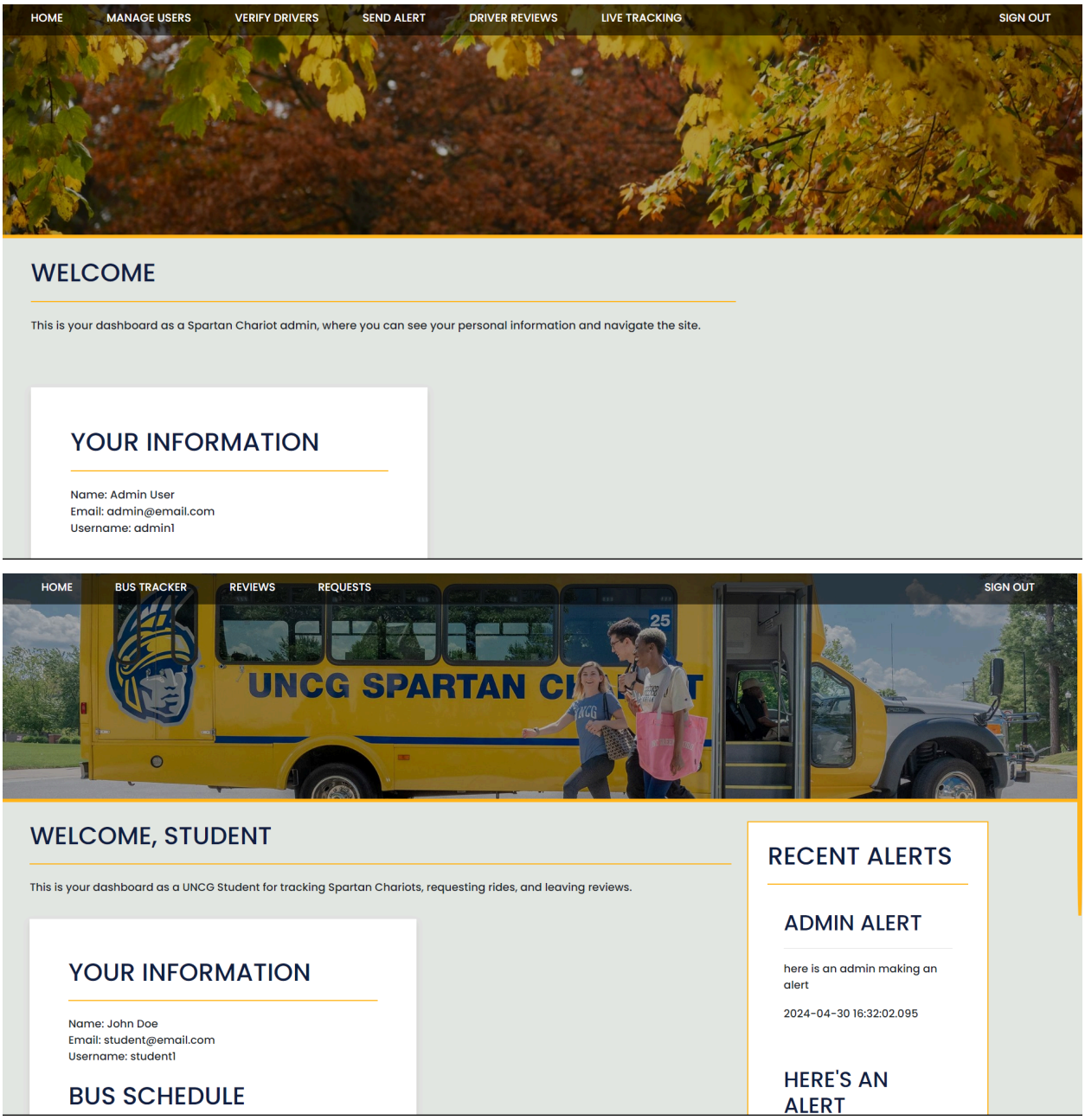
4.1. Interface Requirements

4.1.1. User Interfaces

Every page will feature a header, navigation bar, login or sign out.







4.1.2. Hardware Interfaces

The web application will run on any hardware device that has access to the internet, the ability to display webpages, and the ability to interact with web pages. This includes, but is not limited to, smartphones, tablets, desktop computers, and laptops.

4.1.3. Communications Interfaces

It must be able to connect to the Internet as well as the local database running on PostgreSQL. The communication protocols, HTTP and WebSocket, must be able to connect to the Transitstatus API and the Passio Go! WebSocket and API and return bus coordinates, bus information, routes, stops, and destinations.

4.1.4. Software Interfaces

We will use Bootstrap, Jquery, and Spring Boot ThymeLeaf to help build the front-end. We will also use JPA for the backend database functionality, as well as using Spring Boot with Java to connect the front-end to the back-end.

5. Non-Functional Requirements

5.1. Performance Requirements

- NFR0(R): The local copy of the Go Chariot database will consume less than 50 MB of memory
- NFR1(R): The system (including the local copy of the Go Chariot database) will consume less than 150MB of memory
- NFR2(R): The novice user will be able to intuitively navigate the map for viewing bus routes.
- NFR3(R): The expert user will be able to view bus driver reviews/routes and use it to make an informed decision for requesting/deciding on rides.

5.2. Safety Requirements

Drivers can sign up for GoChariot, however for the safety of users, a manual verification process is done to ensure that no one impersonates a driver. Verified drivers are also able to make safety alerts.

5.3. Security Requirements

- NFR4(R): controller endpoints that write to the database will only be usable by authorized users.
- NFR5(R): Passwords for accounts will be stored securely (hashed).
- NFR6(R): API endpoints will be secured based on the currently authenticated role of the logged in user.

5.4. Software Quality Attributes

5.4.1. Availability

Go Chariot will be available to users at all times.

5.4.2. Correctness

All displayed information on Go Chariot will be correct and not outdated.

5.4.3. Maintainability

Go Chariot does not have a set lifespan. Go Chariot's code will be sufficiently documented, use modular programming, and be cost effective for great maintainability. If the software faces system failure, administrators will be alerted so that it can be resolved as quickly as possible.

5.4.4. Reusability

Go Chariot's code will be modular and use consistent and well-established libraries/software, which means that the components will have good reusability.

5.4.5. Portability

All components behind Go Chariot are cross-platform, meaning the software will be able to be deployed on any modern system. The front-end will also be able to be accessed by any device that can connect to the Internet and access a website.

5.5. Process Requirements

5.5.1. Development Process Used

Agile

5.5.2. Time Constraints

Rolling due dates with presentations and before the Spartan Chariots stop driving around campus.

5.5.3. Cost and Delivery Date

$0 and end of the semester (May 1st, 2024).

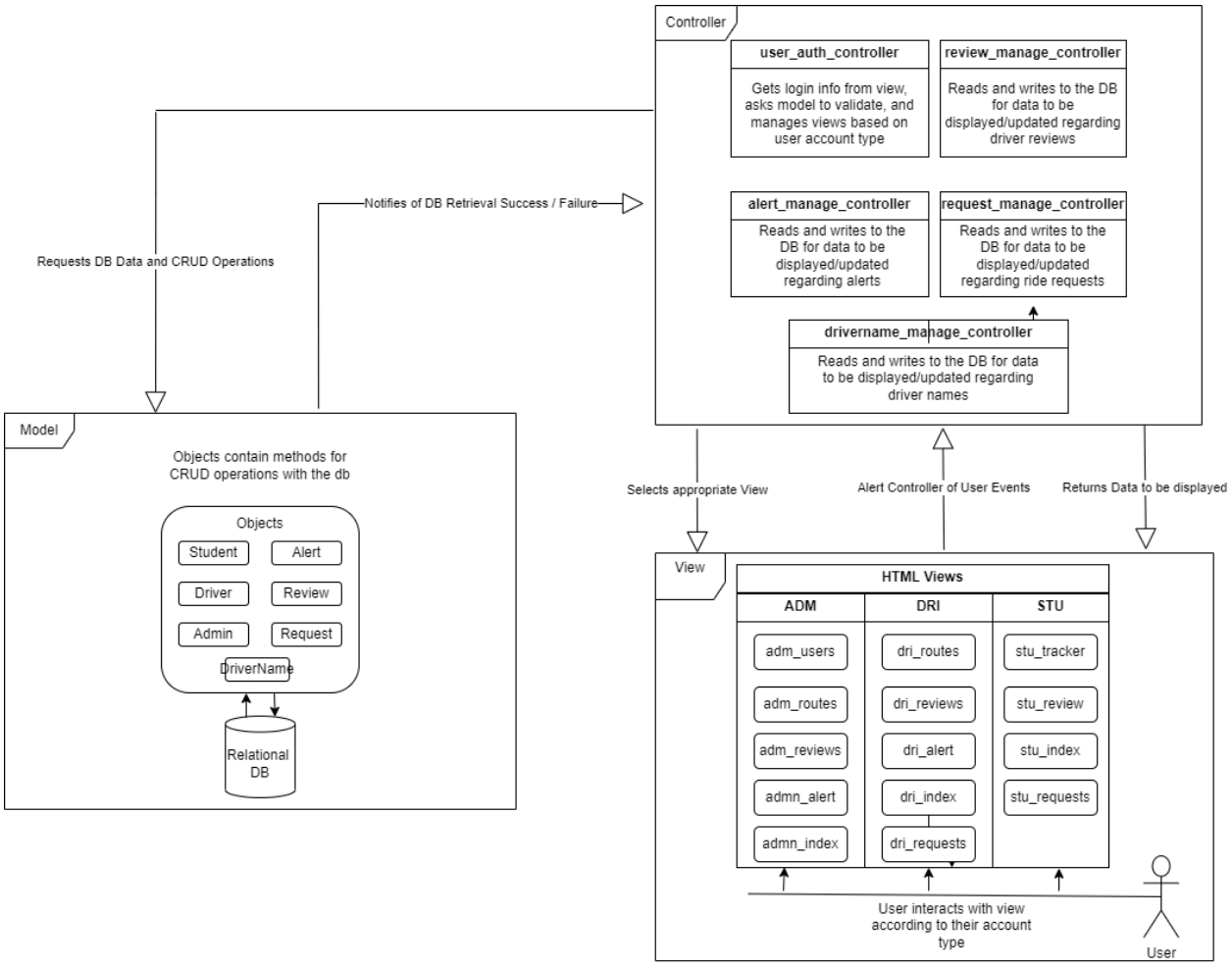
5.6. Other Requirements

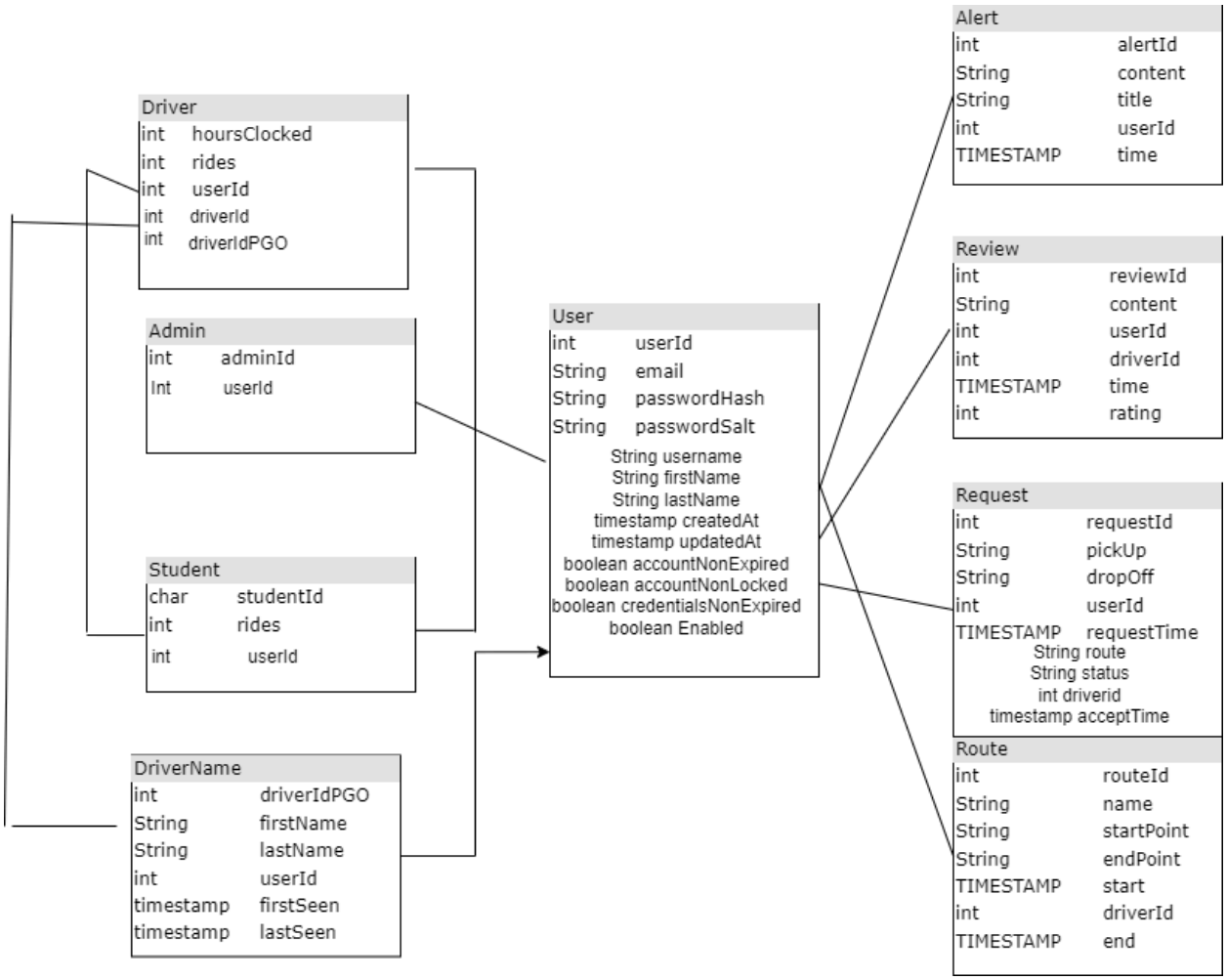
6. Design Documents

6.1 Software Architecture

6.2 High-Level Database Schema

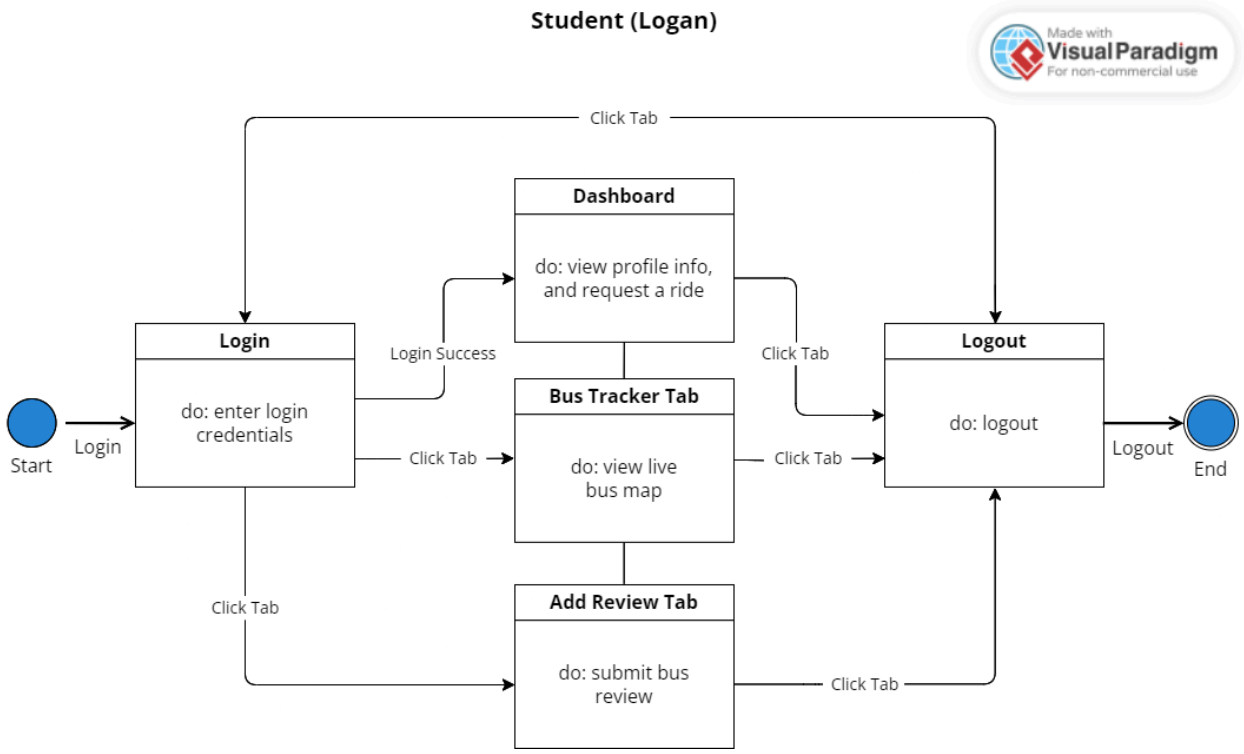
The data being stored on the system will be stored using a relational database. It will use a local PostgreSQL server. The data will be created and updated via interacting with endpoints in the Controller that query the local PostgreSQL server. The data being stored will be information about admins, students, drivers, login information, alerts, reviews, routes, and requests – more can be seen on this in the diagram below.



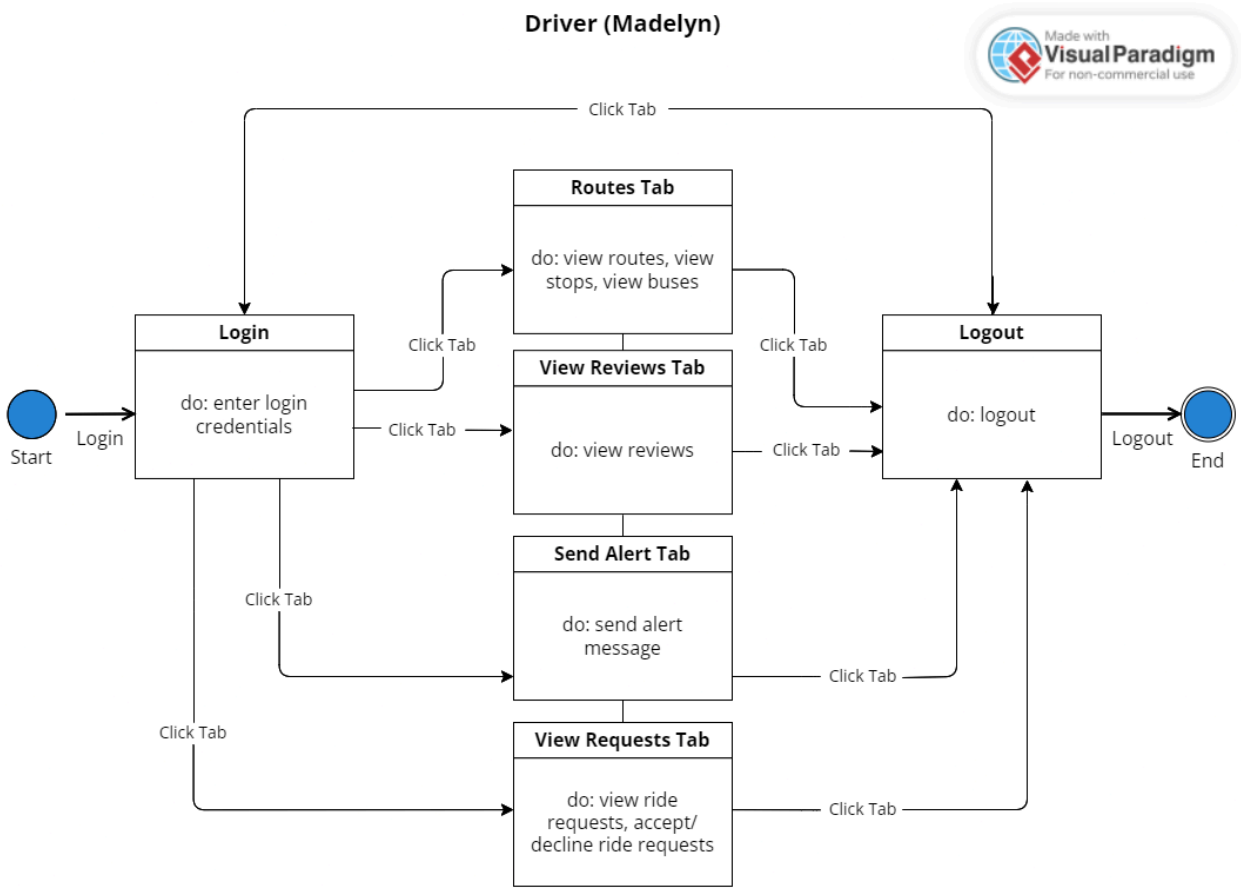

Alert
int alertId
String content
String title
int userId
TIMESTAMP time
Driver
int hoursClocked
int rides
int userId
int driverId
int driverIdPGO
Review
int reviewId
String content
int userId
int driverId
TIMESTAMP time
int rating
Admin
int adminId
Int userId
User
int userId
String email
String passwordHash
String passwordSalt
String username
String firstName
String lastName
timestamp createdAt
timestamp updatedAt
boolean accountNonExpired
boolean accountNonLocked
boolean credentialsNonExpired
boolean Enabled
Request
int requestId
String pickUp
String dropOff
int userId
TIMESTAMP requestTime
String route
String status
int driverid
timestamp acceptTime
Student
char studentId
int rides
int userId
Route
int routeId
String name
String startPoint
String endPoint
TIMESTAMP start
int driverId
TIMESTAMP end
DriverName
int driverIdPGO
String firstName
String lastName
int userId
timestamp firstSeen
timestamp lastSeen

6.3 Software Design

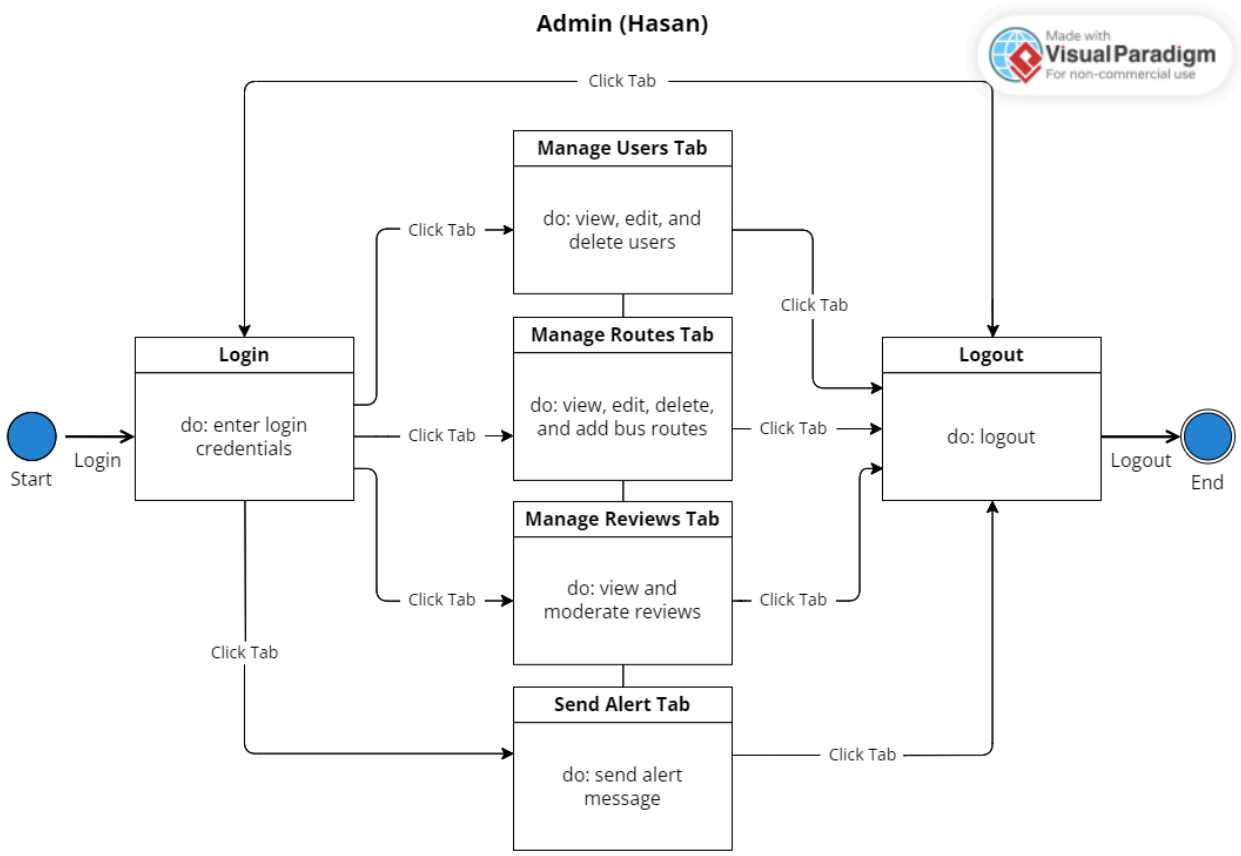
6.3.1 State Machine Diagram: Student (Logan Phillips)



6.3.2 State Machine Diagram: Driver (Madelyn Good)



6.3.3 State Machine Diagram: Admin (Hasan Hashim)



6.4 UML Class Diagram