Software Engineering
Software Requirements Specification (SRS) Document

Race Management System

9-26-2023

Version 1

By: Sergio Ramirez, Christian Leonard, Blake Joye

WE HAVE ABIDED BY THE UNCG *Academic Integrity Policy* ON THIS ASSIGNMENT.

Table of Contents

1. Introduction

1.1. Purpose

This Race Management System will assist all members of a race team in preparing for and competing in any motorsport division. This app will aid team owners in managing their personnel. Race engineers can use this program to create and alter race strategies and view track and weather info about upcoming races. They can also store information on car performance. Drivers can use this program to deliver feedback on their cars as well as check previous race results in order to gauge their performance better. This application will be written in Java, HTML, and operated through a web browser.

1.2. Document Conventions

The purpose of this Software Requirements Document (SRD) is to describe the client-view and developer-view requirements for the Race Mangement System (RMS). Client-oriented requirements describe the system from the client's perspective. These requirements include a description of the different types of users served by the system. Developer-oriented requirements describe the system from a software developer's perspective. These requirements include a detailed description of functional, data, performance, and other important requirements.

1.3. Definitions, Acronyms, and Abbreviations

Java	A programming language originally developed by James Gosling at Sun Microsystems. We will be using this language to build the Restaurant Manager.
.HTML	Hypertext Markup Language. This is the code that will be used to structure and design the web application and its content.
SpringBoot	An open-source Java-based framework used to create a micro Service. This will be used to create and run our application.
MVC	Model-View-Controller. This is the architectural pattern that will be used to implement our system.
Spring Web	Will be used to build our web application by using Spring MVC. This is one of the dependencies of our system.
Thymeleaf	A modern server-side Java template engine for our web environment. This is one of the dependencies of our system.
NetBeans	An integrated development environment (IDE) for Java. This is where our system will be created.
API	Application Programming Interface. This will be used to implement a function within the software where the current date and time is displayed on the homepage.
Spring Data MongoDB	Store data in JSON documents, meaning fields and vary from document to document and data can be added over time.

1.4. Intended Audience

- The ‘Intoduction’ and ‘General Description’ is intended for any person who was is a part of the devolpment process of the racing team. The intended users Team Manager, Driver, and Race Engineer should read the entire document to understand the purpose of this project, learn the functionality, and project requirements.

1.5. Project Scope

The benefits of the project to business include:

- Team managers can use the system to access personnel management, making it easier to coordinate with team members.
- Increasing driver communication by allowing the driver to provide feedback on car performance after every race.
- Improving overall race performance by allowing race engineers to create race strategies through weather forcast, car performance data, and driver feedback.

1.6. Technology Challenges

1.7. References

2. General Description

2.1. Product Perspective

The RMS found its origin in hopes to simplify race management tasks for race teams with an easy to use interface.

2.2. Product Features

- User Accounts: Team Manager, Driver, Race Engineer
- Race Budget: The Team Manager has the ability to manage the budget, allocate expenses and track financials
- Personnel Management: Team Managers can manage the team personnel(hire/fire)
- Weather Viewing: All users have the ability to see the weather for race day
- Track Info: All users have the ability to see information about the track
- Race Strategy: The Race Engineer can develop a plan for the race
- Live communications: The Race Engineer has the ability to communicate with the driver mid race
- Driver Stats: The Driver can view his stats based on their previous races
- Driver Feedback: The Driver can give feedback on his vehicle and how they felt it performed throughout the race

2.3. User Class and Characteristics

The RMS is designed to cater to users with different levels of expertise in motosports, users are categorized as followed:

- Team Manager: This user oversees and manages the race team and budget. They do not require . Expertise in computers but will need to be very knowledgeable in motorsport operations, budget management, leadership and team strategy
- Race Engineer: The engineer focuses on race strategies, car performance, and live communication with the driver during a race. They need to have a strong background in motorsport engineering and strategy

- Driver: The driver is the core of the team as they will be on the track, They will use this system to keep track of their personal performance and to provide feedback about the cars performance. They will primarily need expertise in racing and communication skills.

2.4. Operating Environment

RMS is a web-based application designed to operate smoothly across various browsers.

2.5. Constraints

.

2.6. Assumptions and Dependencies

- The software will be dependent on java for development and command line operation. It will use Spring Boot as the framework for the web application.
- The application will rely on an external API such as a weather API for information about the race day weather or track size API for the track information

3. Functional Requirements

3.1. Primary

- FR0: The system will allow the user to add and remove information into tables.
- FR1: The system will allow the user to approve and deny requests.
- FR2: The system will allow the user to access the weather.
- FR3: The system will allow the user to run calculations based on information contained in tables.

3.2. Secondary

- FR4: An authorization scheme is implemented to ensure that managers, race engineers, and drivers can only access information related to their respective roles.

4. Technical Requirements

4.1. Operating System and Compatibility

The application will be compatible with any operating system that is able to view and to interact with traditional web pages.

4.2. Interface Requirements

4.2.1. User Interfaces

The first screen after logging into the website will be the dashboard. The dashboard will include upcoming races, access to the weather API, and a menu for users to access their specific functions. The menu will act as a directory that takes users to their page. The page specific for each user will include their function and be able to edit things where necessary. The race engineer will be able to view driver feedback through the click of a button on their page. Each page will contain a menu to access different pages and to be able to log out.

4.2.2. 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.

4.2.3. Communications Interfaces

It must be able to connect to the internet as well as the local database on mongoDB.
The communication protocol, HTTP, must be able to connect to the weather API and return the current date, location, and weather status.

4.2.4. Software Interfaces

We will use HTML and Spring Boot ThymeLeaf to help build the frontend, as well as mongoDB for the backend database functionality. We will also use Spring Boot with Java to connect the frontend to the backend.

5. Non-Functional Requirements

5.1. Performance Requirements

- NFR0(R): The system will consume less than 50MB of memory including any local data files
- NFR1(R): The average user will be able to access team and weather information in less than 30 seconds
- NFR2(R): The average user will be able to push information into the database in under 15 seconds
- NFR3(R): The average user will be able to calculate projected race time in under 5 minutes

5.2. Safety Requirements

– NFR4(R): regular data backup processes to ensure that user data is continuously backed up to secure locations.

5.3. Security Requirements

- NFR5(R): The system will only be usable by authorized users.

5.4. Software Quality Attributes

5.4.1. Availability

The website will be online at all times unless for scheduled maintenance.

5.4.2. Correctness

All functions will work correctly to match expectations and performance of the software description.

5.4.3. Maintainability

The RMS will be constantly updated to ensure it adpats to race rules and user expectations.

5.4.4. Reusability

Reusable components and structures will be implmented to accelerate the development of new features.

5.4.5. Portability

The RMS will be allowed to be used on any device that has access to an internet connection and a web browser.

5.5. Process Requirements

5.5.1. Development Process Used

SCRUM

5.5.2. Time Constraints

– The project will be managed in sprints with each sprint allowing for new features to be released to the users. Enabling for quick adaption and fast delivery of software.

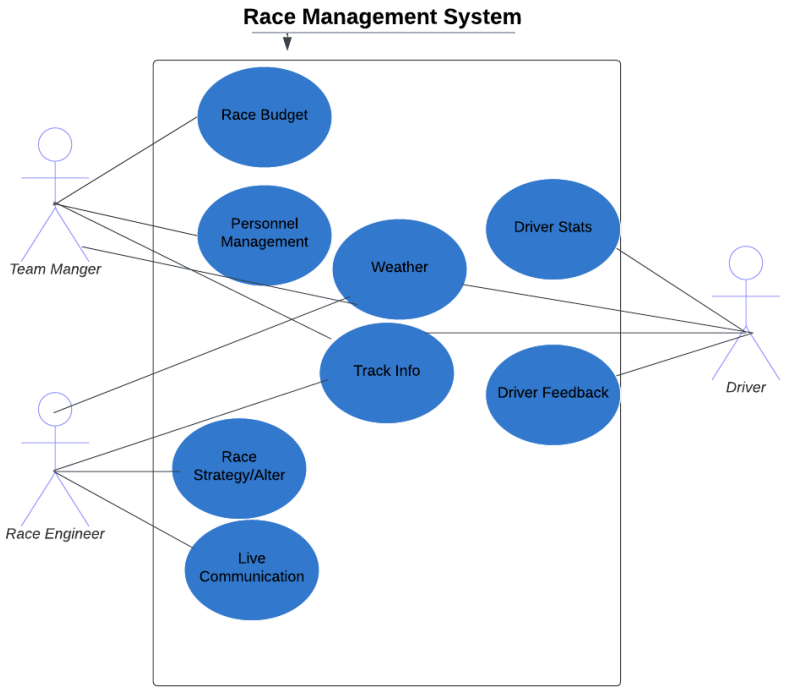
5.5.3. Cost and Delivery Date

– By using Scrum the project's budget will be controlled by focusing on the most valuable features in the beginning based on feedback. This allows for a better turnaround for the users to be able to use the Race Management System (RMS) more quickly and effectively. The iterative nature of Scrum ensures that high-priority features are delivered in short cycles.

5.6. Other Requirements

TBD

5.7. Use-Case Model Diagram



5.8. Use-Case Model Descriptions

5.8.1. Actor: Team Manager (Sergio Ramirez)

- **Race Budget**: The feature allows the team manager to create and display a table-format budget plan for racing events on the website, ensuring calirty over spending.
- **Personnel Management**: This feature allows the team manager to efficiently manage personnel information by adding employees, removing employees, and viewing job titles and other related details in a table format on the website.

5.8.2. Actor: Race Engineer (Christian Leonard)

- **Create Race Stategy**: This feature uses stored car data, track data, and weather forcast to create multiple races strategies. The race engineer can view these strategies and select the best option.

- **Live Communication**: This feature allows the race engineer to communicate with the drivers. It can be used to make car changes or adjustments to race strategy.

5.8.3. Actor: Driver (Blake Joye)

- **Driver Feedback**: In this use case, the primary actor(driver) will provide valuable feedback on the car's performance during the race. The process begins with the user logging into the system, and selecting "Driver Feedback". In this section, they can detail their observations about the car such as handling, performance, or tire grip. Once they are finished, they submit their feedback and it is stored in the system and analyzed.
- **Driver Stats**: Here, the driver can view their personal race statistics such as, race placement, lap times, and podium finishes. The driver can filter through the data by criteria such as race type or location. With this the dirver can identify strengths and weaknesses

5.9. Use-Case Model Scenarios

5.9.1. Actor: Team Manager (Sergio Ramirez)

- **Use-Case Name**: **Race Budget**
  - **Initial Assumption**: The team has a set budget for a race.
  - **Normal**: The team manager creates a race budget plan based on aspects of the race program.
  - **What Can Go Wrong: The system doesnt correctly calculate budget speending causing an error in the budget plan.**
  - **Other Activities**: The team manager will be able to copy and share the budget plan.
  - **System State on Completion**: The function correctly allocates funds and saves the budget plan.
- **Use-Case Name**: **Personnel Managment**
  - **Initial Assumption**: A complete list/table of all employee information.
  - **Normal**: Manager can edit table of employees to add or remove.
  - **What Can Go Wrong**: The system doesnt save changes to the table.
  - **Other Activities**: Add new slots for different information into the table.
  - **System State on Completion**: Table is updated with user information.

5.9.2. Actor: Driver (Blake Joye)

- **Use-Case Name**: Driver Feedback
  - **Initial Assumption**: The Driver is logged into the RMS
  - **Normal**: Driver selects "Driver Feedback" and the Driver provides their feedback then submits it
  - **What Can Go Wrong**: Network connectivity may disrupt submission
  - **Other Activities**: Race engineer and manager can view the feedback
  - **System State on Completion**: Feedback is stored into the system and ready for analysis
- **Use-Case Name**: Driver Stats
  - **Initial Assumption**: Driver is logged into RMS
  - **Normal**: System displays the drivers records
  - **What Can Go Wrong**: May experience difficulties filtering and searching
  - **Other Activities**: Driver can use this information to assess past performancesand make decisions for future races
  - **System State on Completion**: Driver has access to their performance stats.

5.9.3. Actor: Race Engineer (Christian leonard)

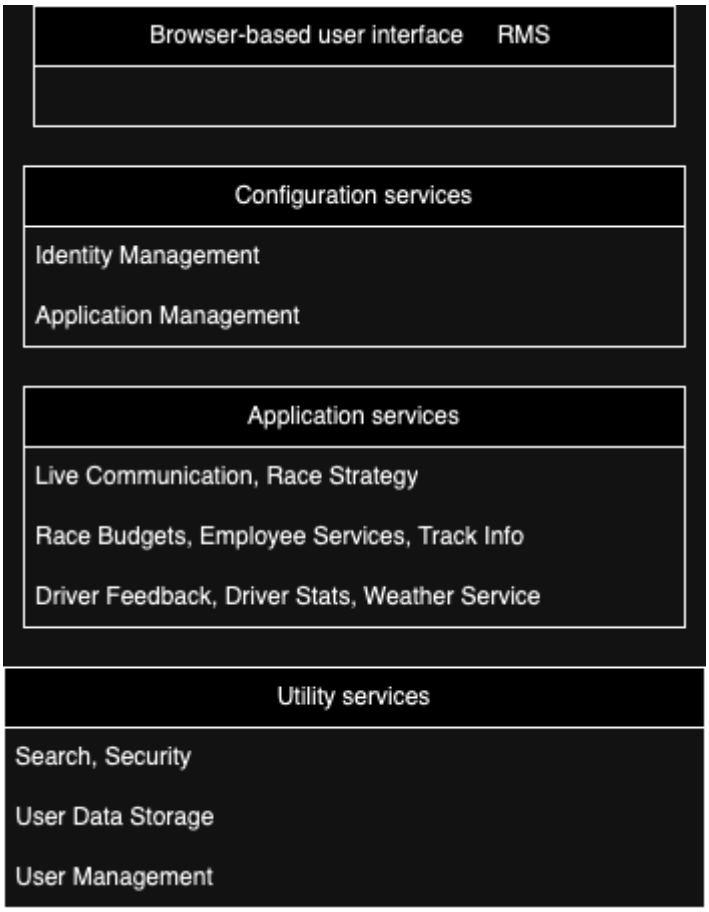
- **Use-Case Name**: Race Strategy

- **Initial Assumption**: Race Engineer is logged into RMS
- **Normal**: Race Engineer selects “Calculate Race Strategy” and selects the best option of those presented be the program.
- **What Can Go Wrong**: May gather incorrect weather forcast, may make incorrect assumption of driver strengths
- **Other Activities**: Race Engineer can use these calculation to predict car and driver difficulties and predict finishing position
- **System State on Completion**: Race Engineer has selected race strategy

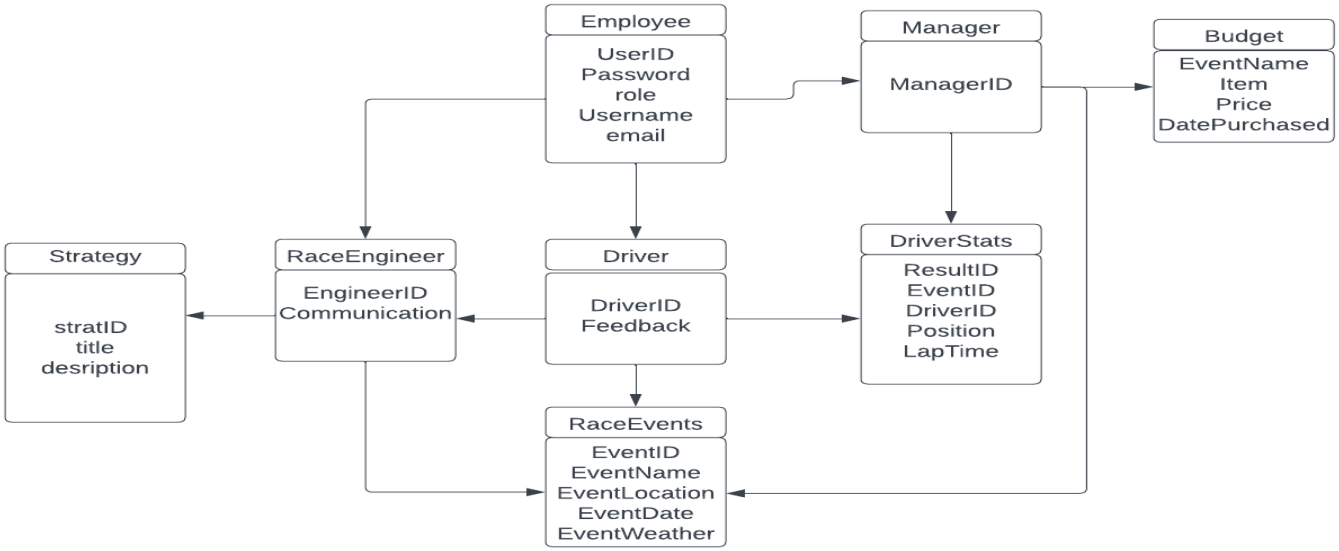
- **Use-Case Name**: Live Communication
  - **Initial Assumption**: Race Engineer is logged into RMS
  - **Normal**: Race Engineer selects “Communicate with Driver,” selects which Driver to communicate with, and writes or reads messages
  - **What Can Go Wrong**: May select incorrect feedback files
  - **Other Activities**: Can be used to gather car and Driver performance data
  - **System State on Completion**: Race Engineer and Driver have sent and read all pending messages

6. Design Documents

6.1. Software Architecture

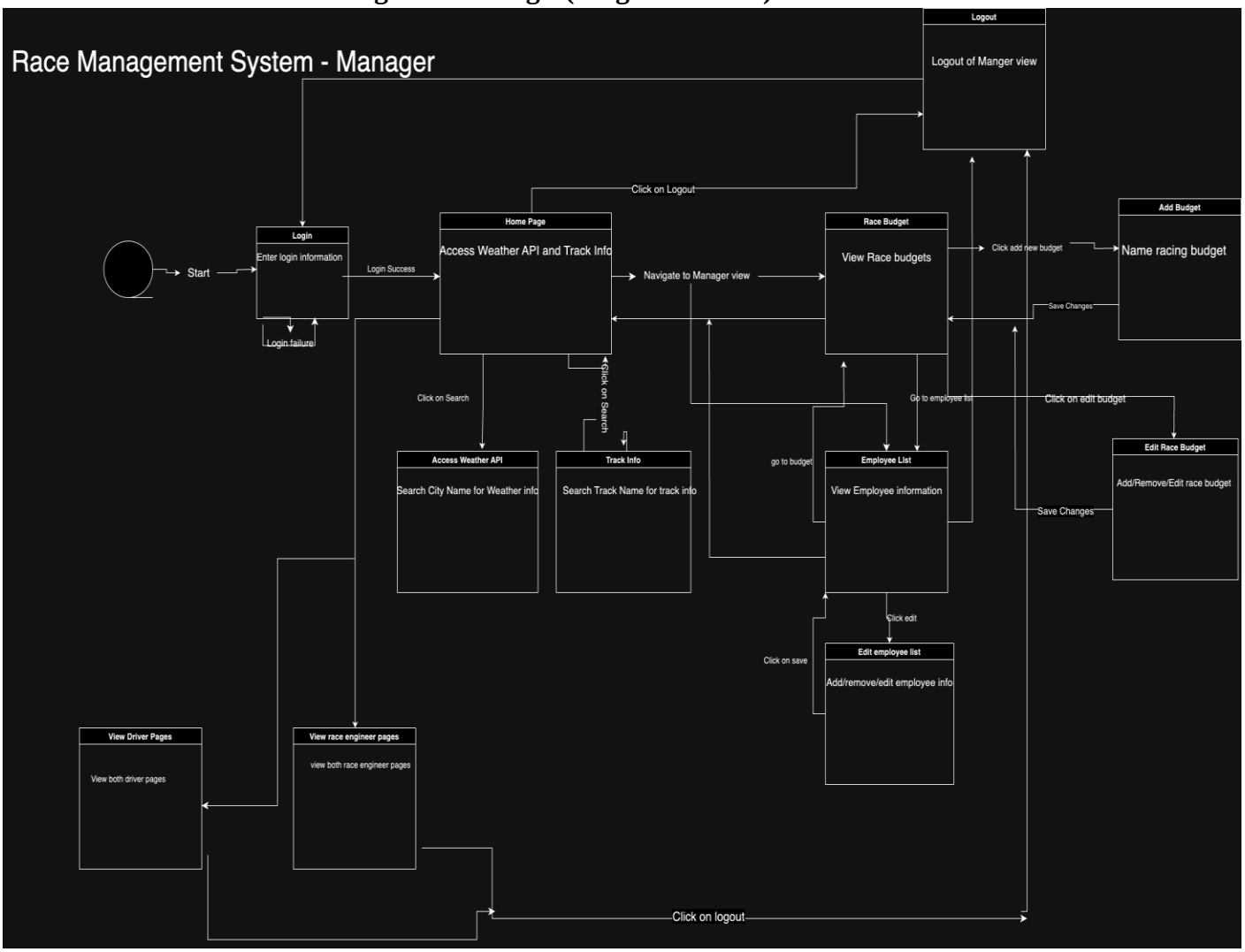


6.2. High-Level Database Schema

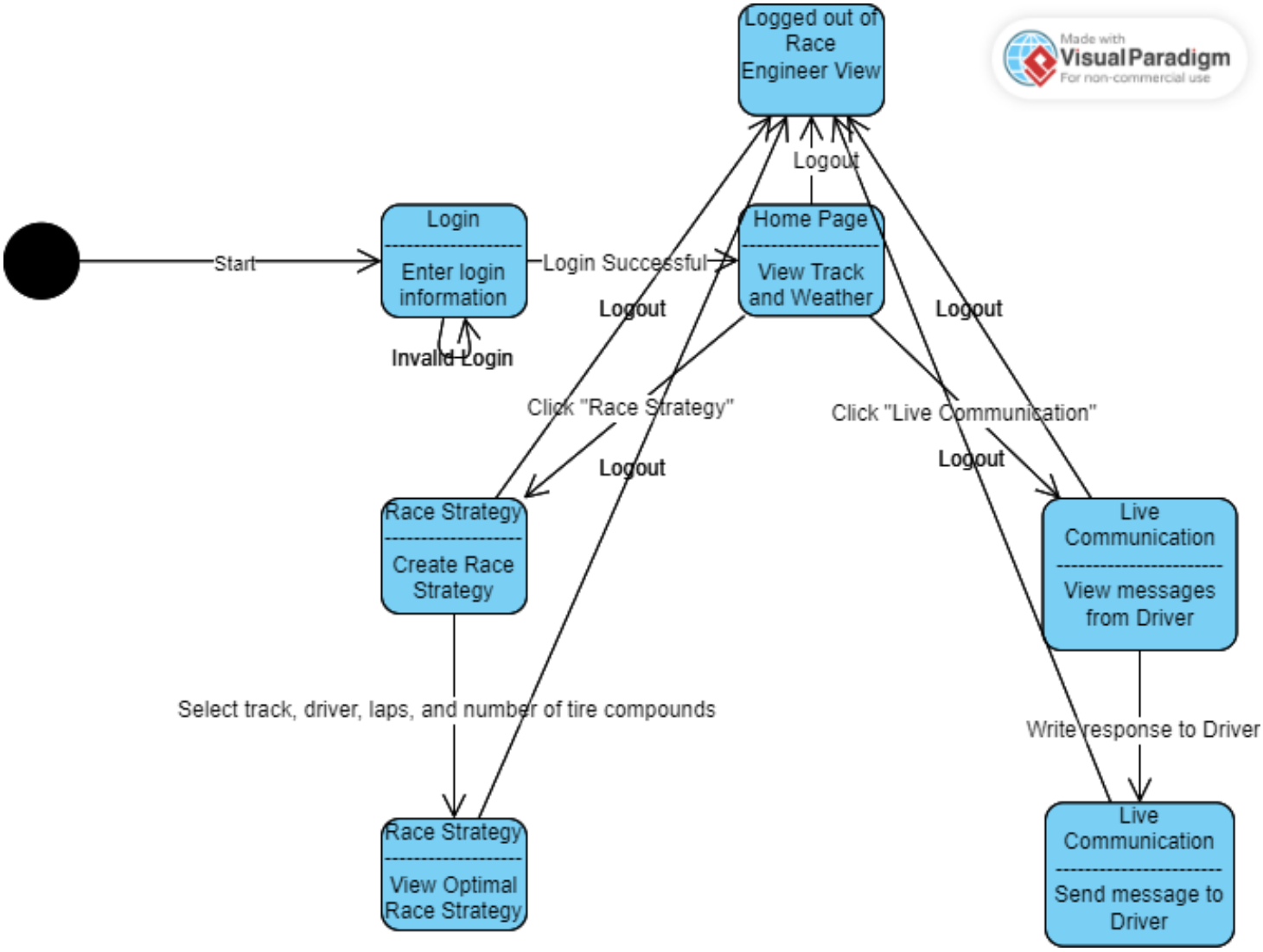


6.3. Software Design

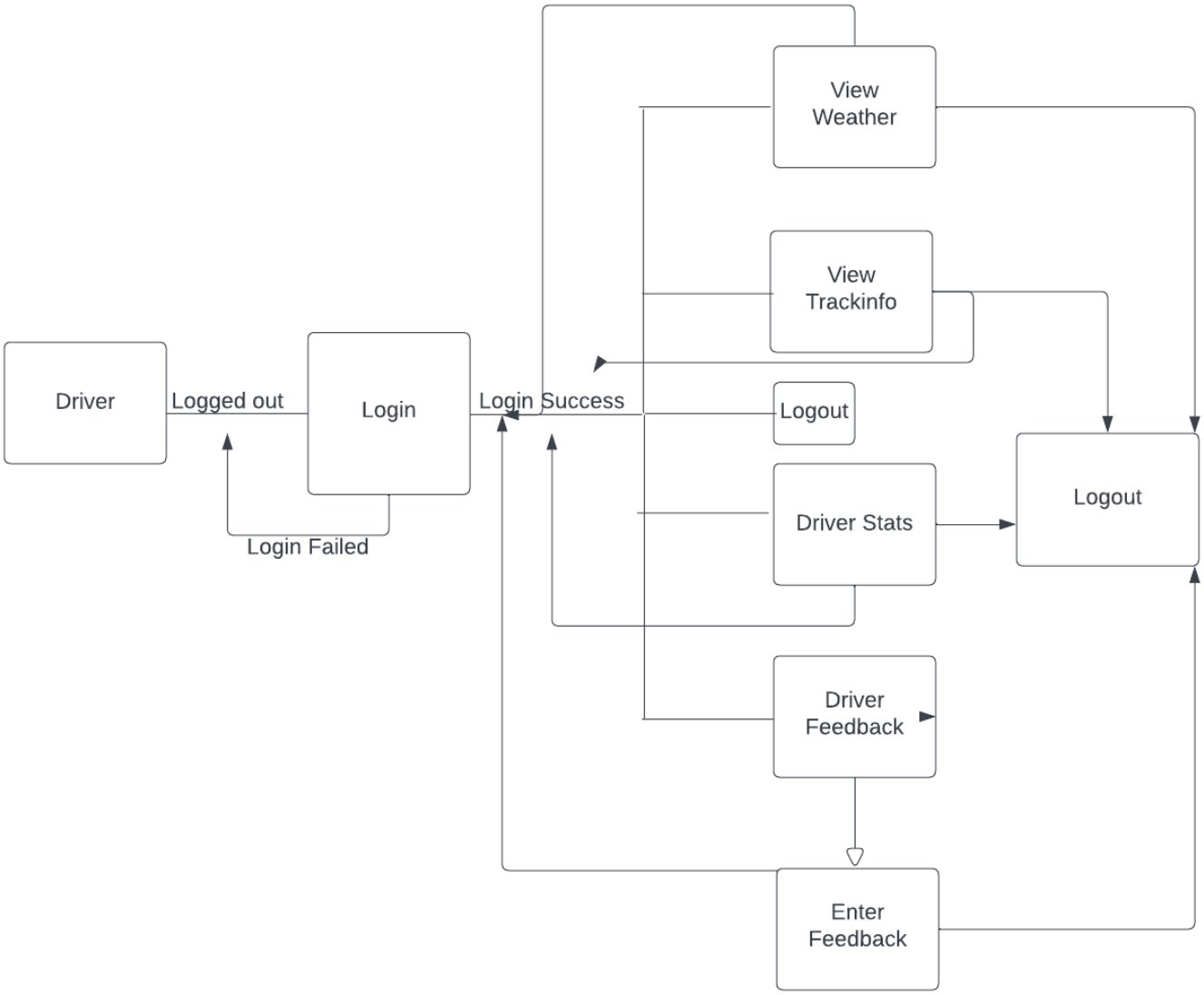
6.3.1. State Machine Diagram: Manager(Sergio Ramirez)



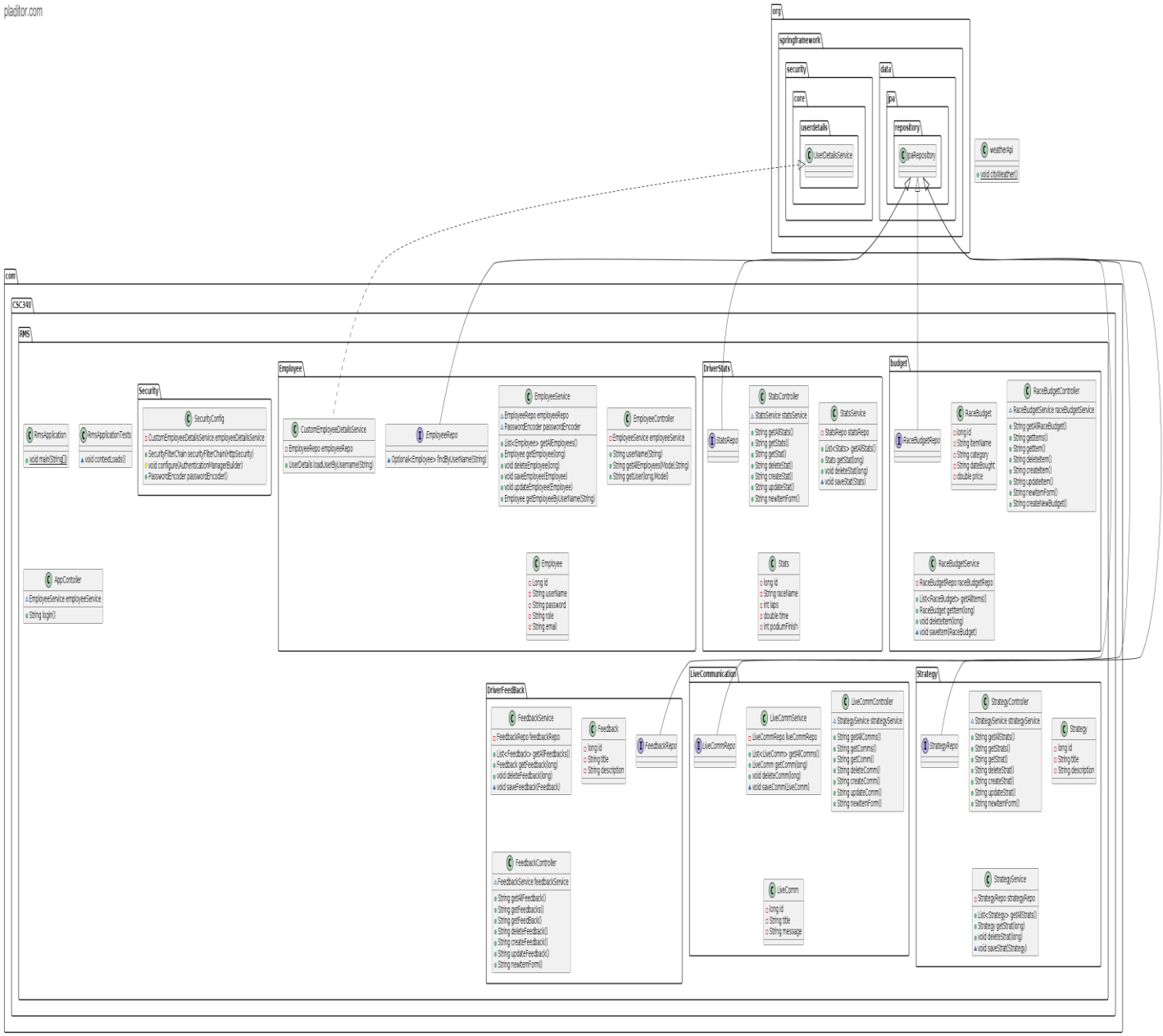
6.3.2. State Machine Diagram: Race Engineer(Christian Leonard)



6.3.3. State Machine Diagram: Driver(Blake Joye)



6.4. UML Class Diagram



7. Scenario

7.1. Brief Written Scenario with Screenshots