Software Engineering Software Requirements Specification (SRS) Document

Thank-The-Bus-Driver

2/14/2023

Version 0.0.1

**By: Joseph Calcagno
Kennedy Ninh
Landon Alison**

Table of Contents

1. Introduction

1.1. Purpose

Thank-The-Bus-Driver (TTBD) is a bus tracking system designed to make public transportation more accessible to the public. This is to reduce wait-times for the average rider and make public transit more efficient for everyone involved.

1.2. Document Conventions

The purpose of this Software Requirements Document (SRD) is to describe client-view (riders, drivers, and administrators), and developer view requirements for our application Thank The Bus Driver (TTBD). The client-oriented requirements include the clients' perspectives, and the different views based on the type of user (rider, driver, admin). The developer-oriented requirements will include a front-end GUI and that accesses an API to use a database of bus system/location data.

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 Thank The Bus Driver application.
MySQL	Open-source relational database management system.
.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	This will be used to implement a function within the software to provide location services and real-time vehicle tracking services.
Github	This is what is used to create a collaborative repository of shared product code.

1.4. Intended Audience

The intended audience of this document is:

- Project developers: Joseph Calcagno, Kennedy Ninh, and Landon Alison.
- Users: people using and/or working for the public transportation system.

- Class instructor: Sunny Ntini.

1.5. Project Scope

The goal of the software is to provide an easy-to-use interface for all involved in the public transportation system. This aligns with the overall
community goals of safe and reliable public transportation for the local community.

The benefits of the project to the local community:

- Reduce wait times at bus stops, saving riders time and stress of guessing and inconsistent bus scheduling.
- Provide bus drivers a more accurate view and scope of their workday.
- Allows administrators to monitor/edit routes and drivers to provide more seamless transportation experience.

1.6. Technology Challenges

[Any technological constraints that the project will be under. Any new technologies you may need to use]
N/A

1.7. References

[Mention books, articles, web sites, worksheets, people who are sources of information about the application domain, etc. Use proper and complete reference notation. Give links to documents as appropriate. You should use the APA Documentation model (Alred, 2003, p. 144).]
Alred, F., Brusaw, C., and Oliu, W. (2003). Handbook of Technical Writing (7th ed.). Boston: Bedford/St. Martin's.
N/A

2. General Description

2.1. Product Perspective

TTBD found its origin in a few college students' desire for public transportation to be more accessible and an easier experience. It was made for people who want to use public transportation by people who are passionate about public transportation.

2.2. Product Features

The product features include the riders and drivers to create accounts and administrators to manipulate those accounts. Riders can look at bus stops around them as well as put money into their bus fare account. Drivers can look at their routes for the day as well as check how much money they have earned. Administrators hold the power to register new bus drivers as well as delete drivers and riders.

2.3. User Class and Characteristics

Our web application does not expect users to have any prior knowledge of a computer, apart from using a web browser. Our application has removed the need for them to know where bus stops in their city are located and removed the need for a rider to guess the location of their bus. It has also removed the need for drivers and riders to manually keep track of money regarding their bus transportation experience.

2.4. Operating Environment

The application is designed to be used on all internet accessible devices.

2.5. Constraints

Due to the knowledge and data in the API we are using, we have limited information about certain cities, including arrival times, stops, and route information.

2.6. Assumptions and Dependencies

The software will be dependent on Spring Web and Thymeleaf to create and execute the MVC architecture that will be developed within NetBeans. The application will also use the TransLoc API (https://rapidapi.com/transloc/api/openapi-1-2/) that will display information about routes, stops, and vehicle locations.

3. Functional Requirements

[Statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations.]

3.1. Primary

- FR0: The system will allow all users to login to their own role in the bus system account using their own email address and password.
- FR1: The system will allow the rider user to enter a location and based on their location the system will provide the nearest bus stops and routes.
- FR2: The system will allow the rider user to deposit money into their bus fare account.
- FR3: The system will allow the driver user to get paid based on their driving records per day.
- FR4: The system will allow the driver user to choose/pick up bus routes that are available.
- FR5: The system will allow the admin user to add and/or remove bus stops and routes.
- FR6: The system will allow the admin user to pay the drivers their respective total pay for the week based on their daily driving record.
- FR7: The system will allow the admin user to add new drivers and remove any drivers.

3.2. Secondary

- Password protection for information only accessible to admins and drivers.
- Location services for all users.

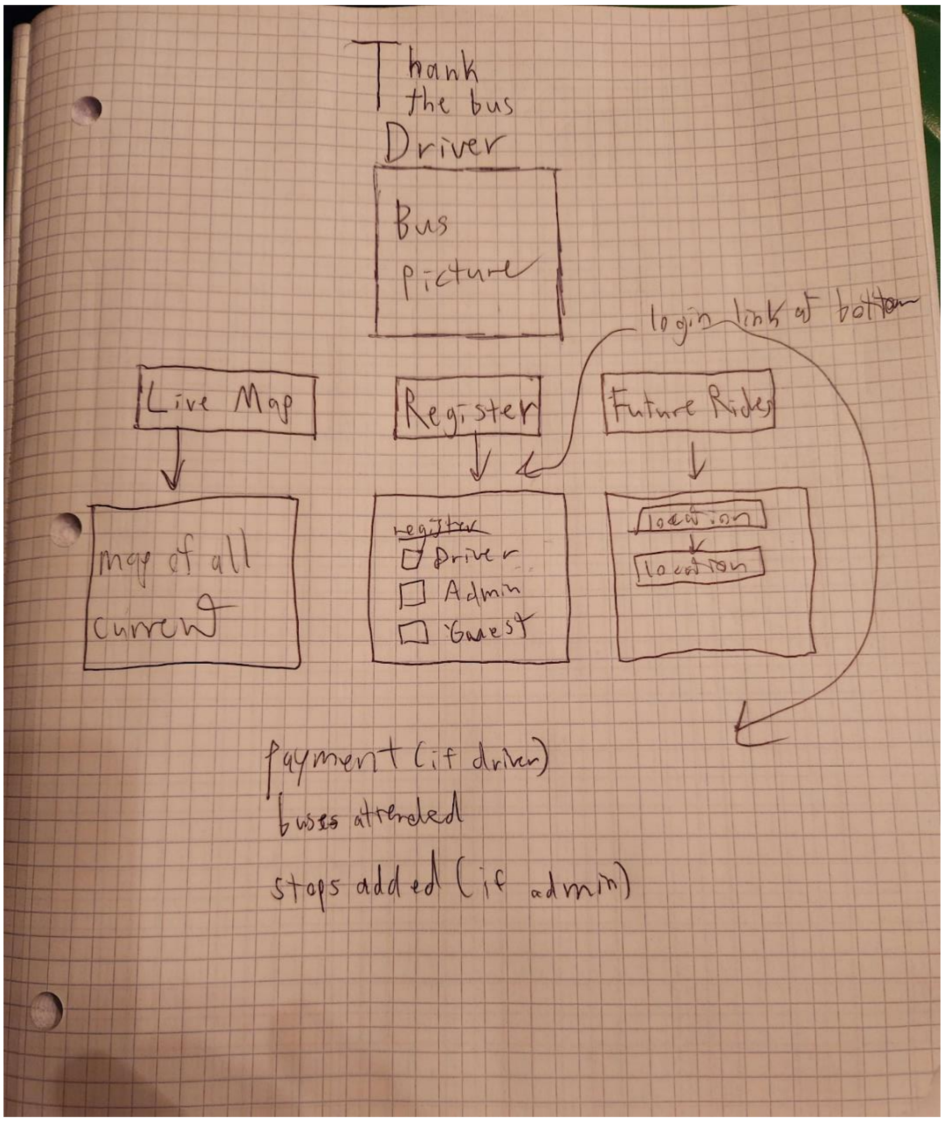
4. Technical Requirements

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



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. However, this program will run best on laptops and desktops.

4.2.3. Communications Interfaces

It must be able to connect to the internet as well as the local database on phpMyAdmin. The communication protocol, HTTP(HTTPS), must be able to connect to the API that is being used and return the routes, bus fares, and other bus data.

4.2.4. Software Interfaces

We will use React and Spring Boot ThymeLeaf to help build the frontend, as well as JPA for the backend database functionality. We will also use Spring Boot with Java to connect the frontend to the backend. We will use our API to connect the backend to the internet application.

5. Non-Functional Requirements

Timing and operational constraints will be the biggest constraints on the program… We will have no control over unpredictable behavior of the base system of transit that we are relying on. If something cancels/runs late, we have no control over those aspects.

5.1. Performance Requirements

- NFR0(R): The local copy of the driver and map database will consume less than – MB of memory
- NFR1(R): The system will consume less than – MB of memory
- NFR2(R): The novice user will be able to schedule in less than – minutes.
- NFR3(R): The expert user will be able to schedule in less than – minute.

5.2. Safety Requirements

There may be some aspects of the application that may unintentionally overlook the qualifications of some of the drivers due to possible loopholes in the driver checking system. A possibility of untrue timings to bus arrival may occur due to faulty circumstances or unpredictable timings also may occur.

5.3. Security Requirements

- NFR4(R): The system will only be usable by authorized users.
- NFR5(R): The private data of a user will only be available to the specified user or an administrator.

5.4. Software Quality Attributes

Below is the attribute list that will be important to know when it comes to using the software.

5.4.1. Availability

This will be available on any device which is available to be portable too. This device needs the necessary software to be developed, as well as to be available to be run on the local machine.

5.4.2. Correctness

The correctness of this software is reliant on the data given by the API, and by proxy the data of the available bus driver data given to the software.

5.4.3. Maintainability

If the device still maintains the ability to run java 17, and the API used is still being maintained and/or available to us, then the software will be accurately maintained.

5.4.4. Reusability

This software is reusable as far as the API and hardware allows, leaving the hardware as the most limiting factor for reusability.

5.4.5. Portability

This program will be portable from all respective platforms that support the software. (See 4.2.2)

5.5. Process Requirements

The process requirements include team members being frequently available to meet and discuss the project, as well as "scrum" meetings to work out bugs and kinks in our current process.

5.5.1. Development Process Used

The project will be completed in an agile environment. We have an individual over processes environment and have frequent meetings to plan and work on the project.

5.5.2. Time Constraints

The project is expected to be completed before the date of April 18th 2023. This falls right before the end of the UNCG spring 2023 semester.

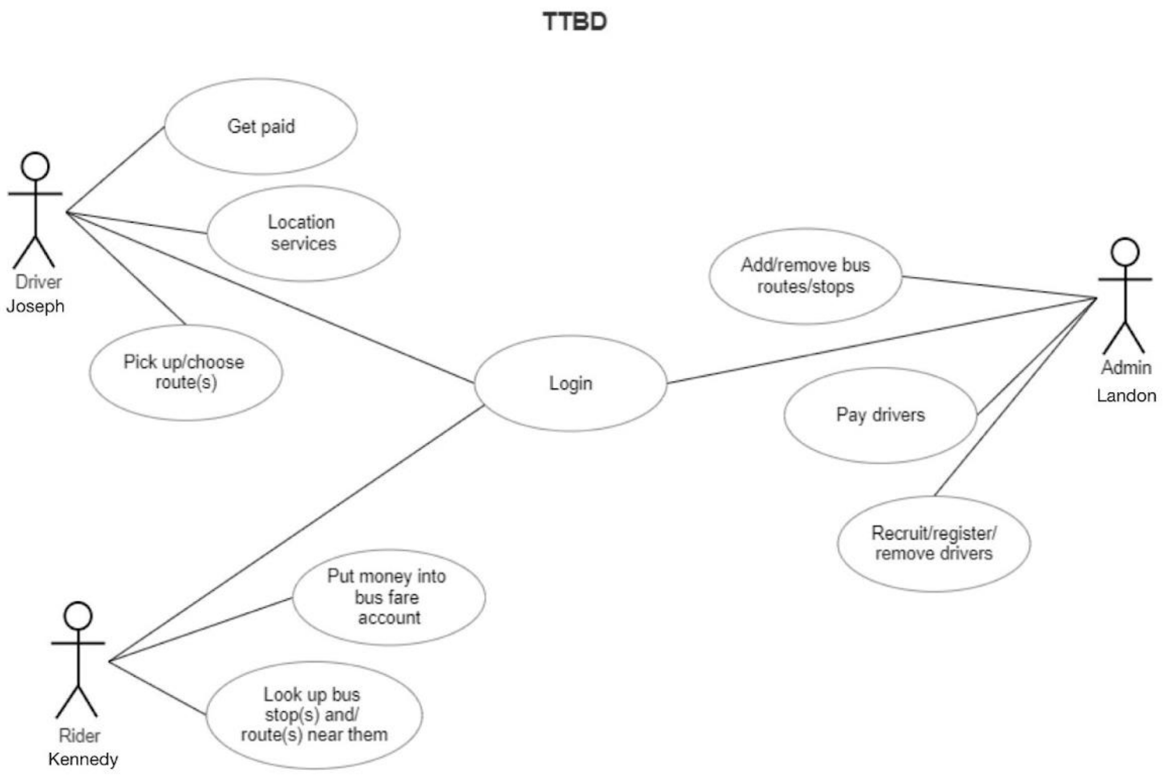
5.5.3. Cost and Delivery Date

The cost of the project should be free, we are accessing a free API service and as with most traditional school projects, we are not getting paid. The delivery date, where we present our project, is targeted to be either April 18th or April 20th.

5.6. Other Requirements

We have been required as a group to construct this project along with this SRD, a use case diagram, a formal project proposal, code reviews, design documents and a GitHub repository.

5.7. Use-Case Model Diagram


TTBD
Get paid
Location services
Driver
Joseph
Pick up/choose route(s)
Login
Add/remove bus routes/stops
Admin
Landon
Pay drivers
Recruit/register/ remove drivers
Put money into bus fare account
Look up bus stop(s) and/ route(s) near them
Rider
Kennedy

5.8. Use-Case Model Descriptions

5.8.1. Driver: Joseph

- **Get Paid**: The driver will be able to see how much money they have made depending on the routes they drove and/or the time spent driving.
- **Pick up/ Choose routes**: The driver will be able to look at routes and stops beforehand and pick which routes they prefer to drive throughout their shift.
- **Location Services:** The bus driver should be able to keep track of where they are during their route and see where other buses currently are.
- **Login:** The driver shall be able to login into the system using the credentials they input themselves such as email address and password. After logging into the system, the rider can view different options to choose from.

5.8.2. Rider: Kennedy

- **Login:** The rider shall be able to login into the system using the credentials they input themselves such as email address and password. After logging into the system, the rider can view different options to choose from.
- **Put money into bus fare account:** The rider can add their desired amount of money into their bus fare account to be able to pay for their bus ride.
- **Look up bus stop(s) and/ route(s) near them:** The rider shall be able to input their location and based on their location they can choose their desired stop/route closest to them.

5.8.3. Admin: Landon

- **Add/remove bus routes/bus stops:** The admin will be able to control routes and stops in local area.
- **Pay drivers:** The admin will be able to disperse funds to drivers based on driver work time.
- **Recruit/register/remove drivers:** Delete and manipulate accounts for various reasons.

5.9. Use-Case Model Scenarios

5.9.1. Admin: Landon

- **Use-Case Admin**: Add/remove bus routes/bus stops
  - **Initial Assumption:** There is a set of registered bus stops that can be manipulated
  - **Normal:** The admin adds potential bus stops, removes potentially unruly bus stops that currently exist.
  - **Initial Assumption:** There is a set of bus stops that are available to the public, with a favorable route.
  - **What Can Go Wrong:** There can be a faulty route/stop due to construction, vandalism, or other activities that can inhibit the ability of the admin to make more informed decisions.
  - **Other Activities:** Use current route and traffic news to make more informed decisions regarding the efficacy of certain routes.
  - **System State on Completion:** Can view map, drivers, and routes.

- **Use-Case Admin:** Pay drivers
  - **Initial Assumption:** each driver has a specific way that they can be paid through the app
  - **Normal:** Each driver has a rate that is paid and gets paid at the end of each ride.
  - **What Can Go Wrong:** A driver can bail, and falsify the driving that they claim they did, claiming payment that should not go to them.
  - **Other Activities:** choose to not take a route and not get paid for it
  - **System State on Completion:** The driver can check their balance and see how much they will get paid for a job.

- **Use-Case Name:** recruit/register/remove drivers
  - **Initial Assumption:** There is a pool of drivers that can be recruited and can be registered through the admin.
  - **Normal:** A group of drivers get registered, and a couple get removed for various circumstances.
  - **What Can Go Wrong:** Faulty approval of a driver may occur
  - **Other Activities:** A driver could be recruited but may be denied registration.

● **System State on Completion:** A driver could register and wait for approval from the administrator.

5.9.2. Rider: Kennedy

- **Use-Case Name**: Login
    - ● **Initial Assumption**: The rider has a registered account to login to the system. The rider's account is saved in the database.
    - ● **Normal**: The rider will enter an email address and password to log into their account.
    - ● **What Can Go Wrong**: The rider's credentials are not accepted because either the email address or password does not match the credentials stored in the database. The rider should be able to request a password reset.
    - ● **Other Activities:** The rider can reset the password using the forgotten password link.
    - ● **System State on Completion:** The rider is logged in. They can view the main page.

- **Use-Case Name:** Put money into bus fare account
    - ● **Initial Assumption:** The rider has logged into the system and can view their account and their current fare balance.
    - ● **Normal:** The rider can connect their payment method to the system and link to their account and from their payment method they can add their desired amount into their fare account.
    - ● **What Can Go Wrong:** The payment method the rider added could be declined. Therefore, they could not add money to their account.
    - ● **Other Activities:** The rider can remove funds from their account.
    - ● **System State on Completion:** The funds are added successfully. It is updated to show their balance.

- **Use-Case Name**: Look up bus stop(s) and/ routes near them
    - ● **Initial Assumption**: The rider has logged into the system and has entered their location and can view the bus stops and routes near them.
    - ● **Normal:** The rider will choose their preferred stop and will be provided directions to the stop.
    - ● **What Can Go Wrong:** The rider's location services could be turned off. The routes and stops could potentially be out of date/out of service and may have not been updated to show.
    - ● **Other Activities**: The rider can set their preferred stop and route to check on how far out a bus is to their stop.
    - ● **System State on Completion:** The rider is seeing the routes and stops accurately. The rider is also able to get to their stop correctly.

5.9.3. Driver: Joseph

- **Get Paid:**
    - ● **Initial Assumption:** The driver will be able to a dollar amount available to them based on routes and time spent driving.
    - ● **Normal:** The driver can view their pay for their time spent working.
    - ● **What Can Go Wrong:** Due to inconsistent access to location services or vehicle locations, or route and stops, pay might not be accurately calculated 100 percent of the time.
    - ● **Other Activities:** The driver can cash out funds that are available to them.

- **System State on Completion:** The driver is paid for their work. Their account is updated to show their balance.

- Location Services:

- **Initial Assumption:** The driver will be able to see the current bus they are driving as well as other buses currently in route.
- **Normal:** The application displays a location of a bus.
- **What Can Go Wrong:** API not being able to track the bus or the location services not being online causing the bus to not be able to be tracked.
- **Other Activities:** Seeing how long to their next stop and being able to alert and update passengers on travel time.
- **System State on Completion:** The driver can look at current buses working and follow their bus/ other busses along routes.

-Pick/Chose routes

- **Initial Assumption:** The driver will be able to see a list of routes that need to be driven and pick their preference as to which route they are driving.
- **Normal:** The routes will be in list form and display stops and other relevant information.
- **What Can Go Wrong:** Route information from API might not be up to date or available.
- **Other Activities:** The driver can inform admin or riders of position to update travel times an information.
- **System State on Completion**: Driver sees a list of routes to pick from and puts in a request to drive that route that day.

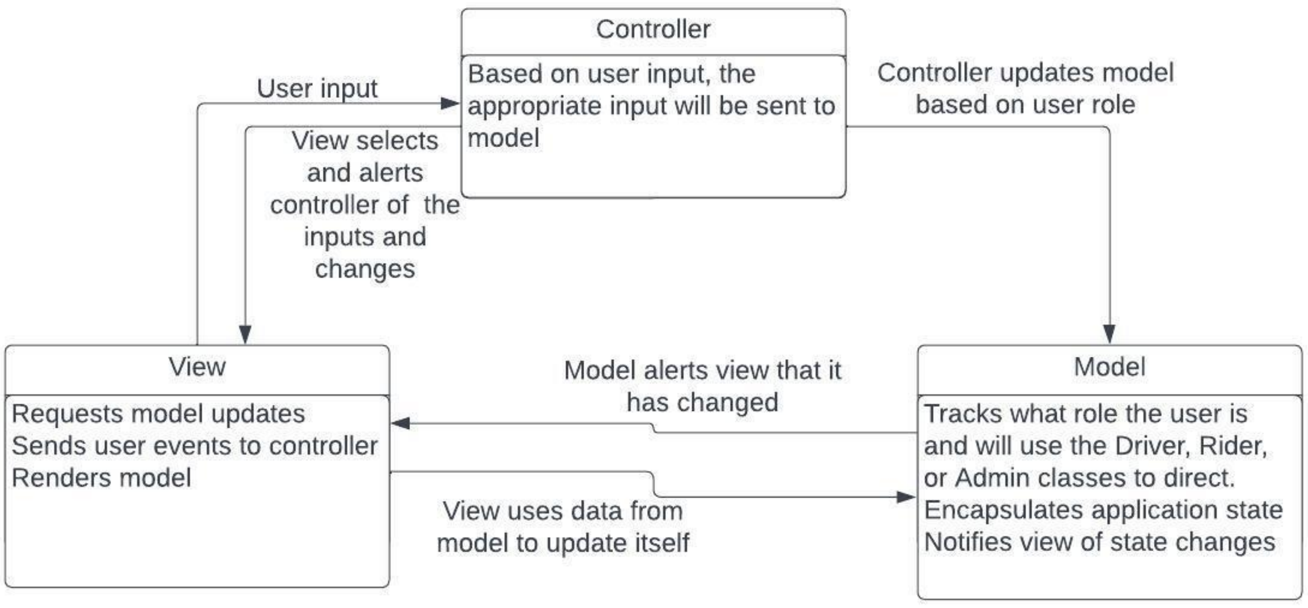
-Login

- **Initial Assumption:** The driver has a registered account to login to the system. The driver's account is saved in the database.
- **Normal:** The driver will enter an email address and password to log into their account.
- **What Can Go Wrong:** The rider's credentials are not accepted because either the email address or password does not match the credentials stored in the database. The rider should be able to request a password reset.
- **Other Activities:** The driver can reset the password using the forgotten password link.
- **System State on Completion:** The driver is logged in. They can view the main page.

6. Design Documents

6.1. Software Architecture

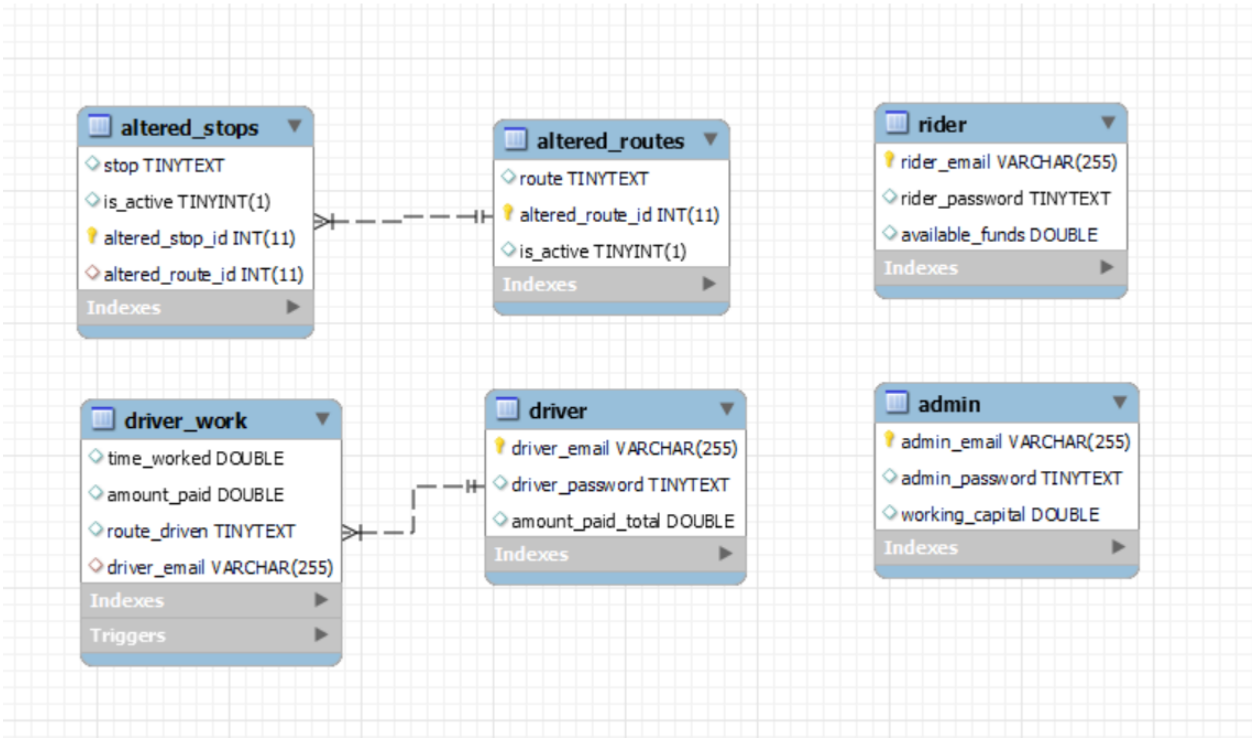
MVC:



Controller
Based on user input, the appropriate input will be sent to model
User input
View selects and alerts controller of the inputs and changes
Controller updates model based on user role
View
Requests model updates
Sends user events to controller
Renders model
Model alerts view that it has changed
Model
Tracks what role the user is and will use the Driver, Rider, or Admin classes to direct.
Encapsulates application state
Notifies view of state changes
View uses data from model to update itself

Controller uses the 'User' interface and it's methods to determine the user's role.
Model uses the Driver, Rider, and Admin classes based on the user's role to get to their appropriate views
View uses the data from the selected class from Model to keep up to date with user input and information changes

6.2. High-Level Database Schema



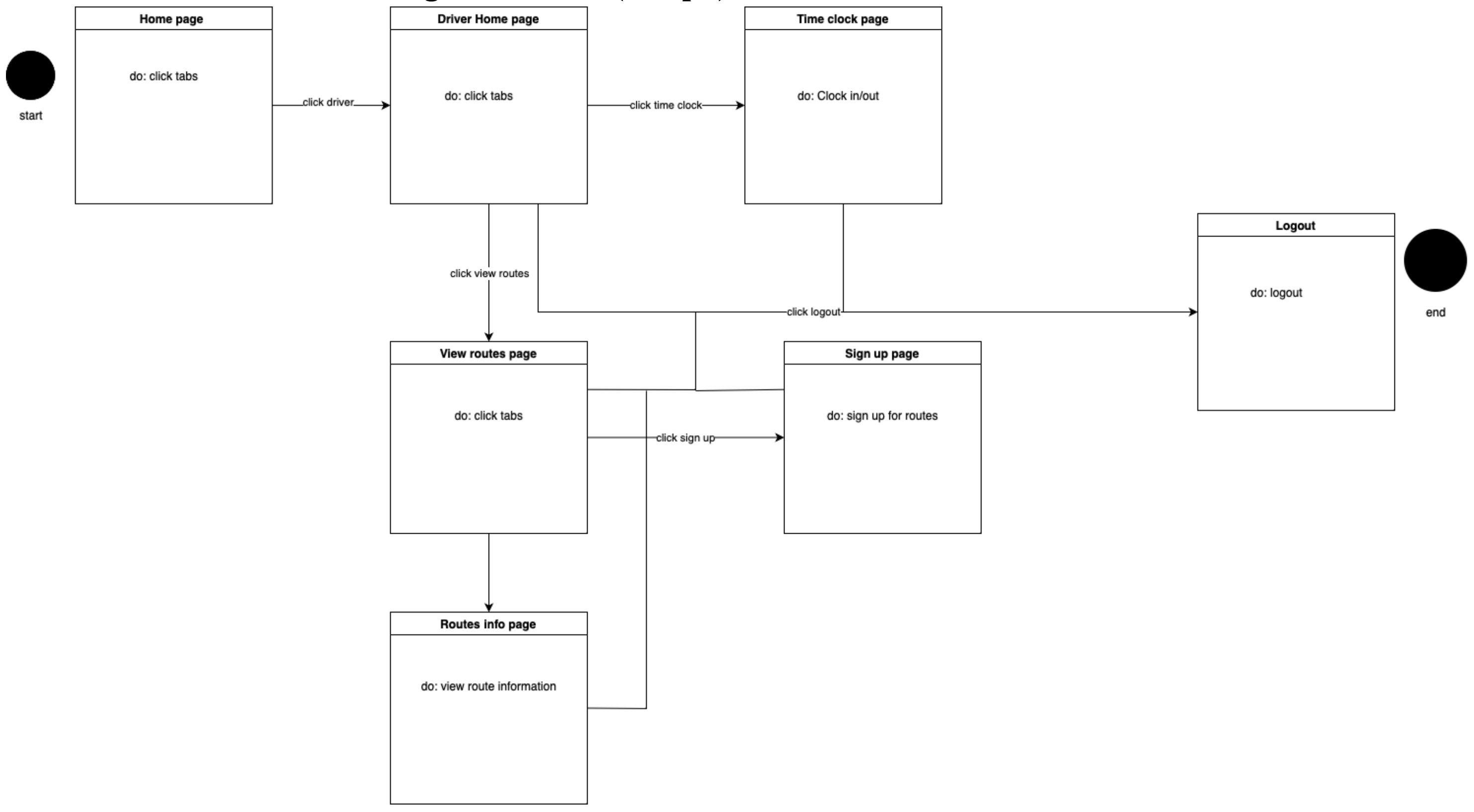
The assumptions made are the following:

1. There will be a trigger that will automatically deduct the amount of money from working_capital.admin to amount_paid.driver…

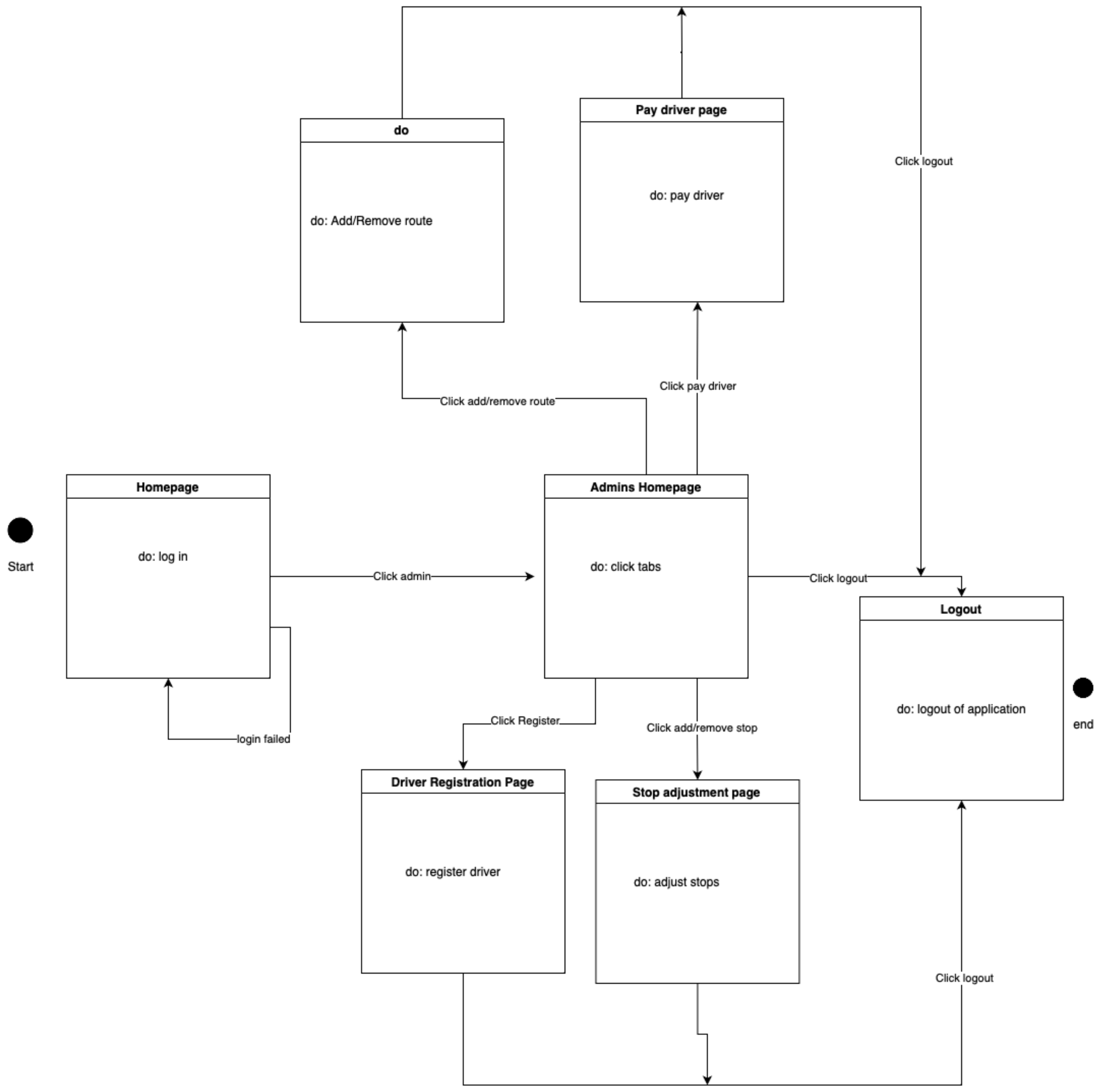
2. Amount_paid_total will be a composite attribute that will have the sum of all the rows from the amount_paid.driver_work attribute for the matching driver_email.
3. The added and removed routes (altered_routes)will be used to modify the available routes, these entities will only be available to the admin. The same applies to the stop entities(altered_stops).

6.3. Software Design

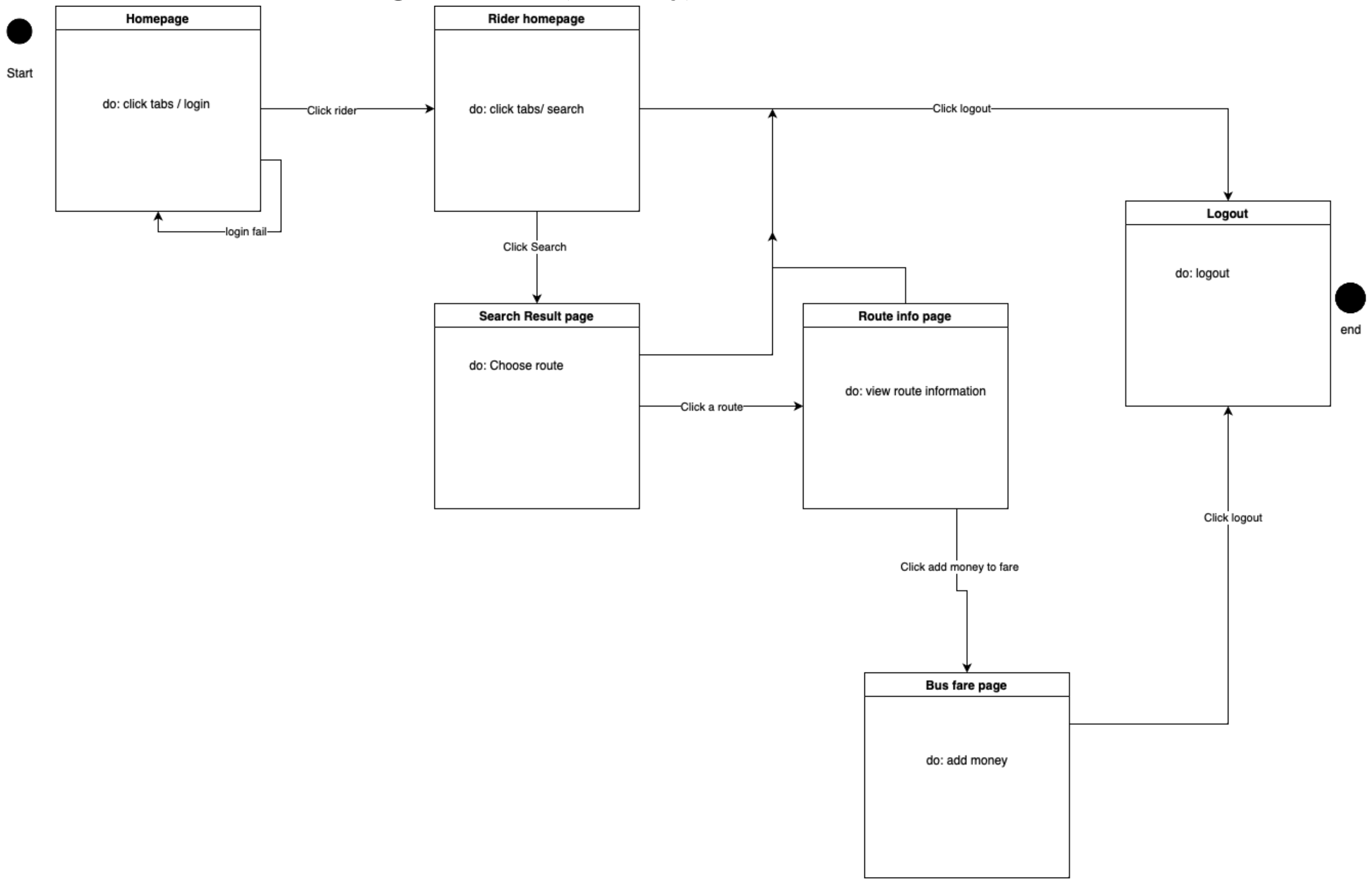
6.3.1. State Machine Diagram: Driver (Joseph)



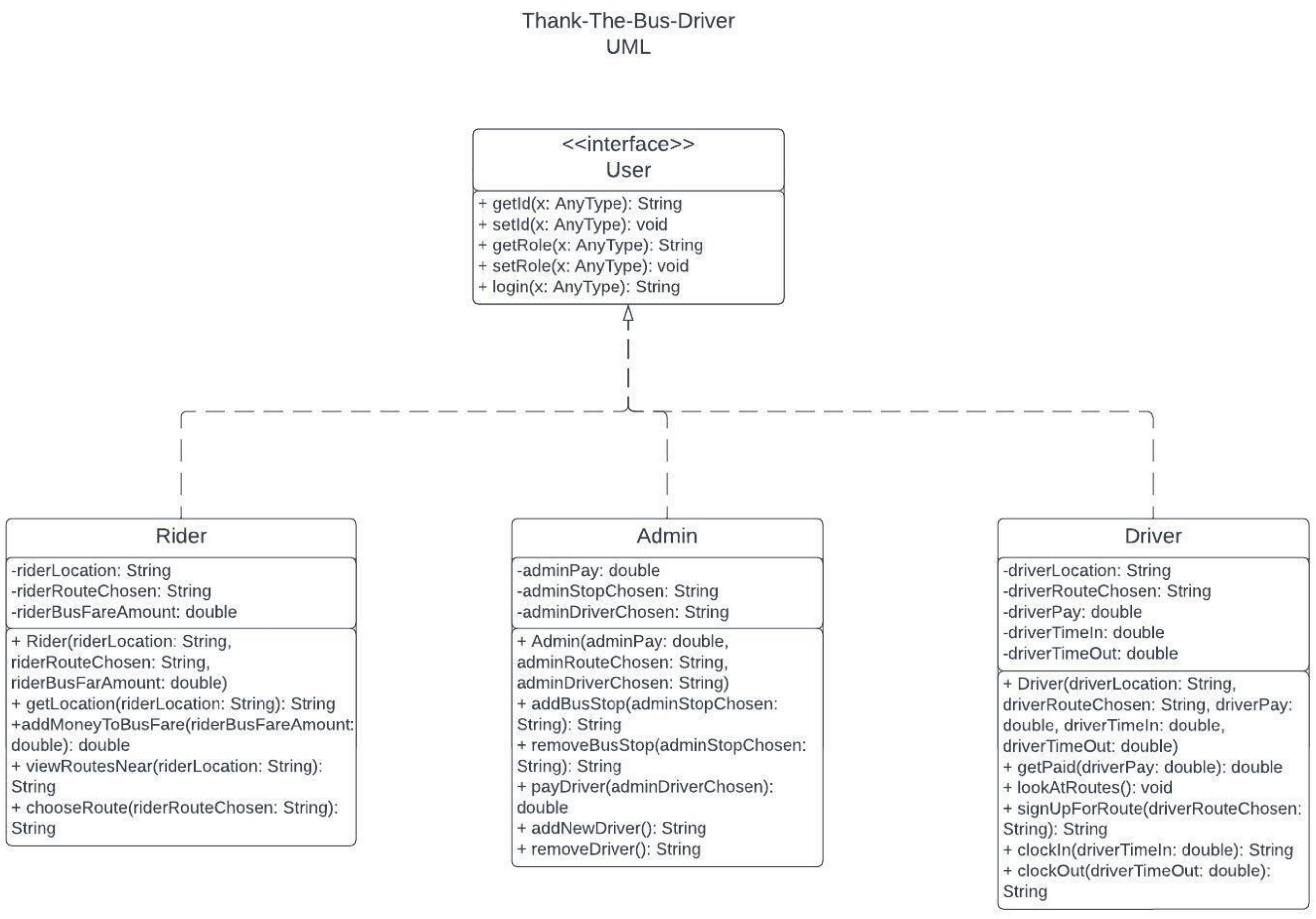
6.3.2. State Machine Diagram: Admin (Landon)



6.3.3. State Machine Diagram: Rider (Kennedy)



6.4. UML Class Diagram


Thank-The-Bus-Driver
UML
<<interface>>
User
+ getId(x: AnyType): String
+ setId(x: AnyType): void
+ getRole(x: AnyType): String
+ setRole(x: AnyType): void
+ login(x: AnyType): String
Rider
-riderLocation: String
-riderRouteChosen: String
-riderBusFareAmount: double
+ Rider(riderLocation: String, riderRouteChosen: String, riderBusFarAmount: double)
+ getLocation(riderLocation: String): String
+addMoneyToBusFare(riderBusFareAmount: double): double
+ viewRoutesNear(riderLocation: String): String
+ chooseRoute(riderRouteChosen: String): String
Admin
-adminPay: double
-adminStopChosen: String
-adminDriverChosen: String
+ Admin(adminPay: double, adminRouteChosen: String, adminDriverChosen: String)
+ addBusStop(adminStopChosen: String): String
+ removeBusStop(adminStopChosen: String): String
+ payDriver(adminDriverChosen): double
+ addNewDriver(): String
+ removeDriver(): String
Driver
-driverLocation: String
-driverRouteChosen: String
-driverPay: double
-driverTimeIn: double
-driverTimeOut: double
+ Driver(driverLocation: String, driverRouteChosen: String, driverPay: double, driverTimeIn: double, driverTimeOut: double)
+ getPaid(driverPay: double): double
+ lookAtRoutes(): void
+ signUpForRoute(driverRouteChosen: String): String
+ clockIn(driverTimeIn: double): String
+ clockOut(driverTimeOut: double): String