# CSC 431 <Train Tracker Miami Dade> System Architecture Specification (SAS)

**<Team 1>**

|  |  |
| --- | --- |
| <Ben Oberg> | <Back-end Developer> |
| <Spencer Terwilliger> | <Risk Management> |
| <Jacob Smyth> | <Front End Developer> |

# Version History

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Author(s)** | **Change Comments** |
| **1.1** | 02/20/2023 | Ben Oberg  Spencer Terwilliger  Jacob Smyth | First Draft (SRS) |
| **1.2** | 04/01/2023 | Ben Oberg  Spencer Terwilliger  Jacob Smyth | SAS |

Table of Contents

1. System Analysis 4

1.1 System Overview 4

1.2 System Diagram 4

1.3 Actor Identification 4

1.4 Design Rationale 4

1.4.1 Architectural Style 4

1.4.2 Design Pattern(s) 4

1.4.3 Framework 4

2. Functional Design 5

3. Structural Design 6

3.1 Class Diagram of System 6

# Table of Figures

1.2 System Diagram

2.1 Sequence Diagrams

2.1.1 Account Services

2.1.2 Searching for Train Stations

3.1 Class Diagrams

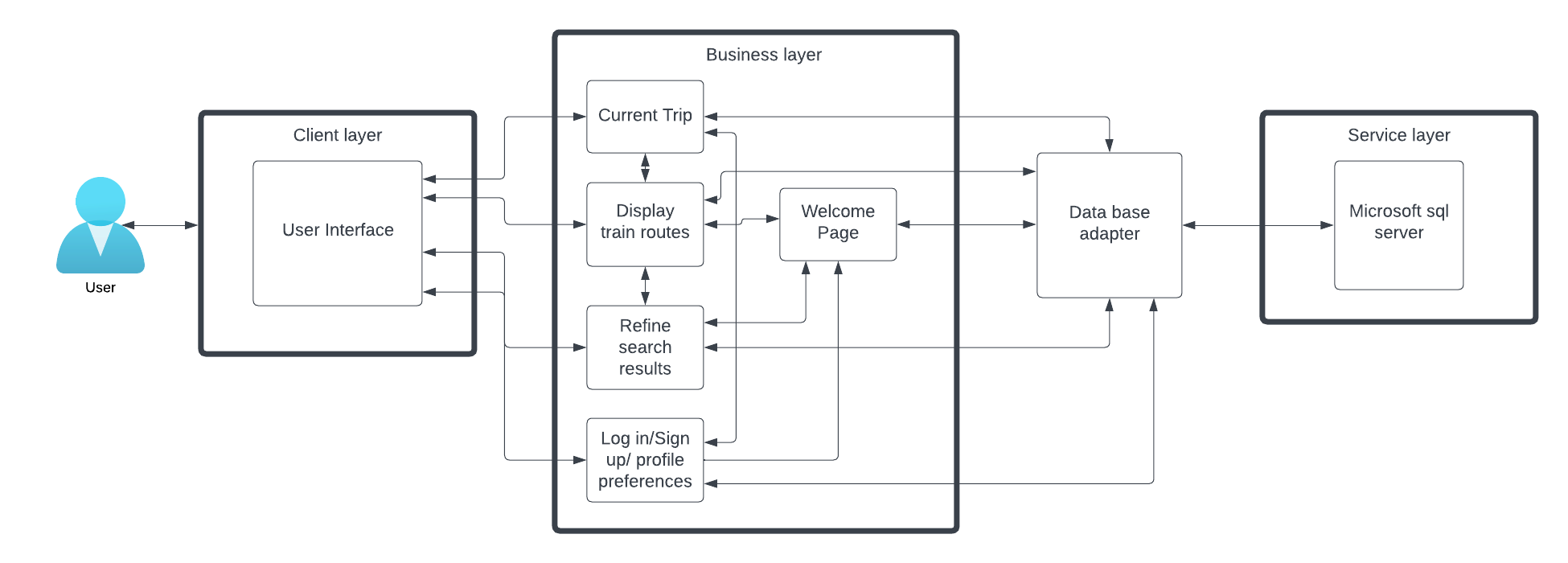
1. System Analysis

#### 1.1 System Overview

*The system we designed for Train Tracker: Miami Dade will be using a three-tier architecture system. This system will include a client, business, and service layer. Our client server will support both IOS and Android devices. The client server will access the business layer in order to properly display the user interface. The business layer will take input from the client server such as the name of a train station or their location. It will use this information to display arrival times of trains at nearby stations on the user interface. The business* *layer will also search for the fastest routes based on the user’s location and destination.*

*The client layer will display the user interface. The business layer will allow the user to click on the search bar, look at suggested routes, and make changes to their account all through various components. The business* *layer will make calls to the service layer in order to retrieve the information needed.*

#### 1.2 System Diagram

**

#### 1.3 Actor Identification

*User- The people using our app in order to save time throughout their day.*

*Admin- The people updating our app and ensuring there are no bugs.*

*Train Station Worker- Workers who ensure that the times are accurate.*

*Server- This actor will store our user data and train data within the database.*

#### 1.4 Design Rationale

##### 1.4.1 Architectural Style

*We will be using a three-tiered architecture style. This fits our needs perfectly because the user interface will reside on our users' phones. The application logic residing on the server side is also good for a mobile application because we do not want to take up too much storage on our users' phones. We will also need our database to be quick and effective to save our user’s time.*

##### 1.4.2 Design Pattern(s)

*Factory- We will be using the factory method because we wish to eventually expand from just train stations to other forms of public transportation. It will be best to leave the options of subclasses available in our future.*

*Decorator- We will be using the decorator method to create relationships between our classes. We chose this method because we wish to add on to our source code for each class instead of creating another class.*

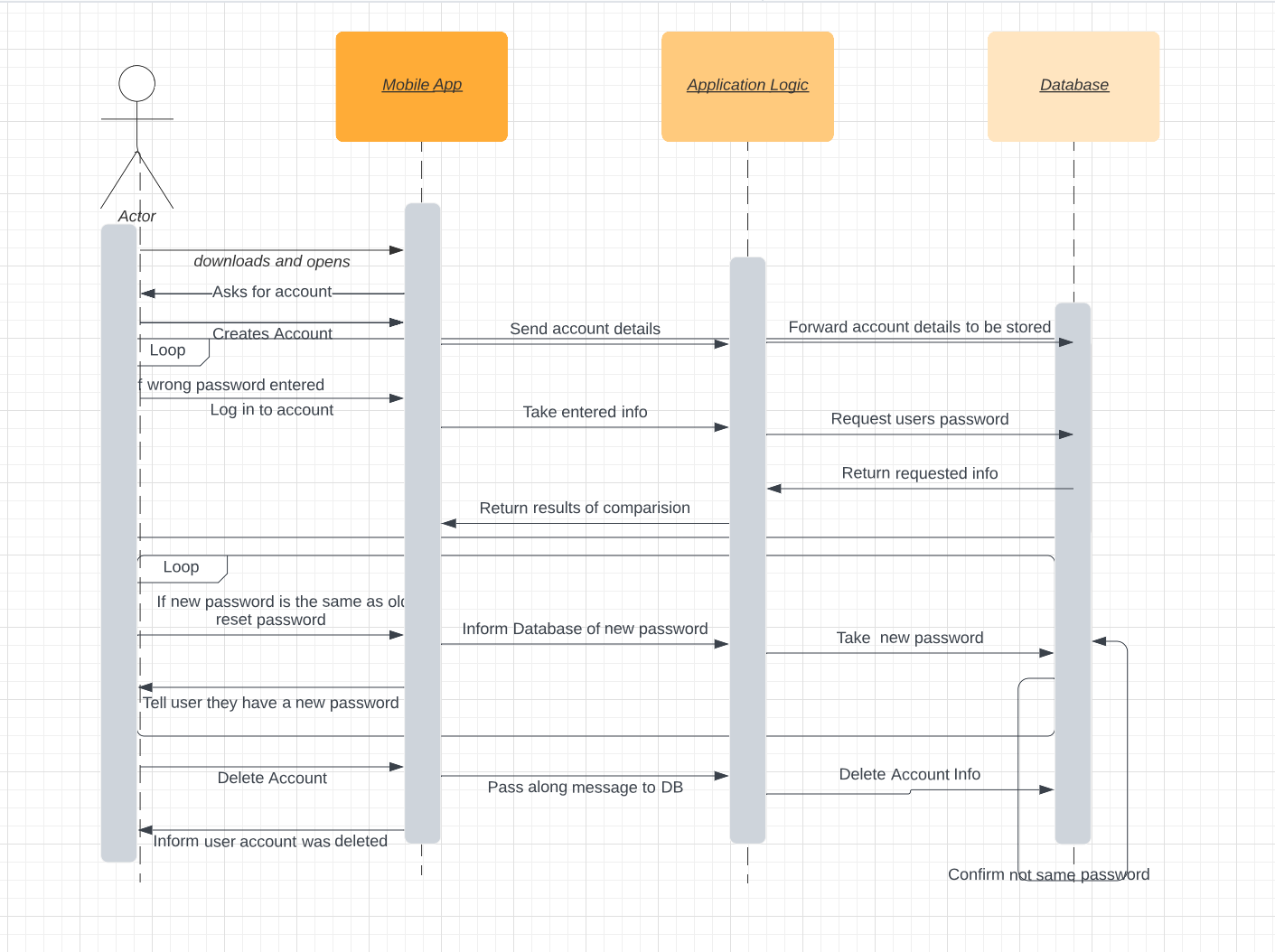
*Command- We will be using the command method for our classes to communicate. We believe it will be best for our classes to communicate through another object as it will help standardize communication throughout our architecture.*

##### 1.4.3 Framework

*React Native- We will be using React Native for our framework. We landed on this framework because it makes creating a cross platform app simple. React works with IOS and Android which is perfect for our app. This will allow us to reach as many users as possible. It also makes it easier for code to be reused, which is another feature we looked for.*

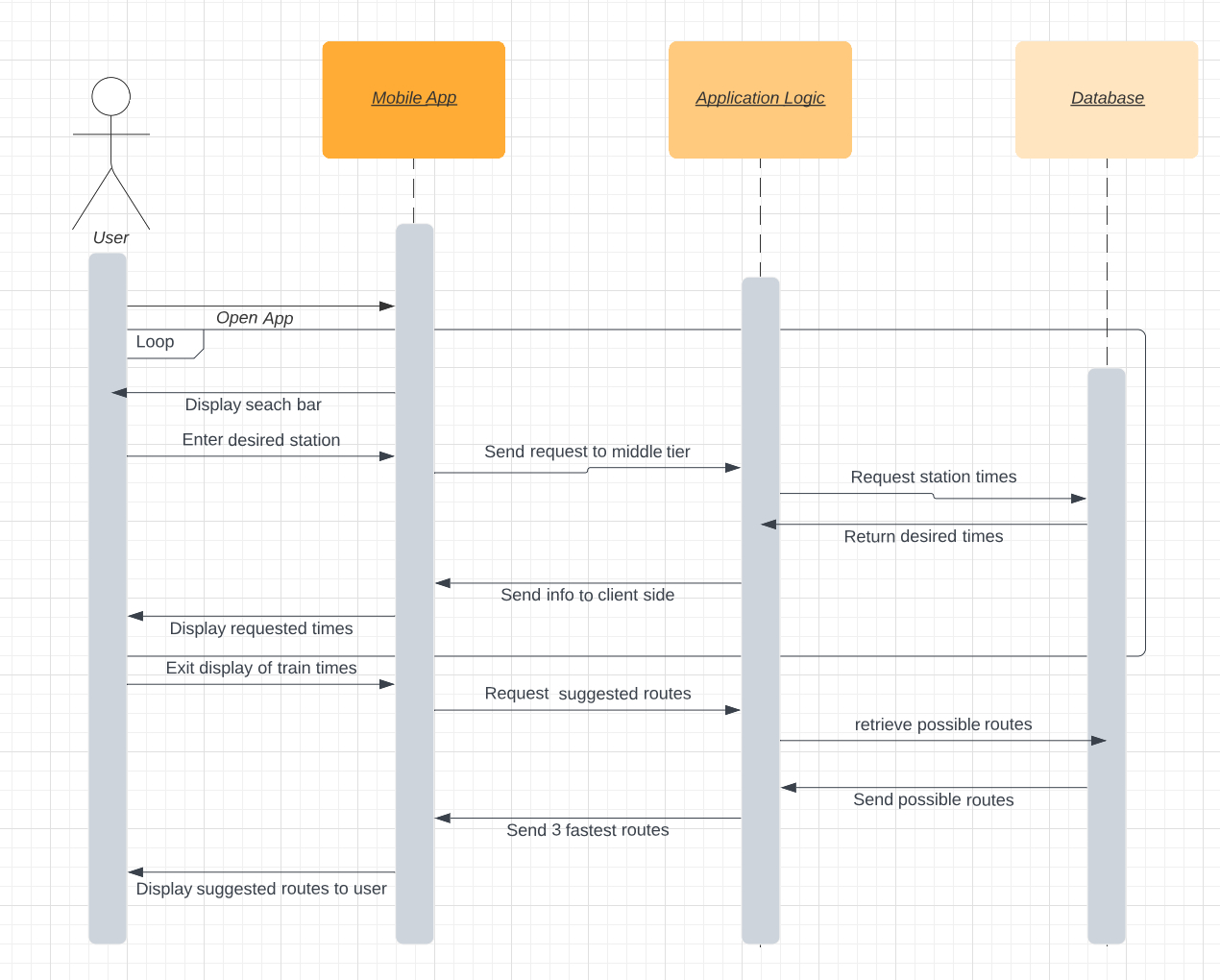
### 2. Functional Design

2.1 Account Services



* First the user will download and open the app
* They will be prompted to create an account or log in
* If creating an account, their information will be stored in the database.
* If logging in, their password will be retrieved from the database to ensure it is correct. If it is incorrect there will be a loop until they get it right.
* The user will also have the option to reset their password which will overwrite the password in the database
* The user will also be able to delete their account. This will wipe their data from the database.

### 2.2 Searching for Train Stations



### To search for train stations the user must log in to their account which routes them to the home page

* At the top of the home page there is a search bar for search for specifying what train/route you want to track
* The constraints of the user's search are sent over to the database where the correct information (Train ID, departure and arrival times, leaving from and destination station and gate, as well as other important information affecting that trains travel) is then relayed and displayed back to the user.

### 3. Structural Design

#### 3.1 Class Diagram of System

