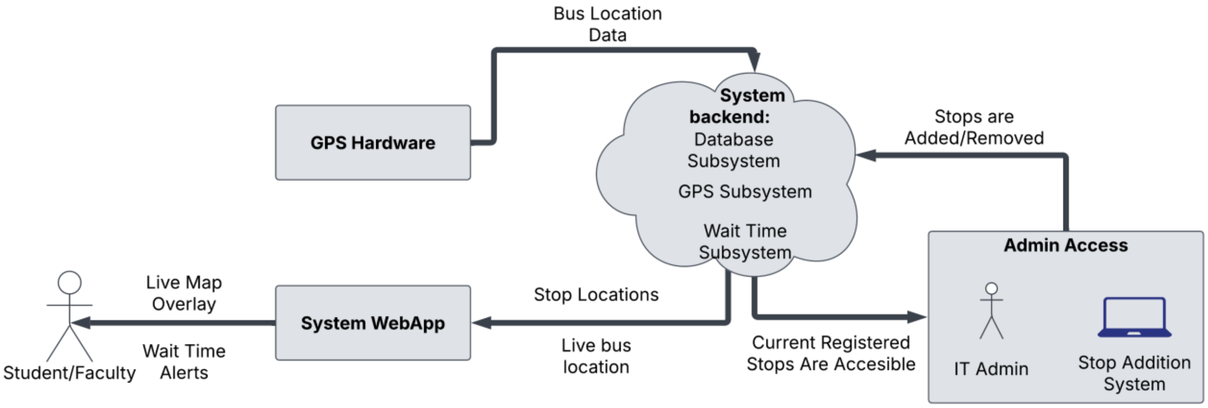
Contributors: Olivia Sharpston, Christian Miller, Nolen Robinson, and Jacobe Johnson

1. Define the system of interest:
For this research, the University of North Alabama's bus system application has at least one subsystem. This subsystem goes into the global positioning system (GPS) for the bus app. The times displayed to users for the estimated arrivals times associated with the GPS via the Internet. These times pulled for the GPS will be updated every minute to ensure up-to-date and realistic wait and arrival times. The application may be written as an app for both iOS and Android operating systems; therefore, the code will be written twice. This app allows users to request from the back end such as being able to find estimated bus arrivals, the bus locations live, and be able to display if any issue occurs with the buses travel, such as an accident or reroute. Users will be able to interact and look up the future wait times and stops for the bus(es). Below is a diagram of our system of interest.

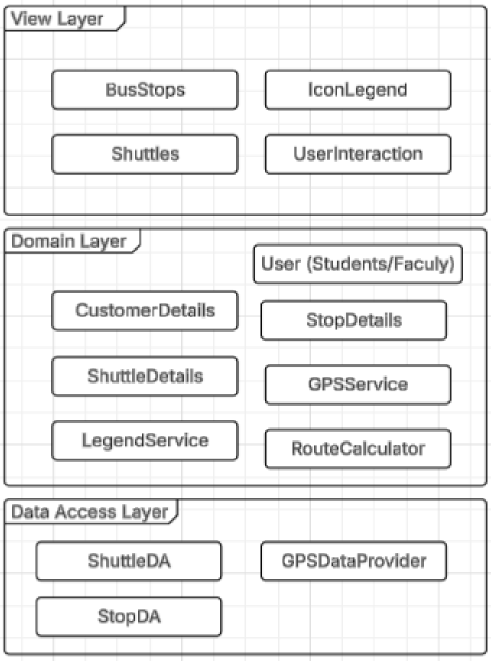


2. Describe the environment:
In this bus system for the University of North Alabama, the relevant stakeholders include students, faculty, the university's IT department, and the bus driver(s). The expectations are different depending on the intended stakeholder. For students and faculty, they need to be able to expect to have a reliable, easy to use, and efficient bus system application to be able to maneuver around, specifically throughout the campus. Secondly, the expectations of the university's IT department should include being able to conduct the maintenance needed for the GPS hardware and the system application. This could include making sure that the global positioning system (GPS) from the Internet is working currently and updating times efficiently. This system would include a form of GPS transmitter hardware, a localized network that is located on the bus(es), and the online-hosted GPS system. Like with most, if not all systems, there are some risks that could arise. This could cause delays, going over budget, or failure to meet the requirements or expectations.

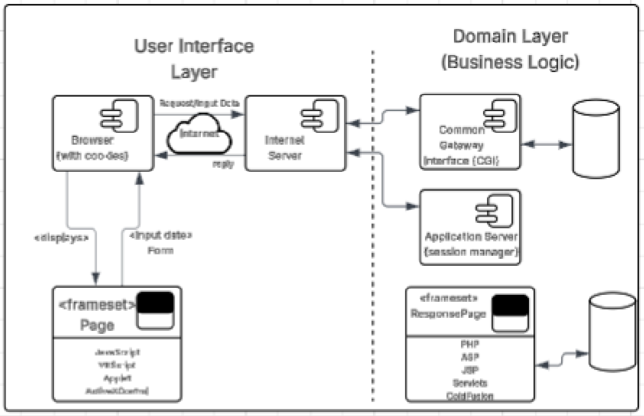
Factors that could cause these would include the online-hosted GPS system going down, there is not enough funds for the application and/or the system, or the bus(es) are not easily accessible for everyone needing to use the bus. Nonetheless, there are several opportunities that can arise from having this bus system in place at the university. Opportunities include being able to transport across the campus easier, allowing new students to know the layout of the campus more efficiently, and possibly reduce the issue of lack of parking, as people could use the bus system to get to campus from a nearby location from the university.

3. Design Application Components

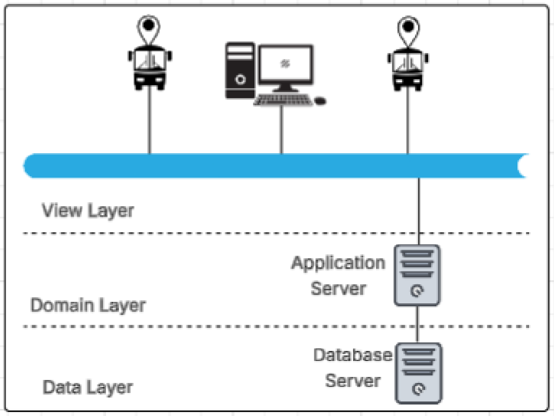
Key components that are included in this system include the users' browsers, such as a smartphone, hosts UI and business logic components, and central databases for the shuttle, users, and data GPS. Our bus system is a system with subsystems, such as the GPS subsystem. There would be some components that could be bought. This includes the GPS data provider, map and route services, an authentication system, and application hosting.



Package Diagram



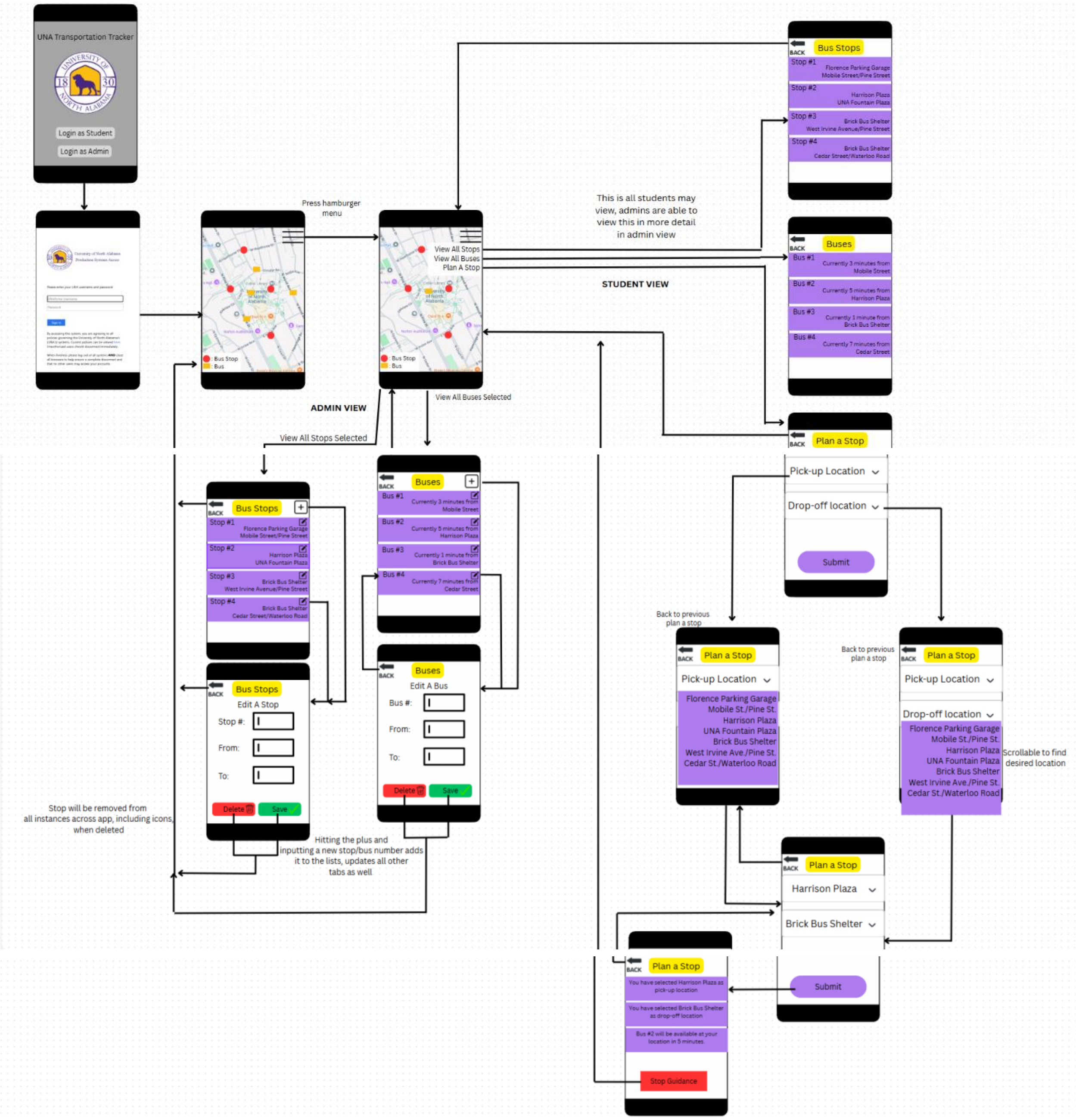
Component Diagram



Deployment Diagram

4. Design the user interface:

User Interface Screens:



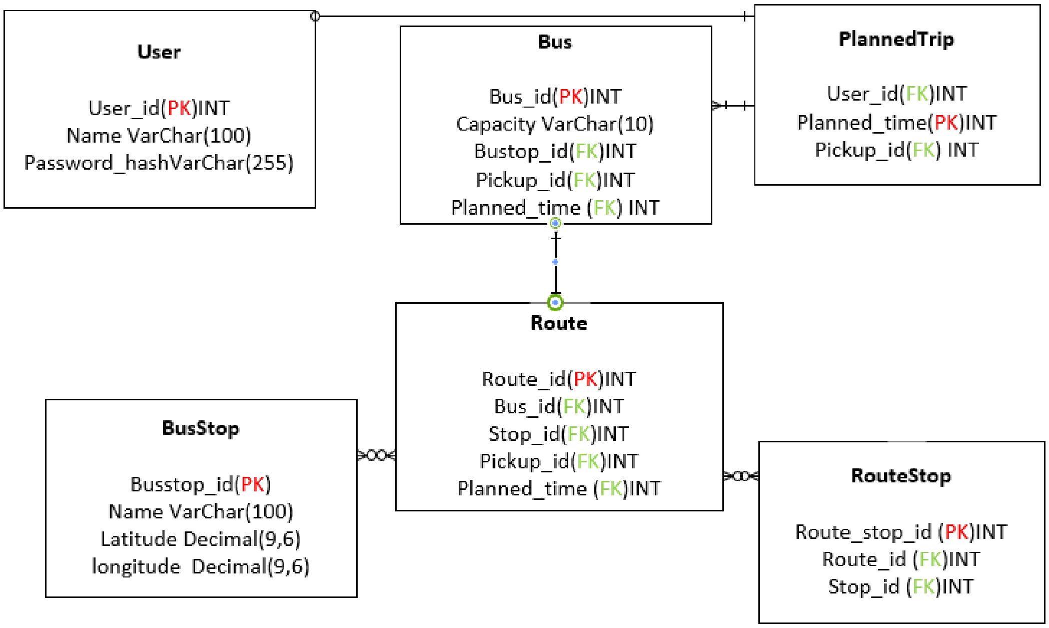
This is all students may view, admins are able to view this in more detail in admin view

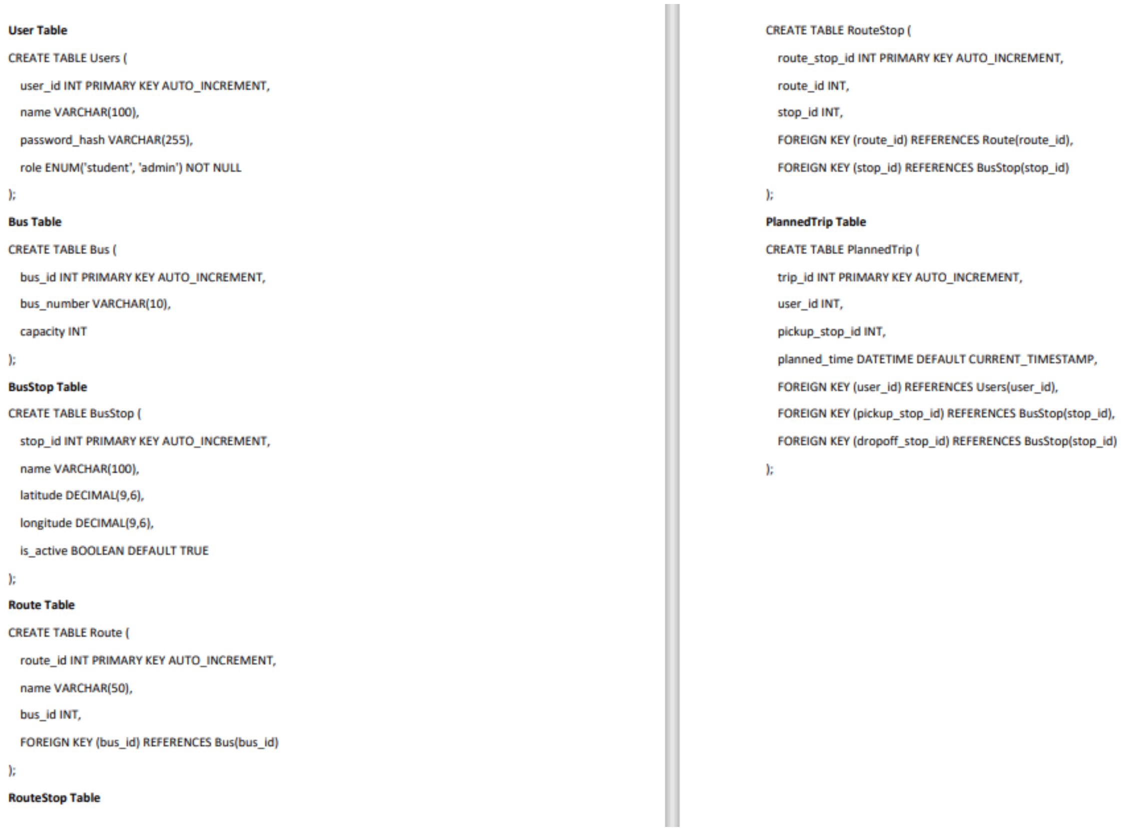
Stop will be removed from all instances across app, including icons, when deleted

Hitting the plus and inputting a new stop/bus number adds it to the lists, updates all other tabs as well

Scrollable to find desired location

5. Designing the database:





User Table

```
CREATE TABLE Users (
   user_id INT PRIMARY KEY AUTO_INCREMENT,
   name VARCHAR(100),
   password_hash VARCHAR(255),
   role ENUM('student', 'admin') NOT NULL
);
```

Bus Table

```
CREATE TABLE Bus (
   bus_id INT PRIMARY KEY AUTO_INCREMENT,
   bus_number VARCHAR(10),
   capacity INT
);
```

BusStop Table

```
CREATE TABLE BusStop (
   stop_id INT PRIMARY KEY AUTO_INCREMENT,
   name VARCHAR(100),
   latitude DECIMAL(9,6),
   longitude DECIMAL(9,6),
   is_active BOOLEAN DEFAULT TRUE
);
```

Route Table

```
CREATE TABLE Route (
   route_id INT PRIMARY KEY AUTO_INCREMENT,
   name VARCHAR(50),
   bus_id INT,
   FOREIGN KEY (bus_id) REFERENCES Bus(bus_id)
);
```

RouteStop Table

```
CREATE TABLE RouteStop (
   route_stop_id INT PRIMARY KEY AUTO_INCREMENT,
   route_id INT,
   stop_id INT,
   FOREIGN KEY (route_id) REFERENCES Route(route_id),
   FOREIGN KEY (stop_id) REFERENCES BusStop(stop_id)
);
```

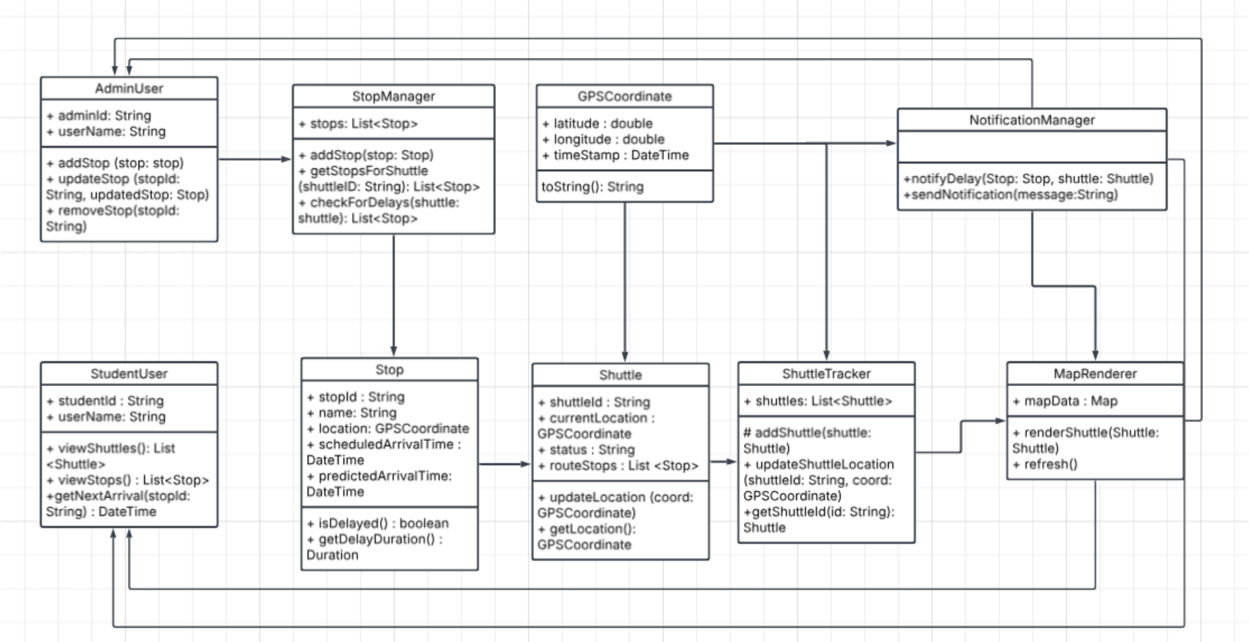
PlannedTrip Table

```
CREATE TABLE PlannedTrip (
   trip_id INT PRIMARY KEY AUTO_INCREMENT,
   user_id INT,
   pickup_stop_id INT,
   planned_time DATETIME DEFAULT CURRENT_TIMESTAMP,
   FOREIGN KEY (user_id) REFERENCES Users(user_id),
   FOREIGN KEY (pickup_stop_id) REFERENCES BusStop(stop_id),
   FOREIGN KEY (dropoff_stop_id) REFERENCES BusStop(stop_id)
);
```

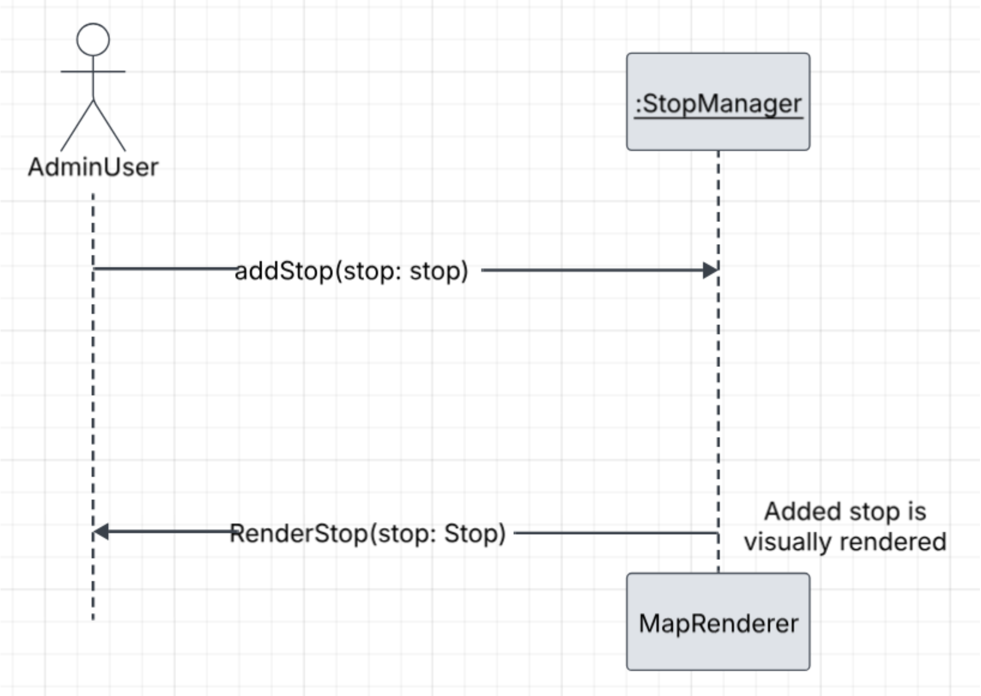
An example of a database structure is shown below:

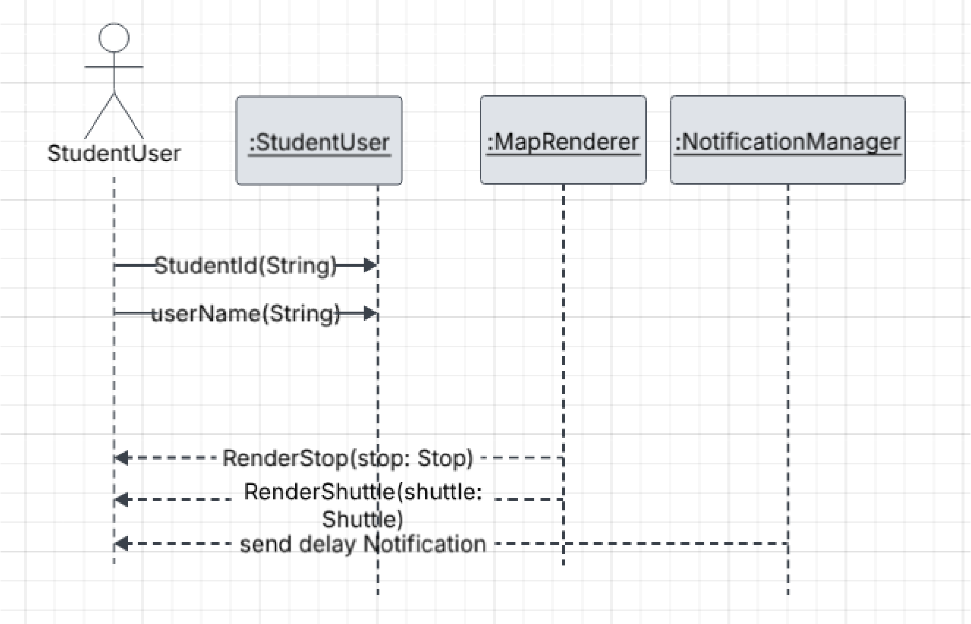
6. Design the software classes and methods:

Software Classes:

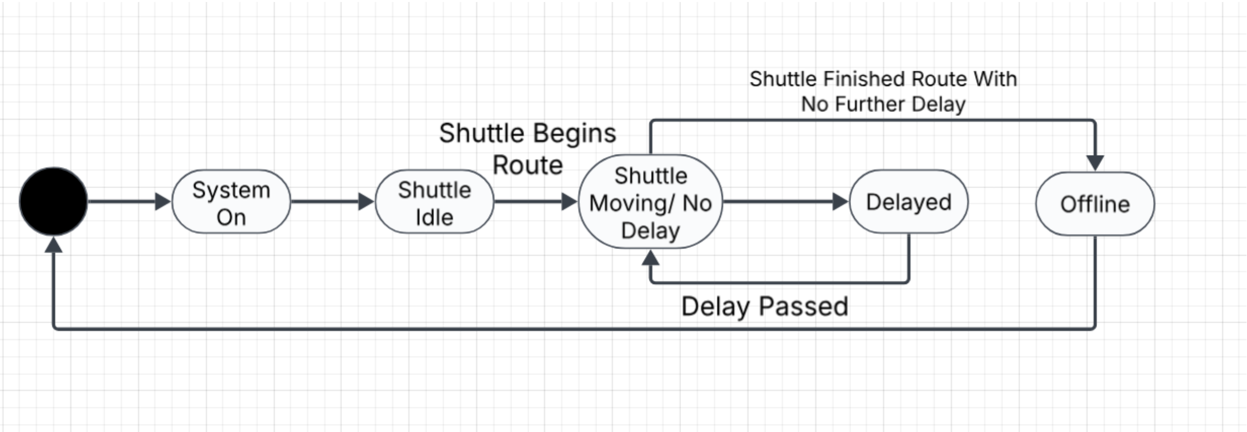

AdminUser
+ adminId: String
+ userName: String
+ addStop (stop: stop)
+ updateStop (stopId: String, updatedStop: Stop)
+ removeStop(stopId: String)
StopManager
+ stops: List<Stop>
+ addStop(stop: Stop)
+ getStopsForShuttle (shuttleID: String): List<Stop>
+ checkForDelays(shuttle: shuttle): List<Stop>
GPSCoordinate
+ latitude : double
+ longitude : double
+ timeStamp : DateTime
toString(): String
NotificationManager
+notifyDelay(Stop: Stop, shuttle: Shuttle)
+sendNotification(message:String)
StudentUser
+ studentId : String
+ userName: String
+ viewShuttles(): List <Shuttle>
+ viewStops() : List<Stop>
+getNextArrival(stopId: String) : DateTime
Stop
+ stopId : String
+ name: String
+ location: GPSCoordinate
+ scheduledArrivalTime : DateTime
+ predictedArrivalTime: DateTime
+ isDelayed() : boolean
+ getDelayDuration() : Duration
Shuttle
+ shuttleId : String
+ currentLocation : GPSCoordinate
+ status : String
+ routeStops : List <Stop>
+ updateLocation (coord: GPSCoordinate)
+ getLocation(): GPSCoordinate
ShuttleTracker
+ shuttles: List<Shuttle>
# addShuttle(shuttle: Shuttle)
+ updateShuttleLocation (shuttleId: String, coord: GPSCoordinate)
+getShuttleId(id: String): Shuttle
MapRenderer
+ mapData : Map
+ renderShuttle(Shuttle: Shuttle)
+ refresh()

System Sequence Diagram:





State Machine Diagram:


Shuttle Finished Route With No Further Delay
Shuttle Begins Route
System On
Shuttle Idle
Shuttle Moving/ No Delay
Delayed
Offline
Delay Passed