Software Requirements Specification
CSC4351 Capstone 1

Team #: 1

Team Name: Panther Transit

Members:

Victoria Robles
Sebastian Sanchez
Hassaan Arshadd
MD Rahman
Kamrul Tarafder

Section: Wednesday

Table of Contents

Introduction / Overview

- The GSU Commute Helper is an application designed to help Georgia State's students commute in a smarter and safer way. The app provides real time updates from multiple sources, including, MARTA transportation system, GSU's transportation system, traffic monitoring services, and weather update systems. To do this, we use multiple APIs, such as MARTA Bus Real-Time RESTful API, Marta Real-Time Rail API, Google Transit Feed Specification - Real Time (GTFS-RT), Google Maps Traffic API, GSU Parking and Bus API, Weather API, etc. The application provides this information through an intuitive, user-friendly interface that allows students to view updates on MARTA delays, nearby traffic conditions, parking availability across GSU lots, and weather changes that could impact their travel.

General Description

- The target user community is students with multiple options for travel to campus. Many of them are budget conscious (often using public transport) and tech-savvy (due to frequent use of technology). Students can use this app to help them access transportation information (such as MARTA times and parking garage space availability). This is also supposed to send notifications based on the user's needs (such as a train arrival alert). This has the benefit of the student knowing important details about their transportation to make the most effective commute plan (whether there are problems or not.). Without knowing such information, it can make commuting more difficult and inflexible, harming their psychological state and academic performance.

Functional Requirements

1. User Registration (Username Validation):
    - **Description:** The app should validate the username to ensure it meets the specific requirements for format and uniqueness.
    - **Input:** Username string: Entered by the user during the registration process.
    - **Source:** User input via the 'username' text box field in the registration page.
    - **Units of Measure:**
        - ➢ Username input: Text field
        - ➢ 'Next' button: Tap to submit
    - **Range of Valid Inputs:**
        - ➢ Email format: Must follow a valid email format.
        - ➢ The username must be unique and not previously registered.
    - **Output:**
        - ➔ Success: Username is accepted, and the user proceeds to the next step of the registration process.

➔ Failure: Error message displayed, indicating that the username is either invalid or already in use.

2. User Registration (Password Validation):
    - **Description:** The app should validate the password to ensure it meets the specific requirements for security.
    - **Input:** Password string: Entered by the user during the registration process.
    - **Source:** User input via the ‘password’ text box field.
    - **Units of Measure:**
        - ➢ Password input: Masked text field.
        - ➢ ‘Sign up’ button: Tap to submit and validate.
    - **Range of Valid Inputs:**
      Password must:
        - ➢ Be at least 9 characters long
        - ➢ Contain at least one uppercase letter (A-Z).
        - ➢ Contain at least one lowercase letter (a-z).
        - ➢ Contain at least one numeric digit (0-9).
        - ➢ Contain at least one special character (e.g., #, $, @, etc.)
    - **Output:**
        - ➔ Success: Password is accepted and stored securely in the database, and the user is able to access the application.
        - ➔ Failure: Error message displayed, indicating which validation rule was not met.

3. Parking Availability Check:
    - **Description:** The app should allow users to check the availability of parking spaces in a specific GSU parking deck. The system will return real time data indicating whether parking spaces are available.
    - **Input:** Parking Deck Selection. The user selects a specific GSU parking deck from a dropdown list.
    - **Source:** User input via the parking deck selection interface.
    - **Units of Measure:**
        - ➢ Parking deck input: Dropdown list.
        - ➢ ‘Check Availability' button: Tap to submit.
    - **Range of Valid Inputs:**
        - ➢ Parking Deck: Must be a valid parking deck in the GSU system (e.g., Parking Deck M, Parking Deck G, etc.)
    - **Output:**
        - ➔ Success: The system will display the number of available parking spaces in the selected deck in real-time.
        - ➔ Failure: If no data is available or the parking deck is full, the appropriate message will be displayed (e.g., “Data not available” or “No parking spaces available).

4. Check MARTA Arrival Times:
    - **Description:** The app should allow users to check real-time arrival and departure times for MARTA buses and trains based on the nearest station or stop.
    - **Input:** Station/Stop ID: Selected automatically based on the user's location or chosen manually from a dropdown list.
    - **Source:** MARTA Bus Real-Time RESTful API and MARTA Real-Time Rail API for station/stop data.
    - **Units of Measure:**
        - ➢ Station/Stop ID: Text field or selected via dropdown.
        - ➢ 'Check Times' button: Tap to submit.
    - **Range of Valid Inputs:**
        - ➢ Station/Stop ID: Must be a valid MARTA station or stop.
    - **Output:**
        - ➔ Success: The app will display the next available arrival and departure times for the selected station or stop.
        - ➔ Failure: If no data is available, an appropriate error message will be displayed (e.g., "Data not available" or "No arrivals scheduled in the next hour").
5. Real-Time Traffic Updates:
    - **Description:** The app should provide users with real-time traffic updates for routes near GSU or commonly traveled routes to campus.
    - **Input:** User's current location (determined via GPS) or manual input of starting and destination locations.
    - **Source:** Google Maps Traffic API for traffic data.
    - **Units of Measure:**
        - ➢ Location data: GPS coordinates or manually entered text fields (starting and destination addresses).
        - ➢ 'Get Traffic Updates' button: Tap to submit.
    - **Range of Valid Inputs:**
        - ➢ Valid GPS coordinates or manually entered addresses.
    - **Output:**
        - ➔ Success: Real-time traffic conditions, including delays, congestion, and accidents, displayed visually on a map or in list format.
        - ➔ Failure: If no traffic data is available, a message will be displayed (e.g., "Traffic data unavailable").
6. Weather Alert Notifications:
    - **Description:** The app should provide weather updates and send notifications if severe weather conditions are detected that may impact the user's commute.
    - **Input:** User's location (automatically determined or manually entered).
    - **Source:** Weather API for real-time weather data.
    - **Units of Measure:**
        - ➢ Location data: GPS coordinates or manually entered address.
        - ➢ 'Get Weather Updates' button: Tap to submit.
    - **Range of Valid Inputs:**

    - ➢ Valid GPS coordinates or manually entered address.
- **Output:**
    - ➔ Success: Weather conditions (e.g., clear, rainy, stormy) and alerts (e.g., severe storm warnings) will be displayed, with notifications sent to the user if necessary.
    - ➔ Failure: If no weather data is available, a message will be displayed (e.g., “Weather data unavailable”).

Interface Requirements

- The main interface requirements that are software will communicate with will be all the external APIs we plan to use these include:
  Google Maps API: the app must communicate with google maps to provide real time traffic data and updates as well as geolocation services and route mapping. The map interface must be embedded within the app and allow users to interact with live traffic data.
  Marta API: the system must communicate with Marta API to fetch real time public transit data including buses and trains including their schedules, delays and route updates. The data pulled via API calls must be displayed within the apps transit section.
  GSU parking API: the system must communicate with GSU parking API to retrieve real time parking availability for all designated panther parking lots across the campus including each deck(M-deck, T-deck, …ect). The communication between our systems must provide data including available spaces, lot capacity and current occupancy.

  All these external APIs must be combined and embedded into our application in order to build a powerful and robust transit app that has all the information commuters would need stored in one complete app.

Performance Requirements

- Three performance requirements for our software system are:
    1. **Real time database updates**: our system must be able to update and retrieve real time traffic, transit, parking and weather information; this update and retrieval from external API’s such as marta and google maps must happen with a delay no longer than a couple seconds (1-10 sec) because this will ensure that users always have the most current and accurate information to make commute decisions
    2. **User interaction low latency**: in order to ensure a responsive and seamless user experience our software must have a maximum response time of around 2-5 seconds for user interactions such as accessing and moving around maps, viewing commute updates and switching between

different screens. This will ensure a smooth user interface when navigating the software system

3. **Concurrent Users scalability**: our software system is intended to be used by the GSU student body and must therefore be able to handle thousands of users all accessing real time commute data and receive notifications all at the same time without any significant degradation in performance among thousands of users. And must maintain the two previous performance requirements to ensure that the system does not degrade with many concurrent users and be able to scale to even ten thousand users or even more.

Design Constraints

1. **Hardware and Software Requirements:**

   The app should run efficiently on standard web browsers and devices that GSU students commonly use, such as laptops, tablets, and smartphones.

2. **Update Frequency:**
   The app must refresh regularly to provide real-time updates but should avoid overloading the server or consuming excessive data, especially on mobile devices.

3. **Reliability:**
   The app should be stable and handle unexpected issues without crashing.

4. **Security:** Protect any user data collected and ensure secure communication with data sources, especially when accessing external APIs.

5. **User Interface Standards:** The app should be user-friendly, following common web design standards to make it easy for students to navigate and understand.

6. **Performance:** The app should load quickly and function smoothly, even with high traffic or slow internet connections.

7. **Scalability**: The app should be designed to handle an increasing number of users as more GSU students start using it, without compromising performance or functionality.

Non-Functional Attributes

- **Constant Secure API Connection:** The system uses real-time data from APIs that aggregate data in real time. Connection must always be active and stream data in real time through API, and if API is down there must be a fallback. Also, the API should be secured with protection from attacks like DDOS. A user should not be able exploit the API into our system so there should be a layer of protection there.
- **Menu switching:** Switching between menus of different public transit systems or parking garage information must be quick, and not have users staring at a blank screen. Potential fallback is a skeleton loading system for slow connections.
- **Peak Commute Time Load Handling:** Peak commute times are usually between 7 AM - 9 AM and 4 PM - 7 PM, so it's crucial to ensure the system maintains reliability with sufficient resources to handle an increased server load compared to other times. There could be a time specific load balancer and auto scaling group involved here.
- **Cached Data:** The system should cache recent data as a fallback option in case of API failure or temporary user network issues. This cache ensures users are still able to view recent data in network dead zones and the app isn't completely useless at this time.
- **Responsive Design + Portability:** Different devices have different sizes so the application should accommodate for this with responsive design to ensure no important information is being covered up due to device size differences. The system should also be portable across different browsers that may have slightly different standards like Chrome, Safari, Firefox, etc.
- **System GPS:** Application must have constant access to user's system GPS in order to deliver accurate commute information.

Appendices

- **Acronyms and Abbreviations**
  - ➢ API: Application Programming Interface.
  - ➢ GTFS-RT: Google Transit Feed Specification - Real Time.
  - ➢ GSU: Georgia State University.
  - ➢ MARTA: Metropolitan Atlanta Rapid Transit Authority.
  - ➢ GPS: Global Positioning System
  - ➢ DDOS: Distributed Denial of Service
- **References**
  - ➢ MARTA Bus Real-Time RESTful API: Provides real-time bus location and schedule data.
  - ➢ MARTA Real-Time Rail API: Delivers real-time rail schedule updates.
  - ➢ Google Transit Feed Specification - Real Time (GTFS-RT): Used for accessing transit data like route information and real time updates.
- **Definition of Terms**
  - ➢ API: A set of methods that allow applications to access data from external sources.

- Load Balancer: A device that sits between the user and the server group and acts as an invisible facilitator, ensuring that all resource servers are used equally i.e. ensures a server isn't overwhelmed and crashes
- Auto Scaling Group: Service that automatically spins up new servers to meet demand as system load increases beyond certain thresholds
- Cache: High speed data storage layer