CSc4351 CS Capstone 1
Software Design Document
Fall 2024

Panther Transit

GSU Commute Helper

Team Members:
1. Victoria Robles
2. Kamrul Tarafder
3. MD Rahman
4. Hassaan Arshad
5. Sebastian Sanchez

Date: (12/12/2024)

GSU Commute Helper

Table of Contents

1. Introduction

1.1 Purpose

The purpose of the GSU Commute Helper application is to facilitate and enhance the commuting experience for Georgia State's students by serving as a real-time transportation data platform. Its primary goal is to help students make safer and smarter commuting decisions by providing timely updates on traffic conditions, MARTA services, campus parking availability, and weather impacts. By consolidating this information into one accessible app, the platform aims to save students' time and reduce transportation-related stress.

1.2 Scope

The GSU Commute Helper application will integrate data from multiple APIs to provide real-time updates on traffic conditions, MARTA bus and rail services, weather changes, and GSU parking availability. Key APIs include the MARTA Bus Real-Time API, Google Maps Traffic API, and Weather API. The application will feature user authentication, traffic incident reporting, weather-related delay updates, and push notifications for transportation alerts. Designed with a user-friendly interface tailored to Georgia State students, the app focuses on enhancing their commuting experience. However, features like MARTA card purchasing or reloading are outside the scope of this project.

1.3 Definitions, Acronyms, and Abbreviations

- **UI**: User Interface
- **API**: Application Programming Interface
- **SDD**: Software Design Document
- **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
- **OS**: Operating System

1.4 References

- IEEE Std 830-1998, IEEE Recommended Practice for Software Requirements Specifications
- MARTA Bus Real-Time RESTful API Documentation
- MARTA Real-Time Rail API Documentation
- Google Maps Platform Documentation

- GSU Parking and Transportation Services Website
- GTFS-RT Feed Specification Documentation

1.5 Overview

The GSU Commute Helper is a mobile application designed to help Georgia State's students commute more efficiently and safely. It provides real-time updates from multiple sources, including, MARTA transportation system, GSU's transportation system, traffic monitoring services, and weather update systems. Through an intuitive, user-friendly interface, students can access updates on MARTA delays, nearby traffic conditions, parking availability across GSU lots, and weather changes that could impact their travel. The app focuses on enhancing safety and convenience for GSU commuters with personalized notifications and timely updates.

2. System Overview

2.1 System Architecture

The Panther Transit mobile application will be developed using a three tier architecture:

- **Presentation Layer**: The mobile user interface enables users (students, faculty, and administrators) to access real-time traffic, transit, and parking information. This layer will include interactive maps, commute updates, and notification panels for delays or alerts
- **Business Logic Layer**: The core functionality of the system. This layer will handle tasks such as processing API requests, managing user preferences, and integrating external services like MARTA, Google Maps, and the GSU Parking API.
- **Data Access Layer**: The backend responsible for storing and retrieving user data, commute preferences, and system logs. This layer will also manage real-time database syncing to ensure the app displays up-to-date information.

2.2 System Context

The Panther Transit app will operate in a mobile environment, accessible on Android and iOS devices. It will interact with external APIs, such as MARTA, Google Maps, and the GSU Parking API, to gather and display relevant commute information. The following diagram illustrates the system context:

```
+-----------------------+
|   Mobile App UI       |
|      (Front End)      |
+-----------------------+
          |
          v
+-----------------------+
|   Presentation Layer   |
| (Maps, Alerts, Stats)  |
+-----------------------+
          |
          v
+-----------------------+
| Business Logic Layer  |
|        (APIs)          |
+-----------------------+
          |
          v
```

```
+-----------------------+
|   Data Access Layer    |
|     (backend)         |
+-----------------------+
          |
          v
+-----------------------+
|  External APIs         |
| (MARTA, Google Maps,   |
|  GSU Parking API)      |
+-----------------------+
```

3. Functional Requirements

3.1 Use Case Diagram

The following use case diagram illustrates the interactions between users and the GSU Commute Helper App:

```
+--------------------+
|     Users          |
+--------------------+
| - Register Account |
| - Check Parking    |
| - View Traffic     |
| - Check MARTA Times|
| - Get Weather Alerts|
+--------------------+
         |
         v
+--------------------+
|    System API      |
+--------------------+
| - Validate Login   |
| - Fetch Parking    |
| - Fetch Traffic    |
| - Fetch MARTA Data |
| - Fetch Weather    |
+--------------------+
         |
         v
+--------------------+
| External Services  |
+--------------------+
| - MARTA API        |
| - GSU Parking API  |
| - Google Maps API  |
| - Weather API      |
+--------------------+
```

3.2 Use Cases

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”).

4. Non-Functional Requirements

4.1 Performance Requirements

- 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)
- 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

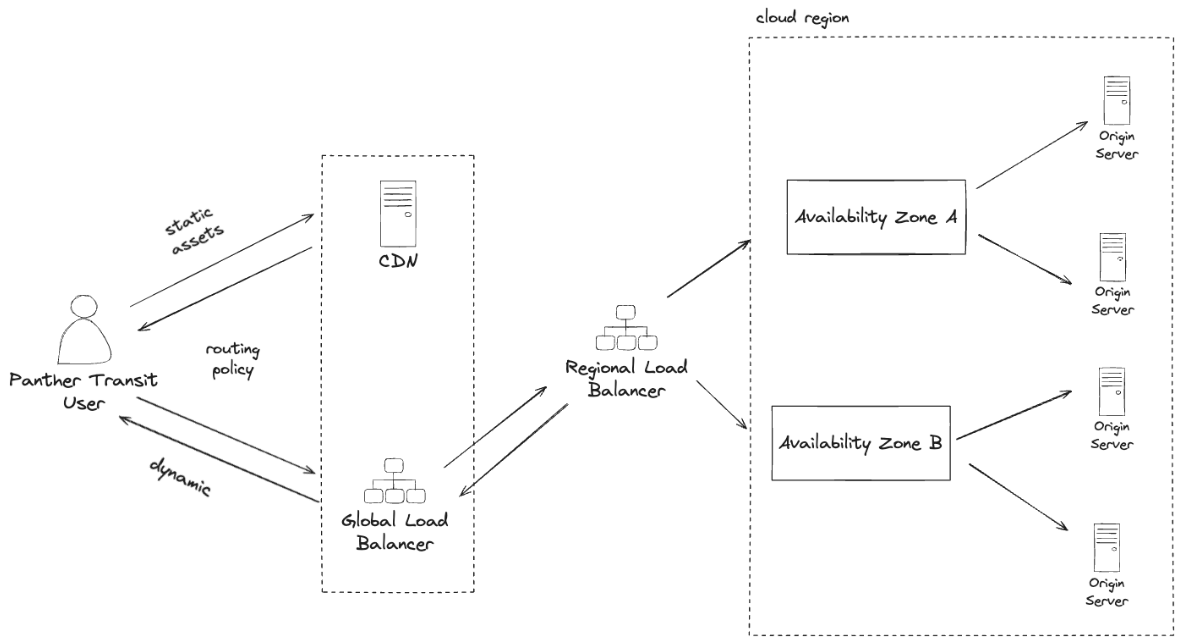
4.2 Security Requirements

- Protect any user data collected by requiring user authentication with secure password storage utilizing hashing algorithms
- Ensure secure communication with data sources, especially when accessing external APIs.

4.3 Usability Requirements

- The app should be user-friendly, following common web design standards to make it easy for students to navigate and understand.
- Switching between menus of different public transit systems or parking garage information must be quick, and not have users staring at a blank screen.

5. System Design



5.1 High-Level Design

The high-level design of the SMS includes the following components:

- **User Interface**: The plan is to develop a mobile application, so for the UI we plan to use a cross platform mobile application framework called React Native. This allows us to build the mobile app using HTML, CSS, and Javascript or Typescript.
- **Business Logic**: The business logic will be implemented using server-side languages like Node.js and Python. We plan to use frameworks built on top of Node.js in Express and on top of Python in Flask to allow for easier development. Node.js can handle real-time API interactions and also provide server side rendering for application content and components. The Flask server can provide support for data and analytics for the application.
- **Database**: The database will be a mix of a relational database and NoSQL data. The relational database will be a PostgreSQL database that stores structured user information on AWS RDS or similar cloud service. The NoSQL database will be a MongoDB database to capture data like user statistics and also logs.

5.2 Detailed Design

- **User Interface Components**:
  - Sign Up Page & Login Page
  - Parking Deck Selection Page
  - Maps Page with Traffic Data

    - Weather Page
    - Marta Train & Bus Routes Page/Components
    - Notification Management Page
    - Bookmarked Transit Route Page
    - Menu Component to Switch Transit Routes
- **Business Logic Components**:
    - User Authentication Module
    - MARTA API Module
    - Notification Sending Module
    - Weather API Module
    - Parking Deck Sensor & Spot Communication Module
    - GSU Traffic Data API

5.3 Database Design

The database schema will include the following tables:

- **Users**: Stores user information (user_id, username, email, password_hash, preferences, created_at).
- **Notifications**: Stores user notification settings (notification_id, user_id, notification_type, enabled, created_at).
- **Parking**: Tracks GSU parking deck data (deck_id, deck_name, total_spaces, available_spaces, last_updated).
- **TransitLogs**: Logs transit details for MARTA (log_id, user_id, transit_type, station_name, arrival_time, delay_minutes, created_at).
- **WeatherAlerts**: Tracks weather alerts (alert_id, user_id, alert_message, severity, alert_time, created_at).
- **BookmarkedRoutes**: Stores user bookmarked transit routes (bookmark_id, user_id, route_name, route_details, created_at).

6. Implementation Plan

6.1 Development Environment

- **IDE**: Visual Studio Code
- **Version Control**: Git and GitHub for source code management and version control
- **Database Management**: AWS RDS & DBeaver to view and update database records

6.2 Tools and Technologies

- Frontend: React Native
- Backend: Node.js with Express and Python with Flask or FastAPI
- Database: PostgreSQL & MongoDB

6.3 Coding Standards

- Follow the PEP 8 style guide for Python & W3 standards for coding conventions
- Use meaningful variable and function names with consistent camel casing across variables, components, and files
- Write comments and documentation for all major functions, classes, and components
- Aim for readability and simplicity over complexity within coding functions
- Enforce linting rules to ensure code quality and consistency in codebase through tools like ESLint

7. Testing Plan

7.1 Testing Strategy

- **Unit Testing**: Verify the functionality of individual components, such as the login module and API integrations, ensuring they perform as expected in isolation.
- **Integration Testing**: Test interactions between modules (e.g., MARTA API integration with the user interface) to confirm seamless data flow and proper communication.
- **User Acceptance Testing (UAT)**: Conduct hands-on testing with GSU students to validate that the app meets real-world usability needs and aligns with user expectations.

7.2 Test Cases

1. **Test Case for User Login**
   Objective: Verify that users can log into the application securely and seamlessly.

   Steps:

   Open the GSU Commuter Helper application.

   Navigate to the login page.

   Enter a valid GSU email and password.

   Click on the "Login" button.

   Observe the result.

   Expected Output:

   User is successfully redirected to the home screen of the app.

   If incorrect credentials are entered, an error message is displayed.

2. **Test Case 2: Marta Arrival Display Times**
   Objective:
   Ensure that the app correctly fetches and displays MARTA train or bus arrival times.

   Steps:

Open the GSU Commuter Helper application.

Navigate to the "MARTA Arrival Times" section.

Enter a station or stop name.

Wait for the app to fetch real-time arrival data.

Expected Output:

Real-time arrival times for MARTA services are displayed accurately.

Data refreshes periodically or on user request.

3. **Test Case 3: Transit Notification Alerts**
Objective:
Ensure users are getting the right alerts that they subscribed to for different transit options and lines

Steps:

Open the GSU Commuter Helper application.

Navigate to the "Notifications" section.

Toggle notifications on for a specific line or station during a specified time

Look for results.

Expected Output:

During the specified time user sees alert for transit line or station

User only sees the alert they subscribed to and not any alerts they didn’t consent or pick

8. Deployment Plan

8.1 Deployment Strategy

- The application will be deployed on AWS.
- Use a CI/CD pipeline for automated testing and deployment with tools like TravisCI and Docker
- Verify all APIs (MARTA, GSU Bus System, Parking, etc.) are integrated and functioning correctly.
- Conduct a staged rollout—initially deploy to a small user group for beta testing.

8.2 User Training

- Include an interactive walkthrough for new users upon first launch.
- Provide tooltips and explanations for key features like MARTA tracking, parking updates, and traffic information.
- Create a help section within the app with FAQs, video tutorials, and troubleshooting guides.
- Include a contact form for reporting issues or requesting assistance.

9. Future Enhancements

- **User Customization**: Enable users to set their preferred transportation modes, routes, and notification types based on their commute habits.
- **Language Support**: Add support for additional languages to cater to a diverse student population.
- **UI Customization**: Allow users to personalize the app's appearance by changing background colors and the app icon.
- **MARTA Card Purchasing**: Redirect users to the official MARTA Breezecard website for card purchasing.
- **Multiple OS Support**: Develop and release app versions compatible with tablet/iPad operating systems.
- **Parking Availability Prediction**: Implement machine learning algorithms to predict parking lot fill rates based on historical data and trends.
- **Feedback System**: Introduce a feature that allows users to submit feedback and suggestions on app features and performance.

10. Appendices

- Appendix A: Glossary of Terms
    - API (Application Programming Interface): A set of protocols for building and interacting with software applications.
    - GTFS-RT (Google Transit Feed Specification - Real Time): A standard format for exchanging real-time public transportation information.
    - MARTA (Metropolitan Atlanta Rapid Transit Authority): Atlanta's public transit system.
    - GPS (Global Positioning System): A satellite-based navigation system used to determine location.
    - DDOS (Distributed Denial of Service): A type of cyber-attack that disrupts network resources.
    - OS (Operating System): Software that manages hardware and software resources on a device.
- Appendix B: User Interface Mockups
    - Login Page: Displays input fields for email and password with a "Forgot Password" link.
    - Home Page: Features buttons to navigate to Parking, Traffic, MARTA, and Weather sections.
    - Maps Page: An interactive map showing real-time traffic updates and transit routes.
    - Notification Settings: A toggle-based interface to enable or disable specific alerts.
    - Parking Availability Page: Lists available spaces for GSU parking decks.
    - MARTA Page: Displays next arrival times and delays for selected stations or stops.
- Appendix C: Database Schema Diagrams
    - Users Table
        - user_id, username, email, password_hash, preferences, created_at
    - Notifications Table
        - notification_id, user_id, notification_type, enabled, created_at
    - Parking Table
        - deck_id, deck_name, total_spaces, available_spaces, last_updated
    - TransitLogs Table
        - log_id, user_id, transit_type, station_name, arrival_time, delay_minutes, created_at
    - WeatherAlerts Table
        - alert_id, user_id, alert_message, severity, alert_time, created_at
    - BookmarkedRoutes Table

            - bookmark_id, user_id, route_name, route_details, created_at

---

This Software Design Document serves as a comprehensive guide for the development of Panther Transit, the GSU Commute Helper application, detailing every aspect from requirements to deployment and maintenance. It is intended to ensure that all stakeholders are aligned and that the development team has a clear roadmap for building the system.