Accessible Interactive Map

SOFTWARE Design Documentation (SDD)

CS 4850 Section 3

Spring 2025

Professor Perry

1/26/2025

Justin Connick

Megan Ingram

Derrick Novack

Spencer Williams

Emily Zhu

Table of Contents

1. Introduction and Overview

1.1. Document Outline

This document provides a comprehensive design framework for the Accessible Interactive Map project, which aims to create a web-based application that enables users to select starting and ending points on an interactive map, view optimized routes, and access

lifestyle-related information. The primary goal is to ensure that the application is accessible to individuals with varying disabilities and adaptable for use in different institutions, beginning with Kennesaw State University (KSU) with hopes to expand to other areas.

1.2. Document Description

1.2.1. Introduction

The Accessible Interactive Map project addresses the need for an inclusive navigation tool designed to accommodate diverse user needs, particularly those with disabilities. This Software Design Document (SDD) outlines the design components, technical specifications, and functional requirements necessary to implement and maintain the system. The intended audience includes software developers, quality assurance teams, project managers, and stakeholders invested in promoting accessibility and enhancing user experience.

1.2.2. System Overview

The Accessible Interactive Map project is designed as a responsive, web-based platform powered by modern web technologies and frameworks such as Flask. The system will provide users with:

1) Interactive Routing: Users can select start and end points on an accessible map and view routes optimized based on user preference and physical accessibility.
2) Lifestyle Recommendations: Features such as calorie count for different routes will support fitness and weight management goals.
3) User Personalization: A user management system will allow storing preferences like types of disabilities and lifestyle goals to tailor the application experience.
4) Device Compatibility: The platform will function seamlessly across desktops, tablets, and mobile devices.

The document does not cover elements such as project scheduling, cost estimates, or testing procedures, as these are addressed in separate project management and quality assurance plans. Prototypes, user guides, and user interface storyboards will serve as supplemental reference and are documented separately.

1.3. Intended Audience and Reading Suggestions

This document is tailored for the following groups:

- Developers: To guide the implementation of the system's technical architecture.
- Testers: To understand the functional requirements and test scenarios.
- Project Managers: To ensure alignment between system design and project goals.

- Stakeholders and Clients: To validate that the design meets their requirements and expectations.

Readers are advised to begin with the system overview to understand the project's objectives and proceed to detailed design components relevant to their roles. Sections on architecture and design specifications are particularly useful for technical teams.

1.4. Product Scope

The Accessible Interactive Map project's main objective is to create an inclusive, user-friendly navigation tool. The project aligns with institutional goals to foster accessibility and support lifestyle improvement. By offering adaptable configurations, the project will pave the way for broader application practices in different settings, promoting universal design principles.

1.5. References

- Accessible Interactive Map Project Proposal (2025-01-20)
- Flask Web Framework Documentation

2. Design Considerations

2.1. Assumptions and Dependencies

Our project relies on the following assumptions and dependencies:

1) Related Software or Hardware:
    a) The platform assumes integration with web browsers supporting modern HTML5, CSS3, and JavaScript Standards, with React as the primary front-end framework.
    b) Compatibility with assistive technologies such as screen readers and voice control systems.
2) Operating Systems:
    a) The application is designed to be platform-agnostic, accessible via standard web browsers on both desktop and mobile devices (i.e. Microsoft Edge, Chrome, Firefox, Safari)
3) End-User Characteristics:
    a) Users will have varying degrees of technical proficiency, including individuals with physical, visual, or cognitive disabilities.
4) Possible and Probable Changes in Functionality:
    a) The system should be modular to accommodate future enhancements such as additional accessibility features or integration with third-party APIs.

2.2. General Constraints

The design of the Accessible Map Project is subject to the following constraints:

1. **Hardware or Software Environment:**
    a. Limited to the processing and memory capacities of standard consumer devices.
2. **End-User Environment:**
    a. Must perform efficiently in both high-speed and low-speed network conditions.
3. **Availability or Volatility of Resources:**
    a. Cloud-based hosting may incur downtime due to maintenance or unexpected outages.
4. **Standards Compliance:**
    a. Adheres to WCAG 2.1 guidelines for web accessibility.
5. **Interoperability Requirements:**
    a. Ensures compatibility with GIS data formats and external APIs for map services.
6. **Interface/Protocol Requirements:**
    a. Utilizes HTTPS for secure communications and JSON for data exchange.
7. **Data Repository and Distribution Requirements:**
    a. Centralized cloud-based database for storing user preferences and map data.
8. **Security Requirements:**
    a. Implements role-based access control and data encryption to protect sensitive information.
9. **Memory and Capacity Limitations:**
    a. Optimized to function within the memory and storage limits of typical consumer devices.
10. **Performance Requirements:**
    a. Routes must be calculated and displayed within two seconds of a user request.
11. **Network Communications:**
    a. Designed to handle intermittent connectivity gracefully by caching critical data.
12. **Verification and Validation Requirements:**
    a. Comprehensive unit testing and user acceptance testing are required to ensure functionality and usability.
13. **Other Requirements:**
    a. Documentation and training materials must be accessible and user-friendly.

2.3. Development Methods

The development of the Accessible Map Project follows an iterative and agile methodology to ensure continuous feedback and adaptability. Key components include:

1. **Agile Framework:**
    a. Features are prioritized in sprints, allowing incremental delivery of functional components.
2. **User-Centered Design:**
    a. Involves regular usability testing with individuals from diverse backgrounds to refine accessibility features.
3. **Component-Based Architecture:**
    a. The system is built using reusable components to facilitate maintenance and scalability.
4. **Code Standards and Best Practices:**
    a. Adherence to React development standards and other industry coding practices ensures readability and maintainability.
5. **Automated Testing:**
    a. Continuous integration and deployment pipelines include automated testing to detect and resolve issues early.
6. **Toolset:**
    a. The project uses Flask for backend development, Bootstrap for responsive design, and SQLAlchemy for database management.

By employing these methodologies, the Accessible Map Project aims to deliver a robust, accessible, and user-friendly platform that meets and exceeds stakeholder expectations.

3. Architectural Strategies

The architectural strategies for the project are designed to ensure scalability, maintainability, and user-centric functionality, while addressing key technological and design considerations.

3.1. Use of technologies

The project leverages React for the front-end due to its component-based structure and extensive community support. While Flask was initially considered for the backend, it may be replaced with another framework better suited for JavaScript-centric development. For data persistence, a cloud-based relational database like PostgreSQL has been selected, offering scalability and robust query capabilities. Additionally, the system integrates with GIS APIs for map rendering and route calculation, along with accessibility-focused libraries to ensure compatibility with assistive technologies.

3.2. Reuse of Software components

To accelerate development and maintain consistent language design, existing open-source components such as React UI libraries (e.g., Material-UI or Ant Design) will be utilized. On the backend, reusable modules for authentication and data validation will provide modularity and facilitate future extensibility.

3.3. Future Extension Plans

The system is designed with modularity in mind, enabling seamless integration with additional APIs, such as elevation tracking, or weather or traffic services, to enhance route recommendations. Future enhancements include the implementation of multi-language support and advanced personalization features, which can be easily incorporated due to the flexible architecture.

3.4. User Interface Paradigm

The application employs a responsive and accessible UI paradigm, ensuring usability across a wide range of devices and compliance with WCAG 2.1 standards. Input is designed to support multiple modalities, including keyboard navigation, touch gestures, and voice commands, to enhance accessibility.

3.5. Error Detection and Recovery

Real-time input validation is implemented to handle user actions, supported by robust error logging mechanisms to identify and resolve issues promptly. The system also incorporates graceful degradation strategies to ensure continued functionality in the event of non-critical failures.

3.6. Data and Persistence Management

User preferences will be saved using a combination of caching, cookies, and account creation. Temporary preferences or session-based data will be managed through caching or browser cookies for quick access. For persistent storage, user account information, preferences, and route data will be securely stored in cloud-hosted databases with nightly backups to ensure data integrity. Performance optimization will be achieved through caching mechanisms, such as Redis, to improve access times for frequently used data.

3.7. Communication and Networking

Client-server communication will rely on a suitable API framework, either RESTful APIs or GraphQL, depending on the project's evolving requirements. RESTful APIs are currently a strong candidate due to their simplicity and wide adoption, while GraphQL may be considered later for its query optimization capabilities. Regardless of the choice, secure HTTPS protocols will ensure data integrity and protection against interception.

3.8. Concurrent and Synchronization

The project employs web workers or service workers to handle concurrent processing tasks, such as route calculations, without blocking the main thread. Synchronization mechanisms are implemented to ensure consistent updates to user preferences across devices.

By integrating these strategies, the project will achieve a robust, scalable, and user-focused system architecture that aligns with the project's goals and requirements.

4. System Architecture

The system is designed as a web-based application that will allow users to select starting and ending points on an interactive map with the primary goal of displaying routes based on accessibility requirements. This architecture is a client server architecture where our backend will be in Flask and our frontend will be React. We will also be using PostgreSQL as our database in which the backend will communicate with to store and retrieve user information and route data. We will also be using the libraries Google Map API, NetworkX, and OSRM which will contribute to the processing of the routes.

These are the components of our application:

1. **Backend (Flask)**

   This is where much of the processing will be done in our application. It will utilize the libraries Google Map API, NetworkX, and OSRM to process routes. The backend will also receive requests from the frontend and send the required data back for the frontend to display the information. Finally, the backend will also communicate with PostgreSQL in order to store information regarding user information and routes.

2. **Frontend (React)**
   The frontend will be responsible for displaying information for the user to select and navigate. It will also send requests to the backend in order to get information to display.

3. **Database (PostgreSQL)**
   The database will store any user information and route data. This includes accessibility related data.

4. **Frameworks**
   We will be utilizing Google Maps API, NetworkX, and OSRM in a hybrid combination in order to give the best data when it comes to calculating and displaying the routes.

4.1. System Architecture Diagram

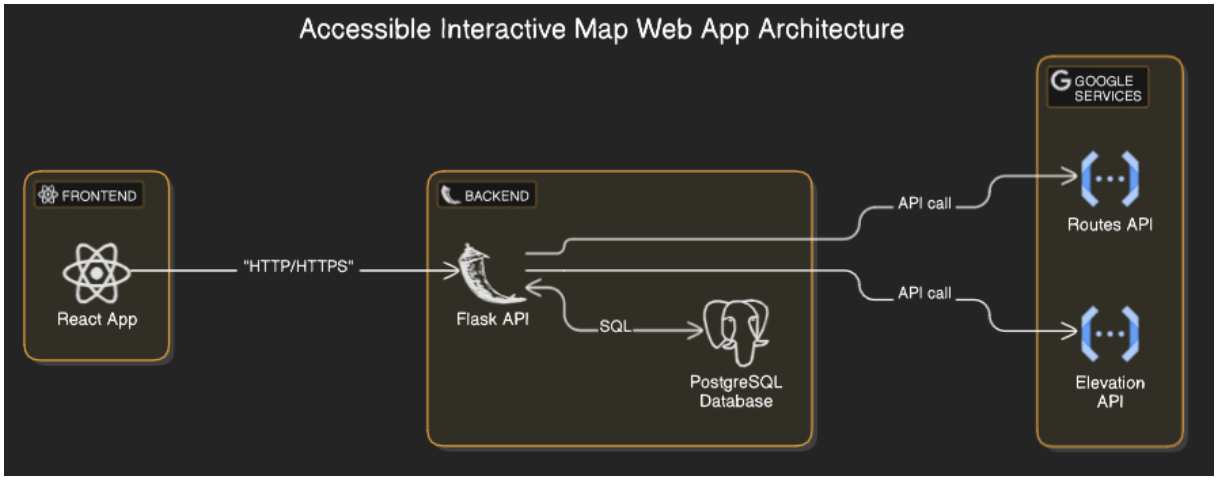

Figure 1:

5. Detailed System Design

Most components described in the System Architecture section will require a more detailed discussion. Other lower-level components and subcomponents may need to be described as well. Each subsection of this section will refer to or contain a detailed description of a system software component. The discussion provided should cover the following software component attributes:

5.1. Classification

The kind of component, such as a subsystem, module, class, package, function, file, etc. ....

- **Modules**
  - Frontend (React, JavaScript, HTML, CSS)
    - Components – UI elements (i.e Search bar, Tokens, Map, Buttons, Forms, Dashboard)
    - API Services – Functions handling HTTP requests to backend
  - Backend (Flask)
    - Routes – API handling route related requests from the user
    - Services – Functions processing requests and interacting with the database
  - Database (PostgreSQL)
    - Tables – Entities like Users, Routes, Feedback, Preferences, Map data
    - Relationships – Foreign Key relationships and constrains
- **Frameworks/API**
  - Google Maps API
  - OSRM
  - NetworkX

5.2. Definition

The specific purpose and semantic meaning of the component. This may need to refer to the requirements specification.

- **Frontend**
  - Display an interactive map with accessibility options along with a user-friendly interface.
  - Send requests to the backend module for route, calorie calculations, and user information
  - Utilize Google Maps API to render the map to the user
- **Backend**
  - Handle communication between the frontend and the database.
  - Will contain the logic to calculate the routes that is requested from the user
  - Process the calorie calculations of a given route
  - Communicate with the frontend to retrieve and store data related to the user and routes.
- **Database**
  - Stores user information and calculated routes.
  - Will also contain assessable information regarding the routes.

5.3. Process Flow

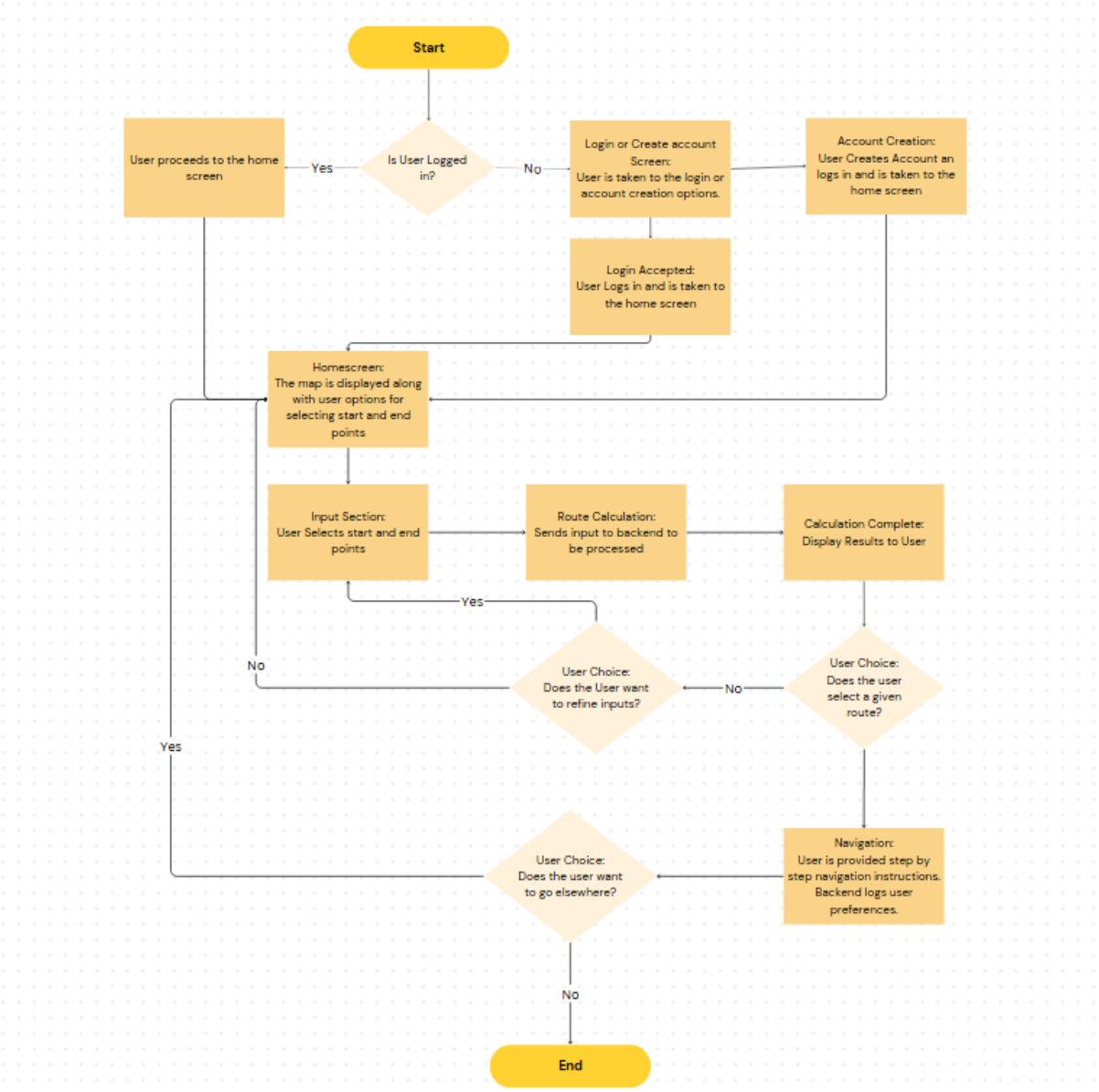

Figure 2:

5.4. Mock-Ups

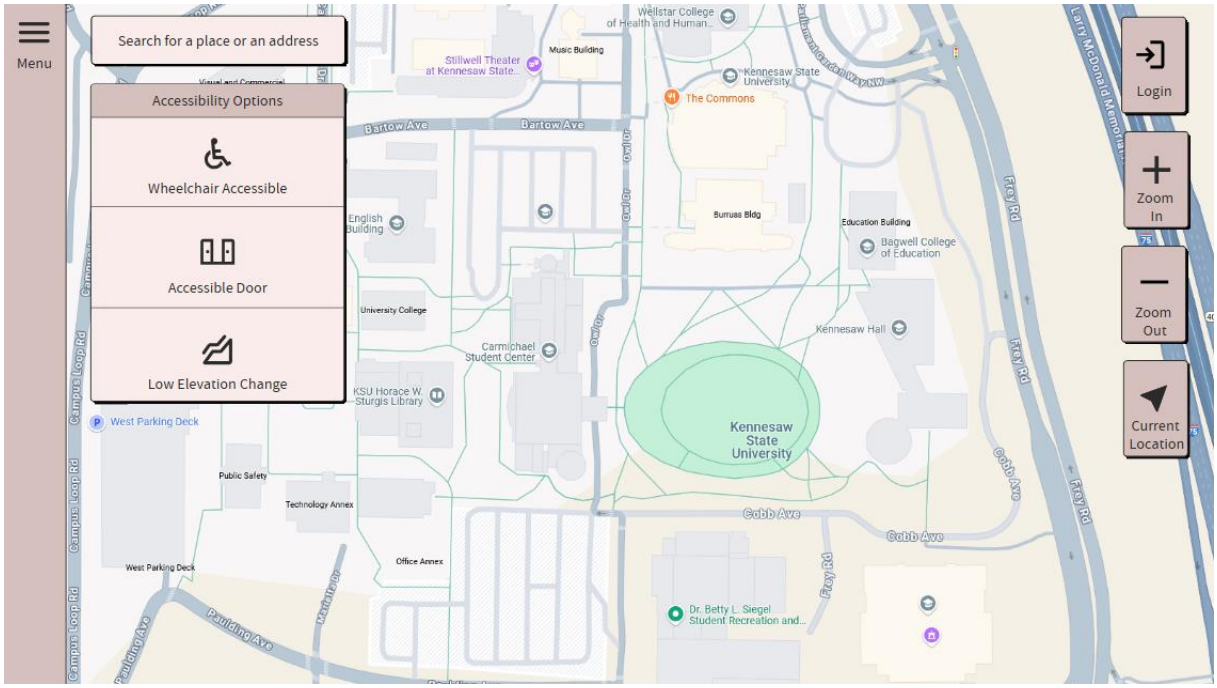

Figure 3:

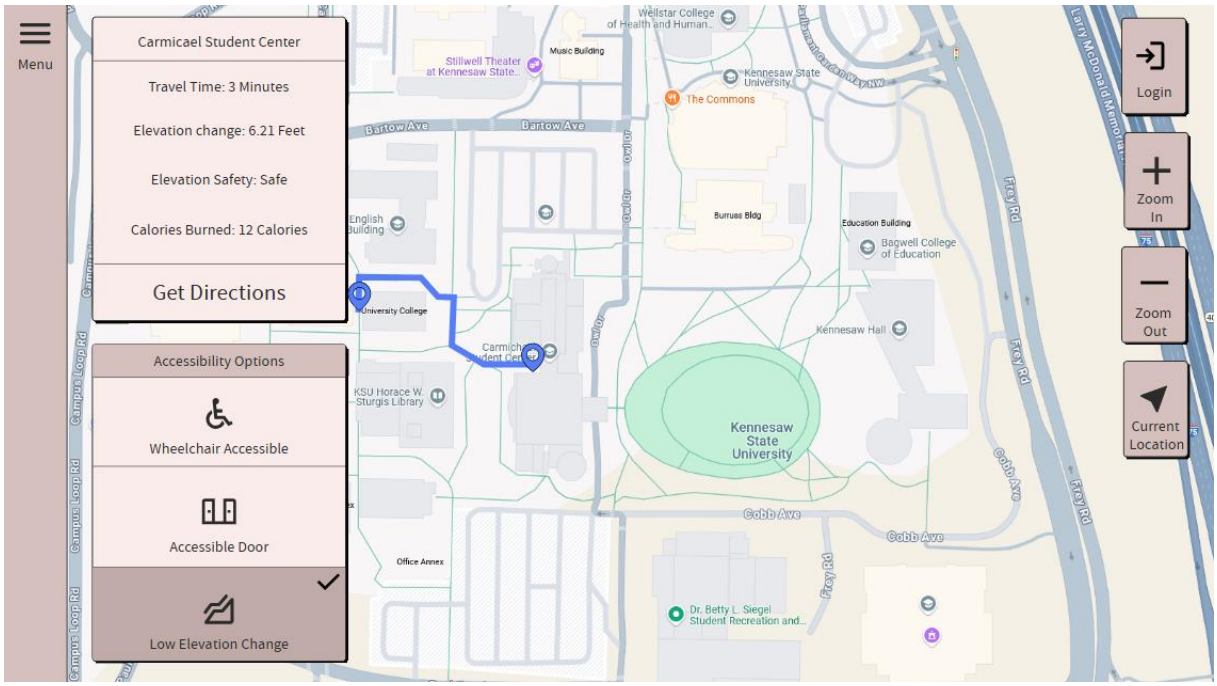

Figure 4:

Figure 5:

5.5. Constraints

Any relevant assumptions, limitations, or constraints for this component. This should include constraints on timing, storage, or component state, and might include rules for interacting with this component (encompassing preconditions, postconditions, invariants, other constraints on input or output values and local or global values, data formats and data access, synchronization, exceptions, etc.)

- **Performance (Frontend, Backend, Database)**
    - This is one of the more important metrics to get right in the project.
    - For the Frontend, we need the route to display in a timely manner as it will have a significant impact on the user's experience.
    - The Backend will need to calculate the routes in a timely manner after receiving a request from the Frontend.
    - The Database is extremely important to get query results in a timely manner, especially as the Database grows in size.
- **Scalability (Database)**
    - Scalability is more of a worry when it comes to providing the product to other institutions. Institutions that have bigger campuses and a larger student population would need to be taken into account during the development of the application.
    - The Database could be a bottleneck if it cannot handle a large number of requests. These requests can be due to more users and/or more routes.
- **API Limits (Backend)**
    - One limiting factor would be the API rate limit that Google Maps API has. This limit can hinder the process of getting coordinates for a destination and even the calculation of routes. This limit will need to be taken into consideration especially when it comes to an increase in users using the product.

5.6. Resources

A description of any and all resources that are managed, affected, or needed by this entity. Resources are entities external to the design such as memory, processors, printers, databases, or a software library. This should include a discussion of any possible race conditions and/or deadlock situations, and how they might be resolved.

- **React**
    - This will be the Frontend framework which will display a user-friendly interface. It will have components that the user can interact with such as the interactive map, search, and user account information.
    - Most of what will be displayed on the Frontend will run through React and any data retrievals will be requested from Flask (Backend).
- **Flask**
    - This is the Backend framework that handles the logic and communication between the database and the Frontend.
    - Flask will also communicate with the Frameworks/APIs to process the calorie and route calculations.
- **Google Maps API**
    - This will provide location and map rendering for the Frontend and specific locations for the Backend.

- **OSRM (Open Source Routing Machine)**
    - Will assist in providing the shortest paths for desired routes that are already mapped out in the real world.
- **NetworkX**
    - Compute the shortest paths of a custom graph that can resemble real life routes.
- **PostgreSQL**
    - Will be used as our database for the application. It will need to store information and be available to be queried to retrieve for the Backend to process the information.

5.7. Interface/Exports

The set of services (resources, data, types, constants, subroutines, and exceptions) that are provided by this component. The precise definition or declaration of each such element should be present, along with comments or annotations describing the meanings of values, parameters, etc. .... For each service element described, include (or provide a reference) in its discussion a description of its important software component attributes (Classification, Definition, Responsibilities, Constraints, Composition, Uses, Resources, Processing, and Interface).

- **Database**
    - It will provide data to the backend to retrieve for the API calls via queries. This data includes user and route information that the Backend will need to process and send to the Frontend.
    - This process is vital since it will help speed route selection for recently selected ones from the user.
- **Backend**
    - The Backend will provide an API consisting of endpoints for the Frontend to make HTTP requests.
    - The Backend will also be calling from external libraries and APIs such as Google Maps API and OSRM.

6. Glossary

An ordered list of defined terms and concepts used throughout the document.

Term	Meaning
API	Application Processing Interface
GIS	Geographic Information System
HTML	HyperText Markup Language
HTTPS	HyperText Transfer Protocol Secure

OSRM	Open-Source Routing Machine
SQL	Structured Query Language
WCAG 2.2	Web Content Accessibility Guidelines

7. Bibliography

Accessible Interactive Map Project Proposal. (2025, January 20). Kennesaw State University Internal Document.

This internal proposal outlines the project's goals, scope, and initial design considerations for the Accessible Interactive Map.

Flask Web Framework Documentation. (n.d.). Retrieved from https://flask.palletsprojects.com/

The official documentation for Flask provides detailed information on building backend web applications using Python.

React Documentation. (n.d.). Retrieved from https://reactjs.org/docs/getting-started.html

The React documentation explains the framework's component-based architecture and best practices for creating responsive user interfaces.

PostgreSQL Global Development Group. (n.d.). PostgreSQL Documentation. Retrieved from https://www.postgresql.org/docs/

This resource provides comprehensive information on using PostgreSQL for data storage and management within the project.

Google Maps Platform Documentation. (n.d.). Retrieved from https://developers.google.com/maps/documentation

Google Maps API documentation covers guidelines for integrating mapping and location-based services into web applications.

Open Source Routing Machine (OSRM) Documentation. (n.d.). Retrieved from http://project-osrm.org/docs/

OSRM documentation offers technical details for implementing route calculation features, which are integral to the project's routing functionality.

NetworkX Documentation. (n.d.). Retrieved from https://networkx.github.io/documentation/

The NetworkX documentation explains how to work with graphs and compute routes, supporting the project's backend processing of route data.