Scenic Navigation App

Kevin Buss, Jacob Marvel, Nicholas Santone, Robert Silver

CSC 468 – Group 1

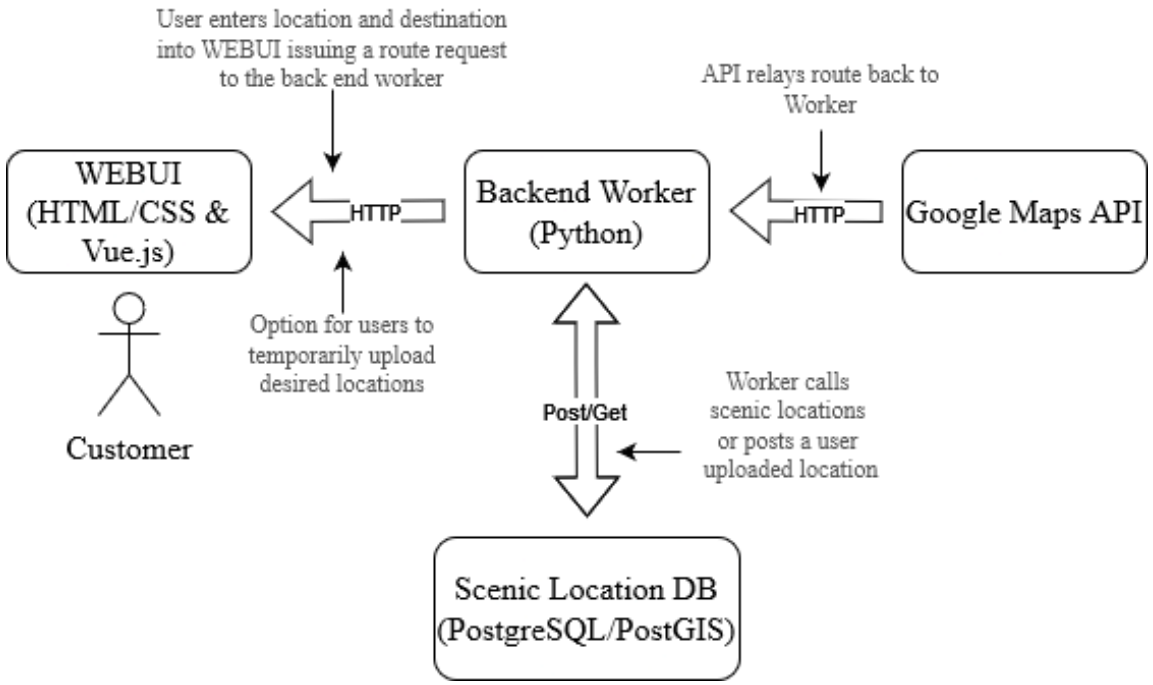
Chapter 1: Team Vision

1.1 Motivation

"It's not about the destination, it's about the journey." With our Scenic Navigation App, we take this adage to heart, transforming mundane routes into something memorable. Today's navigation apps are optimized to reach your destination as quickly as possible. We aim to reimagine everyday travel by prioritizing enjoyment over speed. Input your destination, and our app crafts a personalized journey, weaving through picturesque roads and landmarks to achieve this goal.

1.2 System Architecture Overview

Our cloud-based application follows a modern, microservices-oriented architecture. The frontend, built with Vue.js, provides an intuitive user interface for entering locations and viewing scenic routes. The backend, powered by Python, handles route generation logic and interfaces with the Google Maps API and our scenic locations database. The database, managed with PostgreSQL and PostGIS, stores curated scenic points of interest. The entire system is containerized using Docker for portability and scalability, and will be orchestrated using Kubernetes. Continuous Integration and Continuous Deployment (CI/CD) pipelines ensure smooth development and deployment workflows.



Chapter 2: Implementation Proposal

2.1 Web UI

The Scenic Navigation App's frontend will be developed using Vue.js. The user interface will allow users to input their desired start and end locations, and display the generated scenic route with turn-by-turn directions. The map component will be integrated using the Google Maps API, enabling features such as autocomplete location search and dynamic route rendering.

2.2 Backend Development and Data Management

The backend server, written in Python, will serve as the central hub for the application's functionality. It will handle incoming requests from the frontend, retrieve scenic coordinates from the database, and communicate with the Google Maps API to generate optimized scenic routes. The server will also manage the storage and retrieval of user preferences and route history.

2.3 Containerization with Docker

Each component of the application will be containerized using Docker to ensure portability and ease of deployment. Dockerfiles will be created for the frontend, backend, and database components, specifying the necessary dependencies and configurations. Multi-stage builds will be employed to optimize image sizes and improve build efficiency.

```
# Stage 1: Building Vue.js
FROM node:lts-alpine as build-stage
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
RUN npm run build

# Stage 2: Start the http-server
FROM node:lts-alpine
WORKDIR /scenic-navigation-app
COPY --from=build-stage /app/dist /scenic-navigation-app
RUN npm install -g http-server
EXPOSE 8080
CMD ["http-server", "/scenic-navigation-app", "-p", "8080"]
```

2.4 Cloud Integration and CI/CD Pipelines

The application will be deployed on CloudLab infrastructure. CI/CD pipelines will be set up to automate the build, test, and deployment processes. Code changes will trigger the CI pipeline, which will build Docker images, run automated tests, and push the validated images to a container registry. The CD pipeline will then deploy these images to the CloudLab environment.

2.5 Database Development

The database will be developed using PostgreSQL with the PostGIS extension to support spatial data. The logical schema will include tables for storing scenic locations, user preferences, and route history. The database will be accessed by the backend server through a Data Access Object (DAO) pattern, which encapsulates the database operations and provides a clean interface for data retrieval and manipulation.

Chapter 3: Implementation Details

3.1 Building Docker Images and Testing Docker

Images for each component will be built using Dockerfiles. Challenges may include managing dependencies, optimizing image sizes, and ensuring proper configuration. Testing will involve running the containers and verifying their functionality through unit tests and integration tests. Health checks will be implemented to monitor the containers' status and ensure their reliability.

3.2 Data Collection and Management

Initially, data on scenic locations around West Chester will be collected manually by finding coordinates of relevant landmarks using Google Maps. These coordinates will be inserted into the PostgreSQL database table to populate the app's initial dataset. Additional data collection methods will allow users to add scenic points through the app's interface.

The backend server will interact with the scenic locations database through a data access object (DAO) layer, which will encapsulate the logic for querying and inserting location records. This will provide a clean interface for the server to request or upload scenic waypoints when generating routes or users uploading scenic points.

3.3 Limitations and Concerns:

The primary limitation is the tight deadline for completing the project, which may pose challenges in terms of balancing development efforts with other commitments. However, the utilization of the Google Maps API and its generous free usage tier is expected to alleviate some of the technical challenges. Security concerns, particularly related to API key management, will be addressed through secure key storage and rotation practices.