Software Design and Architecture

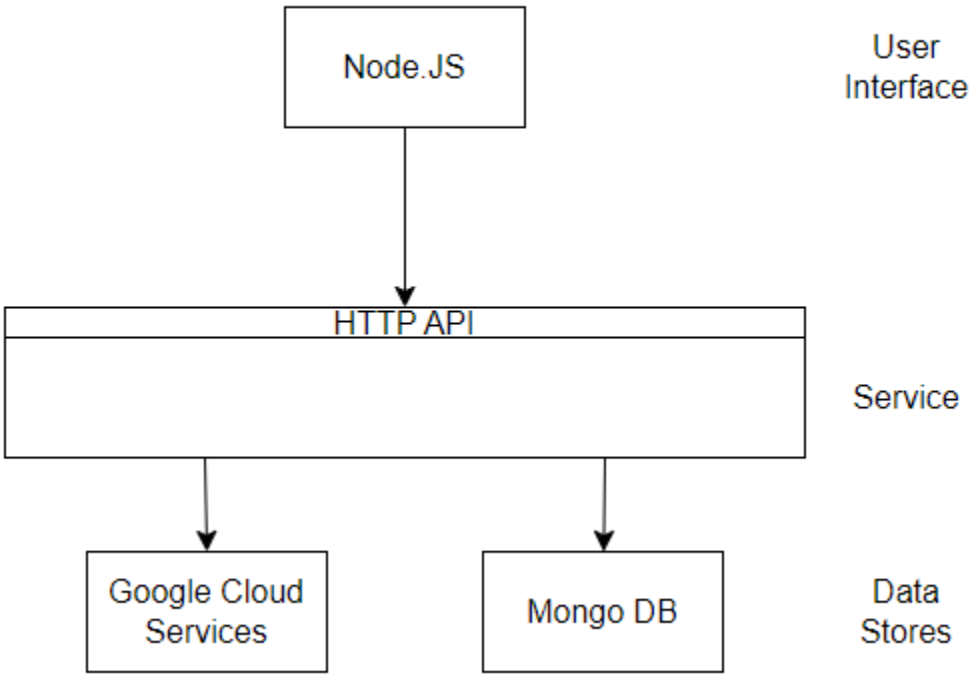
Introduction

This document outlines the architecture for a web-based road monitoring system. A well-defined architecture ensures scalability, maintainability, and reliability, it aligns with our goals as a group and the goals of the class.

Architectural Goals and Principles

The key objectives our architecture aims to achieve are multi-platform access, scalability, and security using the principles of separation of concerns, encapsulation, and modularity.

System Overview



Architectural Patterns

We have chosen to use the microservices design architectural pattern. We chose this architecture as our project will have several small services that will interact with each other to be able to create our finished product.

Component Descriptions

- User Interface: Handles user interactions with our web service/application, displays map data and data associated with flooding or other hazards, provides support for finding data in the local area and viewing common hazards.
- Backend Server: Process API calls, send responses and handle inputs, Interface with the database to enter data and retrieve data.
- Database: Stores and manages data using a non-relational database system.

Data Management

Our data will be structured using a non-relational database scheme for hazard data, user profiles, and location data. The data will be accessed using RESTful API endpoints for CRUD operations on the hazard data, location data, and user profiles.

Interface Definitions

Hazard related API Calls
GET /hazards will return a list of all reported hazards in the database
GET /hazards/location returns the list of hazards in a specified location that the user is viewing on the map
POST /hazards/location will create a new hazard report in the database
GET /hazards/<hazard_id> can show a specific hazard's data
DELETE /hazards/<hazard_id> will delete a specific hazard, will only be accessed by admin profiles
*GET /locations returns a list of all locations, along with positional map data for that location. (for creating the map overview)

User Profile API Calls
GET /users/<user_id> Retrieve user information, accessed by the user themself or admin profiles
POST /users Create a new user profile
PUT /users/<user_id> Update a users profile, accessed by the user themself or admin profiles
DELETE /users/<user_id> Delete a users profiles, accessed by the user themself or admin profiles
*GET /users gets all current user profile names.

*indicates an endpoint that may not make it to the final stage of development, depending on group decisions.

Considerations

Security

One security risk is being able to spam wrong information to our system, which will make our data irrelevant. One method to be able to fix it is to monitor rates that which accounts submit data, or rates that IP's submit data and if too much is submitted in too small of a time frame we can black list that account/ip

If we have user data, a security risk is protecting that data. A way to fix this is by running a strong encryption algorithm to protect the user information.

Performance

Discuss performance requirements and strategies to meet them, i.e., scalability, load balancing, and caching mechanisms.Some examples: horizontal scaling using container orchestration, local storage for the web application, etc.

Maintenance and Support

Once 1.0 is complete this project will be handed to OSU and they will handle any updates that they decide are necessary for the project. It is at this point that Oregon State can determine whether they would like to continue the project with a new group to potentially be developed further, if the project will remain as-is with some dedicated maintenance, or if the project will be terminated upon our departure from the university.

Deployment Strategy

Our architecture will be deployed to either git pages or through google cloud hosting(undecided currently). We will have our main branch be hosted, but will reserve private branches for testing purposes that will be inaccessible to the public.

Testing Strategy

We will primarily use applications like Jest, Postman, and Insomnia to test the API calls, database integration and user interface.

Glossary

API: Application Programming Interface
Crud: Create, Read, Update, Delete.