CS Capstone Design Document

November 28, 2018

HyperRail App

Prepared for

OPENS Lab

Chet Udell

Prepared by

Group 25

Vincent Nguyen
Adam Ruark
Yihong Liu

Abstract

This document explains the tools and methods that the project team will be using to implement the HyperRail web application.

CONTENTS

1 Introduction

1.1 Scope

The scope of this document includes the design decisions and rationale for each aspect of the HyperRail Application project. For each aspect, the desired functionality will be summarized and the methods, techniques, and/or tools that the team has decided to use to achieve said functionality will be discussed.

1.2 Purpose

This design document details the technologies to be used during the development of the HyperRail Application as well as the rationale behind each decision. After reading this document, a developer should know what components are needed for the project as well as the tools that can be used to implement the solution.

1.3 Intended Audience

This document is intended for the project client and the senior capstone advising team. Both the client and capstone advisors will use this document to examine and analyze the project team's design choices as well as monitor their progress during development.

2 Definitions

- HTML: A web markup language that can be used to define the structure of a web page.
- CSS: A web styling language.
- JavaScript: A web scripting language that can be used for the browser and on a server.
- NodeJS: A platform that can run JavaScript and host web pages on a server.
- WebStorm is a powerful IDE for modern JavaScript development. Besides client-side applications, WebStorm helps you develop server-side applications with Node.js, mobile apps with React Native or Cordova, and desktop apps with Electron.
- JSON: JavaScript Object Notation. This is a popular method of handling information with key-value pairs.
- MVC: Model - View - Controller. A traditional client/server representation. The View is the client's perspective of a system (usually a web page), the Controller is the server, and the model is the data passed between the two. It is the job of both the Controller and the View to interpret the data passed to it.
- API: Application program interface. This describes the functions a developer can use with a third-party tool

3 Project Overview

The HyperRail is a small railway where an automated environmental sensor package, which contains a variety of different sensors, can traverse through a space and collect information as it travels along the railway. Using the HyperRail application, the sensor package's settings can be customized and monitored. The application currently allows the user to specify the speed of the package, current length of the rail, and the size of the spool used for the motors. It also monitors the position of the sensor package along the railway and updates its location in real-time. Our proposed solution to this problem is to create a web application that communicates with a central server, which in turn communicates with the sensor system. This web application will be mobile friendly to work on Android, iOS, and personal computers. The web application will let the user configure schedules for the sensors to run, set up intervals for the sensors to collect

data, and also display the current status of the sensor system. Once a user saves a configuration, these settings will be uploaded to a central server where it will be deployed to the corresponding sensor system. Scheduling runs will be controlled and initiated by the central server by sending signals to the HyperRail sensor system with locations to go to, rate of travel, and any points of interest to monitor along the way. The data collected by the sensors will be uploaded to the central server and can be viewed by the web application.

4 Hardware

For the base of the environmental sensor package, the Adafruit Feather M0 board will be used as it has many general purpose input and output (GPIO) pins, useful for connecting multiple sensors, and it also has different variations that support various methods of communication, such as Wi-Fi. With 256KB of flash memory and 32KB of static random access memory, the board has lots of memory to work with as it processes configurations sent by the user.

Because the HyperRail Application is meant to be compatible with any sensor, there are no required sensors. Sensors that are used for the HyperRail will be connected to the GPIO pins on Adafruit Feather M0 board. The data collected by the sensors will be sent to the board, which will then transmit the information to the HyperRail web application so the user can interact with the data.

5 User Interface

The web application is a web interface for users to operate the sensor on the HyperRail system to collect corresponding data for their research. The client-side implementation of the web application user interface (UI) will be made using HTML, CSS, and JavaScript while NodeJS will be used for the server-side hosting. The current list of planned views are as follows:

- Account: Provides views for account management, including user login, registration, password recovery, login failure, email confirmation, other related account information.
- Faculty: Provides views for facility staff to manage users in their facility, and some related information about term of service.
- Home: A home page with general information about the HyperRail system and application, such as its documentation and features and contact information.
- Account Settings: Provides additional views for account management beyond login, including password changes, binding other information to the account, logout, etc.
- Sensor Configurations: A page that requests the user to input and/or displays the current sensor configuration information, such as rail length, spool radius, and velocity. It will also display a list of saved configurations.
- Sensor Scheduling: A page that shows a list of saved schedules and displays the current scheduling information for the HyperRail.
- Data information: The page shows the collected data from the sensor package on the HyperRail system.

6 Data Flow

The main work flow for our application is as follows:

1) The user opens up a browser and puts in the URL for the HyperRail application.

2) The server responds and returns a web page.
3) The user enters in some configuration values and clicks ”Run”.
4) The data gets sent back to the server and is processed.
5) Processing the information involves transforming it into values the Arduino running the HyperRail can understand.
6) The information is sent to the HyperRail and the system begins running.

There are many other work flows possible in our system, but the majority of the data passed around will go through similar steps. Overall, the key component here is that the central server processes the data before it is passed on to the HyperRail system. One other component not mentioned here is a database. Users will have the option to save and load their configurations, so data will need to be stored. This flow also relies on the central server to transform the data passed between the endpoints.

6.1 Server - User connection

There are two ways that data will be passed between the user and the server: static files and dynamic JSON payloads. We plan on using a traditional MVC method to handle the data passed around. This allows us to seamlessly manage information and it is accommodating to changes as new sensors or interface changes come about. On the initial connection between the client and the server, the first piece of information passed between the two is a web page. Our application will only have a couple different pages to serve so this interaction will not be discussed further.

The intensive portion of this connection will the be the JSON payloads passed between the two. These payloads will contain information such as HyperRail configurations, data collected from the HyperRail sensors, the current status of the HyperRail system, and more. Generally, the client will determine what action to perform by sending a request to a specific endpoint on the server and the JSON included in the request will contain the information specific to that unique request. For example, */config* with a payload containing containing the value *load* will tell the server that the user is requesting a configuration. The server will then send the requested configuration back to the client where it will be processed and displayed to the user.

6.2 Server - Database

The database will contain all the information we need to save. This includes HyperRail configurations, data collected from the system, and potentially user login information. One thing that this database will not store is the status of the HyperRail system. That information will be resolved at run time.

The information that will get loaded into the database will come from two sources, the client or the HyperRail. The client will be sending and requesting HyperRail configurations from the database and the HyperRail will be uploading collected data from each run. Since our server will be run off of NodeJS, we will be using MongoDB as our database tool. This is because MongoDB provides a JavaScript API that allows a painless experience between the two technologies.

Internally, there will be a separation of concerns for what is stored in the database. The data collected by each HyperRail will be stored in a unique table, while the configurations uploaded by the user will be held under one table. This should hopefully reduce the amount of processing needed when requesting information from the database.

6.3 Server - HyperRail

Lastly, the server must also communicate with the HyperRail. To simplify this, the connection will also be over HTTP (same as the server-client connection) to reduce the server overhead needed. The HyperRail system will be configured to listen to requests from the server and then execute them. Consequentially, the data sent over this connection will also be in the form of JSON. The HyperRail is tasked with processing the payload and then executing a run based on the information sent. Once a run is done, data is sent back to the server in the form of a JSON payload and is then stored in the database for use later. This connection will be abstracted away from the user and they should never have to worry about this process.

The information sent to the HyperRail will vary depending on the sensors available at the time and also the type of run the user wants to execute.

7 FUNCTIONALITY

The HyperRail application will allow the user to remotely interact with the physical HyperRail system. The application will allow the user to configure the sensor package's settings, such as the package's travel velocity, the motor's spool radius, the HyperRail length, and scheduling information, save them for future use, and upload them to the package itself. The package scheduling will allow the user to specify which sensors the package will use, where along the rail each sensor will collect data, and how frequent the package will traverse the rail. Configurations and schedules can be saved, reloaded, modified, and deleted in case the user wants to reuse certain settings or make room for new ones. Data collected by the sensors will be sent back to the web application where it will be displayed to the user.

Stretch goals for the application include automated data translation, fleet configurations, and actuator interactions. Automated data translation would examine the data collected and dynamically create actionable items or buttons for the HyperRail user to quickly analyze the information or take action. Fleet configurations would allow the user to configure a multitude of sensor packages at once, saving time and effort if they were used for the same tasks. Actuator interactions would allow the package to interact with existing actuators in the environment. Because actuators are used to perform simple motions, such as turning, pushing, or pulling, this would allow the HyperRail to interact with valves and pistons, which can help regulate the environment.