Documentation for Transit-APP Backend API

By Camren Coleman

Introduction

The backend is written in Node js with Express. It handles requests from the frontend and responds when necessary. The backend can invoke three different functions in a separate program to get the necessary information, this program is described below. It runs in a docker container on Google cloud.

Basic URL:

https://transit-app-backend-lybtu35apa-uw.a.run.app/line

Intermediate Routes

/line {`lineNum: string, lineName: string, lineDir: string`}

Line is a request that can act in one of two ways:

Case 1: If no arguments are provided in the query, the response is a list of all bus lines, with UCSC busses listed first.

Case 2: If arguments are provided, a list of stops that the selected line services is sent as a response.

After Case 2, all information is collected, and the client can proceed to the final route.

Final Routes

/linestop {`lineNum: string, stopName: string, lineDir: string`}

A linestop request is the final request in the chain, It provides the estimated time of arrival for that stop for the given line.

All previous fields (Line number, Stop name, and direction) are required to make this request. However direction is automatically filled in if not provided.

Loop Bus Calculation

The location of loop busses (longitude and latitude) is received by the Slugloop API. However, the calculation of time to the selected bus stop needs to be calculated.

This calculation is performed by the python script and the response is identical to that of the Santa Cruz metro web scraper.

Communication

The Node js backend invokes a running python script whenever it receives a valid request. This python script runs alongside Node to provide web scraping services and calculate the loop bus location and ETA.

To remedy complications with cross language communication, the Socket.IO library allows for communication between them.

Whenever a Route with a query is processed, the information in the query is sent as a JSON to the python script, using Socket.IO events.

After the information is processed, the python script sends the result back as a JSON, which the Backend parses and returns relevant information to the frontend.

The Socket.IO server/client runs on port 5000 in the docker file.

Program flow: