

MEAN Travlr

# **CS 465 Project Software Design Document**

Version 1.2

## Table of Contents

[**CS 465 Project Software Design Document** 1](#_Toc36198462)

[Table of Contents 2](#_Toc36198463)

[Document Revision History 2](#_Toc36198464)

[Instructions 2](#_Toc36198465)

[Executive Summary 3](#_Toc36198466)

[Design Constraints 3](#_Toc36198467)

[System Architecture View 4](#_Toc36198468)

[Component Diagram 4](#_Toc36198469)

[Sequence Diagram 5](#_Toc36198470)

[Class Diagram 6](#_Toc36198471)

[API Endpoints 7](#_Toc36198472)

[The User Interface 7](#_Toc36198473)

## [Document Revision History](#_heading=h.lnxbz9)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | <3/21/21> | Christopher George | Filled out a draft for the Executive Summary, Design Constraints, and Component Diagram sections. |
| 1.1 | <4/4/21> | Christopher George | Added a sequence diagram of events with a description, a class diagram of JavaScript classes with a description, and current api endpoints (subject to change) |
| 1.2 | <4/14/21> | Christopher George | Added the user interface section with images and summery.  Updated API endpoints to reflect current project state. |

## Instructions

## Fill in all bracketed information on page one (the cover page), in the Document Revision History table, and below each header. Under each header, remove the bracketed prompt and write your own paragraph response covering the indicated information.

## [Executive Summary](#_heading=h.35nkun2)

In the information age, a clean, fast, and well-designed website can mean the difference between the life and death of a business. To compete, it is essential that a company has a website which accomplishes several goals at once:

1. Fast user access.

Users must be able to connect to a website quickly with quick loading times between pages.

1. Comprehensive admin dashboard.

A powerful admin dashboard can enable the easy long-term maintenance of a website while also keeping website content up-to-date and supporting users in their trip purchases.

1. Scalable data storage.

Scalable storage for user data, login information, and other data is essential to enable an easy growth of the business and the website.

With these goals in mind, the structure that seems most accommodating to the needs of the business would involve the use of a MEAN stack. A MEAN stack is a type of software stack makes use of MongoDB, Express, Angular, and Node.js. This stack will enable us to accomplish all of the goals listed. With this stack, we can create a user facing website that is quick to load and navigate with Express. With Angular we can create a powerful Single Page Application (SPA) for the admin side of the website. With MongoDB we can create a scalable data storage system which can include json webpage content, user data, login data, and analytics. And node.js runs the whole site and glues the components together.

For the front-end, we can create navigable pages which contain json content using Express and MongoDB. This will allow for quick load times for the user which can be essential for user retention. As mentioned, the admin dashboard will be created as a SPA (Single Page Application) which will allow for quickly making changes to the site and handling users at the cost of a little load time on launch.

## [Design Constraints](#_heading=h.1ksv4uv)

For this project, there are several constraints for this website.

Users must be able to:

1. Create an account
2. Search for travel locations by price point
3. Search for travel locations by location
4. Book reservations
5. See their itinerary

Admins must be able to:

1. Maintain customer base
2. Maintain available trip packages
3. Maintain pricing packages
4. Maintain pricing for other items

These constraints must be met with a public facing website and a separate, admin only website.

With the constraints in mind, the public website should consist of pages listing content such as packages and items defined in the marketing wireframe, login and create account pages, and a user dashboard so that they can view account information and their itinerary. The admin account should consist of an SPA which allows for the access of certain user data, the uploading of page content, and manual inputting of price information and other content such as links.

## [System Architecture View](#_heading=h.44sinio)

### Component Diagram



A text version of the component diagram is available: [CS 465 Full Stack Component Diagram Text Version](https://learn.snhu.edu/d2l/lor/viewer/view.d2l?ou=6606&loIdentId=24342).

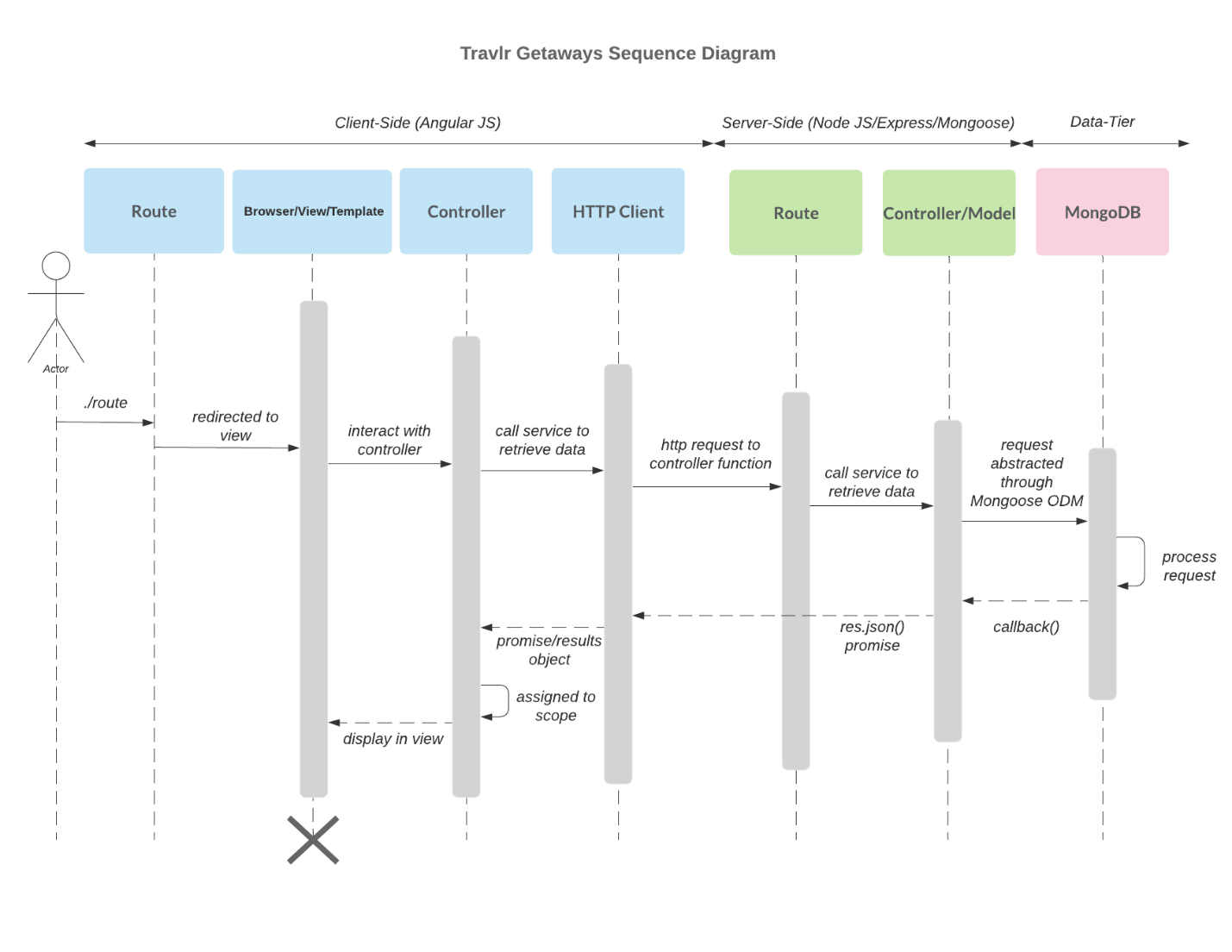
As can be seen from the diagram above, the website will consist of three main component categories, the client, the server, and the database.

When a user opens the public facing website, this will create a new client session on the web browser. In this session, the user will have the opportunity to login or create an account which will talk to the authentication server In the server section and create a new server session.

The server session will then attempt to authenticate the user by talking to the MongoDB database in the database section through Mongoose ODM or will file a new account into the Traveler Database and send encrypted login credentials into MongoDB through Mongoose ODM.

Back in the client section, after logging In or creating an account, the user is then able to access the Traveler portfolio which will be populated by the MongoDB server after a user request has been sent. The Traveler Portfolio will also be populated by the graphic library located on the public website without the need to retrieve it from a database to ensure quick load times.

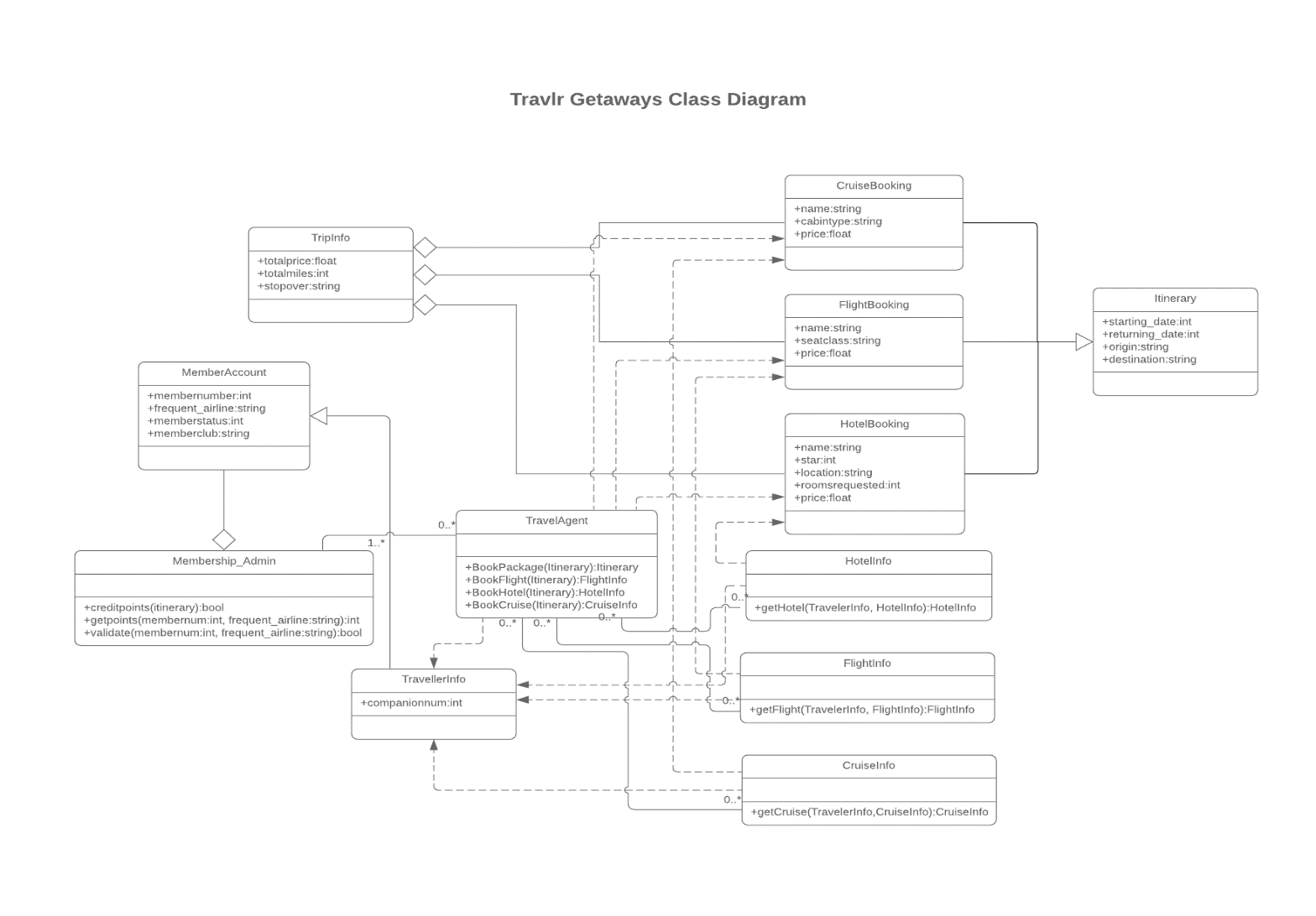
### Sequence Diagram



When a user enters the site, a web page is routed to them with a route in the app\_server. Next the page talks to the app\_server controller for that page to retrieve page content. That request is then set to the app\_api though a HTTP request. That request is then routed to the controller/model which uses mongoose to talk to MongoDB and request the page content from the correct collection.

After all this, the page data is then sent back through the app\_api controller/model as a json file, back through the app\_api router, through the HTTP client to the app\_server controller, finally being interpreted by the controller and relayed to the front-end web page which then uses that data to populate the hbs web page.

## Class Diagram



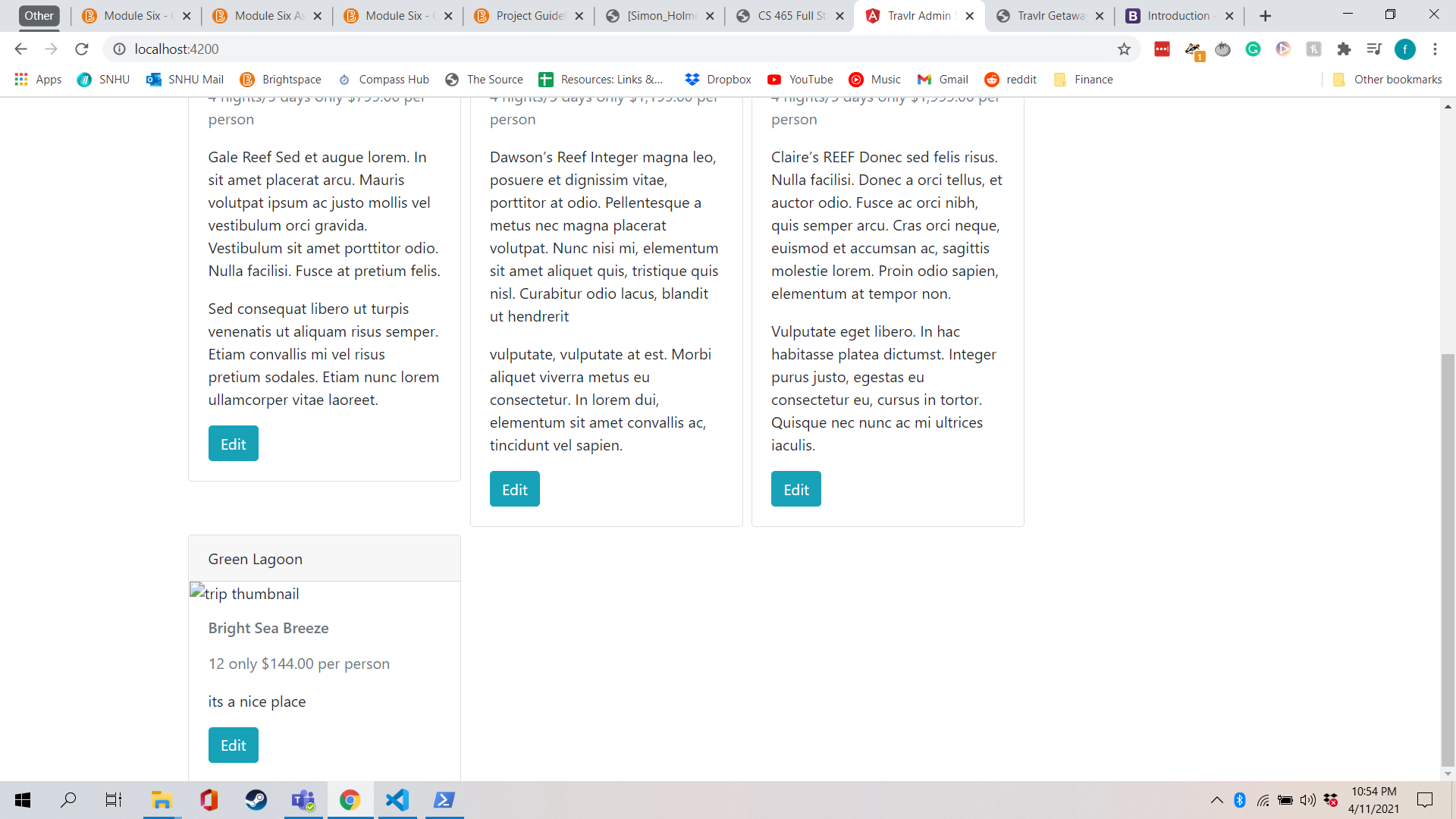
Each of the 12 JavaScript classes shown above serve a purpose to allow for the booking of packages, the displaying of user data, or the verification of membership data. The HotelInfo, Flightinfo, CruiseInfo, and TravelAgent classes are all dependent on the TravellerInfo class which populates or uses each of these classes. When a user books a package, all of the classes dependent upon the travellerInfo class are called and the whole process starts. Next, the member account linked to the travellerInfo then checks for values in the Membership\_Admin class for benefits that the travel agent can use, such as credits and points, which is also validated. The TravelAgent class is then able to take all of the information outlined above and book a cruise, a flight, or a hotel with the respective classes CruiseBooking, FlightBooking, and HotelBooking respectively. After the package has been booked, that data is then aggregated by the TripInfo class, and potentially added to a user facing dashboard. The Itinerary class is then populated with the booking information and displayed to the user through a dashboard.

## [API](#_heading=h.2jxsxqh) Endpoints

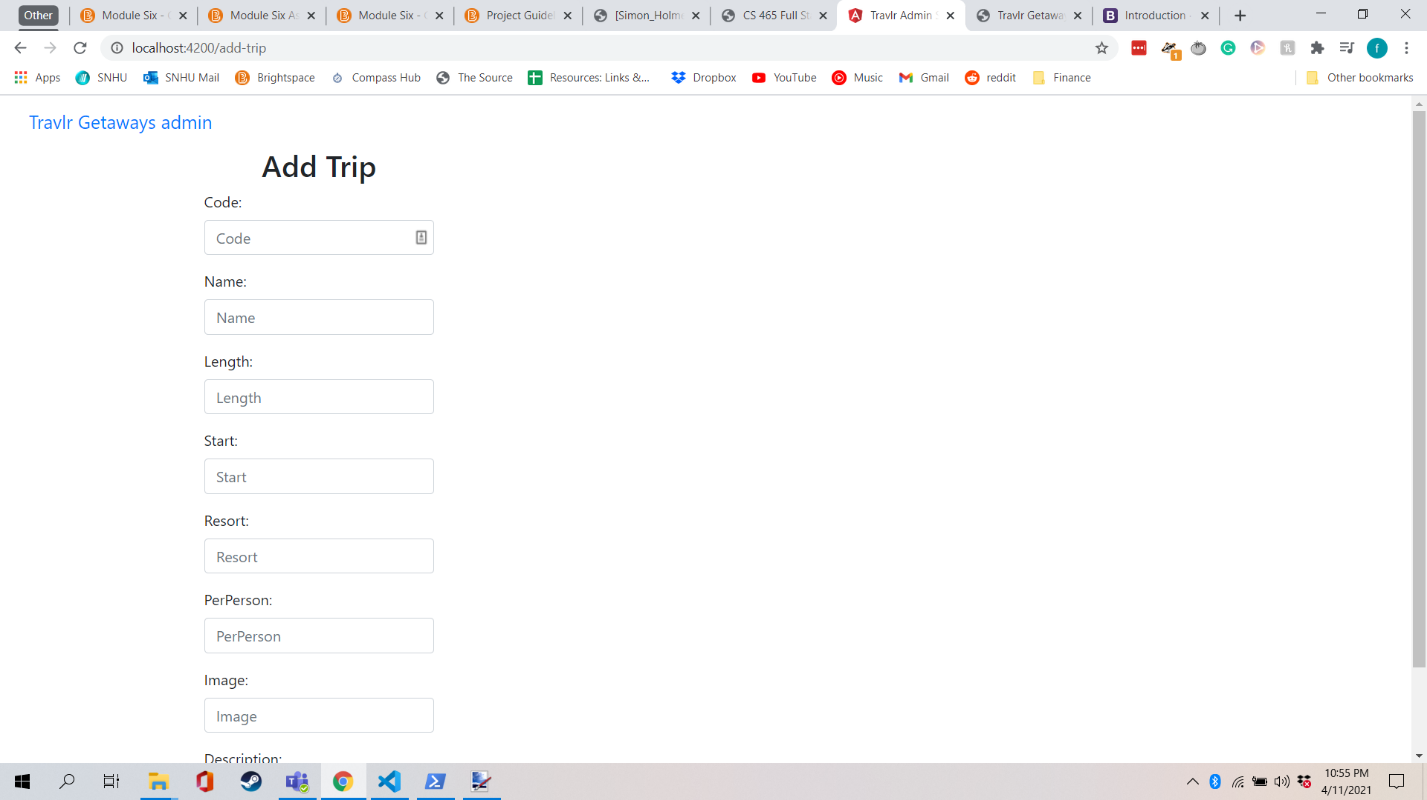
| **Method** | **Purpose** | **URL** | **Notes** |
| --- | --- | --- | --- |
| **GET** | Get all trip locations | /api/trips | Returns a list of all trip locations and data |
| **POST** | Add a new trip | /api/trips | Adds a new trip to the database. |
| **PUT** | Updates an existing trip | /api/trips | Edits an existing trip in the database. |
| **GET** | Get one trip location | /api/trips/:tripsId | Returns a single trip location and data |
| **POST** | Login with credentials | /api/login | Returns a login token when given proper login credentials. |
| **POST** | Register account | /api/register | Registers an account with the given credentials |

## The User Interface

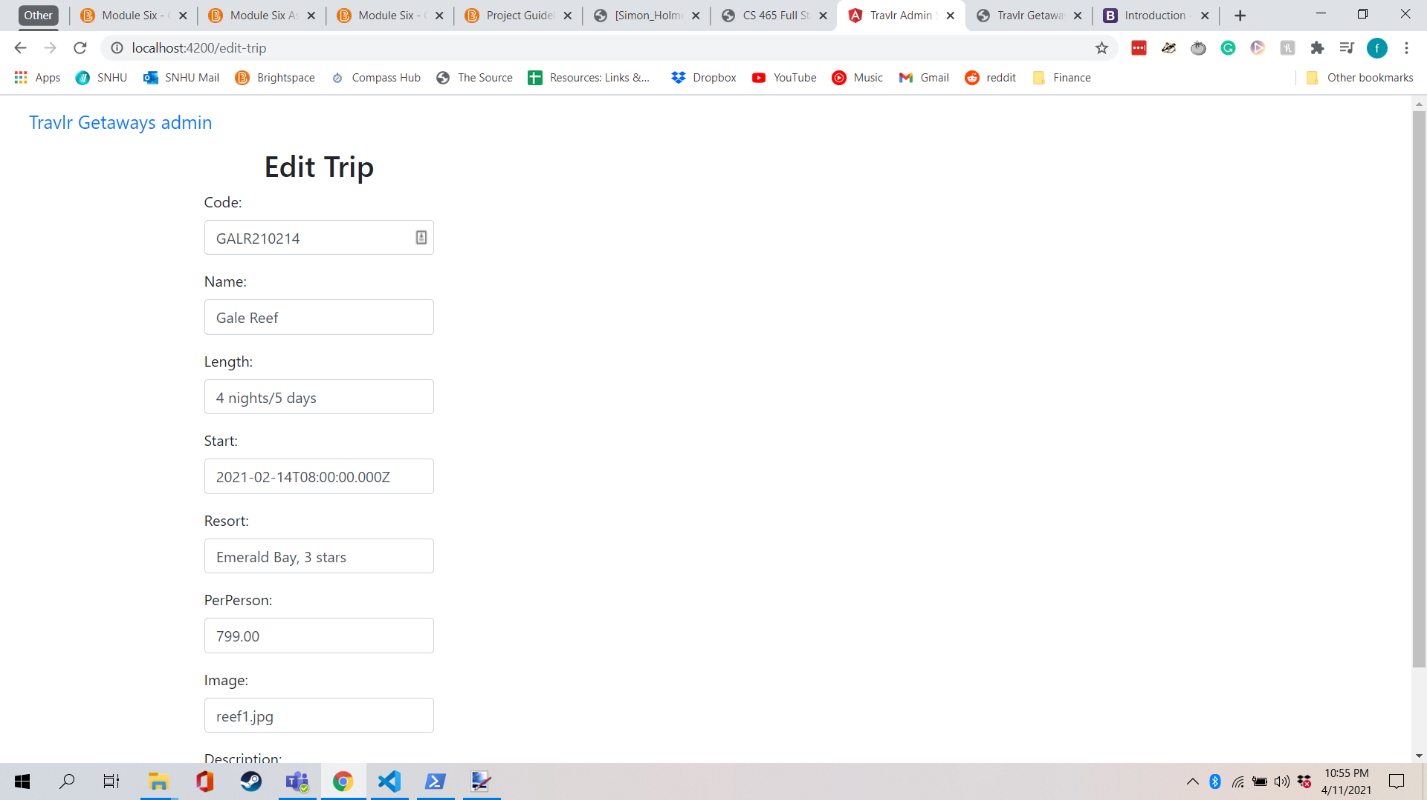
**New Added Trip**



**Add Trip Display**

****

**Edit Trip Display**



The customer facing Express website is set up in a way that is more page based then Angular. Each page has its own hbs view that view where all of the page html is stored. It then links to each other page through a router and the page data is going through the controllers and API. When you switch to another page, it changes the URL to the correct page. The Angular SPA on the other hand doesn’t switch between URL’s and instead displays everything on the same page by switching around components. Instead of hbs files, all of the components have an html file. Each page also needs a component.ts file to handle the page logic. All of these components are displayed on the single page, app.component.html.

There are several disadvantages and advantages when it comes to SPA’s. One of the main advantages is that it allows for doing things on the SPA quickly without the need to load additional pages. This allows for quickly making and seeing changes in database, for example. This does however come at the cost of a slower initial load time since you are loading the whole application at the same time rather then just a single page. Another disadvantage of SPA’s is that search engines have some trouble searching them for search terms. This can end up with a reduced likelihood that a website will show up on searches. There are ways to work around this downside, but it is certainly something to keep in mind.

For testing, I ran the SPA often and used Postman to check that everything working. The one error I ran into was when I was trying to update the individual trips. I kept getting an error where the value of the json variable “code” kept coming up as null. I will have to fix before the final version.