

<Name of Software Application>

# **CS 465 Project Software Design Document**

Version 1.0

## 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 3](#_Toc36198468)

[Component Diagram 3](#_Toc36198469)

[Sequence Diagram 4](#_Toc36198470)

[Class Diagram 4](#_Toc36198471)

[API Endpoints 4](#_Toc36198472)

[The User Interface 4](#_Toc36198473)

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

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | <mm/dd/yy> | <Your Name> | <Brief description of changes in this revision> |

## 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)

This project is for the client Travlr Getaways. The project consists of building a travel booking web site that should have a customer facing site where customers can look for, price, book, and view updates on vacations they are interested in. Also, this site will need an admin only side to it as well, where administrators can maintain a customer base, available trip packages and pricing for everything on the site. The customer facing side of the website will be a server-side rendered application that will display all the customer needs and will include being able to create an account, log in, search, book packages and trips, and check the status of bookings whenever they may need to. The admin side of the site will be a single-page, client rendered application that will enable the admin to maintain the customer base, available trip packages and pricing for the site. Making this side an SPA will ensure the performance and make it more efficient to manage the site when needed.

The tech stack being used for this project will be the MEAN Stack. This stack consists of MongoDB for the database, Express is a web server that will be used for allowing middleware to augment or change its behavior such as with templating using Handlebars, Angular will be used to create the front end of the application to make dynamic pages giving us an interactive user interface, and Node is our JavaScript runtime environment that executes our JavaScript applications. This stack makes it very easy to quickly develop a full-stack application as the base language for all pieces of the stack is JavaScript.

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

Some constraints that will need to be considered in this project are:

* This is a web application.
* Users will need to be able to use a browser to access the application.
* Users must be able to create an account and log in.
* Site must have secure log in as personal information such as payment information will be handled.
* Users must be able to search for packages in a specific way.
* Users must be able to book packages.
* Users must be able to check on packages that they have booked.
* Admin users must be able to maintain customer base.
* Admin users must be able to maintain available packages.
* Admin users must be able to maintain pricing on site.

With these constraints there are many decisions that have been made to deliver this product to the customer. The first is to separate the web site into two types, a server-side rendered site for the customer, and a client rendered site for the admin users. This will help separate the roles and uses of the site and also give a different experience based on the user’s needs. With this site handling personal information there is also a need to keep data safe, and have a secure log in so that we can be sure who is accessing the site when logging in.

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

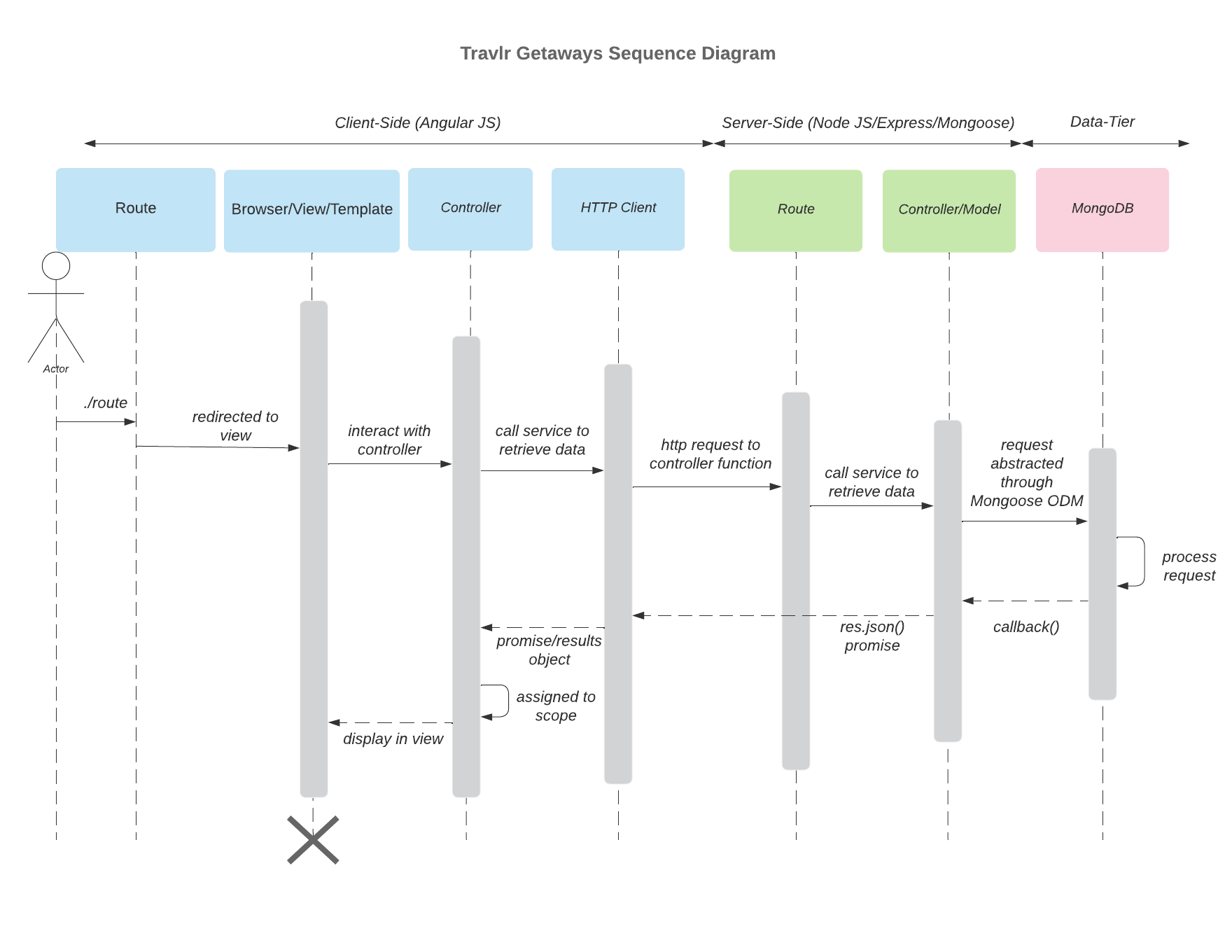
### Component Diagram



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).

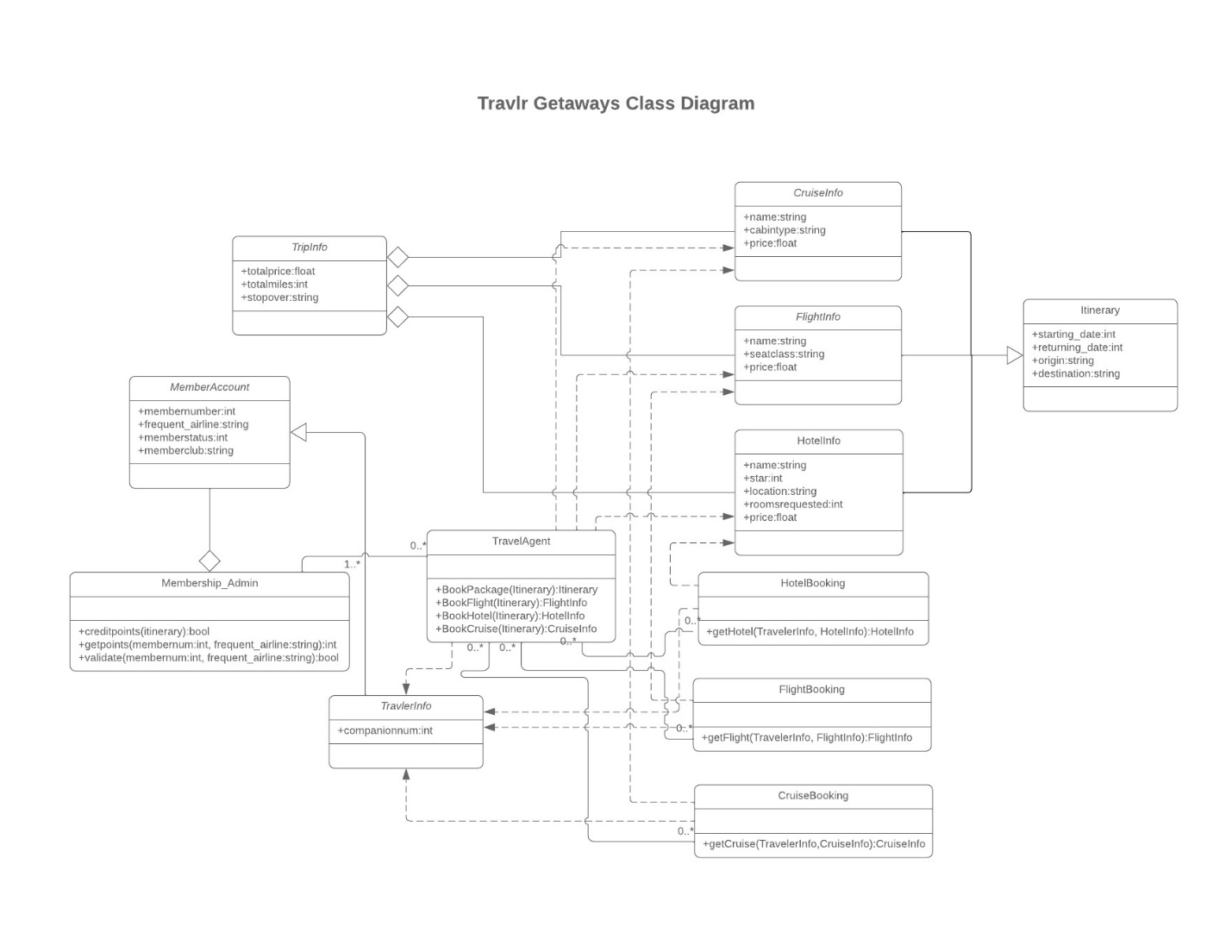
This application uses the MVC or Model View Controller design pattern to achieve a very clear separation of logic with in the application. The model would be the database component and where data is stored. The view is the Client component of the pattern, this is the area of the application only concerned with displaying the data to the user. The controller is the Server component of the system, this component is concerned with being the go between for the View and Model components. This is where the main logic is housed that requests data from the Model component and serves data to display to the View component. The manipulation of the data and how it should be sent to the View component is done in this area of the application.

### Sequence Diagram



Looking at this diagram I will go through the main flow of data within our web application. The flow starts with a user entering a specific route into the web address bar. This is where the browser picks everything up and send this to the controller. The route being requested gets data from the controller who then passes the request to the Http Client. The Http client sends an Http request to the controller function which includes the route requested. The controller/model calls to retrieve the data from the database that the request needs. The database (MongoDB) processes the request and returns the data back up the line. The controller/model passes a Json response back to the Http Client, which then sends the results to the controller. The controller feeds the proper data back to the browser/view/template to be render for the user to see.

## Class Diagram



Looking at this diagram I will explain the classes starting with the MemberAccount class.

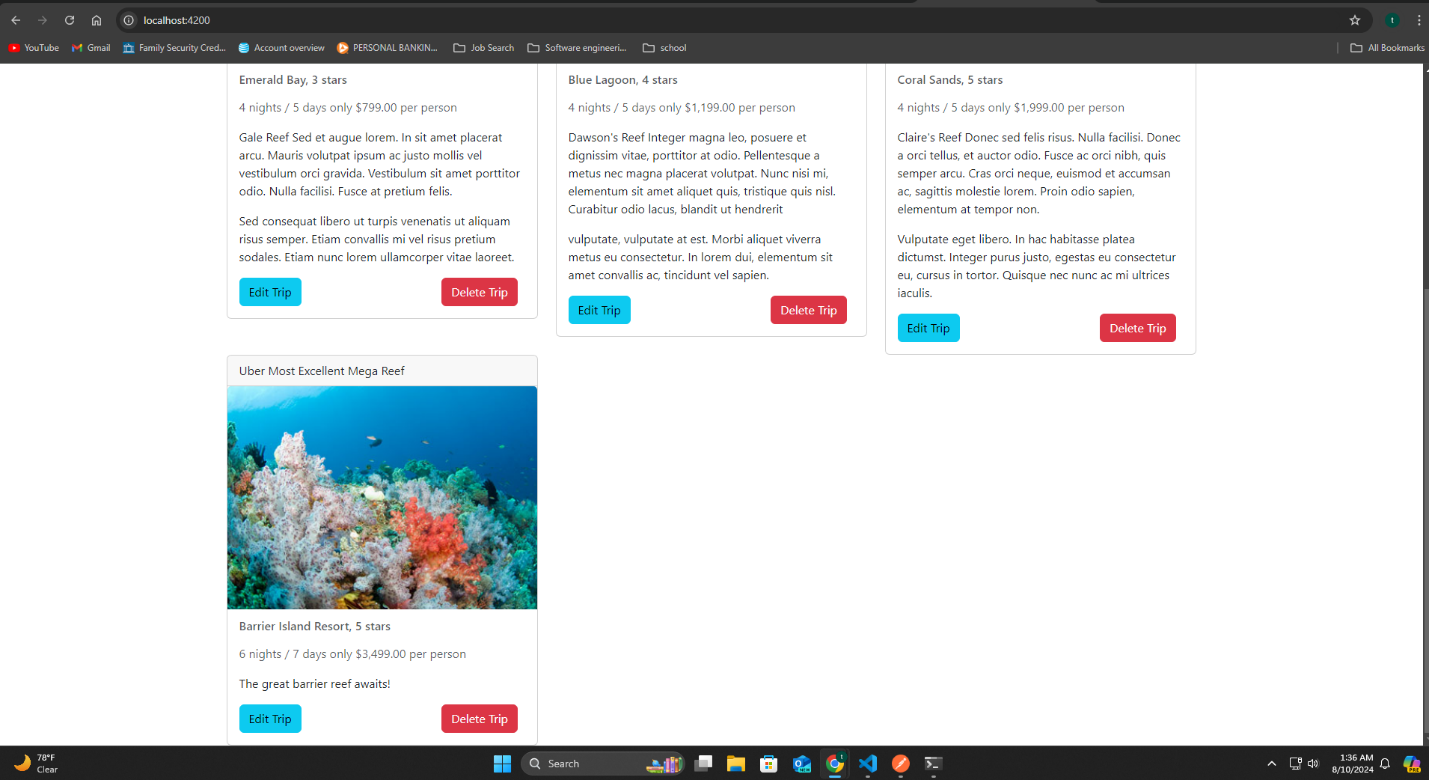
* MemberAccount – This is the class used for a member account, it contains public variables for member number, frequent airlines, member status, and member club. It is related to the Membership\_Admin class and the TravlerInfo class inherits from it.
* Membership\_Admin – This class is a helper class used to change or retrieve information for the MemberAccount class. It contains three public functions which are creditpoints(itinerary), getpoints(membernum, frequent\_airline), and validate(membernum, frequent\_airline). It has a relationship with the TravelAgent class so that it can use current trips to perform its functions.
* TravlerInfo – This class inherits from the MemberAccount class and also adds a public variable for companionnum. This class has dependencies with the TravelAgent, FlightBooking, HotelBooking, and CruiseBooking classes.
* TravelAgent – This class is a class that connects many others. It has 4 functions to book packages, book flights, book hotels, and book cruises. This will be the class that does a lot of the main work for the customers wanting to book a trip. It has relationships with Membership\_Admin to update accounts, HotelBooking, FlightBooking, and CruiseBooking. This class has dependencies with TravlerInfo, HotelInfo, FlightInfo, and CruiseInfo classes.
* HotelBooking – This class is a class for booking a hotel room. It has one public function to get hotel info. This class has a relationship with the TravelAgent class, and dependencies with the TravlerInfo class and the HotelInfo class.
* FlightBooking – This class is a class for booking a flight. It has one public function to get flight info. This class has a relationship with the TravelAgent class, and dependencies with the TravlerInfo class and the flightInfo class.
* CruiseBooking – This class is a class for booking a cruise. It has one public function to get cruise info. This class has a relationship with the TravelAgent class, and dependencies with the TravlerInfo class and the CruiseInfo class.
* TripInfo – This class is for overall trip info. This class gets data from the CruiseInfo, FlightInfo, and HotelInfo classes. It has three public variables for totalprice, totalmiles, and stopover.
* CruiseInfo – This class is where the chosen cruise information is held. It has three public variables, name, cabintype, and price. It passes data to the TripInfo class, and has dependencies with the TravelAgent, and CruiseBooking classes.
* FlightInfo – This class is where the chosen flight information is held. It has three public variables, name, seatclass, and price. It passes data to the TripInfo class, and has dependencies with the TravelAgent, and FlightBooking classes.
* HotelInfo – This class is where the chosen cruise information is held. It has five public variables, name, star, location, roomsrequested, and price. It passes data to the TripInfo class, and has dependencies with the TravelAgent, and HotelBooking classes.
* Itinerary – This is the class that holds the itinerary info from a lot of the classes in the system and provides the itinerary for the customer of their trip. It has four public variables, starting\_date, returning\_date, origin, and destination. This class inherits from the CruiseInfo, FlightInfo, and HotelInfo classes.

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

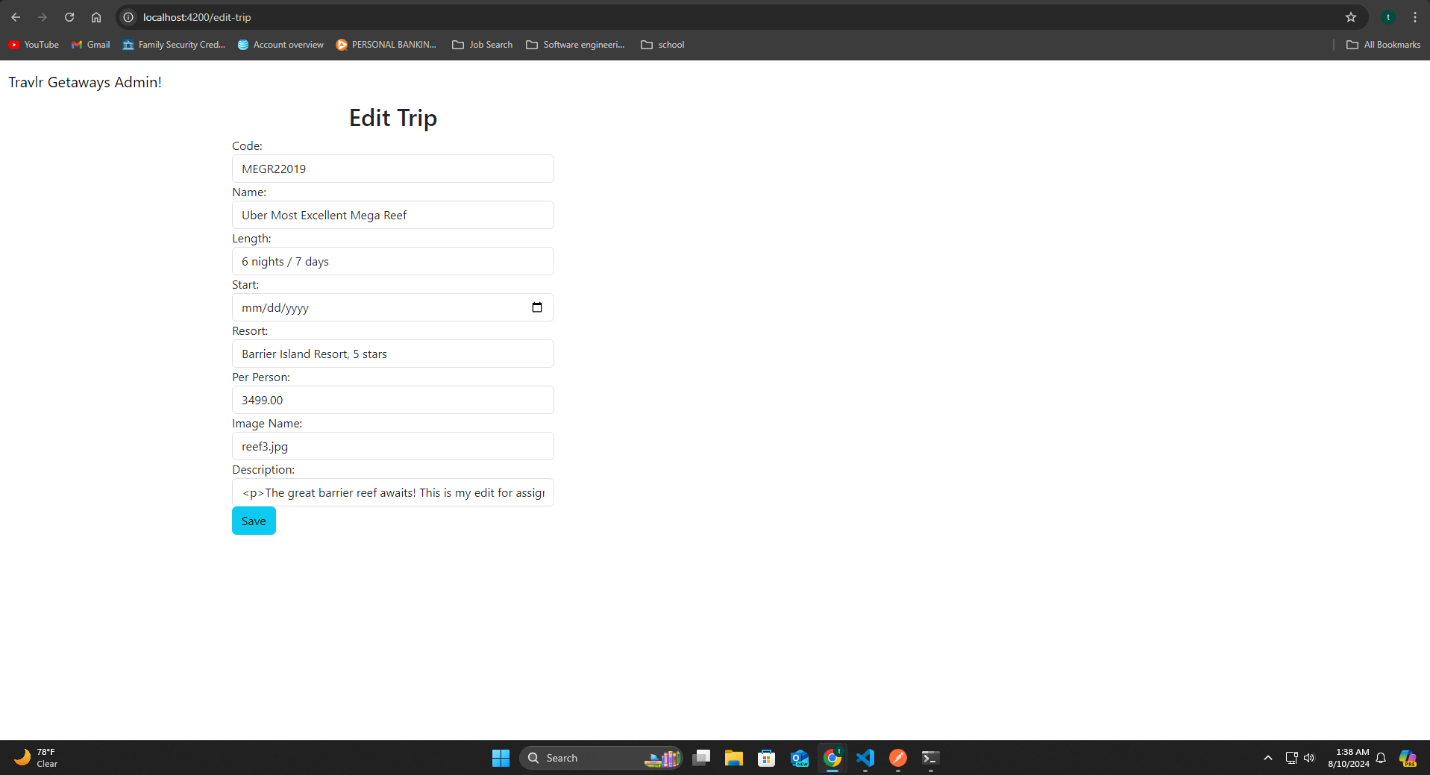
| **Method** | **Purpose** | **URL** | **Notes** |
| --- | --- | --- | --- |
| **POST** | <Create a new account> | </api/register> | <Using a name, email, and password, allows a user to create a new account to be stored in the database> |
| **POST** | <Login user> | <api/login> | <Takes in user info and verifies that they are a user and logs them in to the site> |
| **GET** | <Retrieve list of trips> | </api/trips> | <Returns list of all trips in database> |
| **GET** | <Retrieve single trip> | </api/trips/:tripCode> | <Returns single trip instance from the database, identified by the tripCode passed on the request URL> |
| **PUT** | <Edit a trips info> | </api/trips/:tripCode> | <Returns selected trip in a form to be edited> |
| **Delete** | <Delete selected Trip> | </api/trips/:tripCode> | <Returns the deleted trip, and removes this trip from the database> |

## The User Interface

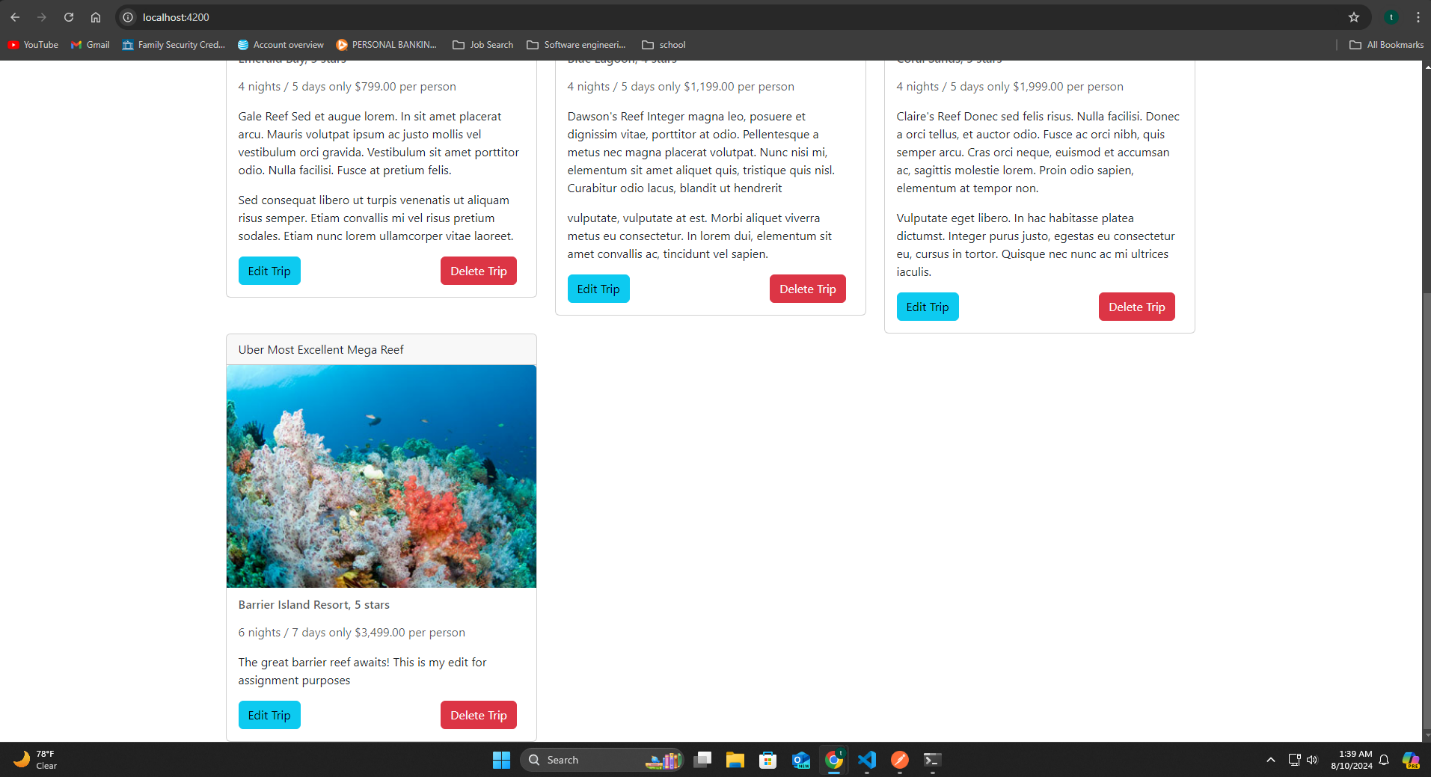
**Added Trip:**



**Edit Screen:**



**Updated trip:**



The Angular portion of the project focuses on having components for each task that needs done in the app. This is a main part of the Angular side of the project. As the whole application is loaded upfront unlike the Express side of the project, this means that components can easily be navigated without a whole reload of the page and allows for a fast responsive user experience. This is very different from the Express portion of the project as with each navigation action the site requests the next page from the server and reloads the site which can make the user feel the application is slower than it should be. The Angular side of the project is the admin site. To use this site, you must create an account an be logged in. Once you are logged in this site allows the user to add trips to the database, edit trips information, and delete whole trips when needed. The main way that I tested these functions was using Postman. Using Postman, you can check that your API endpoints are working correctly by calling the route in Postman Using GET, POST, PUT, or DELETE. You can also add parameters to your API call if needed with Postman. When doing this you will receive the output of your route and be able to confirm that the output is correct. Another helpful tool for testing was MongoDB Compass. This application allows the user to see the databases and their contents which can help paired with Postman to ensure your API calls are actually modifying the database as well as providing the correct output. The last method for testing that I used was using the site how it is intended as I were a user to ensure a great user experience free of bugs.