

Travlr Getaways

# **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 | 11/17/24 | Alex Schmierbach | Completed the following: Executive Summary, Design Constraints, and System Architecture View |
| 1.1 | 11/30/24 | Alex Schmierbach | Completed the following: Sequence Diagram, Class Diagram, and the API Endpoints table |
| 1.2 | 12/21/24 | Alex Schmierbach | Completed the following: User Interface |

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

## The application will be developed by utilizing a framework stack known as MEAN, which stands for MongoDB, Express.js, Angular, and Node.js. The reason such a framework is chosen is due to its ability to scale, remain flexible, and have tremendous efficiency in handling both the admin panel page along with the user and customer-facing web pages. The side that users will utilize is responsible for browsing and booking travel packages. While the admin panel will give authorized users the ability to manage the bookings, update any packages needed, and follow through with any problems that users run into. An approach such as this allows for the ability to give the users and admins a seamless experience.

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

There will be multiple constraints with the development of the Travlr Getaways application, and they are the following:

1. Scalability: The application must be capable of allowing thousands of users to be active at once. This capability requires that database queries be both efficient, but also optimized specifically for peak performance in the server.
2. Security: The data that is to be used is incredibly sensitive information relating to various user data, which includes payment information. In regards, to the sensitive nature it is incredibly important for this information to be not only stored but also transmitted through a robust encryption and authentication pipeline.
3. Responsiveness: In regards, to the performance of the application it must allow for a consistent experience across the board through a multitude of different devices and screen sizes ultimately relating to a seamless and comfortable experience for the end user.
4. Integration: Overall, the system should allow for the ability to plug in various third-party possibilities. Without this type of adoption, it would be hard to maintain some scalability with various aspects such as payment processing and travel package data.

With regard to these various constraints, it is pretty clear that they will have a significant outcome on the development process. It will turn out to be vital for the application to go through a careful planning and implementation process for scalable database schemas along with up to date and secure coding practices while also maintaining design processes that remain responsive. In addition to the following the ability to make integrations with third-party services must be reliable.

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

### Component Diagram



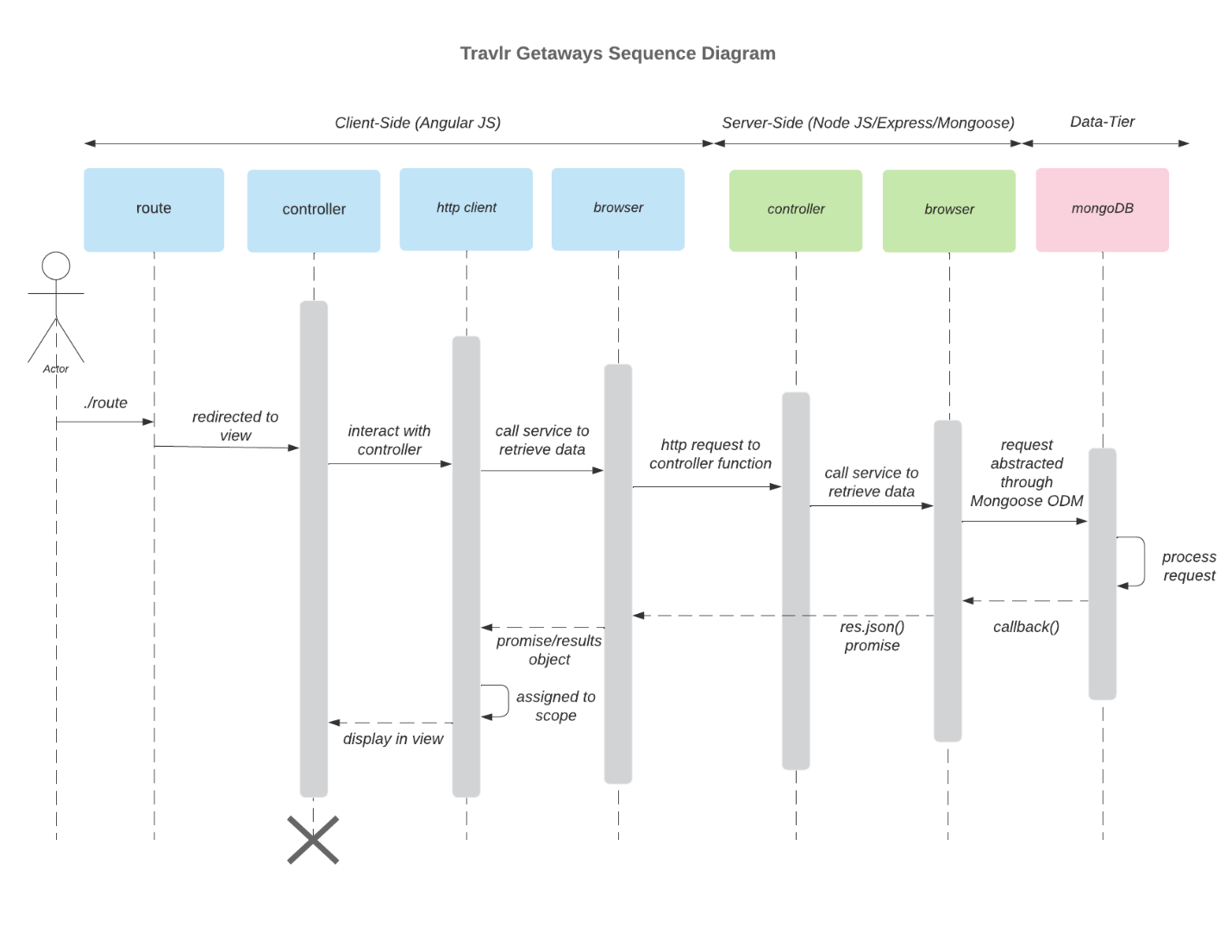
A text version of the component diagram is available: [CS 465 Full Stack Component Diagram Text Version](https://learn.snhu.edu/d2l/lor/viewer/view.d2l?ou=6606&loIdentId=24342).

### The system architecture of the Travlr Getaways application is accurately depicted in the component diagram above. The following are the significant components contained in said diagram.

1. Client-Side Application: This component handles the user interface for both the admin users and customers of the application, which will be built utilizing Angular.
2. Server-Side Application: This component handles API requests, logic that relates to business requirements, and communication with the database and through development with Node.js and Express.js.
3. Database: This component takes care of all information required to be stored such as: user data, travel packages, and booking information.
4. Authentication Service: This component takes care of the user authentication process along with its authorization too.
5. Payment Gateway: This component will adapt and integrate with a third-party service of some sort to handle the payment processing for the application.

### These various components do a great job at interacting with each other in addition to providing an easily understood, performant, secure, and seamless experience for both the authorized users that deal with admin tasks, and the general users that use the application ensuring both efficient management of data and allowing for the admin team to make real-time updates.

### Sequence Diagram

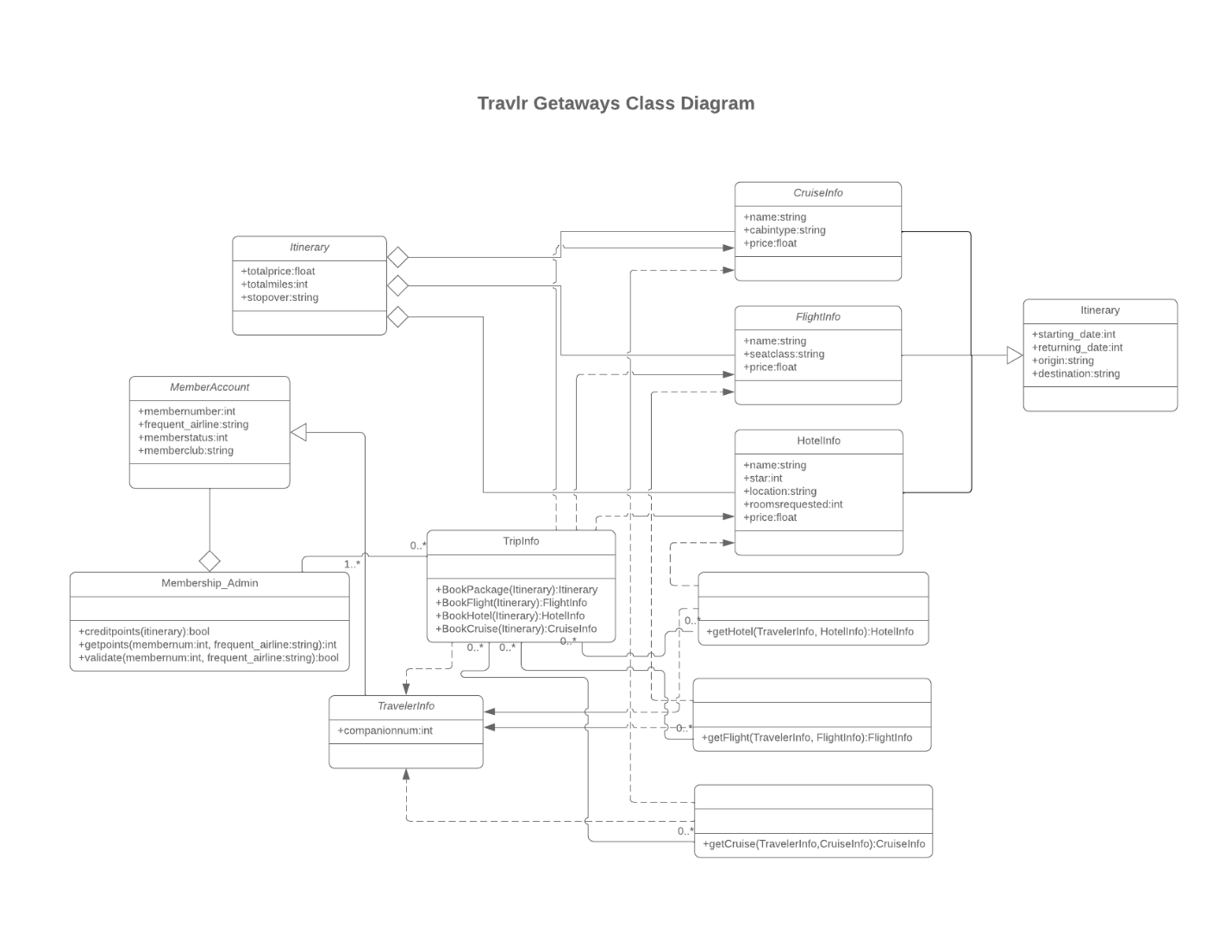


The following sequence diagram deals with several components that work together for the web application. Firstly, the actor which is the user interacts with the application which starts the process. Secondly, this is known as a route which is the entry point which routes the users request to the corresponding and correct controller. Thirdly, the controller deals with handling the logic for the users request such as interacting with various services or obtaining information from the database via a query. Fourth, there are various services which perform specific and concise tasks, for example, simply retrieving data from the database. Fifthly, we have the database layer which is where data is stored, retrieved, or manipulated by either essential tasks or through user input. The sequence that a user might go through would look like the following:

1. The user initiates a request that is correctly routed to the corresponding view or template.
2. Said view interacts with the controller thus making a HTTP request to a specific service.
3. Said service retrieves information that is required and may query the database through MongoDB while utilizing an Object-Document Mapper or ODM like Mongoose.
4. Lastly, the data from the request is processed and sent back to the user through the service to the controller and then displayed in front of the user.

Overall, the above diagram signifies the process in which a user’s interaction may take when interacting with the application to retrieve and view data. A sequence like this is important because it allows for a user’s request to be handled with efficiency and with prompt response.

## Class Diagram

  
Itinerary – Top Left: This class is the representation of the entire travel plan including the total price and miles along with the number of stopovers or layovers.

MemberAccount: This class is the representation for information relating to a member’s data including their ID number, status, and affiliation.

Membership\_Admin: This class is the representation for administrative tasks relating to members including validating their membership and the management of a member’s points.

TripInfo: This class is the representation that has the responsibility for management with trip information and the booking processes dealing with various components like flights, cruises, hotels, and packages.

CruiseInfo: This class is the representation for specific details having to deal with cruises such as the name, type of the cabin, and the price point.

FlightInfo: This class is the representation for specific details having to deal with flights such as the name, seat class, and the price point.

HotelInfo: This class is the representation of specific details having to deal with hotels such as the name, their star rating, location, requested rooms, and the price point.

Itinerary – Top Right: This class is the representation of a different part of the itinerary having to deal with departure and arrival dates including the departure location and arrival location.

TravelerInfo: This class is the representation of the information having to do with travelers and the number of people traveling.

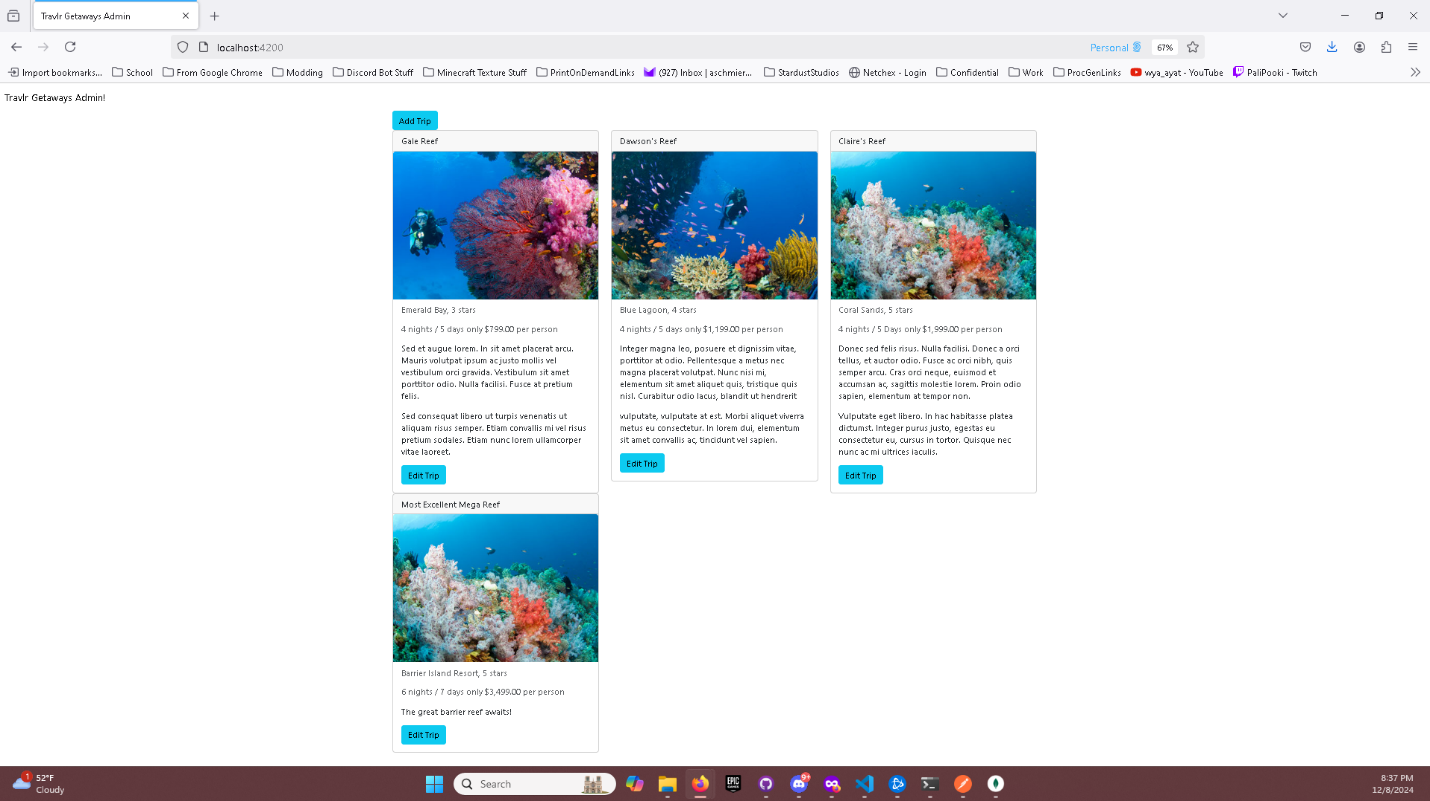
Methods: There are three methods having to deal with obtaining information regarding cruises, flights, or hotels based around a specific traveler’s data.

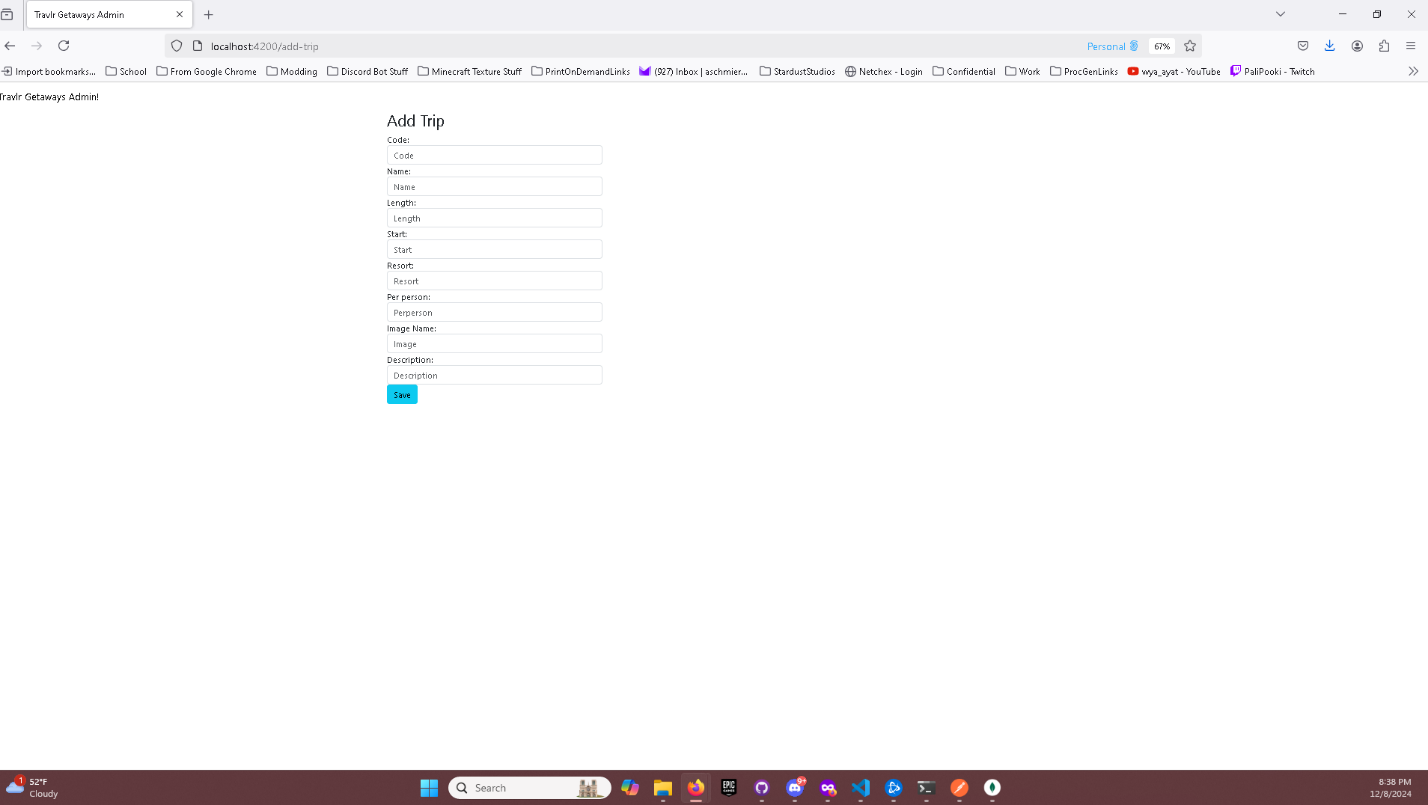
## [API](#_heading=h.2jxsxqh) Endpoints

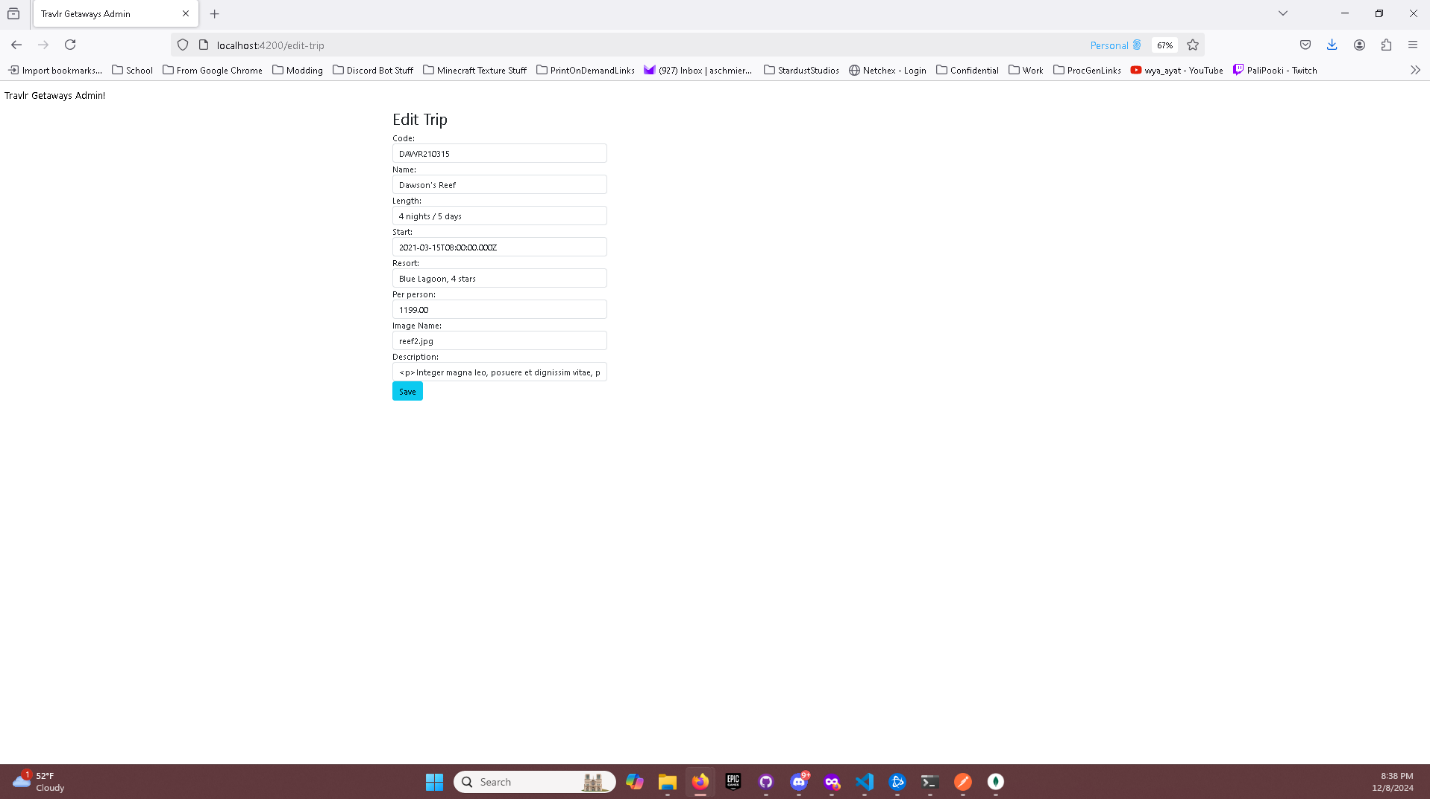
<Exposing RESTful endpoints is a design approach to enable an application to participate in a larger ecosystem. Document each endpoint in the table below, including the HTTP method, purpose, URL, and notes.>

| **Method** | **Purpose** | **URL** | **Notes** |
| --- | --- | --- | --- |
| **GET** | Retrieve list of trips | /api/trips | Returns all current data for related trips |
| **GET** | Retrieve a single trip | /api/trips/:tripCode | Returns a single trip related data identified by the trip ID passed on the request URL |

## The User Interface







<Summarize the Angular project structure and how it compares to the Express project structure. Be sure to describe the rich functionality provided by the SPA compared to a simple web application interaction. Describe the process of testing to make sure the SPA is working with the API to GET and PUT data in the database.>

The angular project structure involves the following: modules, components, services, routing, assets, and environments. Modules group components that are related to components, services and similar other functionalities. Components are made up of the UI and behavior for these specific parts, are controlled by HTML templates, CSS, and TypeScript classes. Services are responsible for the business logic side of things along with data handling and API calls. These are usually injected within components utilizing Angular’s dependency injection tools. Routing oversees managing the navigation within the SPA or Single Page Application which enables transitions between specific views without the need for reloading the associated pages. Assets are just a folder in the directory for holding stylesheets, images, or just various static resources. Environments are utilized mainly for specific development and production configurations. The express project structure involves the following: routes, controllers, models, middleware, views(optional), and static files. Routes are just files that define API endpoints for handling the HTTP requirements such as “/api/” products. Controllers are utilized for fetching or saving data to databases that hold functions for requesting these parts of logic. Models are utilized for representing data structures and schemas often requiring use of ORM libraries like Mongoose. Middleware is utilized with functions to handle authentication, error handling, logging, etc. Views which are optional determine how the app serves HTML or if templates are used like Handlebars. Static files are like resources within Angular and related to images or stylesheets.

When comparing a SPA or Single Page Application versus a Simple Web App we have rich functionality within SPA described below. SPAs like Angular load a single HTML file that allows for dynamic updating of a single view using JavaScript. Thus, eliminating the need for page reloading and providing smoother user interaction along with making this interaction faster. Angular also handles the routing needed on the client side thus improving performance and the overall user experience. The data that is exchanged between front-end and back-end, which is the API, happens all seamlessly via asynchronous calls from the HttpClient within Angular. On the flip side simple web applications which are built using server-side rendering. The users’ interactions usually involve a HTTP request along with a complete page reload. While these are easier to set up, they lack the efficiency and interactivity that come with SPAs.

Testing has to do with using Postman to determine whether the API calls are working correctly. Along with utilizing MongoDB compass to make sure database documents are working as intended. The following steps involved the testing process: Setup Mock API or Test Environment, Check GET Request, Check PUT requests, and then integration testing. Setting up a mock API or Test Environment involves using tools like Postman or a mock API server to verify that the endpoints work independently. Checking GET requests ensures that the Angular service fetches the correct data from the API and then correctly displays it to the components. Checking PUT requests ensures that form submissions or verified or updates based upon the correct sent data to API and reflects those changes within view only upon a successful response. Integration testing is a combination of back-end and front-end within a test environment to confirm communication between the SPA and API is smooth.