

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 | 01/27/2024 | Cody Faircloth | Rapid Prototype of a MEAN stack web application |

## 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)

The Travlr Getaways web application will be built using a MongoDB, Express, Angular, and Node.js (MEAN) stack. The application will be comprised of two portions. The customer-facing side will be a static portion displaying content for the website. The administrator portion of the application will be a Single Page Application (SPA). Node.js is utilized as the application runtime that the entire web application will run on.

The customer-facing side of the web application utilizes Express, Handlebars, and MongoDB. Express is a framework for Node.js that serves static content to the user. MongoDB is used with Handlebars to store and generate the static content of the web application. Handlebars is a templating language that will be used to create dynamic content to populate the webpage. MongoDB is a NoSQL database that will be used to store the static content to be displayed dynamically through Handlebars templates. Working together, Handlebars will iterate through the MongoDB database to generate dynamic content for pages being served by Express.

The administrator portion of the web application will be a SPA. The SPA will be created using Angular. This part of the web application will allow administrators to perform various administrative functions such as managing the web application's content. This includes adding, updating, and removing static content displayed on the customer-facing side of the web application. This will be implemented by allowing documents in the MongoDB databases to be updated.

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

The web application must be easily updatable. When updates are committed, the changes should not require a redeployment of the website. The web application must be extremely responsive. It should also provide an excellent user experience that accommodates various device sizes. The administrator side of the application must allow for the static content being displayed on the customer-facing side to be easily added, updated, and deleted. This will allow for easy web application management.

There are several design constraints associated with portions of the MEAN stack. MongoDB is susceptible to performance issues when the dataset becomes too large. Express, Angular, and Node.js when used in the MEAN stack can increase the complexity of the codebase leading to increased difficulty in post-launch management. SPAs are typically difficult to make crawlable for search engines. However, this should be mitigated since the administrator portion of the web application is the only section utilizing a SPA.

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

### Component Diagram



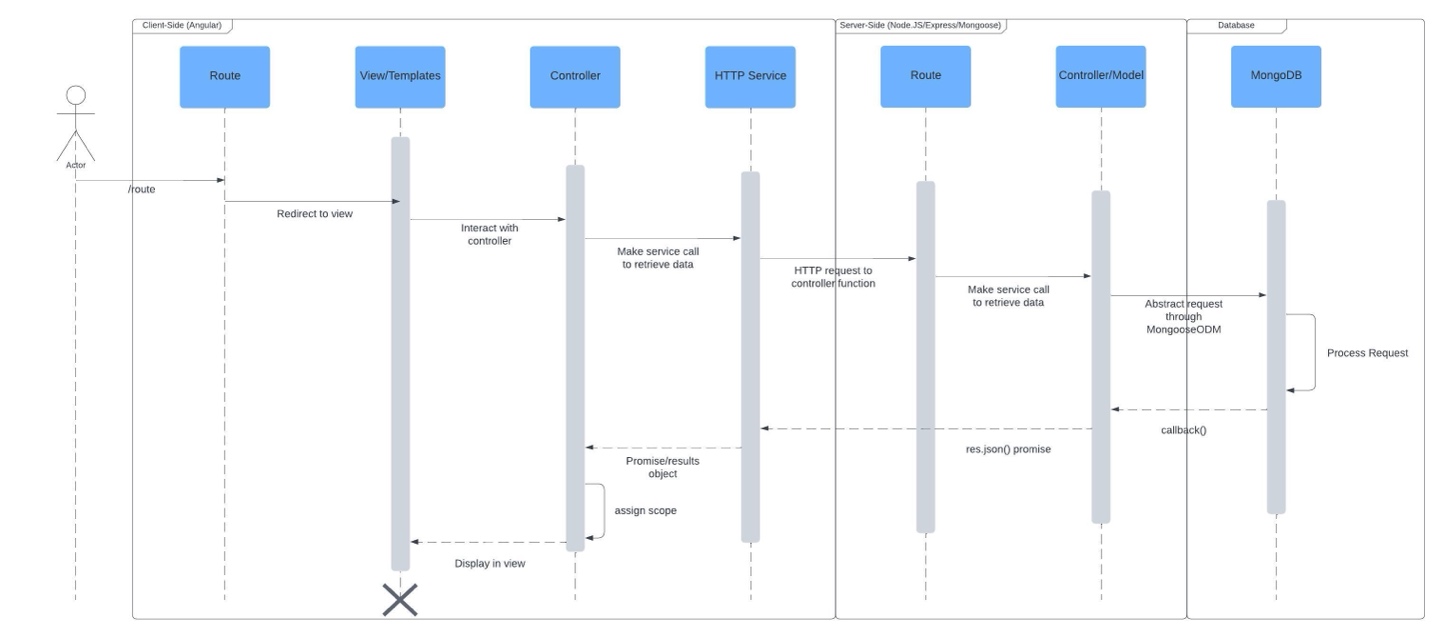
There are three components of the Travlr Getaways web application. These components are the client component, the server component, and the database component. Each major component and its respective subcomponents will be discussed in detail in the following paragraphs.

The client component contains four subcomponents: the client session, the web browser, the traveler portfolio, and the graphic library. The web browser and graphic library provide an interface for the client component of the application. The web browser is required for use as an interface with the client session and traveler portfolio. The traveler portfolio also requires the graphic library as an interface. The client session connects with the server component through port interactions. The traveler portfolio interacts with the database to display information.

The server component contains four subcomponents: the authentication server, the server session, the traveler database, and Mongoose ODM. The server session and Mongoose ODM provide an interface for the server component of the application. The traveler database requires the server session interface. The server session requires the Mongoose ODM interface. Mongoose ODM requires a connection to the database component. The server session and authentication server components interact with a port on the server to provide an interface.

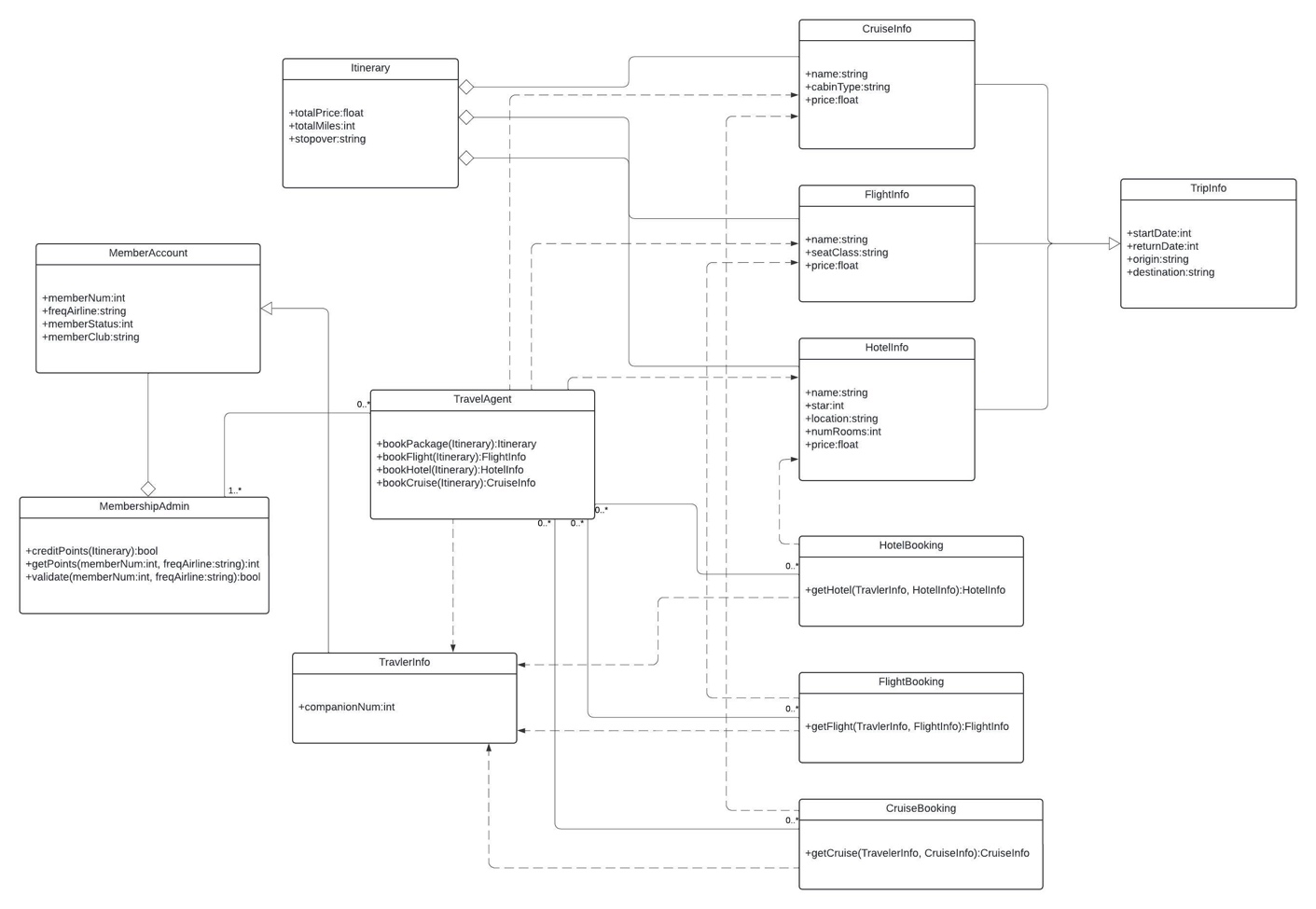
The database component consists of MongoDB. MongoDB provides an interface to the server and client components, providing data for both components.

### Sequence Diagram



In this sequence diagram every request begins with the user, represented by the actor. The user enters a route such as /trips which is directed to the corresponding view or template through the front-end router. The view will call the controller to populate the template with corresponding data to be rendered for display. The controller utilizes HTTP services to retrieve data through requests. The HTTP services utilizes the API to connect the front-end and back-end through API function calls for specific routes. The back-end router receives the request and calls the corresponding back-end controller based on the route. The back-end controller then utilizes Mongoose to retrieve data from the database element, MongoDB. MongoDB processes the request and returns the data to the back-end controller. The backend controller then passes this data directly to the HTTP service. The HTTP service then passes the data to the controller which determines scope before passing the data to the template. The view then renders the received data for display to the user.

## Class Diagram



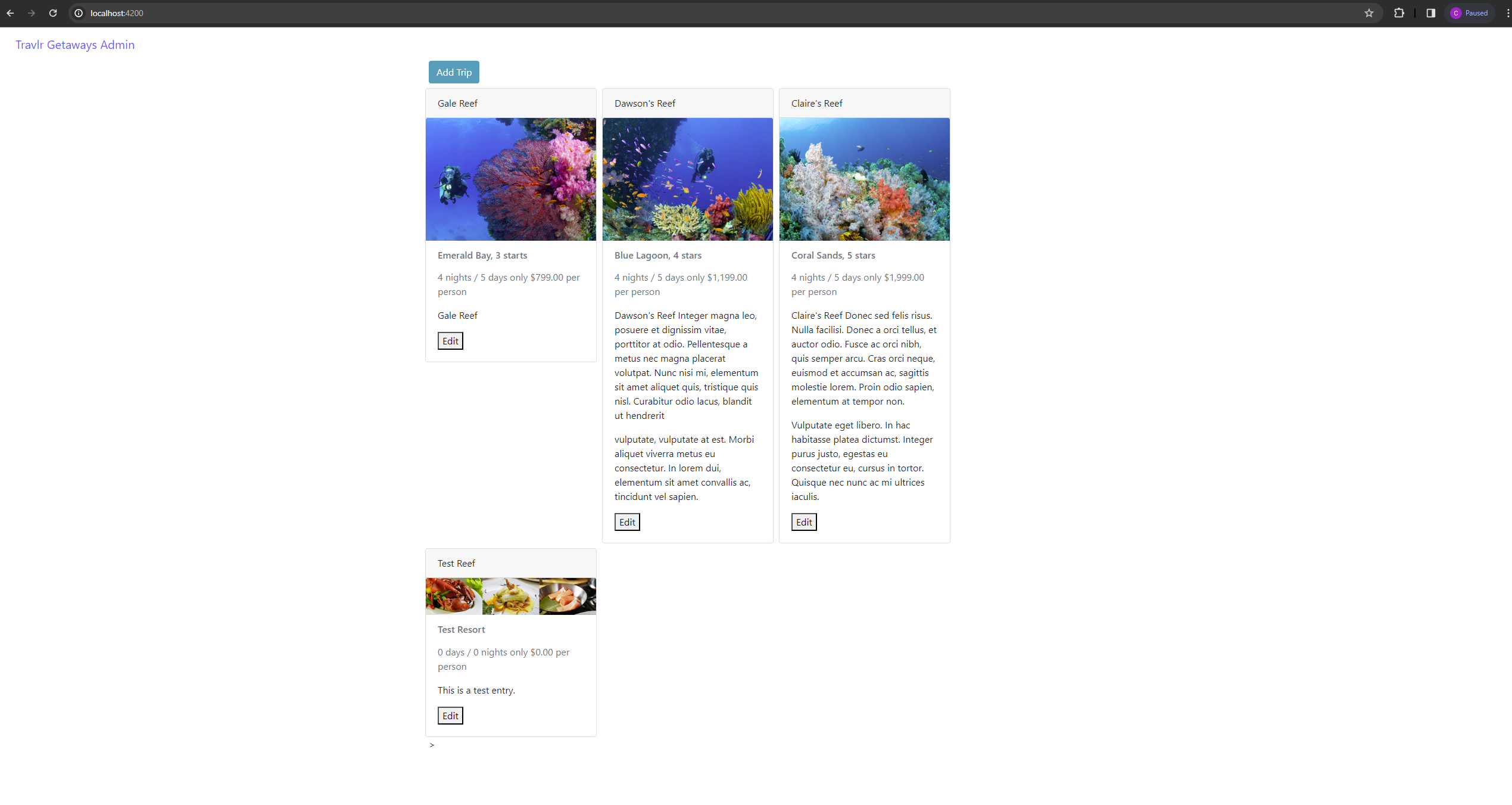
The CruiseInfo, FlightInfo, and HotelInfo classes each contain a string variable to store the name and several class variables to store data relevant to the cruise, flight, or hotel. Each of these classes inherit from the TripInfo class which contains variables to store the start date, return date, origin, and location of the trip. These variables are applicable to the child classes. CruiseBooking, FlightBooking, and HotelBooking have associations with CruiseInfo, FlightInfo, and HotelInfo respectively. Each booking class also has an association with the TravelerInfo class. Additionally, each booking class has a zero to many multiplicities with the TravelAgent class. The TravelAgent class has associations with each Info class including the TravelerInfo class. The TravelAgent class also includes a one to many multiplicity with the MembershipAdmin class. The TravelerInfo class inherits from the MemberAccount class. The MembershipAdmin class aggregates from the MemberAccount class. The Itinerary class aggregates from the CruiseInfo, FlightInfo, and HotelInfo classes.

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

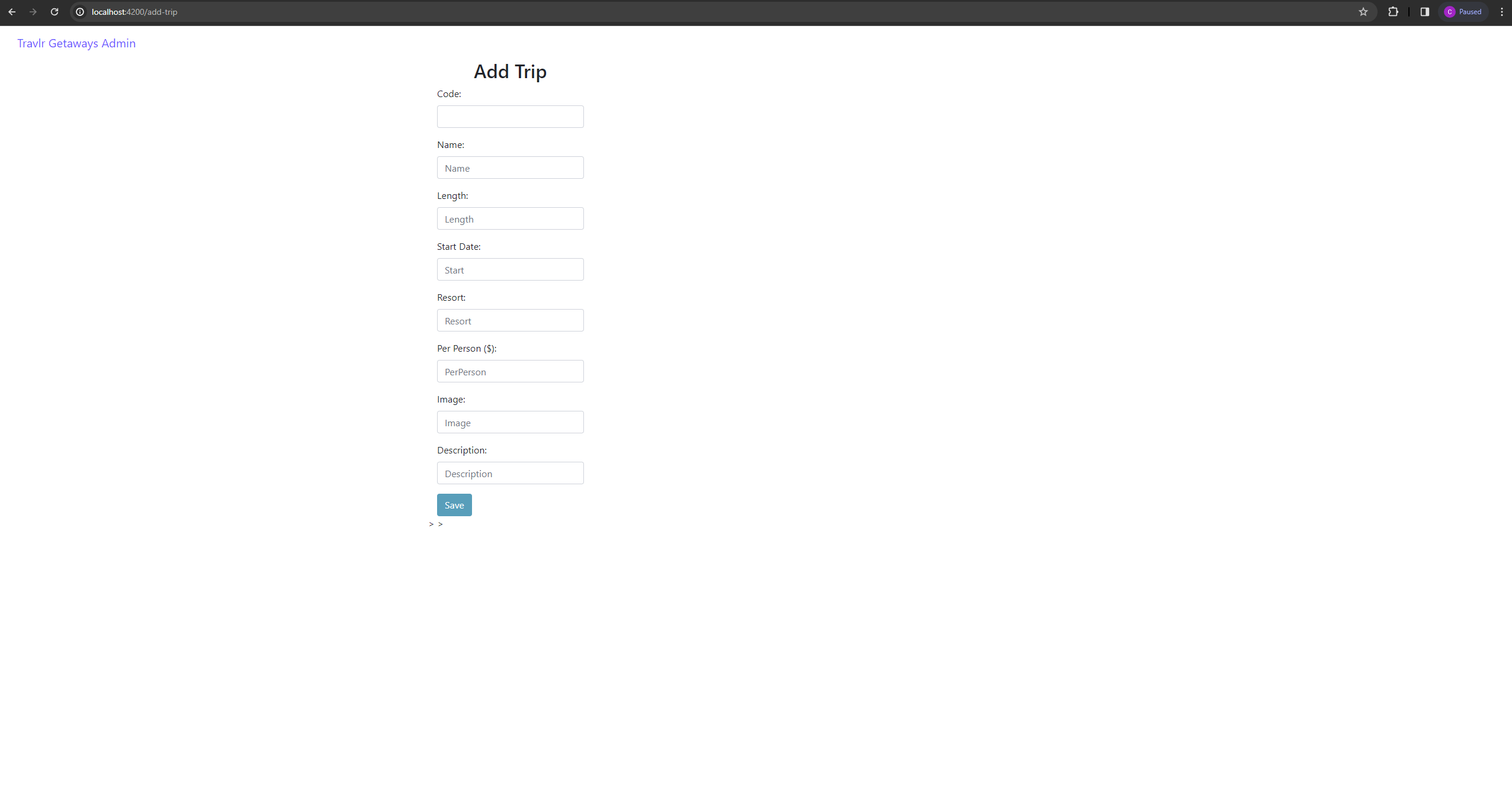
| **Method** | **Purpose** | **URL** | **Notes** |
| --- | --- | --- | --- |
| **POST** | Login a user | /api/login | Authenticates a user, returning a JWT |
| **POST** | Register a user | /api/register | Adds new user to the database, returning a JWT |
| **GET** | Retrieve a list of meals | /api/meals | Returns all meals |
| **GET** | Retrieve a single meal | /api/meals/:mealCode | Returns a single meal identified by the meal code value at the end of the URL |
| **GET** | Retrieve a list of news | /api/news | Returns all news articles |
| **GET** | Retrieve a single news article | /api/news/:newsCode | Returns a single news article identified by the news code at the end of the URL |
| **GET** | Retrieve a list of rooms | /api/rooms | Returns all rooms |
| **GET** | Retrieve a single room | /api/rooms/:newsCode | Returns a single room identified by the room code at the end of the URL |
| **GET** | Retrieve a list of trips | /api/trips | Returns all trips |
| **GET** | Retrieve a single trip | /api/trips/:tripCode | Returns a single trip identified by the trip code at the end of the URL |
| **POST** | Add a trip | /api/trips | Add a new trip to the database |
| **PUT** | Update a single trip | /api/trips/:tripCode | Updates a single trip identified by the trip code at the end of the URL |
| **DELETE** | Delete a single trip | /api/trips/:tripCode | Deletes a single trip identified by the trip code at the end of the URL |

## The User Interface

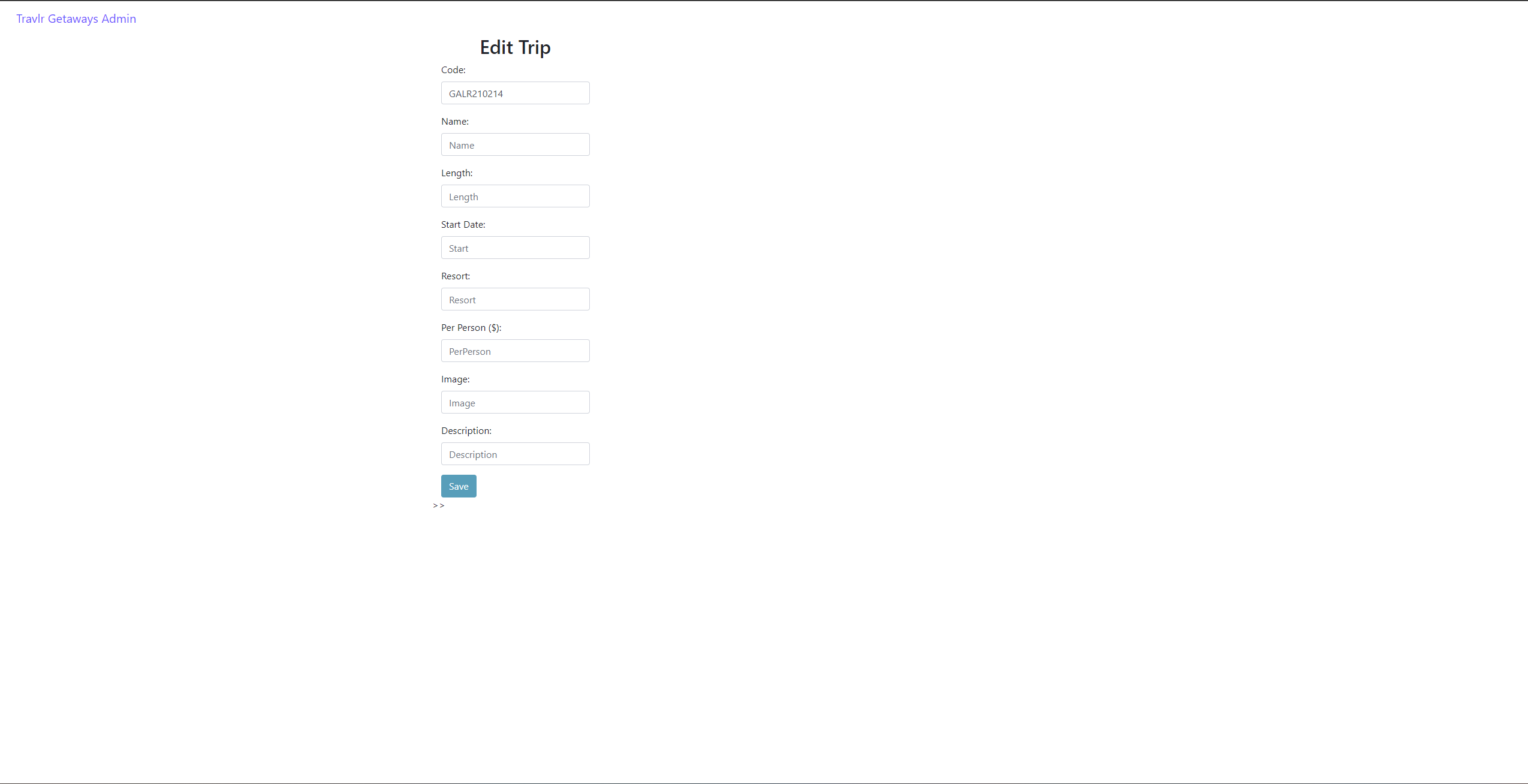
Unique Trip:



Add Trip



Edit Trip:



The structure of the Angular project is a single-page application (SPA). SPA's serve dynamic content by providing reusable content, data binding, and dependency injections. The structure of the Express project is the server-side component of the web application. This portion of the web application utilizes a RESTful API to handle HTTP requests.

SPAs provide several advantages such as enhanced user experience, improved speed, and cross-platform compatibility. However, SPAs often have an increased initial loading time, search engine optimization (SEO) issues, and browser history limitations. Some additional functionality associated with SPAs include a quicker development time, easier debugging, and the potential of a mobile-friendly web application.

The process of testing a SPA requires hosting two separate portions of the web application simultaneously. The Angular application and the Express application must both be hosted to ensure the API allows for communication between each section of the application. Additionally, information is displayed in the console to indicate when components and services are used or when certain HTTP requests are called.