

Travlr Getaways

# **CS 465 Project Software Design Document**

Version 3.0

## Table of Contents

[**CS 465 Project Software Design Document** 1](#_heading=h.gjdgxs)

[Table of Contents 2](#_heading=h.30j0zll)

[Document Revision History 2](#_heading=h.1fob9te)

[Instructions 2](#_heading=h.1fob9te)

[Executive Summary 3](#_heading=h.3znysh7)

[Design Constraints 3](#_heading=h.2et92p0)

[System Architecture View 3](#_heading=h.tyjcwt)

[Component Diagram 3](#_heading=h.3dy6vkm)

[Sequence Diagram 4](#_heading=h.4d34og8)

[Class Diagram 4](#_heading=h.2s8eyo1)

[API Endpoints 4](#_heading=h.17dp8vu)

[The User Interface 4](#_heading=h.4h5b3qv007h8)

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

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 02/01/25 | Brady Weber | Updated the Executive Summary, Design Constraints, and the Component Diagram description. |
| 2.0 | 02/23/25 | Brady Weber | Updated the Component Diagram, Sequence Diagram, and API Endpoints |
| 3.0 | 02/23/25 | Brady Weber | Updated the User Interface section. |

## 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)

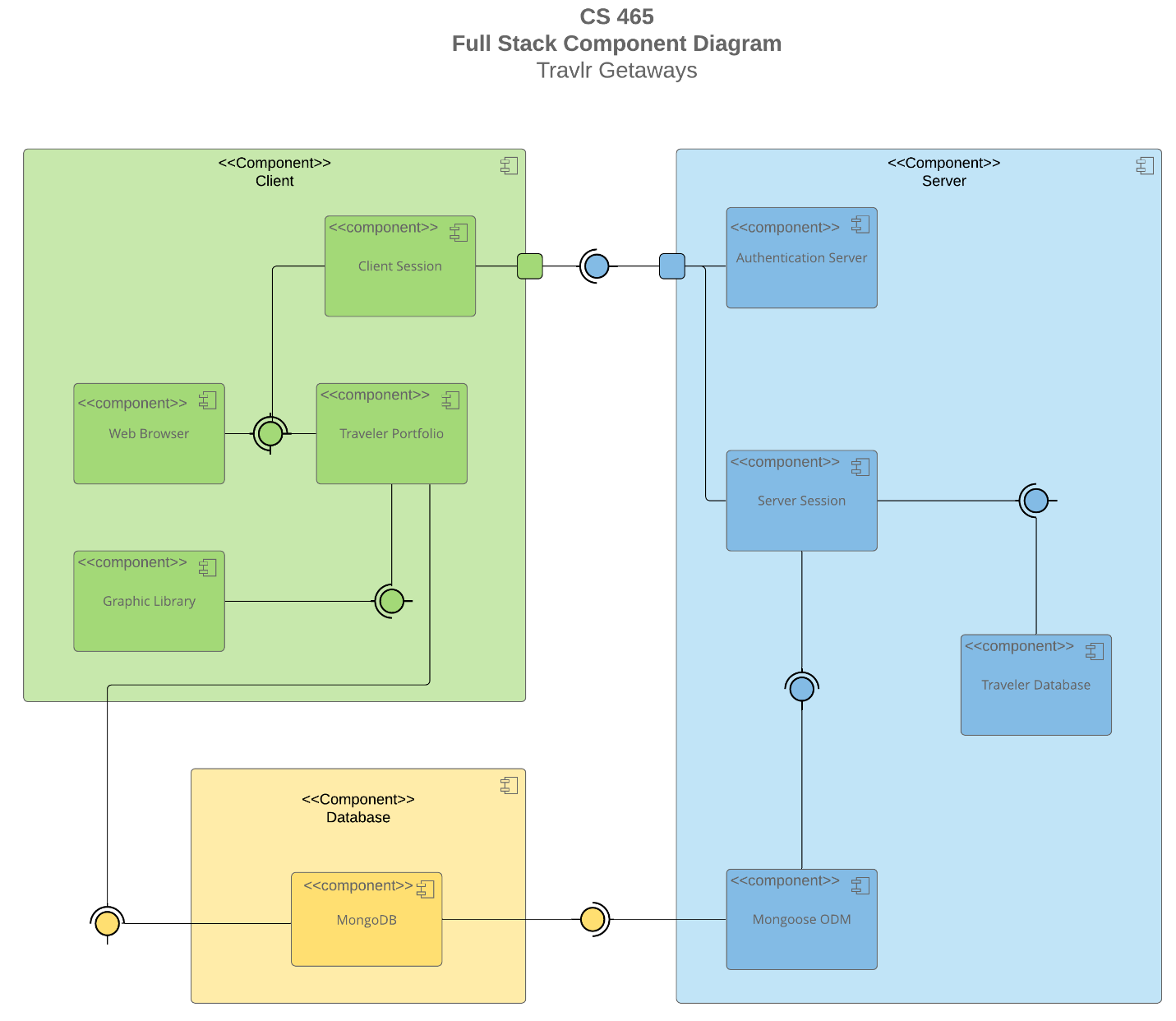
The client, Travlr Getaways, has contracted us to build them a web-based application that allows their customers to browse and book vacation packages. In order to complete this task, we will be creating an application that is built utilizing the MEAN stack. MEAN stands for MongoDB, Express, Angular, and Node.js. These tools will be used to build the customer-facing application, the client-facing admin single-page application, and the NoSQL database to house the information. The customer interface will be a dynamic web page that allows customers to login, view different vacation packages, read the latest news, and book vacations; it will be designed around the wireframe that the client has provided. The admin single-page application allows the client to perform necessary admin activities such as managing users and updating relevant client-facing information i.e. vacation packages and news. MongoDB will be used to store all of the different user information to allow the application to scale appropriately with the client’s business.

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

* Single-Page Application (SPA): the admin page will be designed as a SPA which changes how it will be developed as well as the different security measures that will be implemented to better protect it.
* Authorization and Authentication: The application will feature both customer and client services. Therefore, it will be necessary to implement authorization during development to ensure that users can only access relevant information. The login page will also need to utilize proper authentication methods in order for authorization methods to be accurately implemented.
* MEAN stack: The stack itself limits what libraries and tools can be implemented due to compatibility issues. MongoDB is a NoSQL database which impacts how data will be held and modified.
* Scalable: The application must be able to scale with the client’s user base.
* Security/Legal: The application will hold sensitive user data such as credit card information. This impacts the required security and guidelines that must be followed in order to protect it.
* Accessibility: The application should be able to run on multiple different devices and browsers.

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

### Component Diagram



**Database:**

* MongoDB - the NoSQL Database that stores the data for the application.

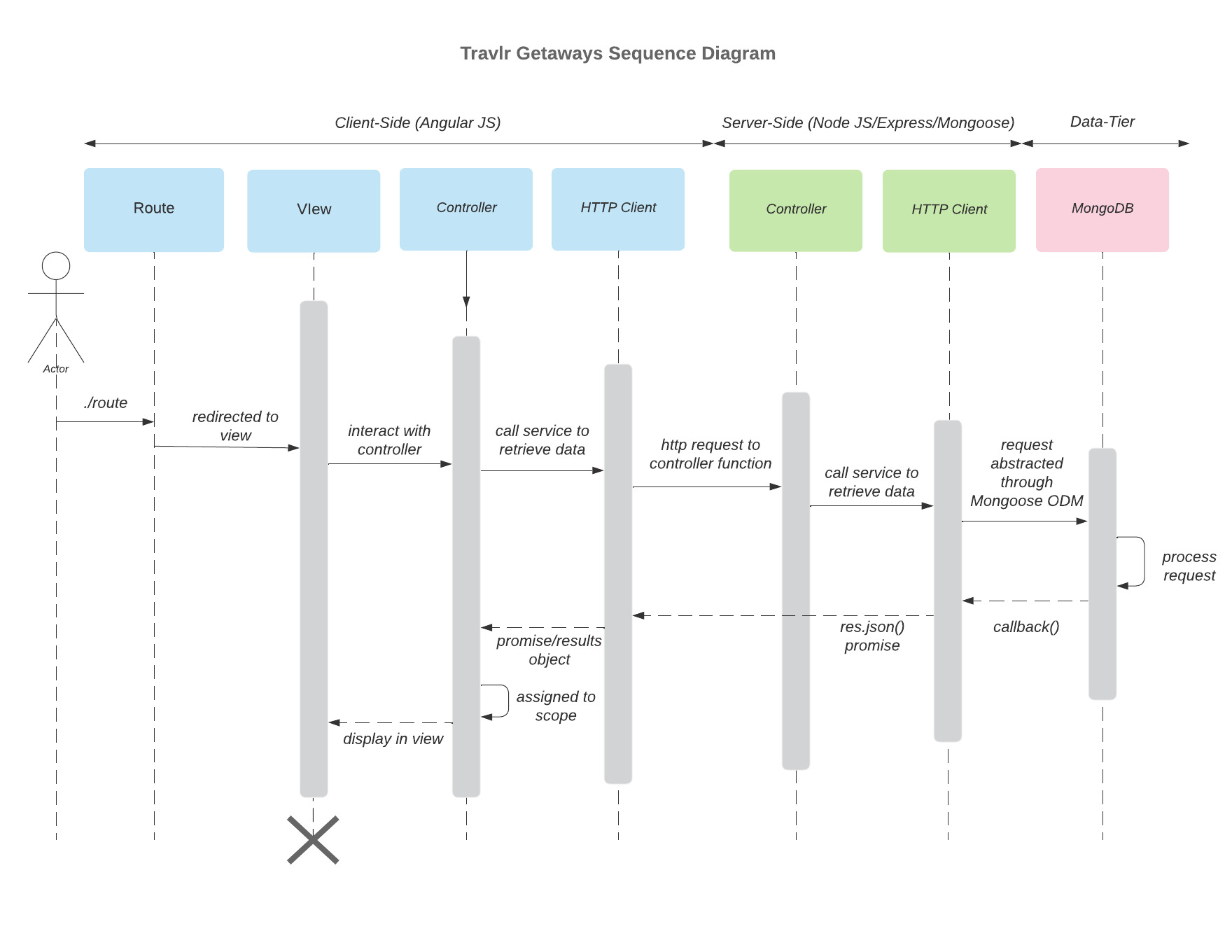
**Server:**

* Mongoose ODM - simplifies the interactions with MongoDB.
* Server Session - manages the backend API interactions and application logic.
* Authentication Server - validates the user is who they claim to be.
* Traveler Database - stores any immediate information to be displayed by the client or accessed by the user.

**Client:**

* Client Session - manages the data sent and received to the server and what to do with the data that is received.
* Web Browser - displays the customer-facing and client-facing web application interfaces.
* Traveler Portfolio - stores user data that is specific to the current user.
* Graphic Library - Stores the needed images to be displayed to the user.

**Sequence Diagram**



For the process of logging in, the user can access the browser view via the specific route /login. Once the view has been given, the user can interact with the page and click the sign in button. The controller then processes and handles this request in order to call the necessary HTTP Client method. The HTTP Client method then takes the information passed by the client-side controller and connects it with the necessary server-side controller; in this case the authentication step. This server-side controller, authentication, handles the logic that is necessary for processing the information and passing it through to the HTTP Client that interacts between the server and the database. The HTTP client then calls the database, in this case, via GET, to find a user with the given credentials. The database then passes the user information back to the server, which validates that the information received from the database is a valid user. The server then returns the valid/invalid result to the client-side which either presents error messages to the user’s view or continues to the given page after successful login.

## Class Diagram

The main sections of this class diagram are the member, the trip, and the booking. The member section comprises the MemberAccount, Membership\_Admin, and TravellerInfo classes. The TravellerInfo class is a type of MemberAccount which is part of the Membership\_Admin class. These classes contain information regarding the specific user. The trip section contains TripInfo, CruiseInfo, FlightInfo, HotelInfo, and Itinerary. TripInfo contains the three subclasses of FlightInfo, CruiseInfo, and HotelInfo. All three of these subclasses are also Itinerary classes. All of the trip section classes contain information about the specific parts of the trip. The booking section contains the Travel\_Agent, HotelBooking, FlightBooking, and CruiseBooking. The HotelBooking, FlightBooking, and CruiseBooking are classes that connect a specific trip subclass with a user. The Travel\_Agent class is responsible for actually booking the specific trip or trip subclass.

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

| **Method** | **Purpose** | **URL** | **Notes** |
| --- | --- | --- | --- |
| **GET** | Retrieve list of trips | /api/trips | Returns all active trips |
| **GET** | Retrieve single trip | /api/trips/:tripCode | Returns single trip instance, identified by the trip code passed on the request URL |
| **PUT** | Update single trip | /api/trips/:tripCode | Updates single trip instance, identified by the trip code passed on the request URL. Information based on form data submitted. |
| **POST** | Add single trip | /api/trips | Adds a single trip to active trips. Information based on submitted form data. |
| **POST** | Admin user login | /login | Login form data sent to server for validation. Available through admin site, not user site. |
| **POST** | Register admin user | /register | Registers new user based on submitted form data. No current view setup for interacting via browser. |

## 

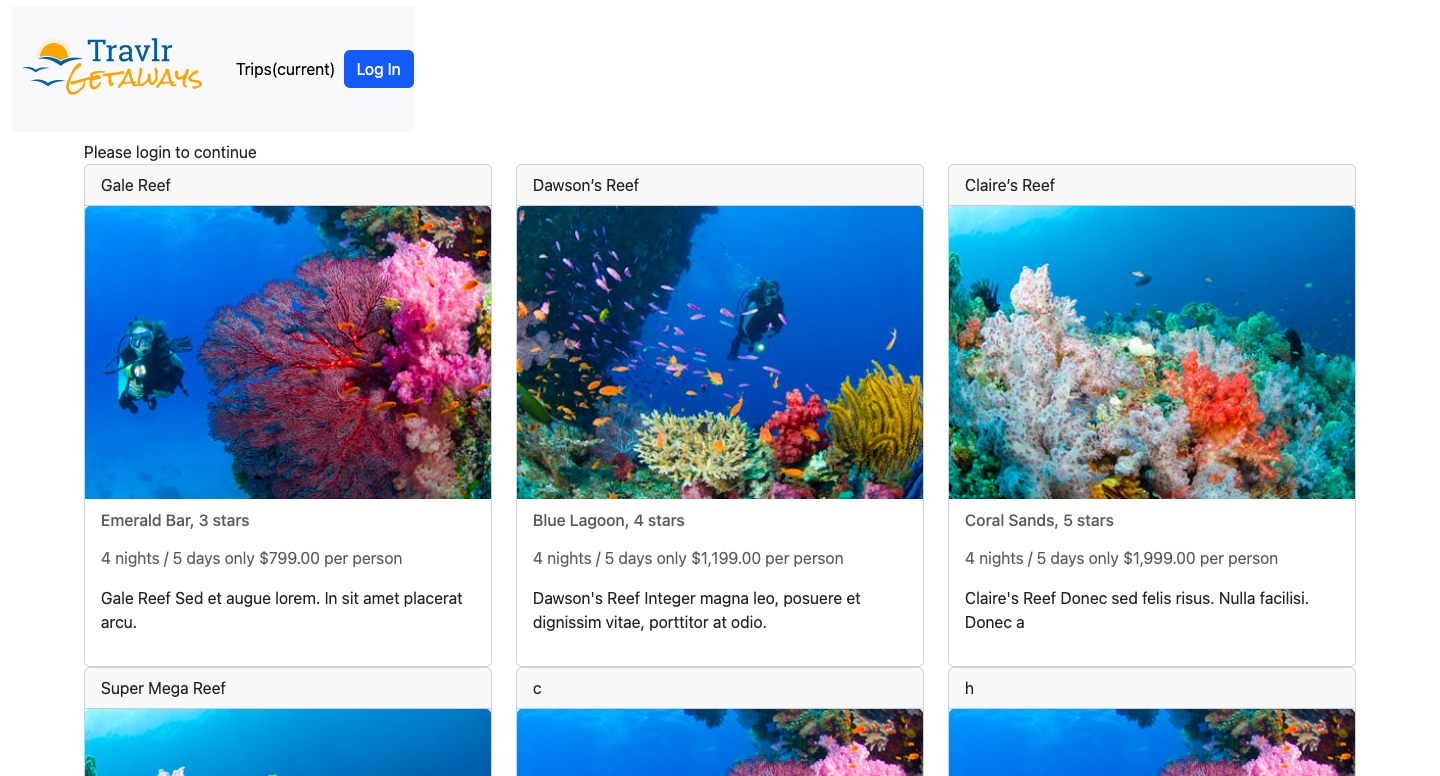
## 

## 

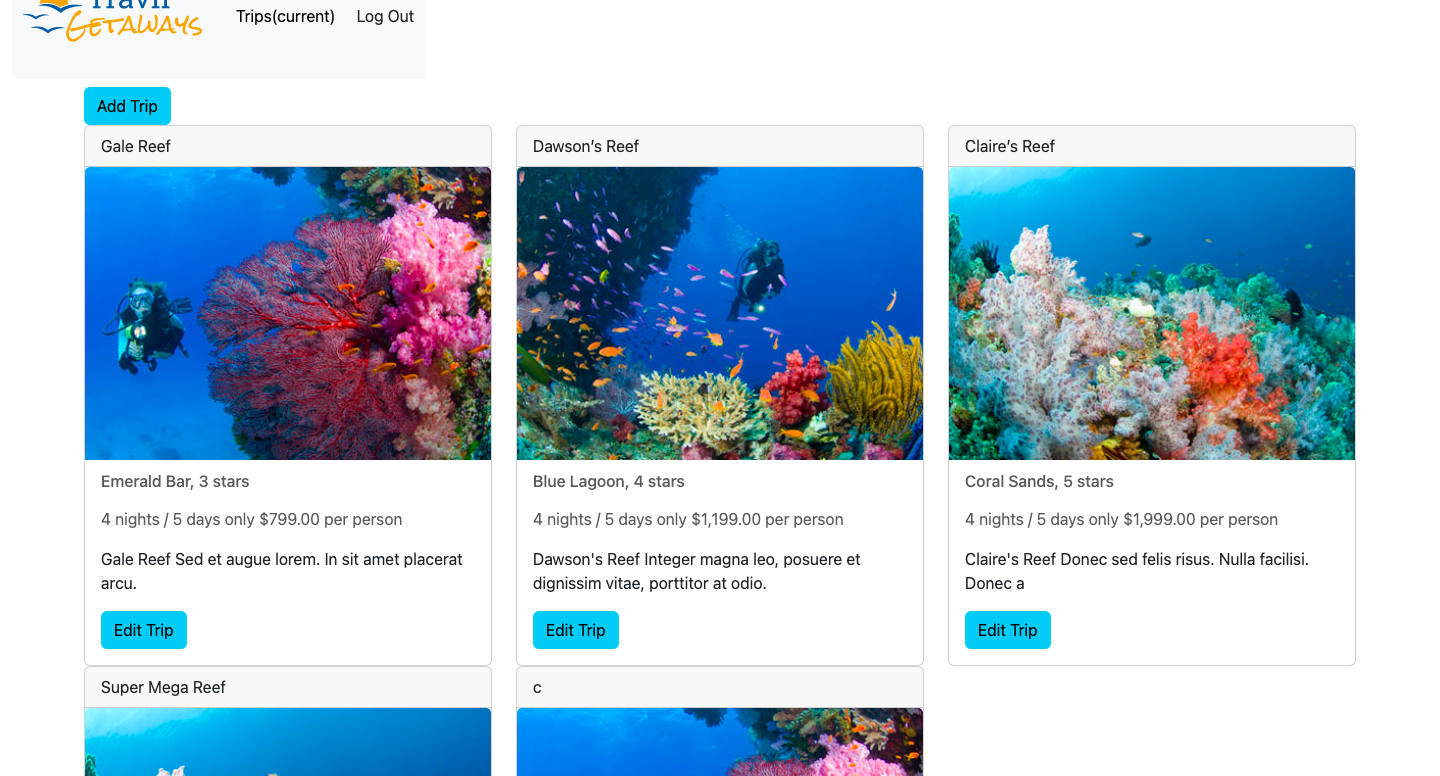
## 

## 

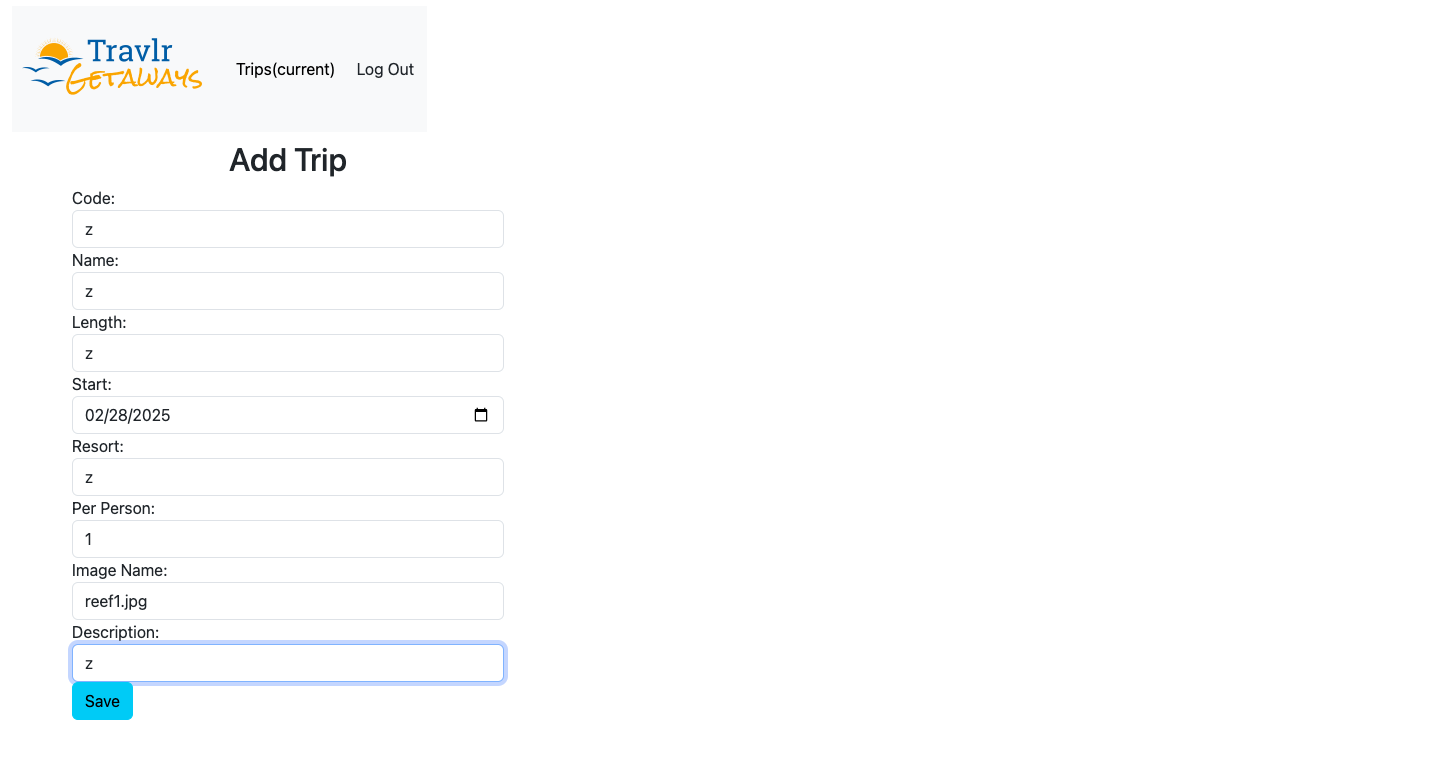
## The User Interface



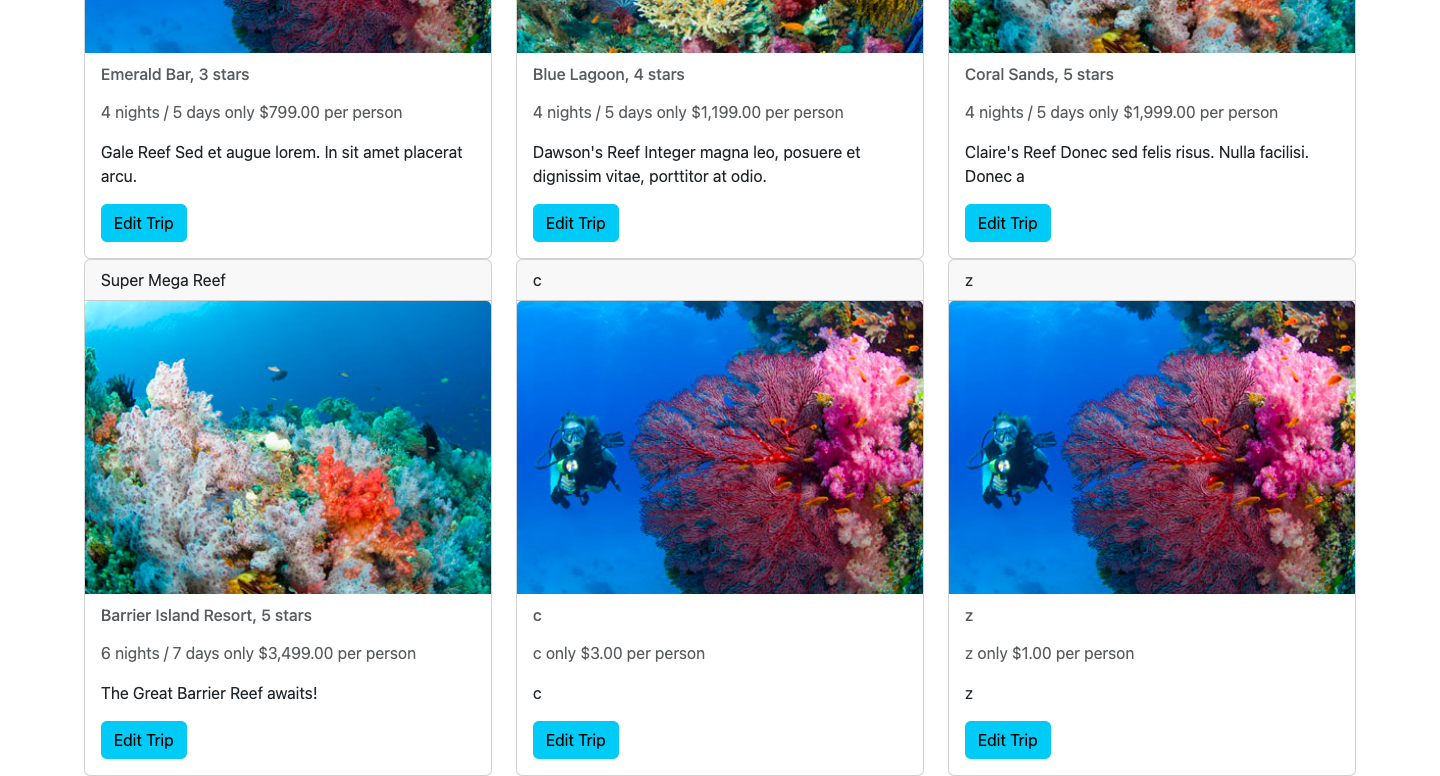
[view trips while logged out]



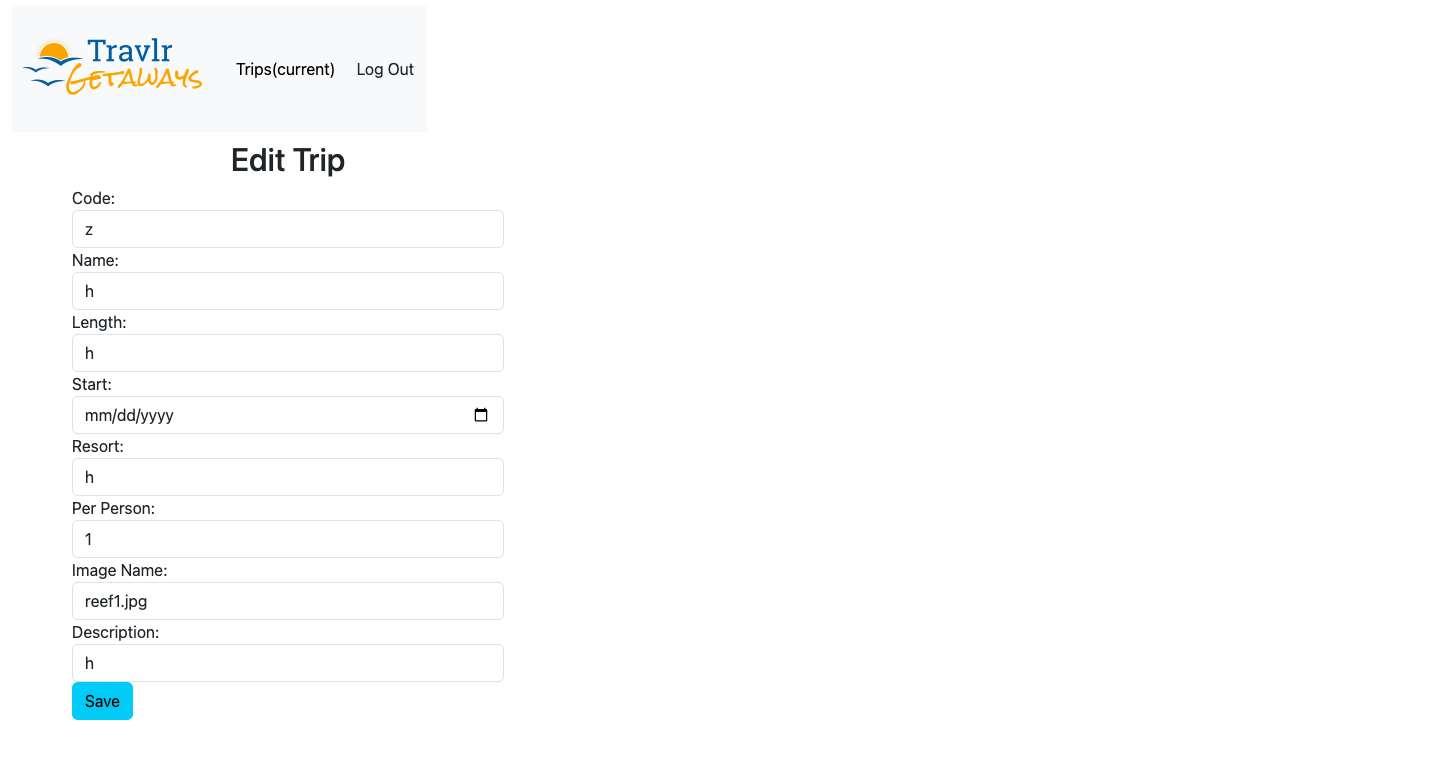
[view trips while logged in]



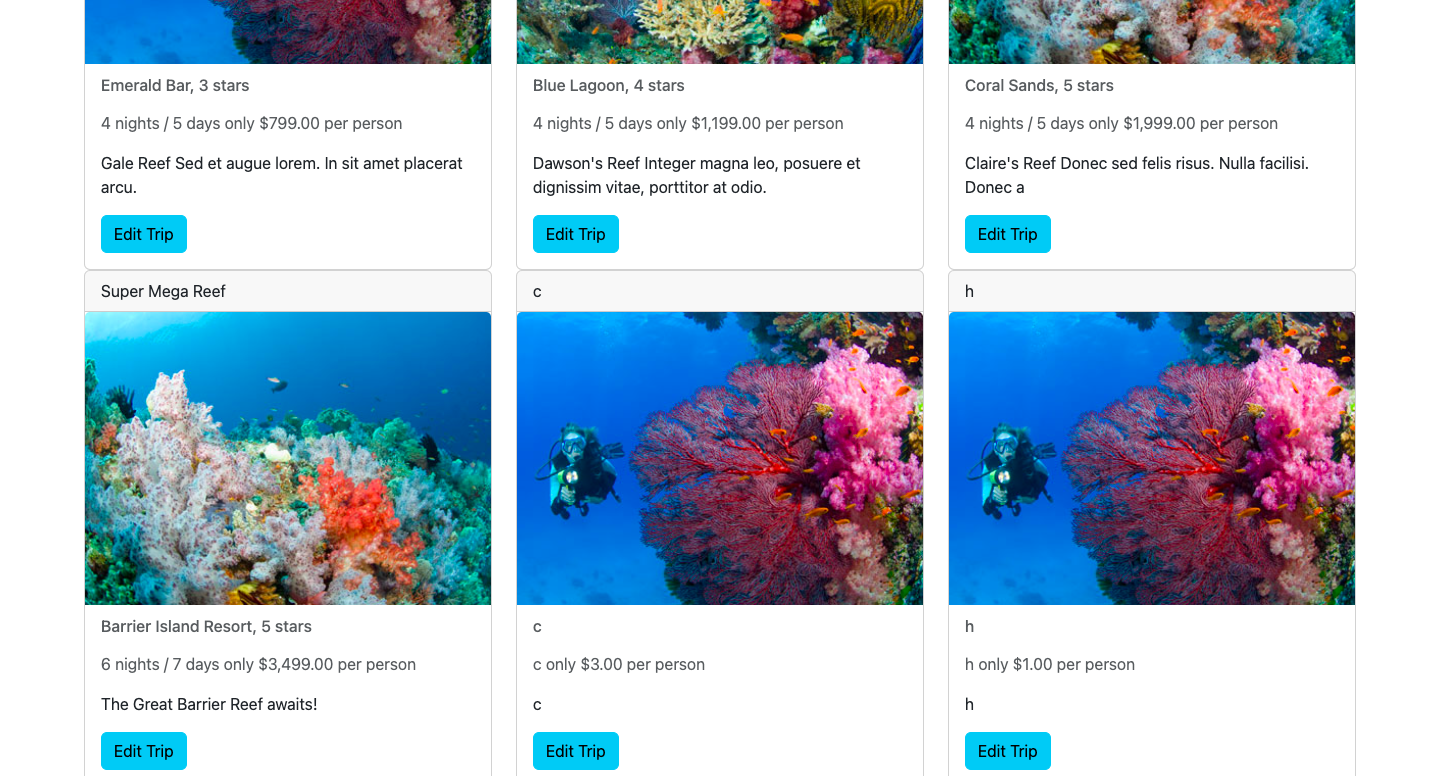
[add trip form]



[trips list shown with added trip]



[edit trip form]



[trips list with updated trip from edit form]

At a base level, the structures of the Angular project versus the Express customer-facing page differ since the Angular portion is a single-page application (SPA) and the Express portion contains multiple pages. This inherently alters the structure of the portions due to the content the files and folders must contain. But, more explicitly, the difference between the two is more apparent when looking at how they apply separation of concerns. Express contains views, controllers, and routes folders that contain the corresponding javascript and handlebars files for each page. Angular breaks the application up by its components first and then gives each component an html, css, and ts file to define it. This allows for a simpler breakdown of the page compared to Express’ entire page being contained in the handlebars file that contains everything on the page. The main benefit of an SPA compared to a simple web application interaction is the SPA’s ability to dynamically refresh content instead of having to reload the entire page’s content. This is the reason why it is a popular structure for things like social media which may have a lot of dynamic content that requires updating often. Another big advantage of an SPA is its ability to do a lot more work on the client-side of the application which can help to reduce server load. Disadvantages of SPAs are mainly due to their reliance on client-side use of JavaScript. Which can reduce compatibility with certain browsers, increase performance requirements, reduce search engine optimization (which wouldn’t be an issue in our case since this isn’t a public site), and increase security concerns due to cross-site scripting. The process that I used to test the SPA’s ability to work with the API involved the use of the third-party application Postman. This allowed me to send specific requests to the API endpoint in question to ensure that it was executing correctly when provided the data and method.