

Travlr Getaways

# **CS 465 Project Software Design Document**

Version 1.0

## Table of Contents

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

[Table of Contents 2](#_30j0zll)

[Document Revision History 2](#_1fob9te)

[Instructions 2](#_3znysh7)

[Executive Summary 3](#_2et92p0)

[Design Constraints 3](#_tyjcwt)

[System Architecture View 3](#_3dy6vkm)

[Component Diagram 3](#_1t3h5sf)

[Sequence Diagram 4](#_17dp8vu)

[Class Diagram 4](#_3rdcrjn)

[API Endpoints 4](#_26in1rg)

[The User Interface 4](#_1ksv4uv)

## [Document Revision History](#_44sinio)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 09/16/23 | Jackson Foster | First draft |

| 1.1 | 10/01/23 | Jackson Foster | second draft |
| --- | --- | --- | --- |

## 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](#_2jxsxqh)

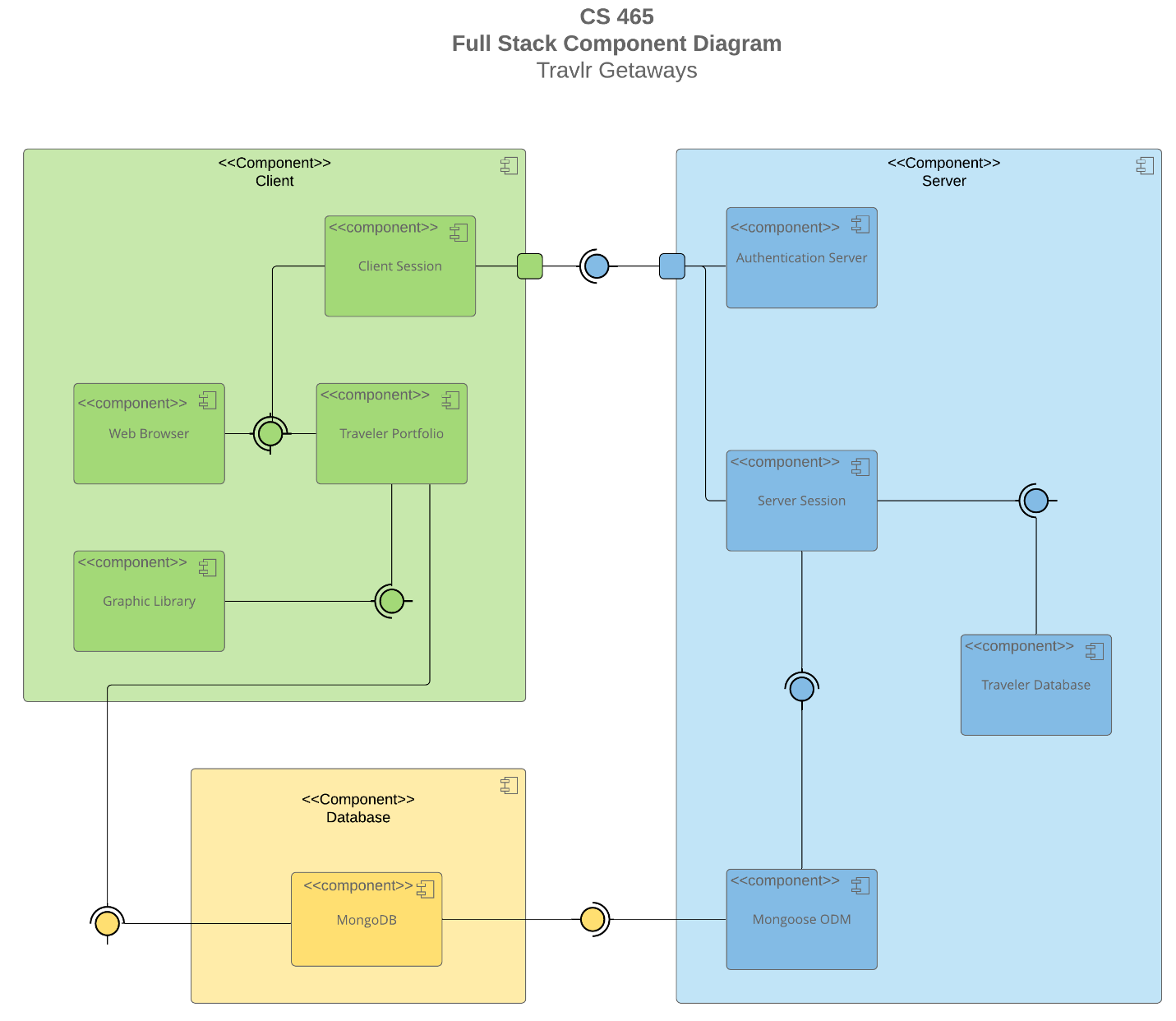
In response to your project's executive summary request, we propose the development of a cutting-edge Travel Web Application with a strong focus on scalability and user experience. Our chosen architecture centers around Node.js and MongoDB, ensuring seamless integration and efficient data handling tailored for travel-related content. The customer-facing application, powered by Node.js, will provide users with features like user registration, an interactive dashboard, advanced destination search, booking capabilities, and responsive design to guarantee an immersive and user-friendly travel experience. Simultaneously, the administrator Single-Page Application (SPA) will empower your team with robust tools for managing travel content, analytics, reporting, and real-time monitoring. Our team is enthusiastic about embarking on this travel-focused project and is confident that our MEAN stack-based solution will not only meet your requirements but also redefine travel web experiences.

## [Design Constraints](#_z337ya)

There are many design constraints when building a travel website include ensuring compatibility with various devices and browsers to be able to have a diverse user base. Data security and privacy measures must be robust, especially when handling user information and payments. Load times and performance should be optimized to provide a seamless experience. Integration with external APIs for travel bookings, weather updates, and location-based services is essential for comprehensive functionality. Compliance with travel industry regulations and accessibility standards is necessary to reach a wide audience, including individuals with disabilities. Finally, scalability and robust server infrastructure are vital to accommodate traffic spikes during peak booking seasons.

## [System Architecture View](#_3j2qqm3)

### 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 CS 465 Full Stack Component Diagram outlines the technical architecture of our travel website. Within the Client Component, we have key components including the Web Browser, which represents the user's browser, the Client Session responsible for managing user sessions, the Traveler Portfolio handling user-specific data and interactions, and the Graphic Library providing graphical resources for the user interface.

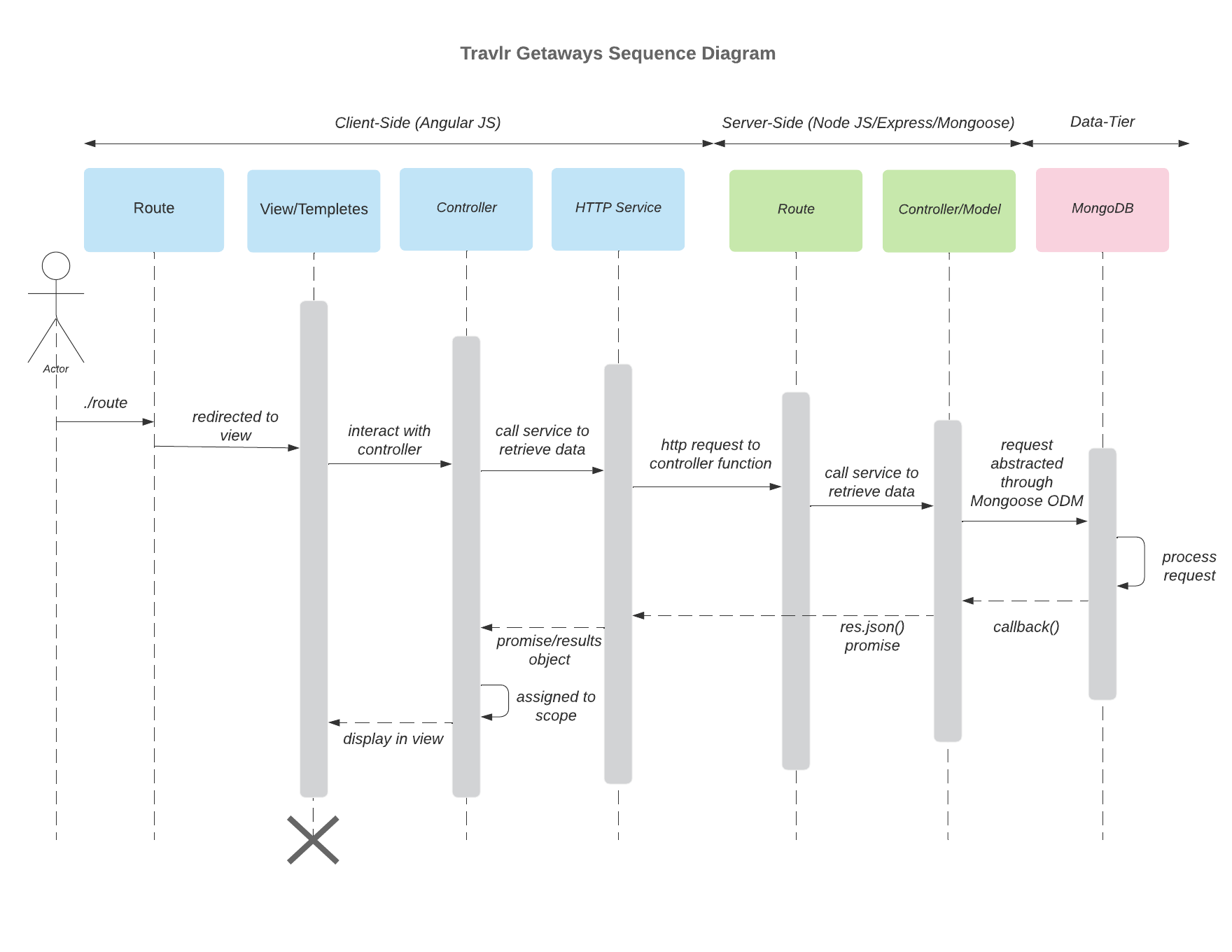
On the Server Component side, we utilize the Authentication Server for user authentication and authorization, the Server Session to maintain server-side user sessions, the Traveler Database for storing traveler-related data, and interact with a MongoDB database for efficient data management.In the Database Component , we have the MongoDB, serving as our NoSQL database to store and manage various application data.

In terms of interactions, the Web Browser component connects to the Traveler Portfolio and Client Session, which is an indication of the required interfaces for user interaction and session management. The Traveler Portfolio component connects to the Graphic Library for graphical resources and interacts with the MongoDB database for data retrieval and storage.

The MongoDB database component interfaces with the Mongoose ODM for streamlined database interactions. The Mongoose ODM component, in turn, connects to the Server Session for server-side session management. The Server Session component communicates with the Traveler Database for data operations and also interacts with the Client Session for user session coordination.

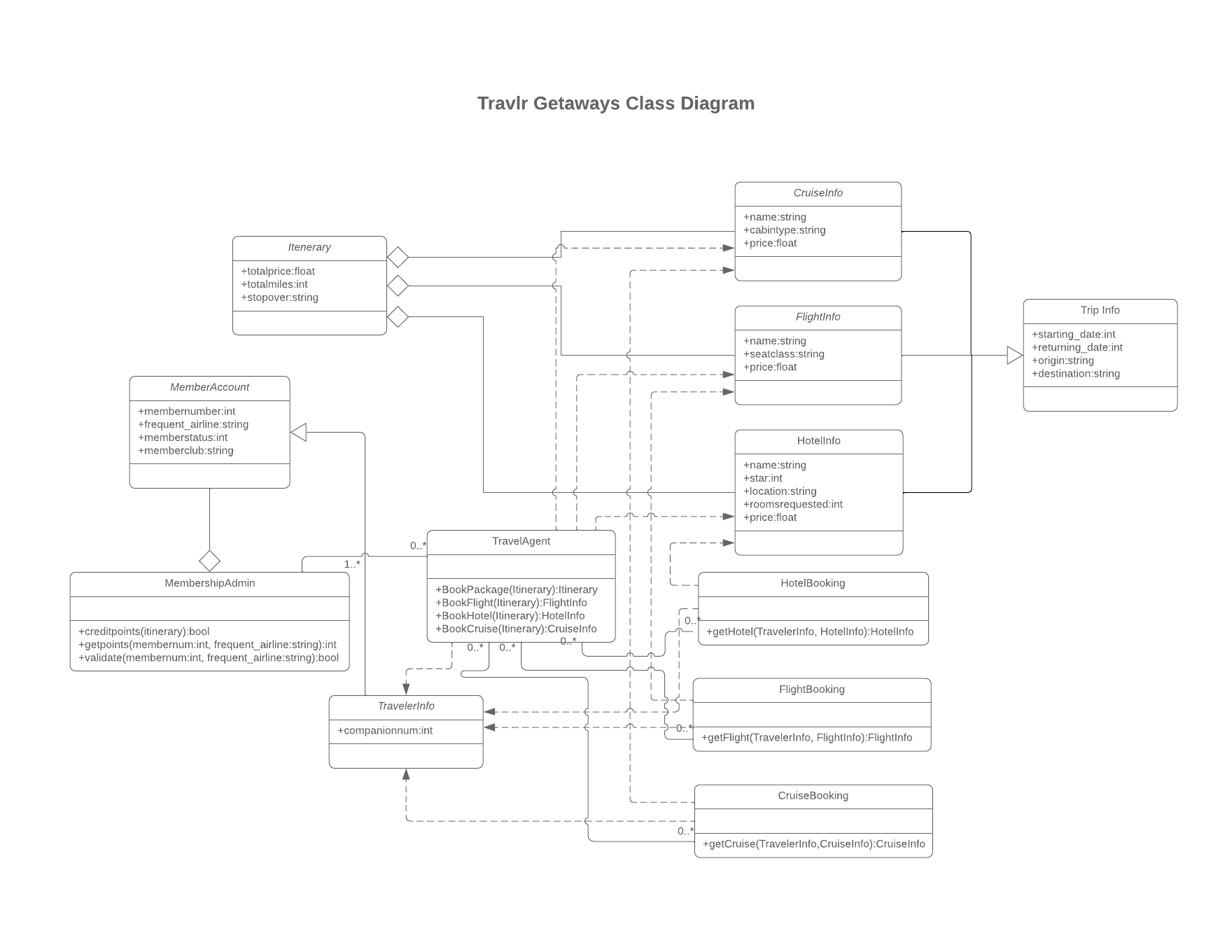
Lastly, the Client Session component connects to the Authentication Server for user authentication and authorization. This comprehensive component diagram illustrates the vital components and their interdependencies within our travel website's architecture, highlighting our use of the MongoDB database for efficient data handling.

### Sequence Diagram



The sequence diagram starts with the actor which represents the user. The actor initiates a route and is guided to one of the views/templates for the site via the frontend router. Subsequently, the view triggers the corresponding controller, which is responsible for populating the template, rendering it, and then returning the view for display to the actor. The frontend controller initiates calls to functions within the HTTP service to fetch specific pieces of information. The outcomes of these function calls are transmitted back to the controller. The HTTP service establishes the connection between the frontend and backend by executing API calls to designated routes. On the backend side, the router receives the route from the frontend and, based on that route, invokes the appropriate backend controller. Once invoked by the router, the backend controller initiates a call to the database using Mongoose. The controller then processes the data returned by the database and transmits the result back to the calling frontend HTTP service. Finally, the MongoDB database receives the query from the backend controller, processes the request, and ultimately returns the desired result.

## Class Diagram



The Cruiselnfo, Flightlnfo, and Hotellnfo classes all contain a 'name' property and other attributes specific to each mode of travel. Additionally, each of them inherits attributes from the Triplnfo class, which includes properties for start and return dates, as well as origin and destination locations.

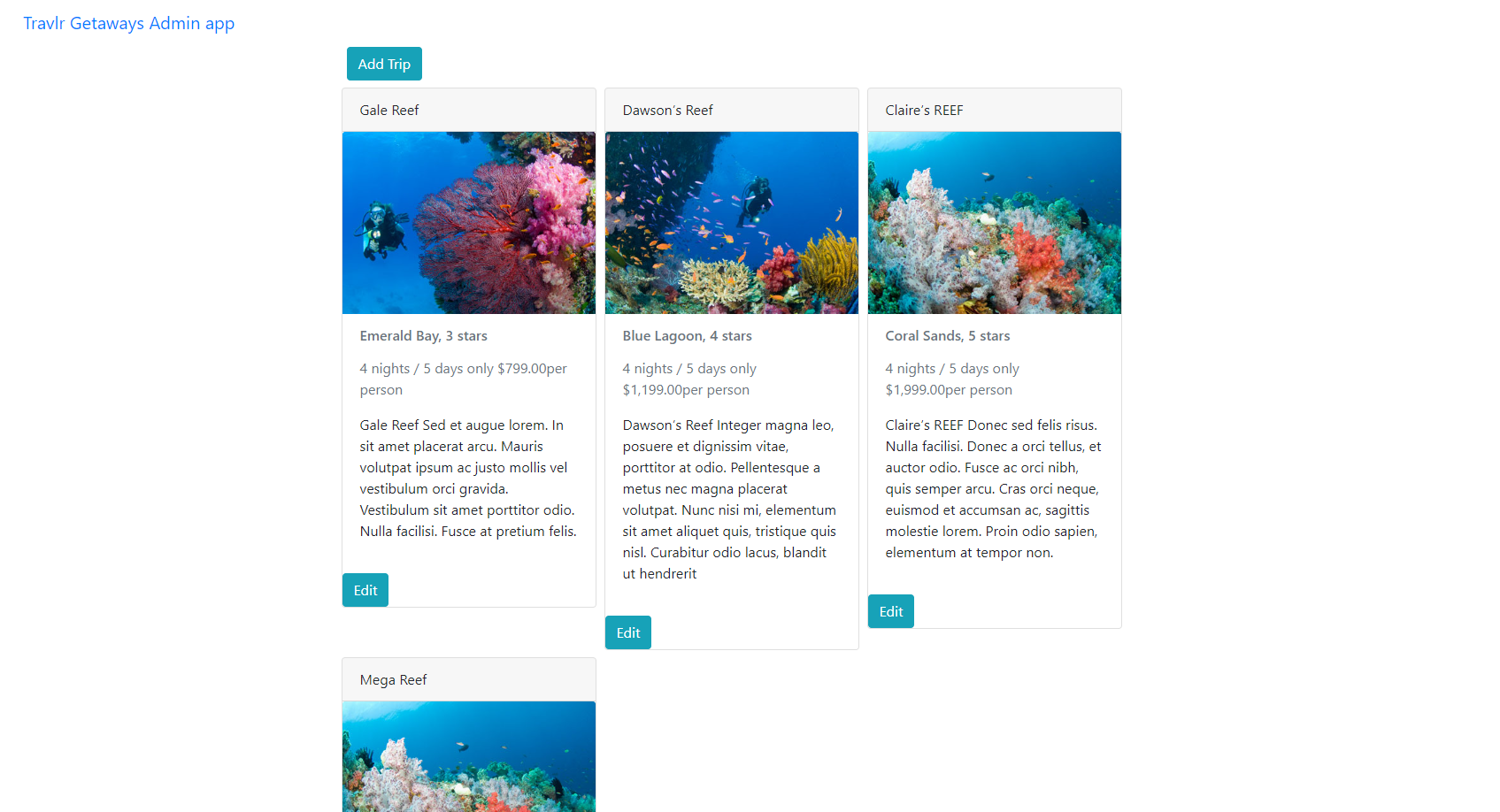
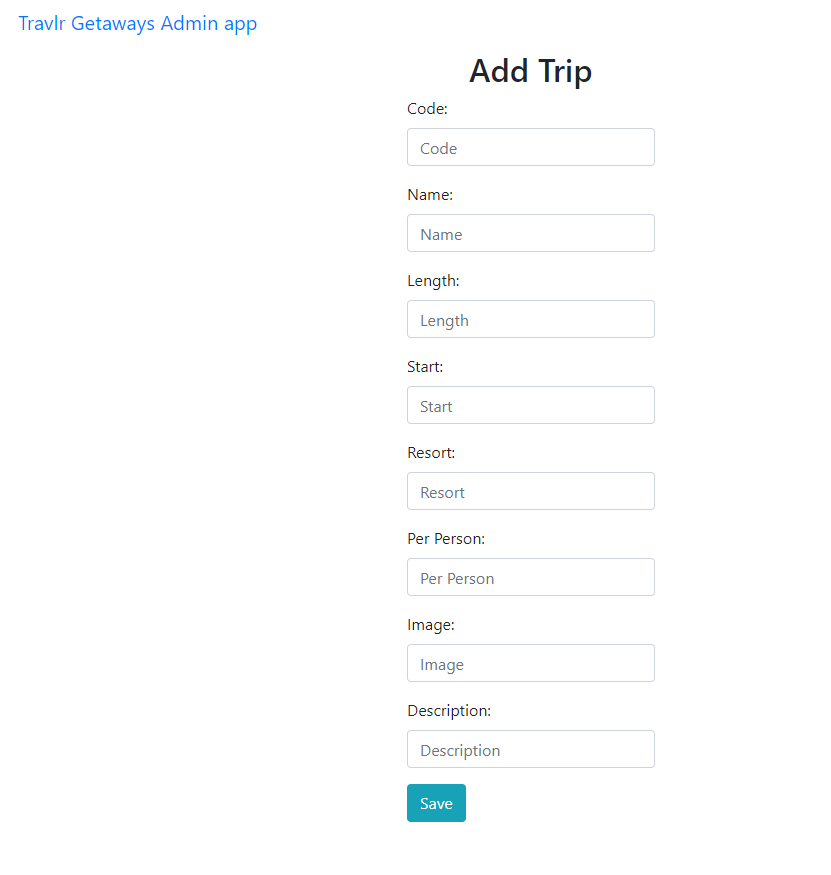
CruiseBooking, FlightBooking, and HotelBooking are each associated with their respective Info class and the Travellerlnfo class. There exist zero-to-many relationships between the Booking classes and the TravelAgent class in both directions. The TravelAgent class also forms associations with the Cruiselnfo, Flightlnfo, Hotellnfo, and Travellerlnfo classes, and it maintains a one-to-many relationship with the MembershipAdmin class.

The Travellerlnfo class inherits attributes from the MemberAccount class, and the MembershipAdmin class has an aggregate relationship with the MemberAccount class. Furthermore, the Itinerary class has an aggregate relationship with the Cruiselnfo, Flightlnfo, and Hotellnfo classes.

## [API](#_1y810tw) Endpoints

| **Method** | **Purpose** | **URL** | **Notes** |
| --- | --- | --- | --- |
| **GET tripsList** | Return list of all trips | </api/trips> | Returns all active trips |
| **GET tripsfindbycode** | return one trip based on its code | </api/trips/:tripId> | Returns single trip instance, identified by the trip ID passed on the request URL |

## The User Interface



Today, my focus was on creating a Single Page Application (SPA) for managing a web server directly from the user's browser. One noticeable difference was that the project structure in Angular is quite distinct from that of Express HTML pages. While Angular employs modular components, services, and routing, Express HTML pages tend to have simpler structures with separate HTML and JavaScript files. In terms of the SPA's functionality, there are both advantages and disadvantages to consider. SPAs provide a more enjoyable user experience with seamless navigation and reduce the strain on the server by reducing the number of page requests. However, they can have longer initial loading times. On the plus side, SPAs offer dynamic content loading without full page refreshes and improved data synchronization capabilities with the server.

Testing the SPA was a crucial part of the process to ensure it interacts smoothly with our API for data retrieval and submission. This involved integration testing to confirm data flowed correctly between components, and end-to-end testing to validate the entire user experience. I had run into many issues and it seems to be some what my own fault. There was one thing that was frustrating was having to figure out my verison of Node was too recent and after doing hours of research it recommended reverting back and that ended up fixing an issue I was stuck on for quite some time. Other issues was after I got the admin server running my client side one would not start so I had to go line by line to find my mistake.