

<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 | 5/26/2024 | Brittney Miller | Executive summary, design constraints, and system architect view changed and updated |
| 2.0 | 6/9/2024 | Brittney Miller | Complete sequence diagram, Complete Class Diagram, API Endpoints |
| 3.0 | 6/22/2024 | Brittney Miller | The final project finished up. Edit unique trips, user interface completed, design finished. |

## 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 paragraph response covering the indicated information.

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

Travel Getaways is built using the MEAN system, which is MongoDB, Express.js, Angular, and Node.jsA static portion will display content for the and an airport that will manage the website. Regarding the customer-facing side of the application, we will use Express.js, Handlebars, and MongoDB. Node.js will use Express.js as the framework and will be used to serve the static content of the web application. MongoDB will work with Handlebars, which will be used to generate the static content that Express.js can serve. Handlebars is a templating language that dynamically populates the static page with content. MongoDB is the NoSQL database that stores static content displayed in the handlebar templates.

With the admin side of the application, a single-page application will be created using Angular. This allows an admin to perform various administration functions. You can access the admin page with authentication and login. Then, they can manage the web application's content and add, update, or remove content for different parts of the static side of the web application. You can do this by updating the documents in MongoDB to reflect what is done in the admin portal.

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

When going over the design, we can run into some design constraints when developing a web application using the MEAN stack. We will need to find an easy way to update the content on the static portion without redeploying the website to lessen downtime or glitches. The design must also be responsive and have a friendly interface that runs smoothly without issues. The website must also be multiplatform accessible and function smoothly on devices without issue. The admin page also needs to be a single-page application and be able to be used dynamically to display components so the admin can manage the web application.

MongoDB will be the database used in the MEAN stack. It is well suited for web applications but can suffer performance issues if the dataset becomes too large. The rest of the stack ( Express, Angular, and Node.js) works well together but can end up with a large codebase that can become difficult to manage.

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

### Component Diagram

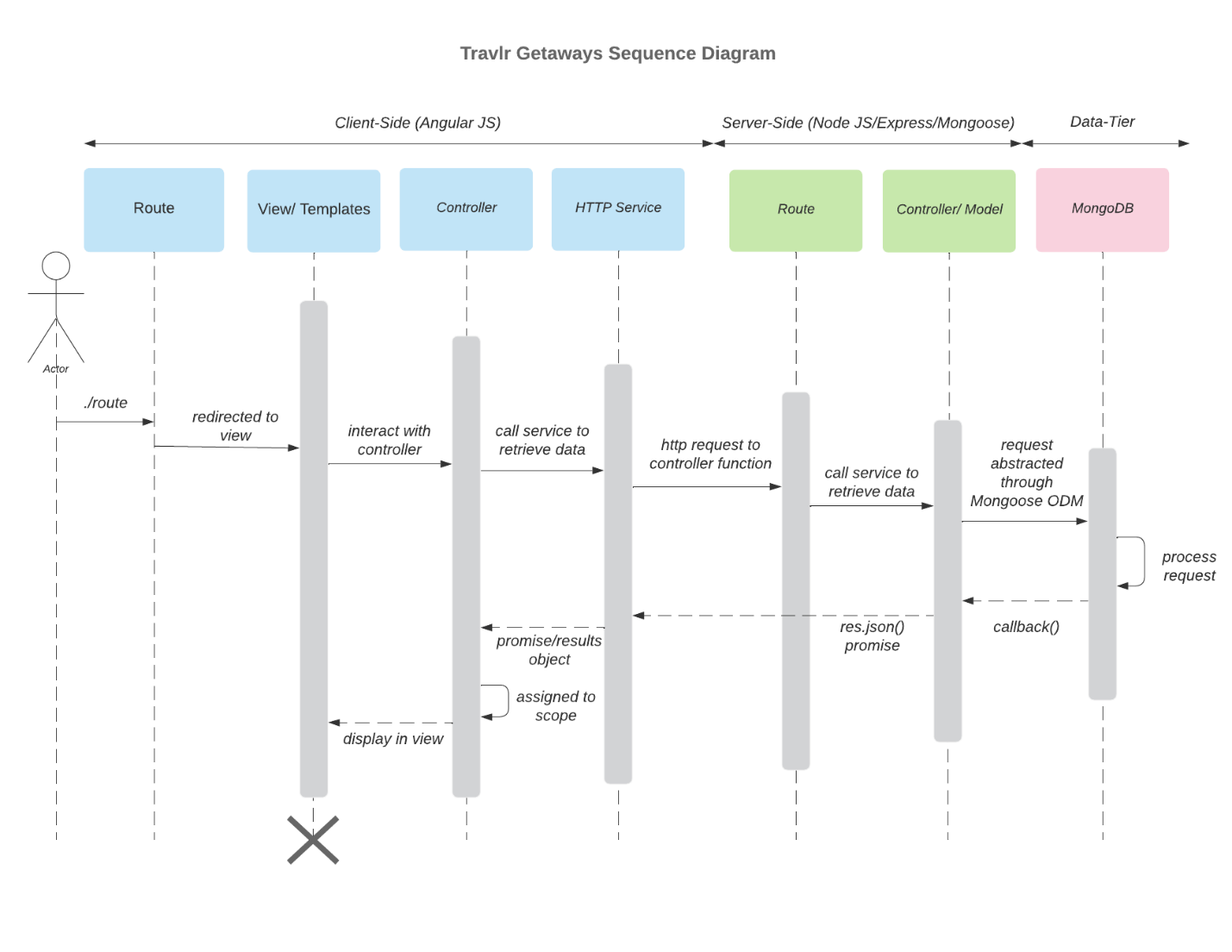


A text version of the component diagram is available: [CS 465 Full Stack Component Diagram Text Version](https://learn.snhu.edu/d2l/lor/viewer/view.d2l?ou=6606&loIdentId=24342).

The Travlr application consists of three major components: the client, the server, and the database. Each element has its owrts to ensure functionality.

The Client component contains components for Client sessions, the web browser, the traveler portfolio, and the graphic library. The web browser and graphic library both have an interface. The client session and traveler portfolio require the web browser interface. The traveler portfolio also requires the graphic library interface and the interface provided by the database component. The client session interacts with a port on the client component to connect to the server's necessary component interface.

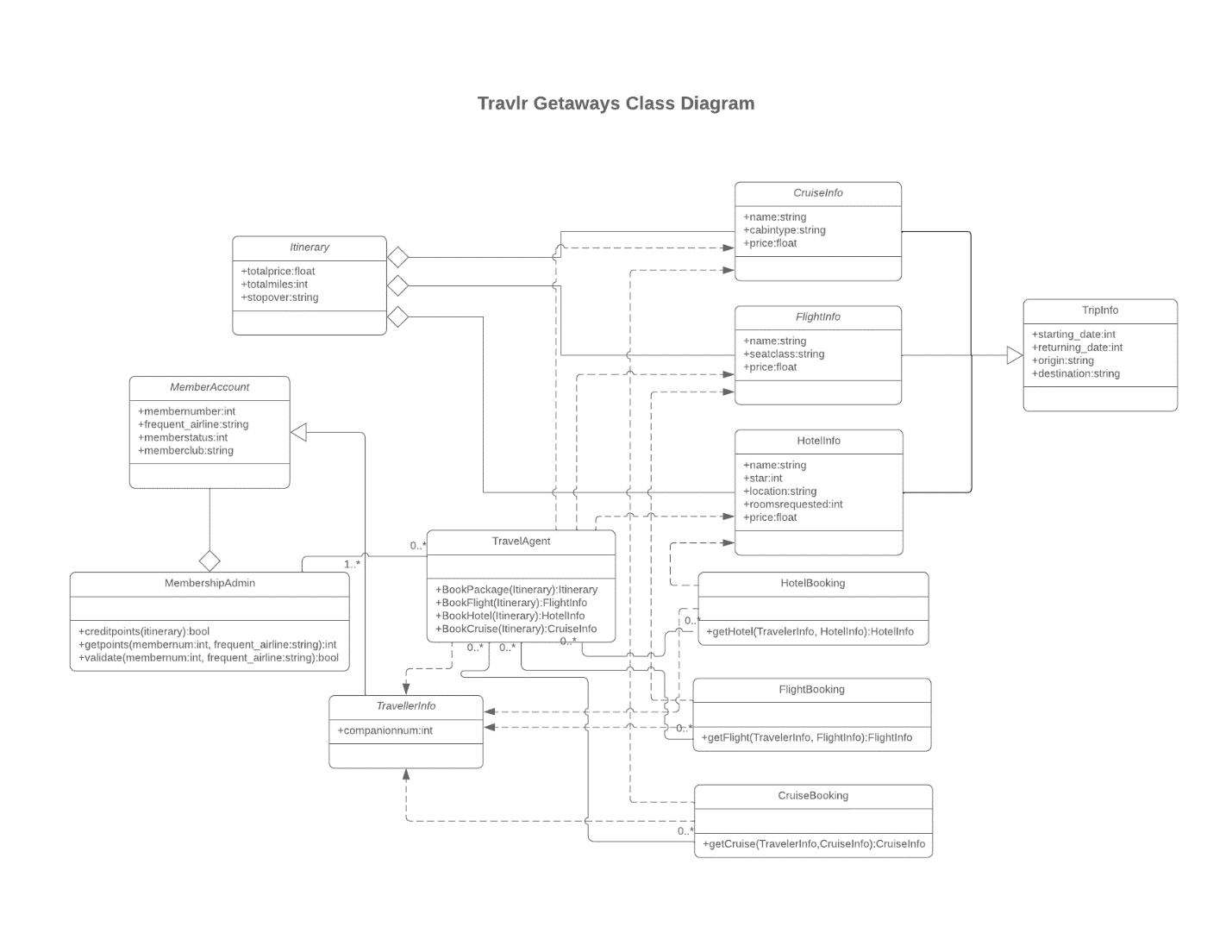
The server component has four components: the authentication server, server session, traveler database, and Mongoose ODM. The Mongoose ODM and server session both provide an interface. The traveler database requires the server session interface, and the server session requires the Mongoose ODM interface. The Mongoose ODM component requires the interface provided by the database component. Both the server session and authentication server components interact with a port on the server component to provide an interface.

The database component has a single MongoDB component, providing an interface to the client and server components. 

### Sequence Diagram

The sequence starts with an actor/user. The user will enter a route and is then directed to one of the views for the site by the frontend router. The view then calls the correct controller, creating a template to showcase the correct info for the user. The Frontend controller calls functions within the HTTP service to gather information. Function results will then be passed back to the controller. HTTP connects the front and back end through API calls to specific routes. The router on the backend will receive the routes from the front end and call the appropriate backend controller. The backend controller will call the database using Mongoose, and the controller uses the returned data to pass the result back up to the frontend HTTP service. MongoDB then receives queries from backend processes and returns the result.

## Class Diagram



The classes (CruiseInfo, FlightInfo, HotelInfo) all have name properties and other fields that make them unique for each mode of travel. They all inherit the TripInfo class, which contains start and return data properties, origin, and destination locations.

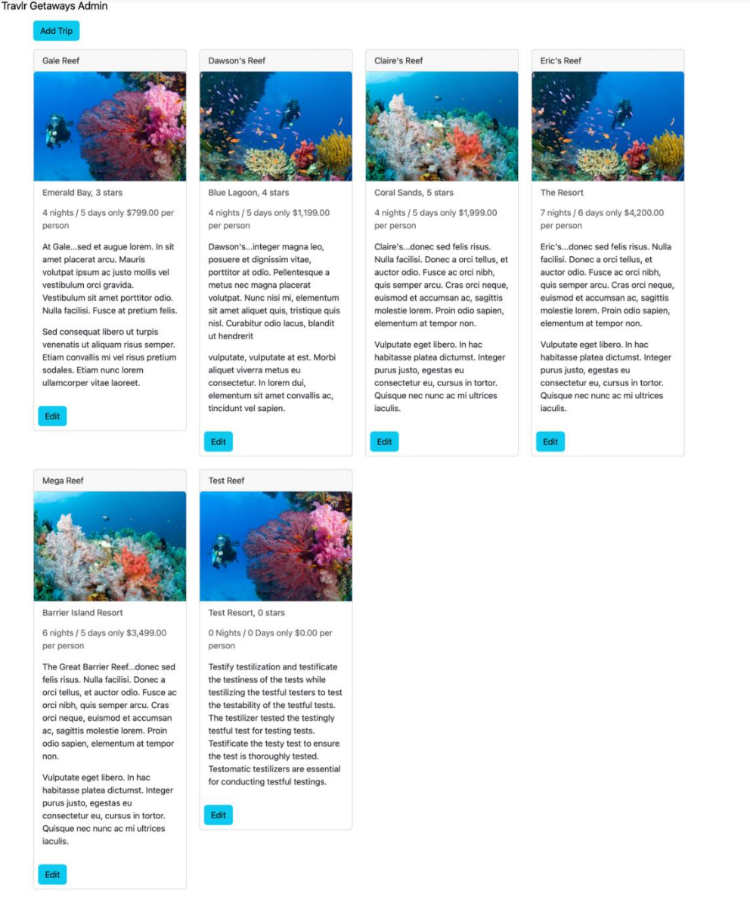
CruiseBooking, FlightBooking, and HotelBooking are all associated with their respectable Info class and the TravellerInfo class.

The Booking and TravelAgent classes had few relationships in both directions. The TravelAgent class is also associated with CruiseInfo, FlightInfo, and TravellerInfo classes and shares a one-to-many relationship with the MembershipAdmin class. TravellerInfo inherits the MemberAccount class. MembershipAdmin is an aggregated relationship with the MemberAccount class. Finally, the Itinerary class has an aggregated relationship with the CruiseInfo, FlightInfo, and HotelInfo classes.

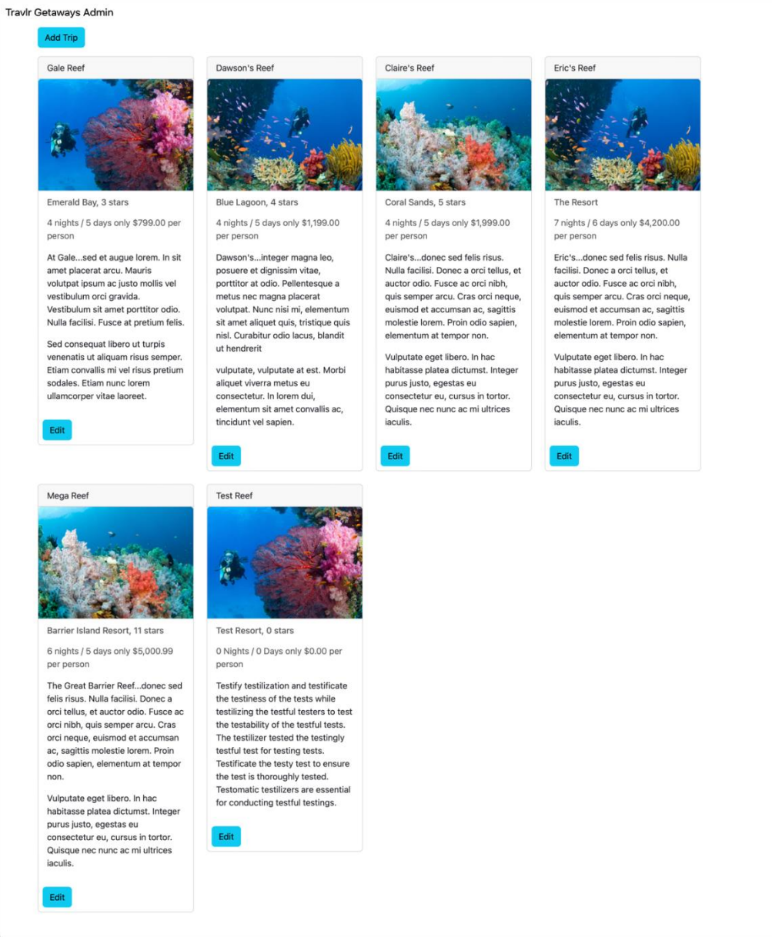
## [API](#_heading=h.2jxsxqh) Endpoints

| **Method** | **Purpose** | **URL** | **Notes** |
| --- | --- | --- | --- |
| **Post** | Login User | /API/login | Authenticates a user and returns a JWT |
| POST | Register a user | /API/Register | Add a new user to the database and return a JWT |
| GET | Retrieve a list of meals | /API/meals | Returns all meals |
| GET | Retrieve single meal | /API/meals/:mealCode | Returns a single meal, identified by the meal code at the end of the URL |
| GET | Retrieve a list of news | /API/news | Returns all news content |
| GET | Retrieve a single piece of news content | /API/news/:newsCode | Returns a single news piece, identified by the news code at the end of the URL |
| GET | Retrieve a list of rooms | /API/rooms | Returns all rooms |
| GET | Retrieve single room | /API/rooms/:roomCode | 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 |
| POST | Add a trip | /API/trips | Add a new trip to the database |
| GET | Retrieve single trip | /API/trips/:tripCode | Returns a single trip, identified by the trip code at the end of the URL |
| **PUT** | Update single trip | /API/trips/:tripCode | Updates single trip, identified by the trip code at the end of the URL |
| **DELETE** | Delete single trip | /API/trips/:tripCode | Deletes a single trip, identified by the trip code at the end of the URL |
|  |  |  |  |

## The User Interface



A screenshot of a computer

Description automatically generated

Angular is the front-end framework where the views are rendered on the client-facing side.

Express is the back-end framework where the views are rendered on the server and sent to the client. Angular in the project is made from the views, controllers, and routes we added to the design. Angular is a single HTML page that dynamically updates the views and page content on the client side. It reuses components to make up the site and parts of it.

Express will use a templating engine. For this project, we used handlebars to generate content on the server before sending it to the client. Therefore, both of these use APIs to send and retrieve data.

The functional advantages of SPAs are the reduced server load since we only need to send the initial page, the fact that SPAs are known to create some pretty interactive sites with a multitude of functionalities, and the fact that user interaction and experience are usually faster since we are eliminating full page reloads and typically only necessary data is retrieved from the server.

The disadvantage of SPA functionality is the longer initial load time since we are retrieving the entire javascript application; they are also tricky for SEO optimization.

SPA functionality that may be useful is client-side routing, which can remove additional server requests and provide a smoother transition between views. SPAs can also offer offline support after the initial page load, which can be resourceful for clients and users.