

Travlr Getaways Website

# **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 | 07/20/2025 | Jaden Williams | First Edit |
| 1.1 | 08/03/2025 | Jaden Williams | Second Edit |
| 1.2 | 08/17/2025 | Jaden Williams | Final edit |

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

The Travlr Getaways website will be built using the **MEAN stack**, an acronym for **MongoDB, Express.js, Angular.js, and Node.js**. This modern web development stack is designed to efficiently handle JSON data across all layers of the application, allowing for streamlined development and high performance.

#### **MEAN Stack Components**

* **Frontend (Client Side):** Handled by **Angular.js**, which supports dynamic, responsive, and elegant user interfaces.
* **Backend (Server Side):** Powered by **Node.js** in combination with the **Express.js** framework, enabling fast and scalable server-side logic.
* **Database:** Managed by **MongoDB**, a NoSQL database optimized for high-speed data retrieval and flexible data structures.

The architecture allows Angular.js to request data from Node.js, which in turn uses Express.js to interact with MongoDB. The data is then returned in JSON format and displayed on the client side, ensuring a seamless and efficient user experience.

The customer-facing Multi‑Page Application (MPA), built with **Express.js** and **Handlebars**, lets users browse and book getaways—featuring a user-friendly **dark mode** for comfortable viewing. Meanwhile, the admin side is a centralized **Single‑Page Application (SPA)** built with **Angular.js**, enabling real-time content management (adding, editing, deleting trips) that instantly reflects on the site.

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

While the Travlr Getaways website leverages the flexibility of the MEAN stack, there are a few inherent limitations to consider, both creatively and technically.

#### **General Limitation:**

* **Creative Scope:** The design and features of the website may be constrained by available development time, frameworks, and resource priorities.

#### **MongoDB:**

* **BSON Document Size:** Maximum size is 16MB to prevent RAM overuse.
* **Nested Depth Limit:** BSON documents can be nested up to 100 levels (each object or array counts as one level).
* **Database Naming:** Names are not case-sensitive, must be lowercase, avoid special characters, and stay under 64 characters.
* **No Duplicate Field Names:** Each document must have unique field identifiers.
* *(For more information, consult the MongoDB documentation.)*

#### **Express.js:**

* Serves content from the server, playing a critical role in how web pages are delivered to end users.

#### **Angular.js:**

* Used to build the interactive user interface for both customers and administrators.
* Works alongside **Handlebars**, which enables dynamic content rendering within JavaScript-based templates.

#### **Node.js:**

* Enables application scalability and seamless handling of increasing traffic, especially when paired with MongoDB for backend operations.

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

### Component Diagram



The Travlr Getaways system is divided into three primary components: **Client**, **Database**, and **Server**. Each plays a distinct role in how the application functions, and they work together to create a seamless experience for both users and administrators.

#### **1. Client Component**

The Client side initiates user interaction and includes the following sub-components:

* **Client Session**
* **Graphic Library**
* **Traveler Portfolio**
* **Web Browser**

The process begins when a user launches the **Web Browser**, which triggers the **Client Session**. This session communicates with the **Authentication Server** to verify that the user isn't already logged in through another port or session. Once authenticated, the **Traveler Portfolio** loads, presenting the user with travel-related content from the website. As the Traveler Portfolio is activated, it also launches the **Graphic Library**, which supports visual elements and UI rendering.

Proceeding from the Client, data flows into the Database.

#### **2. Database Component**

The Database layer consists of a single, essential sub-component:

* **MongoDB**

As users interact with the website, the **MongoDB** database updates reflect any changes made—such as bookings or account information. This data synchronization is crucial to maintaining real-time content accuracy and overall site functionality.

After the database processes update, control shifts to the Server.

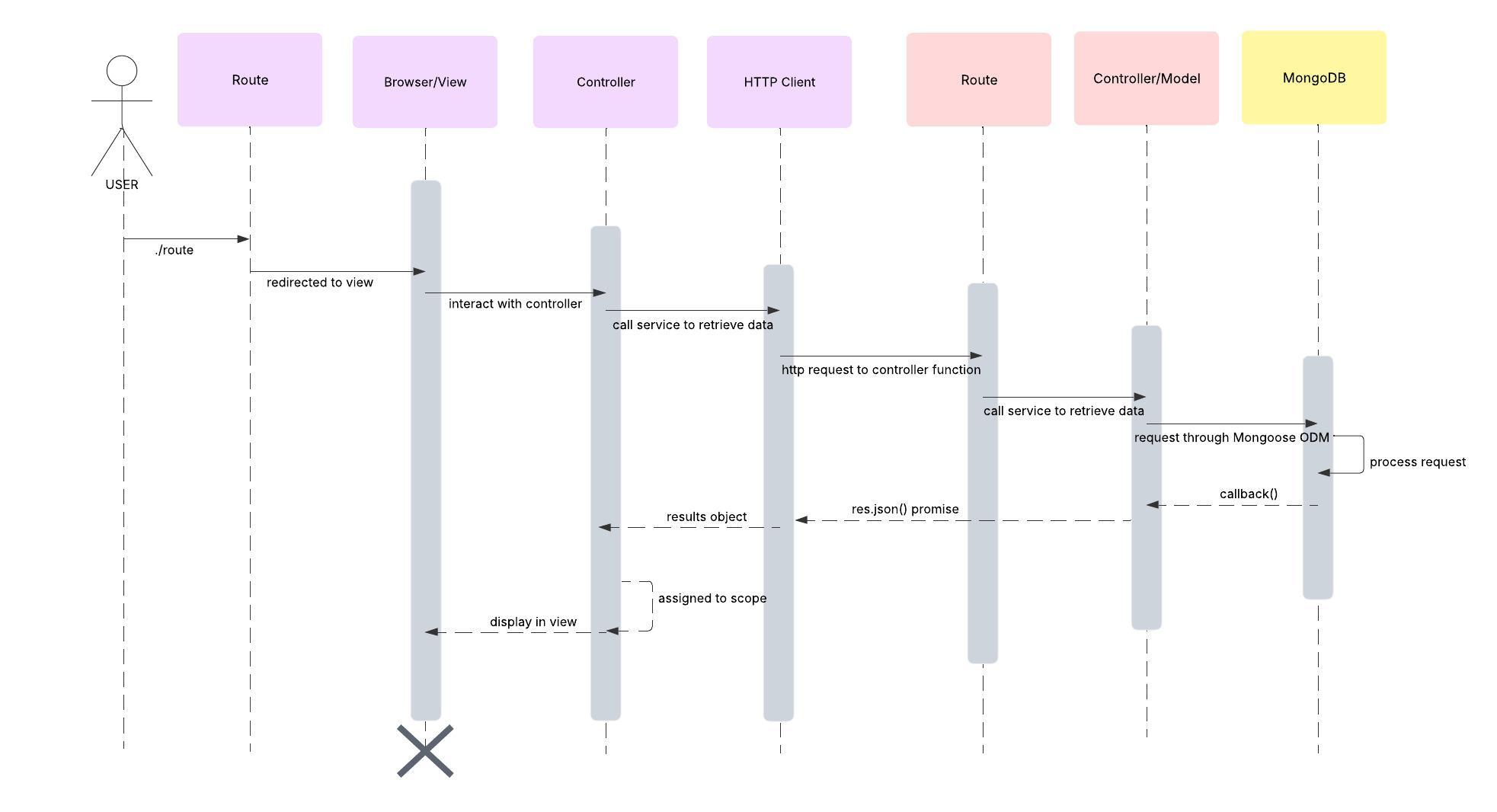
#### **3. Server Component**

The Server side manages logic, sessions, and authentication. It includes:

* **Authentication Server**
* **Mongoose ODM**
* **Server Session**
* **Traveler Database**

The server uses **Mongoose ODM** to interface efficiently with MongoDB, allowing the use of schemas and rapid development of workflows. **Server Session** handles user information and validates it against the current **Traveler Database**. Once verified, this information is passed to the **Authentication Server**, confirming the user’s identity. The result is sent back to the **Client Session**, completing the authentication loop and allowing continuous interaction.

### Sequence Diagram



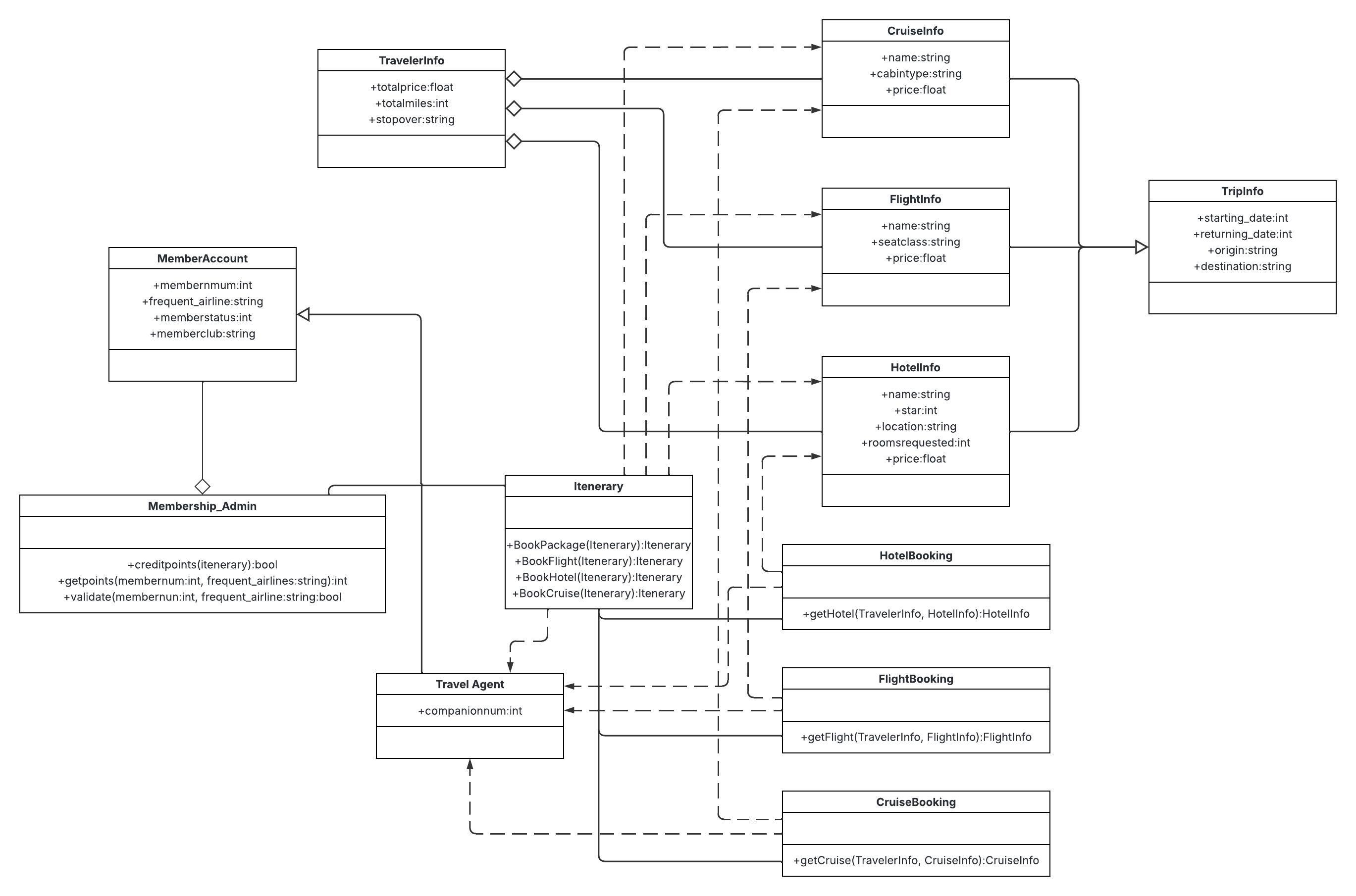
The process begins with an **Actor (the user)** accessing the website by navigating to: https://(insertwebsitelink).com/(insertwebsite-page) Upon doing so, the **Web Browser** loads a client-side template or view, displaying the initial content of the page to the user.

When the user interacts with the page—such as clicking on a link (an element with an anchor tag)—the **client-side controller** acts. This controller invokes the **HTTP Client**, which serves as a bridge between the front end and the back end.

The HTTP Client sends a request to the server-side **API controller**, specifying which resource or page the user is trying to access. The API controller then processes this request and forwards it to the **MongoDB database**.

MongoDB retrieves the requested data and sends it back through the same chain—from the API controller to the HTTP Client—until it reaches the browser. The page is then updated with the new information, and the system continues to listen for additional user actions or data requests.

## Class Diagram



**1. Membership\_Admin**

* **Aggregates:** MemberAccount
* **Associates with:** One or more Itinerary objects
* **Methods:**
  + public boolean creditpoints() → returns Itinerary
  + public int getpoints() → returns int membernum, String frequent\_airline
  + public boolean validate() → returns int membernum, String frequent\_airline

**2. MemberAccount**

* **Inherited by:** TravelAgent
* **Aggregated by:** Membership\_Admin
* **Variables:**
  + public int membernumber
  + public String frequent\_airline
  + public int memberstatus
  + public String memberclub

**3. TravelAgent**

* **Inherits:** MemberAccount
* **Realized by:** Itinerary, HotelBooking, FlightBooking, CruiseBooking
* **Variable:**
  + public int companionnum

**4. Itinerary**

* **Realizes:** TravelAgent
* **Associations:**
  + With Membership\_Admin (1 or more)
  + With: CruiseInfo, FlightInfo, HotelInfo, HotelBooking, FlightBooking, CruiseBooking
* **Methods:**
  + public Itinerary BookPackage() → returns Itinerary
  + public FlightInfo BookFlight() → returns Itinerary
  + public HotelInfo BookHotel() → returns Itinerary
  + public CruiseInfo BookCruise() → returns Itinerary

**5. CruiseBooking**

* **Realizes:** TravelAgent, CruiseInfo
* **Associates with:** Itinerary (0 to many relationships both ways)
* **Method:**
  + public CruiseInfo getCruise() → returns TravelerInfo, CruiseInfo

**6. FlightBooking**

* **Realizes:** TravelAgent, FlightInfo
* **Associates with:** Itinerary (0 to many relationships both ways)
* **Method:**
  + public FlightInfo getFlight() → returns TravelerInfo, FlightInfo

**7. HotelBooking**

* **Realizes:** TravelAgent, HotelInfo
* **Associates with:** Itinerary (0 to many relationships both ways)
* **Method:**
  + public HotelInfo getHotel() → returns TravelerInfo, HotelInfo

**8. CruiseInfo**

* **Aggregated by:** TravellerInfo
* **Inherits from:** TripInfo
* **Realized by:** Itinerary, CruiseBooking
* **Variables:**
  + public String name
  + public String cabintype
  + public float price

**9. FlightInfo**

* **Aggregated by:** TravellerInfo
* **Inherits from:** TripInfo
* **Realized by:** Itinerary, FlightBooking
* **Variables:**
  + public String name
  + public String seatclass
  + public float price

**10. HotelInfo**

* **Aggregated by:** TravellerInfo
* **Inherits from:** TripInfo
* **Realized by:** Itinerary, HotelBooking
* **Variables:**
  + public String name
  + public int star
  + public String location
  + public int roomsrequested
  + public float price

**11. TripInfo**

* **Superclass of:** CruiseInfo, FlightInfo, HotelInfo
* **Variables:**
  + public int starting\_date
  + public int returning\_date
  + public String origin
  + public String destination

**12. TravellerInfo**

* **Aggregates:** CruiseInfo, FlightInfo, HotelInfo
* **Variables:**
  + public float totalprice
  + public int totalmiles
  + public String stopover

**Key Relationship Summary:**

* **Inheritance:**
  + TravelAgent ← MemberAccount
  + CruiseInfo, FlightInfo, HotelInfo ← TripInfo
* **Aggregation:**
  + Membership\_Admin → MemberAccount
  + TravellerInfo → CruiseInfo, FlightInfo, HotelInfo
* **Realization:**
  + TravelAgent → Itinerary, FlightBooking, HotelBooking, CruiseBooking
  + Itinerary, CruiseBooking → CruiseInfo
  + FlightBooking → FlightInfo
  + HotelBooking → HotelInfo
* **Association:**
  + Itinerary ↔ CruiseBooking, FlightBooking, HotelBooking, CruiseInfo, FlightInfo, HotelInfo, Membership\_Admin

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

GET — Retrieve Data

|  |  |  |  |
| --- | --- | --- | --- |
| **Endpoint** | **Purpose** | **URL** | **Notes** |
| Blogs | Get all blogs | /api/blogs | Returns all active blog posts |
| Blogs | Get a single blog | /api/blogs/:blogCode | Identified by title in the request URL |
| Latest Posts | Get all latest posts | /api/latest | Returns all active latest posts |
| Latest Posts | Get a single latest post | /api/latest/:latestCode | Identified by title in the request URL |
| Meals | Get all meals | /api/meals | Returns all active meals |
| Meals | Get a single meal | /api/meals/:mealCode | Identified by mealName in the request URL |
| News | Get all news posts | /api/news | Returns all active news posts |
| News | Get a single news post | /api/news/:newsCode | Identified by posterName in the request URL |
| Rooms | Get all rooms | /api/rooms | Returns all active rooms |
| Rooms | Get a single room | /api/rooms/:roomCode | Identified by name in the request URL |
| Testimonials | Get all testimonials | /api/testimonials | Returns all active testimonials |
| Testimonials | Get a single testimonial | /api/testimonials/:testimonialCode | Identified by person in the request URL |
| Trips | Get all trips | /api/trips | Returns all active trips |
| Trips | Get a single trip | /api/trips/:tripCode | Identified by code in the request URL |

### POST – Create New Entries

|  |  |  |
| --- | --- | --- |
| **Resource** | **URL** | **Purpose** |
| Blog | /api/blogs/ | Creates a new blog post |
| Latest Post | /api/latest/ | Creates a new latest post |
| Meal | /api/meals/ | Creates a new meal |
| News Post | /api/news/ | Creates a new news item |
| Room | /api/rooms/ | Creates a new room |
| Testimonial | /api/testimonials/ | Creates a new testimonial |
| Trip | /api/trips/ | Creates a new trip |

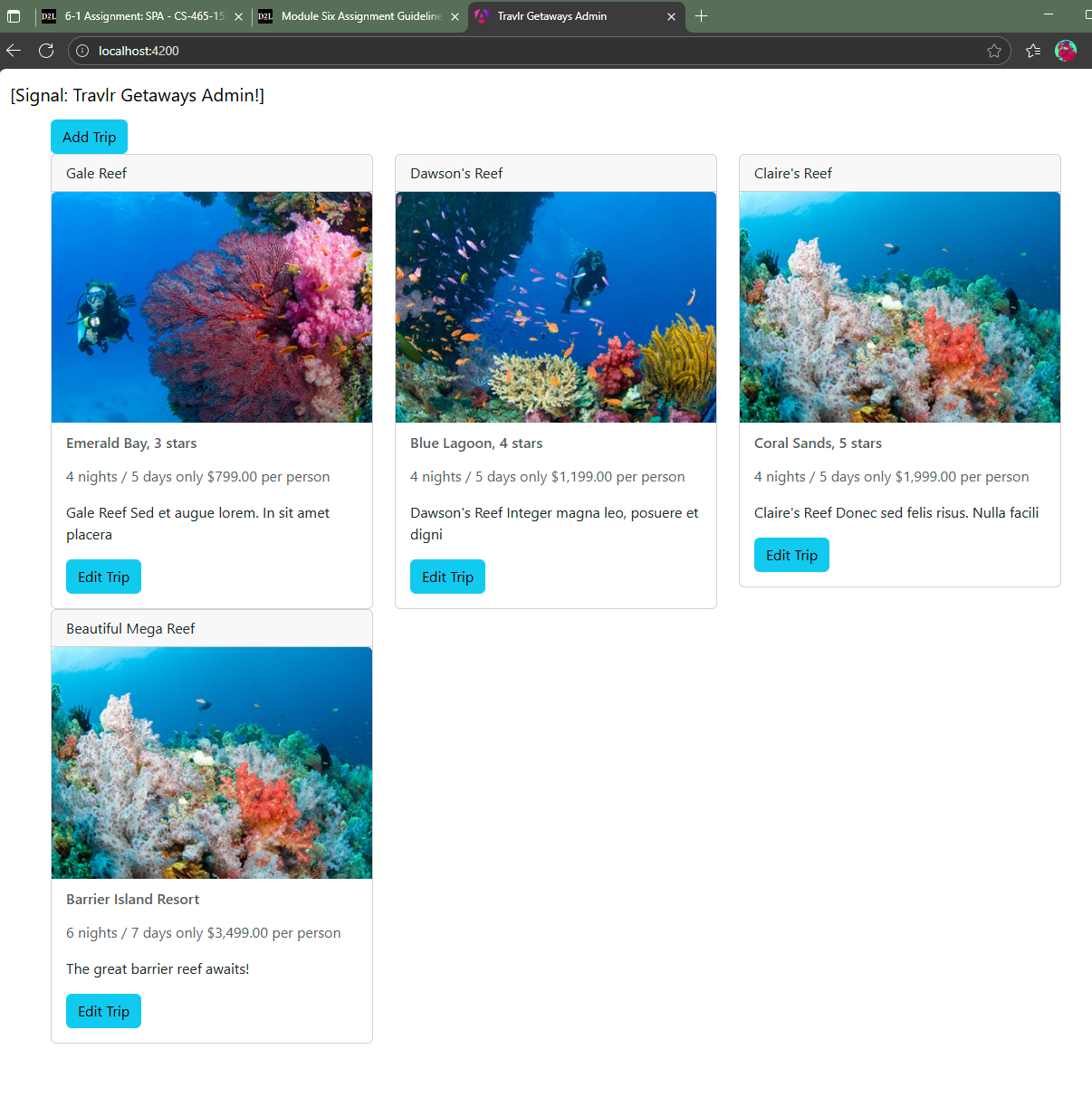
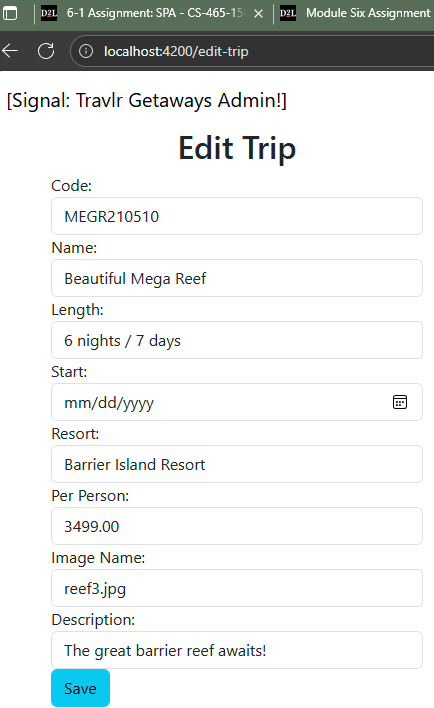
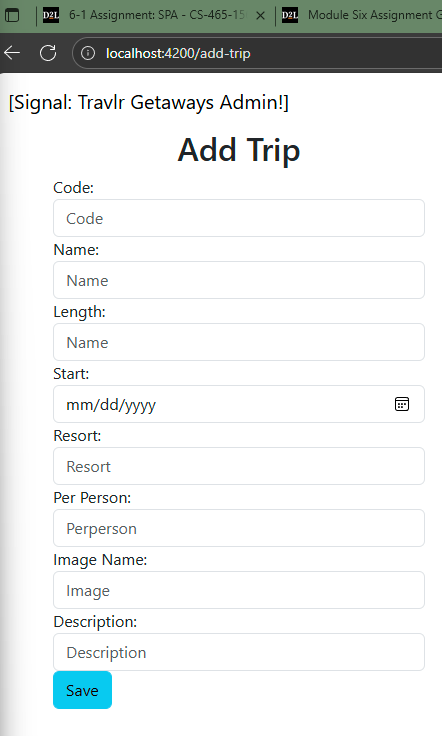
PUT — Update Existing Entries

|  |  |  |
| --- | --- | --- |
| **Resource** | **URL** | **Purpose** |
| Blog | /api/blogs/:blogCode | Updates a blog post (by title) |
| Latest Post | /api/latest/:latestCode | Updates a latest post (by title) |
| Meal | /api/meals/:mealCode | Updates a meal (by mealName) |
| News Post | /api/news/:newsCode | Updates a news item (by posterName) |
| Room | /api/rooms/:roomCode | Updates a room (by name) |
| Testimonial | /api/testimonials/:testimonialCode | Updates a testimonial (by person) |
| Trip | /api/trips/:tripCode | Updates a trip (by code) |

DELETE — Remove Entries

|  |  |  |
| --- | --- | --- |
| **Resource** | **URL** | **Purpose** |
| Blog | /api/blogs/:blogCode | Deletes a blog post (by title) |
| Latest Post | /api/latest/:latestCode | Deletes a latest post (by title) |
| Meal | /api/meals/:mealCode | Deletes a meal (by mealName) |
| News Post | /api/news/:newsCode | Deletes a news item (by posterName) |
| Room | /api/rooms/:roomCode | Deletes a room (by name) |
| Testimonial | /api/testimonials/:testimonialCode | Deletes a testimonial (by person) |
| Trip | /api/trips/:tripCode | Deletes a trip (by code) |

## The User Interface



The Angular portion of the Travlr Getaways project is structured as a Single-Page Application (SPA), while the Express portion serves as the backend API. The key difference between the two lies in workload distribution: Express handles most of the processing on the server, requiring frequent client-server communication, whereas Angular loads most of the application on the client side. After the initial load, the SPA communicates with the server only to retrieve or update data, resulting in faster, more seamless interactions for the user.

SPAs provide a richer user experience compared to traditional web applications. Instead of reloading the entire page for each interaction, Angular dynamically updates components in response to user actions. This enables smoother navigation, real-time updates, and interactive features such as form validation, booking updates, and live trip listings. While SPAs may have a longer initial load time, Travlr Getaways is a smaller application, and users are likely to spend time exploring trip details. Offloading much of the work to the client browser reduces server strain and enhances responsiveness, making an SPA an ideal choice for this project.

To ensure proper integration between the SPA and the backend API, we tested GET, POST, and PUT requests using tools such as Postman, Robo 3T, and the terminal. For example, we used Postman to POST a new trip and verified the request appeared in the terminal logs, then confirmed the data was stored in MongoDB through Robo 3T. Interactions from the Angular frontend, such as retrieving or updating trips, were monitored in the terminal to confirm successful communication. Testing the API independently from the frontend using Postman helped us efficiently validate functionality and ensure the SPA correctly interacts with the database.