Full Stack Web Application
CS 465 Project Software Design Document

Version 1.2

Table of Contents

Document Revision History

Version	Date	Author	Comments
1.0	9/21/2024	Bryce Jensen	Completed the first part of the document.
1.1	10/4/2024	Bryce Jensen	Added Sequence and Class Diagrams and API Endpoints
1.2	10/17/2024	Bryce Jensen	Added more API Endpoints, UI Screenshots, and the Angular Summary

1. Executive Summary

Travlr Getaways has requested a full-stack web application for its site. It will utilize the MEAN stack, which consists of MongoDB, Express, Angular, and Node.js. This architecture allows for developing a full-stack web application, where the customer-facing and single-page applications work seamlessly together.

1.1 – Customer-Facing Application

1.1.1 – **Client-Side**

- The Angular Framework will be used to develop the customer-facing front end. It helps with the creation of dynamic and responsive web pages. It also allows the website to load more efficiently, only changing the areas of the site that need to be updated instead of reloading the entire site every time.
- HBS is the site's initial renderer. It loads a static version of the site before it creates a template and uses handlebars (HBS) to make it dynamic in Express. It also reduces redundancy by injecting the data into templates.
- The site will also be responsive, adjusting to different screen sizes to help with the user experience across different-sized devices.

1.1.2 – **Server-side (Backend)**

- The server's logic is built on Node.js and the Express framework. Express is a framework of Node.js that manages the static content and generally handles some background things so the developer can focus more on development instead of setting it up.
- The backend will follow the Model-View-Controller (MVC) pattern. Models define the data layout, Views render the initial HTML (where the HBS comes in), and Controllers handle the requests, process data, and respond to client actions.
- MongoDB is a NoSQL database. This will use MongoDB to handle customer data, product listings, bookings, and reviews. MongoDB is very flexible and ideal for different data types and complex relationships between them.

1.1.3 – Administrator Single-Page Application (SPA) Dashboard

- The admin side of the application will be a single-page application built with Angular. It will allow users to manage travel packages, user accounts, and bookings.
- Admin will have access to basic CRUD (Create, Read, Update, Delete) functionality to manage travel package data. This can also be achieved through Angular and a RESTful API.

2. Design Constraints

2.1 – Technology Constraints

- The application is required to use the MEAN stack. This restricts the choice of tools and libraries to MEAN. Developers must understand those tools well; otherwise, development can be slowed down.
- The use of MongoDB means that the data will be stored in collections, if not handled correctly this can lead to data quality issues. Source: https://www.koombea.com/blog/mongodb-advantages-and-disadvantages/
- The admin page must be an SPA using Angular. If this isn't planned correctly from the beginning, it can add complexity to the front end.

2.2 – Client Requirements

- The application must allow customers to create accounts, search for travel packages, book reservations, and view itineraries. Admins need to manage user data and travel packages. A wide range of features need to be implemented within the given time. Prioritizing functionality will be essential to meeting deadlines without sacrificing quality.
- The design must align with the client's wireframe requirements. This means that there is little flexibility in changing the layout or design. This can be challenging if the wireframe does not account for responsive design and may require some creativity from the developers.

2.3 – Performance Requirements

- Dynamic data loading is a requirement, and data must be loaded based on user actions such as searching for travel packages or viewing bookings. This means efficient data retrieval and loading are crucial for a smooth user experience (UX).
- The application should be scalable for growing user accounts and travel package options. This means the design should be set up for potential scalability issues, such as database performance and server load. Pagination and caching should be considered in the design to support growth.

2.4 – Security Constraints

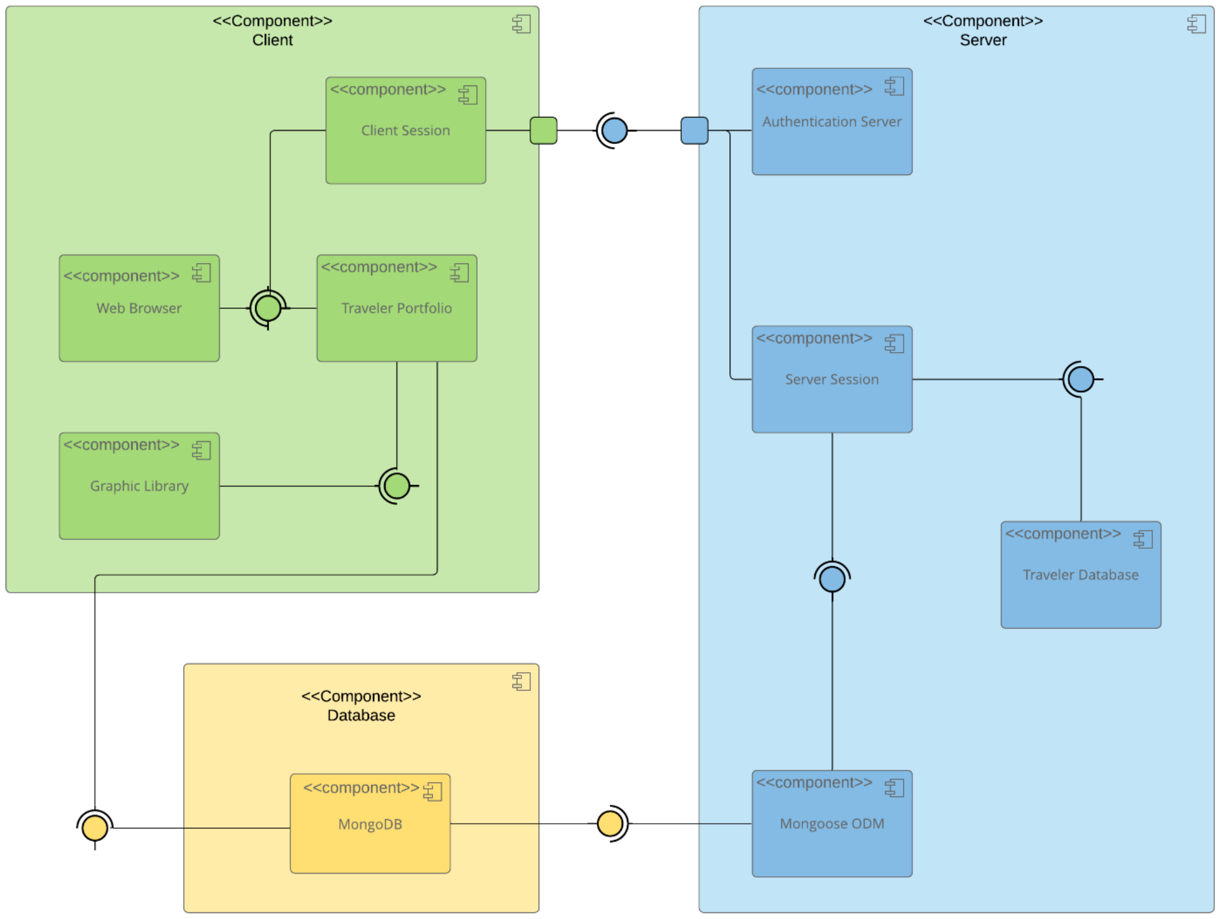
- Secure login and role-based access are required for both customer and admin areas. Implementing security features like authentication and secure API endpoints will complicate development.
- The application should comply with data protection requirements (e.g., GDPR) when handling user data. This may mean that data encryption, secure data storage, and allowing users to manage their data will need to be implemented.

2.5 – Device and Browser Compatibility

- The application must work smoothly across browsers (e.g., Chrome, Firefox, Safari) and devices (e.g., desktop, mobile). Developers will need to test the application in different environments, which can be time-consuming.

3. System Architecture View

3.1 – Component Diagram



A text version of the component diagram is available: [CS 465 Full Stack Component Diagram Text Version]().

3.1.1 – Client Components (Green Section)

- **Web Browser:** This is where users – customers and admins alike – interact with the Travlr Getaways application. It is responsible for rendering the application's front end.
- **Graphic Library:** This component handles the visual elements of the application, such as images and animations.
- **Client Session:** This represents the data stored for the current session.
- **Traveler Portfolio:** This is where users can view and manage their travel bookings, itineraries, and account information. It interacts with both the client session and the database to display the users' data.

3.1.2 – Server Components (Blue section)

- **Authentication Server:** This manages the user authentication. It handles login, logout, and user permissions, ensuring secure access to the application's traveler and admin sides.

- **Server Session:** Stores the session data on the server side, including all the necessary data for server processing.
- **Traveler Database:** This database handles the storage and retrieval of traveler-related data, such as user profiles, bookings, and itineraries.
- **Mongoose ODM:** ODM = Object data modeling. This works between the application and the MongoDB database. It simplifies CRUD operations.

3.1.3 – Database Components (Yellow Section)

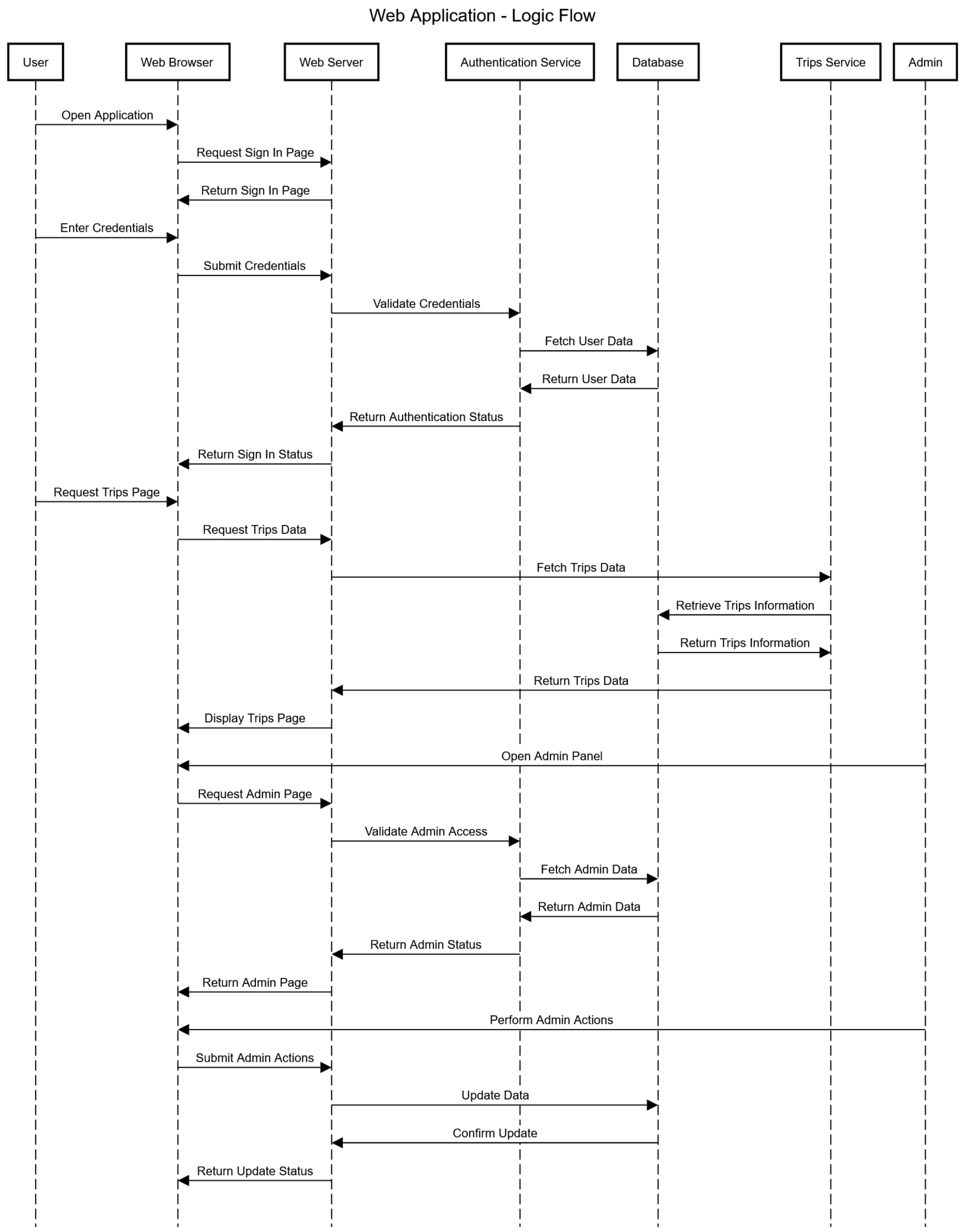
- **Mongo DB:** This is the NoSQL database used to store the application's data. It stores user profiles and their bookings, travel packages, administrative data, and more in a way suitable for scalability and flexibility.

3.1.4 – Relationships Between Components

- The **web browser** interacts with the **traveler portfolio** to display user-specific travel data. It also interacts with the **client session** to store the data from the current browsing session.
- The **traveler portfolio** communicates with **MongoDB** to fetch and update travel-related information. It also interacts with the **graphic library** to pull in images.
- The **client session** interacts with the **authentication server** to get the current user's access.
- The **authentication server** interacts with the **server session** to get the processed data for the current server session.
- The **server session** gets data like the bookings, itineraries, and profiles from the **traveler database**. It also interacts with the **Mongoose ODM** to read information from the **MongoDB** database.

4. Sequence Diagram

4.1 – Image



NOTE: The diagram was created using https://sequencediagram.org/, a fantastic tool I just found.
Link to the above diagram in the tool

4.2 – Sign in Process:

- The user opens the application in their web browser.
- The web browser sends a request to the web server for the Sign In page.
- The web server returns the Sign In page to the web browser.
- The user enters their credentials and submits them via the web browser.
- The web browser sends the credentials to the web server.
- The web server forwards the credentials to the authentication service for validation.
- The authentication service queries the database to fetch user data.
- The database returns the user data to the authentication service.
- The authentication service returns the authentication status to the web server.
- The web server sends the Sign In status back to the web browser.

4.3 – Trips Process:

- The user requests the Trips page via the web browser.
- The web browser sends a request to the web server for trips data.
- The web server forwards the request to the trips service.
- The trips service queries the database to retrieve trips information.
- The database returns the trips information to the trips service.
- The trips service sends the trips data back to the web server.
- The web server returns the Trips page with the data to the web browser.

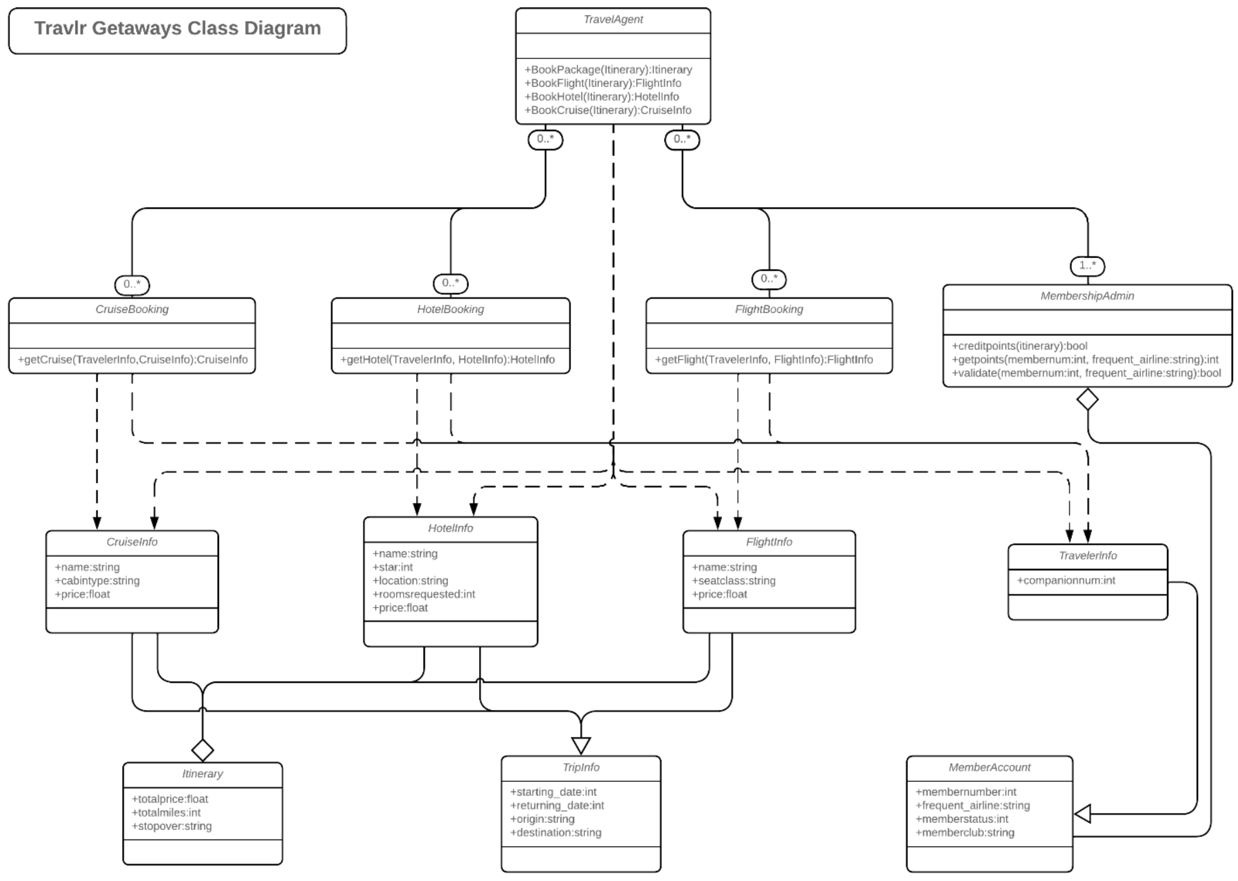
4.4 – Admin Interactions:

- The admin opens the Admin Panel in their web browser.
- The web browser sends a request to the web server for the Admin page.
- The web server forwards the request to the authentication service to validate admin access.
- The authentication service queries the database to fetch admin data.
- The database returns the admin data to the authentication service.
- The authentication service returns the admin status to the web server.
- The web server sends the Admin page back to the web browser.
- The admin performs various actions via the web browser.
- The web browser sends the admin actions to the web server.
- The web server updates the data in the database.
- The database confirms the update to the web server.
- The web server returns the update status to the web browser.

5. Class Diagram

5.1 – Classes

There are 12 classes in the Travlr Getaways application. A description of each follows.



1. TravelAgent
   - Methods:
     - `BookPackage(Itinerary) : Itinerary`
     - `BookFlight(Itinerary) : FlightInfo`
     - `BookHotel(Itinerary) : HotelInfo`
     - `BookCruise(Itinerary) : CruiseInfo`
   - This class represents a travel agent who would be responsible for booking packages, flights, hotels, and cruises. It interacts with the booking classes to retrieve specific travel information. It also gets some information from the Info classes as well.
   - JavaScript Class:

```
class TravelAgent {
  bookPackage(itinerary) {
    // Logic to book a full travel package
    return itinerary;
  }
  bookFlight(itinerary) {
    // Logic to book a flight
    return new FlightInfo();
  }
  bookHotel(itinerary) {
    // Logic to book a hotel
    return new HotelInfo();
  }
  bookCruise(itinerary) {
    // Logic to book a cruise
    return new CruiseInfo();
  }
}
```

2. CruiseBooking
   - Methods:
     - `getCruise(TravelerInfo, CruiseInfo): CruiseInfo`
   - This class handles cruise bookings. It retrieves cruise information for a given traveler.
   - JavaScript Class:

```
class CruiseBooking {
  getCruise(travelerInfo, cruiseInfo) {
  // Logic to retrieve cruise details based on traveler info
  return cruiseInfo;
  }
}
```

3. HotelBooking
   - Methods:
     - `getHotel(TravelerInfo, HotelInfo): HotelInfo`
   - This class deals with hotel bookings. It retrieves hotel details based on the traveler's info and preferences.
   - JavaScript Class:

```
class HotelBooking {
  getHotel(travelerInfo, hotelInfo) {
    // Logic to get hotel info based on traveler preferences
    return hotelInfo;
  }
}
```

4. FlightBooking
   - Methods:
     - `getFlights(TravelerInfo, FlightInfo): FlightInfo`
   - This class manages flight bookings. It retrieves flight information for a traveler based on their itinerary and preferences.
   - JavaScript Class:

```
class FlightBooking {
  getFlight(travelerInfo, flightInfo) {
    // Logic to get flight info based on traveler
    return flightInfo;
  }
}
```

5. MembershipAdmin
   - Methods:
     - `creditPoints(Itinerary): bool`
     - `getPoints(memberNum, frequent_airline): int`
     - `validate(memberNum, frequent_airline): bool`
   - This class is responsible for managing the membership and frequent flyer information for the users. It calculates and credits points based on the user's bookings.
   - JavaScript Class:

```
class MembershipAdmin {
  creditPoints(itinerary) {
    // Logic to credit points for itinerary
    return true;
  }
  getPoints(memberNum, frequentAirline) {
    // Logic to get points for a member
    return 100; // Example points
  }
  validate(memberNum, frequentAirline) {
    // Validate membership details
    return true;
  }
}
```

6. CruiseInfo
   - Attributes:
     - `name: string`
     - `cabinType: string`
     - `price: float`
   - This class contains information specific to a cruise, such as cruise name, cabin type, and price.
   - JavaScript Class:

```
class CruiseInfo {
  constructor(name, cabinType, price) {
    this.name = name;
    this.cabinType = cabinType;
    this.price = price;
  }
}
```

7. HotelInfo
   - Attributes:
     - `name: string`
     - `star: int`
     - `location: string`
     - `roomsRequested: int`
     - `price: float`
   - This class holds information about the hotel, including its name, rating, location, and room details.
   - JavaScript Class:

```
class HotelInfo {
  constructor(name, star, location, roomsRequested, price) {
    this.name = name;
    this.star = star;
    this.location = location;
    this.roomsRequested = roomsRequested;
    this.price = price;
  }
}
```

8. FlightInfo
   - Attributes
     - `name: string`
     - `seatClass: string`
     - `price: float`
   - This class contains flight information such as the airline name, seat class, and price.

- JavaScript Class:

```
class FlightInfo {
  constructor(name, seatClass, price) {
    this.name = name;
    this.seatClass = seatClass;
    this.price = price;
  }
}
```

9. TravelerInfo
    - Attributes:
        - companionNum: int
    - This class holds information about the traveler, including how many companions they are traveling with.
    - JavaScript Class:

```
class TravelerInfo {
  constructor(companionNum) {
    this.companionNum = companionNum;
  }
}
```

10. Itinerary
    - Attributes:
        - `totalPrice: float`
        - `totalMiles: int`
        - `stopOver: string`
    - This class represents an itinerary, which includes the total price, total miles traveled, and stopover information.
    - JavaScript Class:

```
class Itinerary {
  constructor(totalPrice, totalMiles, stopOver) {
    this.totalPrice = totalPrice;
    this.totalMiles = totalMiles;
    this.stopOver = stopOver;
  }
}
```

11. TripInfo
    - Attributes:
        - `startingDate: int`
        - `returningDate: int`
        - `origin: string`
        - `destination: string`
    - This class holds general information about a trip, including the start and return dates and locations.
    - JavaScript Class:

```
class TripInfo {
  constructor(startingDate, returningDate, origin, destination) {
    this.startingDate = startingDate;
    this.returningDate = returningDate;
    this.origin = origin;
    this.destination = destination;
  }
}
```

12. MemberAccount

- Attributes:
    - `memberNum: int`
    - `frequentAirline: string`
    - `memberStatus: int`
    - `memberClub: string`
- This class holds information about the member's account, including their frequent airline, status, and club membership.
- JavaScript Class:

```
class MemberAccount {
  constructor(memberNum, frequentAirline, memberStatus, memberClub) {
    this.memberNum = memberNum;
    this.frequentAirline = frequentAirline;
    this.memberStatus = memberStatus;
    this.memberClub = memberClub;
  }
}
```

5.2 – Relationships and Associations:

- The TravelAgent class interacts with the _Booking classes to handle the booking of different travel components.
- The MembershipAdmin class interacts with MembershipAccount and helps manage membership information.
- Classes like Itinerary, TravelerInfo, CruiseInfo, FlightInfo, and HotelInfo hold details that are used in the booking process.

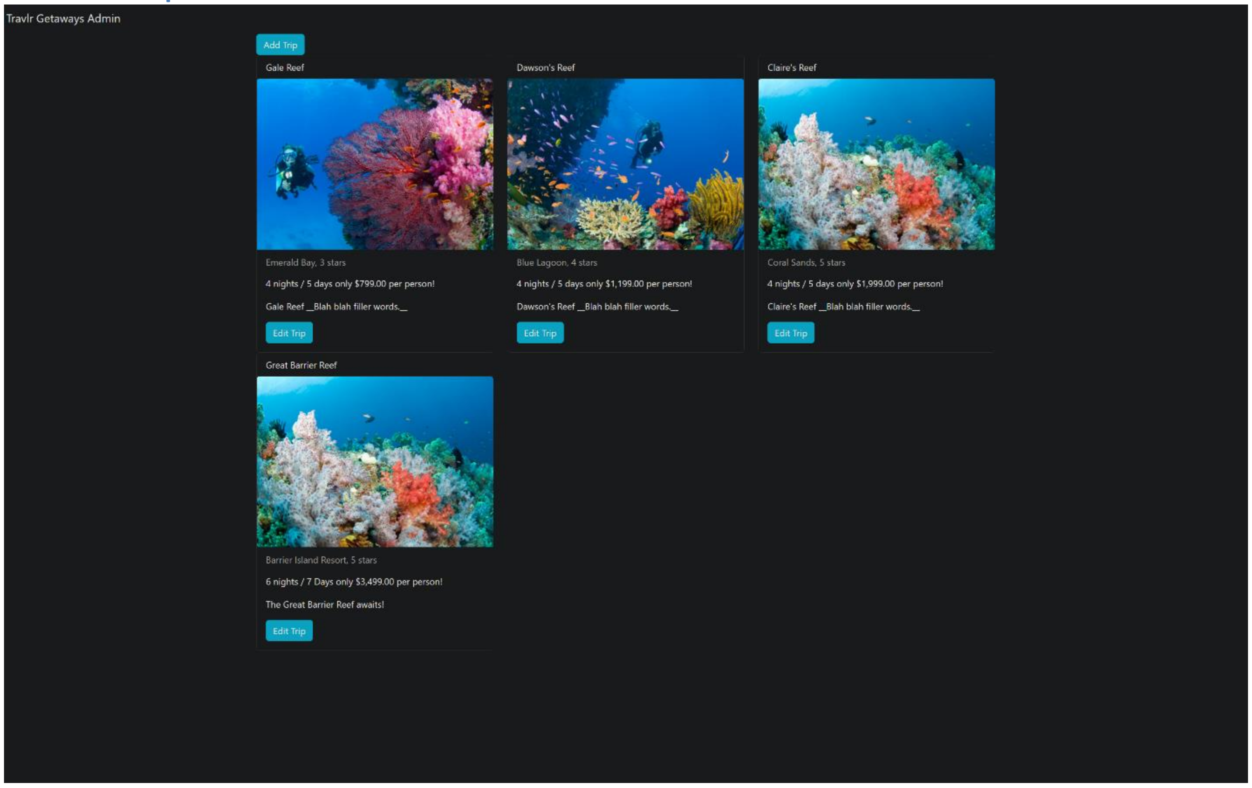
6. API Endpoints

Method	Purpose	URL	Notes
GET	Retrieve a complete list of all trips.	/api/trips	Returns all active trips in the trips DB
GET	Retrieve single trip.	/api/trips/:tripCode	Returns single trip instance, identified by the trip ID passed in the request URL
POST	Login a user	/api/login	Authenticates the user's identity, returns a JSON Web Token (JWT)
POST	Register a new user	/api/register	Adds a new user to the users DB, returns a JWT
POST	Add a trip	/api/trips	Adds a new trip to the trips DB
PUT	Edit an existing trip	/api/trips/:tripCode	Updates the details of a single trip, identified by the trip ID passed in the request URL

Note: This list is incomplete for the completed site; it only contains API Endpoints currently implemented on the Travlr Site.

7. The User Interface

7.1 – New Trip Added



Travlr Getaways Admin

Add Trip

Gale Reef

Emerald Bay, 3 stars

4 nights / 5 days only $799.00 per person!

Gale Reef _Blah blah filler words._

Edit Trip

Dawson's Reef

Blue Lagoon, 4 stars

4 nights / 5 days only $1,199.00 per person!

Dawson's Reef _Blah blah filler words._

Edit Trip

Claire's Reef

Coral Sands, 5 stars

4 nights / 5 days only $1,999.00 per person!

Claire's Reef _Blah blah filler words._

Edit Trip

Great Barrier Reef

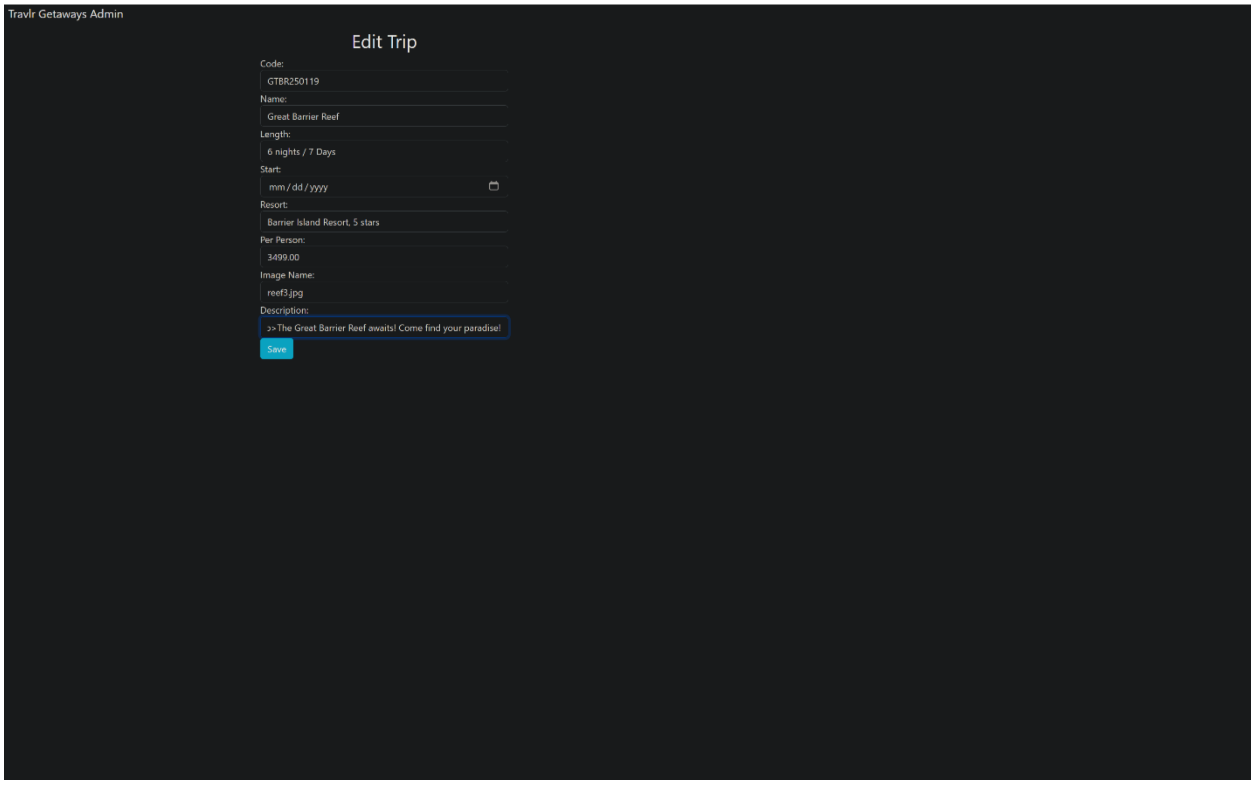
Barrier Island Resort, 5 stars

6 nights / 7 Days only $3,499.00 per person!

The Great Barrier Reef awaits!

Edit Trip

7.2 – Edit Screen



Travlr Getaways Admin

Edit Trip

Code: GTBR250119

Name: Great Barrier Reef

Length: 6 nights / 7 Days

Start: mm / dd / yyyy

Resort: Barrier Island Resort, 5 stars

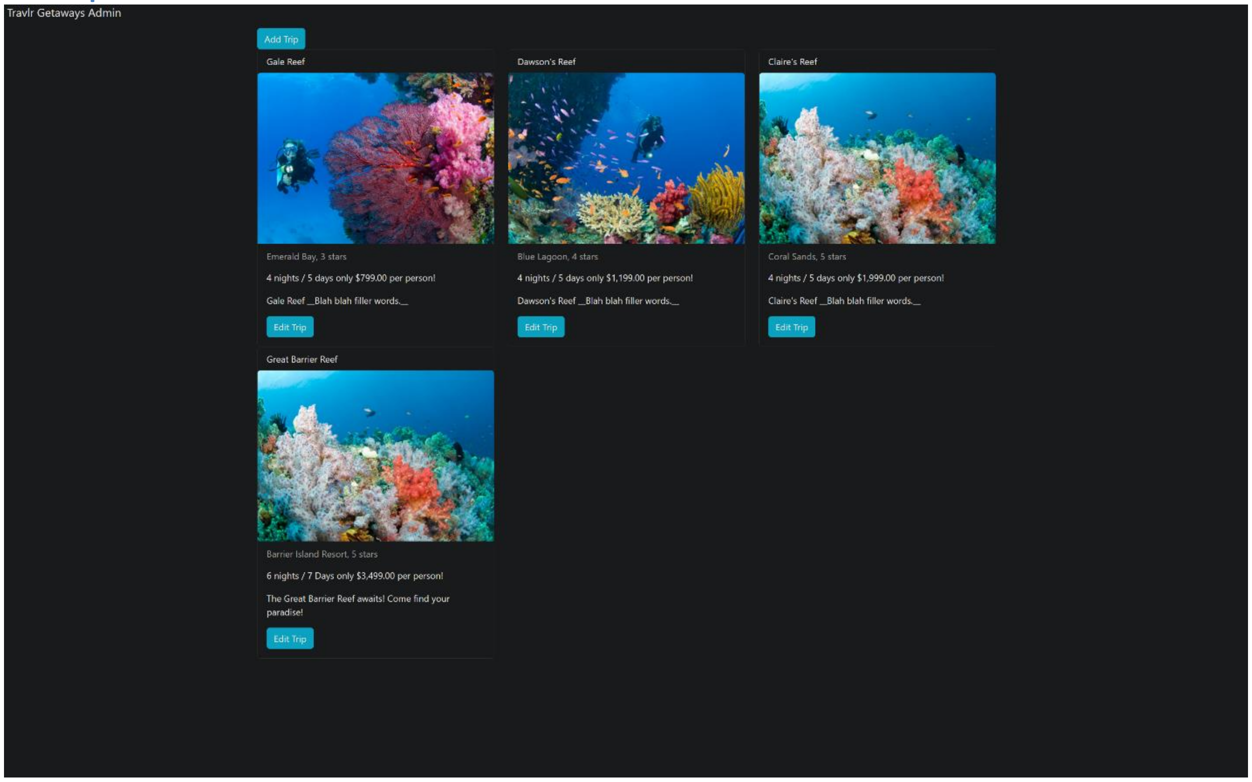
Per Person: 3499.00

Image Name: reef3.jpg

Description: >>The Great Barrier Reef awaits! Come find your paradise!

Save

7.3 – Updated Screen



Travlr Getaways Admin

Gale Reef

Emerald Bay, 3 stars

4 nights / 5 days only $799.00 per person!

Gale Reef __Blah blah filler words.__

Dawson's Reef

Blue Lagoon, 4 stars

4 nights / 5 days only $1,199.00 per person!

Dawson's Reef __Blah blah filler words.__

Claire's Reef

Coral Sands, 5 stars

4 nights / 5 days only $1,999.00 per person!

Claire's Reef __Blah blah filler words.__

Great Barrier Reef

Barrier Island Resort, 5 stars

6 nights / 7 Days only $3,499.00 per person!

The Great Barrier Reef awaits! Come find your paradise!

8. Angular Project Summary

The Angular project is designed as a single-page application (SPA), which is much simpler than the Express multi-page application (MPA). While they contain the same data, the Express MPA used HTML and Handlebars to render static pages that needed to be reloaded to be seen. The Angular SPA used CORS to dynamically load and update the content (without page reloads). The Angular site gave the ability to create an admin and client side of the application, changing the abilities of the page based on who was logged in.

Testing was simple enough. I used Postman (an API testing tool) to send requests to the API and verify that the data was acting as expected on the back end.