Travlr Getaways Web Application

CS 465 Project Software Design Document

Version 1.0

Table of Contents

Document Revision History

Version	Date	Author	Comments
1.0	07/12/2024	Kalee Li	Executive Summary, Design Constraints and System Architecture view.
2.0	07/22/2024	Kalee Li	Updated Design constraints based on Feedback
3.0	7/30/2024	Kalee Li	Sequence Diagram, Class Diagram and API Endpoints.
4.0	8/7/2024	Kalee Li	Updated API Endpoints based on Feedback
5.0	8/13/2024	Kalee Li	The User Interface, Compare Angualar and Express structurally, Testing Process.

Executive Summary

This project is to create a travel booking web application for our client, Travlr Getaways. This application allows Travlr Getaways' customers to create accounts, search for packages by location and pricing, make reservations, and review trip itineraries. Additionally, an admin site will be implemented for Travlr Getaways administrator to manage customer interaction, available trip packages, and pricing for each item and package.

Customer	Admin
• Create Account • Search Packages by using location and Price • Make reservation • Review Trip Itineraries	• Add and edit trips • Check available trip packages and pricing • Check customer reservation

The Travlr Getaways web application will be built using the MEAN stack, which comprises MongoDB, Express, Angular, and Node.js. This architecture supports the customer-facing side and administrator single-page application SPA to ensure optimal user experience.

The outline of the components and features of the MEAN Stack

MongoDB The database	**Express** The web framework for Nodes.js	**Angular** The JavaScript front-end Framework	**Node.js** The Web Server
• Type: NoSQL • Data Format: JSON • Storage: Document/Collection • Schema: No fixed columns	• Functionality: Routing and handling HTTP methods • APIs: Creates RESTful APIs • Use URL routing and MVC pattern	• Purpose: Create the interface for the web application • Features: Data Modeling One/Two-way Data Binding	• Approach: Single-threaded • Capability: Scalable network applications

Design Constraints

The Travel Engine Application has two main components:

- **A public-facing side:**
  - Features: Displays travel package descriptions and pricing, allowing customers to search and view their itineraries.
  - User Access: Accessible via web browsers, following standard HTTP/HTTPS protocols.
- **Administrator interface:**
  - Feature: allows Travlr getaway agents to manage the available packages**.**
  - U**s**er Access: Secure web portal through web browser following Role-based access control.

As mentioned, The Travlr Getaways web application will be built using the MEAN stack. However, Angular poses a few constraints for the public-facing side, such as **Search Engine Optimization (SEO) struggle, analytics integration, and slower initial page load.**

Search Engine Optimization (SEO): Angular (SPAs) relies on client-side rendering content, which can cause SEO crawlers to struggle to index dynamic content on the user's browser. Today, SPA has been significantly improved, and it is crawlable. We can use server-side rendering or pre-rending to tackle the SEO struggle, but it may add complexity to the project.
https://developers.google.com/search/docs/crawling-indexing/javascript/fix-search-javascript

Analytics integration: Angular is a single-page application handling content updates within the same, making it difficult for traditional analytics packages to track user behavior accurately. This complexity can impede the ability to provide valuable insights necessary for data-driven decision-making.

Slower initial page load: SPAs have a slower first-page load due to the complexity of initial rendering. The browser has to load JavaScript to initialize the framework and set up the application's code before rendering the required view as HTML, which takes longer than loading a simple HTML page and affects user experience.

Suggestion: To address the challenges that SPAs pose for the public view interface, I suggest using Express within the MEAN stack to serve content directly from the server. Express, as mentioned, is a web application framework with template engines to build HTML on the server. Moreover, integrating MongoDB with express application to keep track of user sessions and continue to use angular to create individual interactive components. This approach allows us to ensure the web application uses MEAN stack, end-to-end JavaScript, searchability, faster loading speed, and track user sessions and website performance.

Security: Both public-facing and Administrative interfaces will support the creation of unique credentials.

- Customer accounts: manage their itineraries
- Adminstrator accounts: Manage travel packages, pricing, and customer reservations.

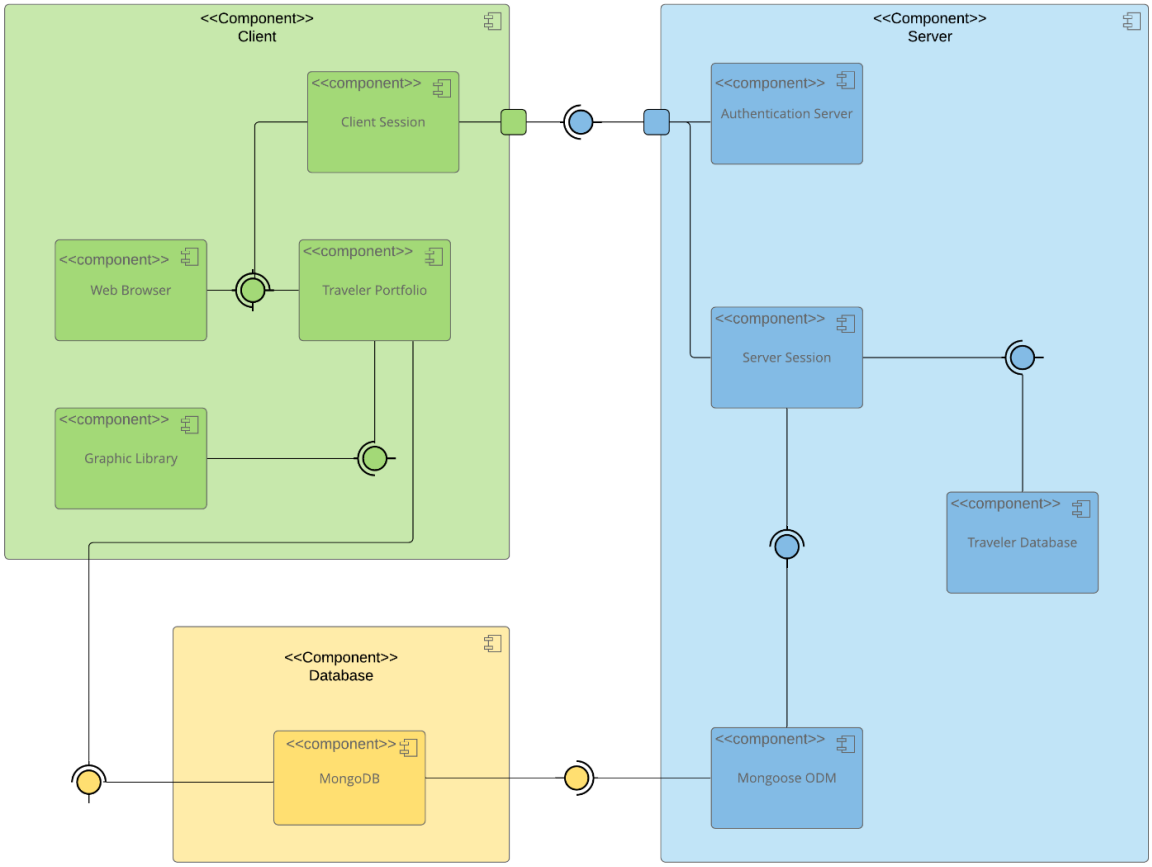
These interfaces will be protected with authentication, such as requiring login with unique credentials (username and password) and Role-based access control to ensure users have access only to the functionality and data necessary for their roles.

Environment: This application will be deployed in a scalable, cloud-based environment to ensure availability and performance.

Regulation: Ensure compliance with GDPR, the General Data Protection Regulation, and security to protect customer data and maintain data privacy standards. This includes but is not limited to data encryption, regular audits, user consent, and data access and deletion.

System Architecture View

Component Diagram



The component diagram above outlined the Travlr Getaway Web Application's architecture. This component diagram is divided into three main sections: Client, Server, and Database.

Client Component Box:

The components within the client component box:

- Web Browser – where users interact with the web application.
- Client session – manage client sessions (Itineraries)
- Traveler Portfolio – User profile and personal data
- Graphic Libaray – graphical content and rendering (Images, Animation, widgets, charts...etc.)

Server Component Box:

The components within the Server component box:

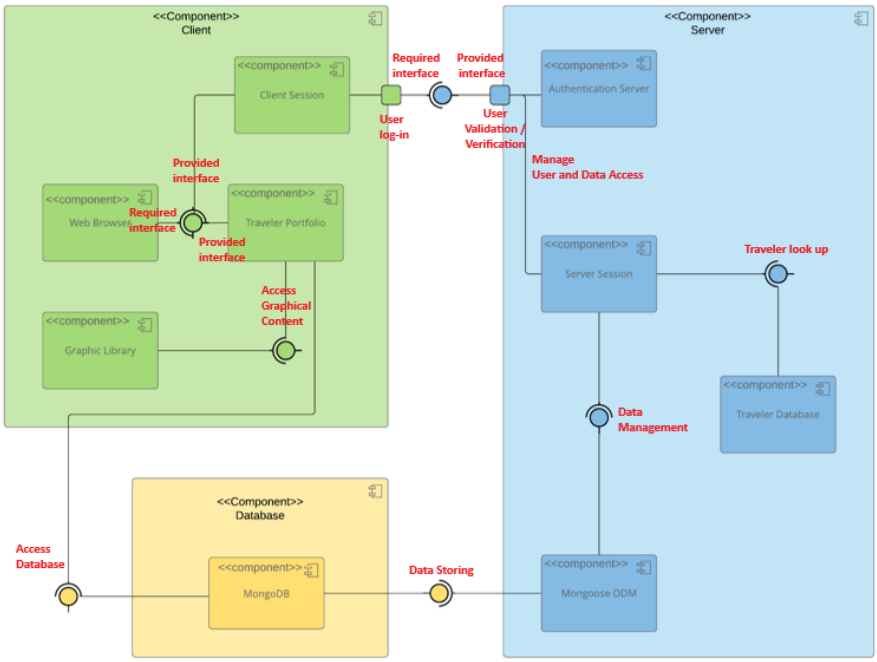
- Authentication Server – handles user authentication and security
- Server Session – manage server session. (available package, pricing, and customer interaction)
- Traveler Database – manage customer's data
- Mongoose ODM – Object Data Modeling library for MongoDB and Node.js to manage data and schema validation.

Database Component Box:

The components within the Database component box:

- MongoDB – A NoSQL database to store application data.

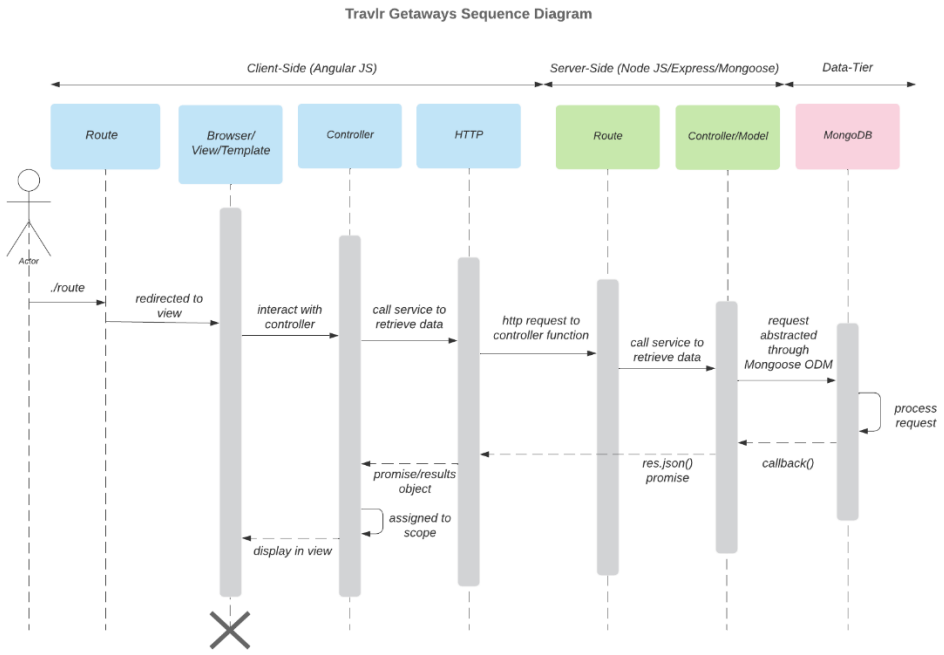
Relationship and interaction



Flow of Data and Request

- **Web Browser ↔ Traveler Portfolio ↔ MongoDB**: The web browser interacts with the traveler portfolio to display user data request data from MongoDB.
- **Web Browser ↔ Client Session ↔ Authentication Server**: The browser manages user sessions through the client session component, requiring an authentication server to provide credential validation for logging in.
- **Traveler Portfolio ↔ Graphic Library**: The traveler portfolio requests necessary graphic content from the graphic library to render user data.
- **Traveler Portfolio ↔ MongoDB**: The traveler portfolio retrieves user data in MongoDB.
- **Server Session ↔ Mongoose ODM ↔ MongoDB**: The server session uses Mongoose ODM to interact with MongoDB for storing and managing session-related data.
- **Authentication Server ↔ Server Session↔ Traveler Database**: The authentication server validates user credentials. The server session accesses traveler-related data managed by the traveler database.

Sequence Diagram



The sequence diagrams describe the flow of logic in the Travlr Getaways web application; detailing the interaction between the Client-Side (Angular JS), Server-side (Node JS/Express/Mongoose) and Data-Tier. Lets look at the Sequence Flow from left to right.

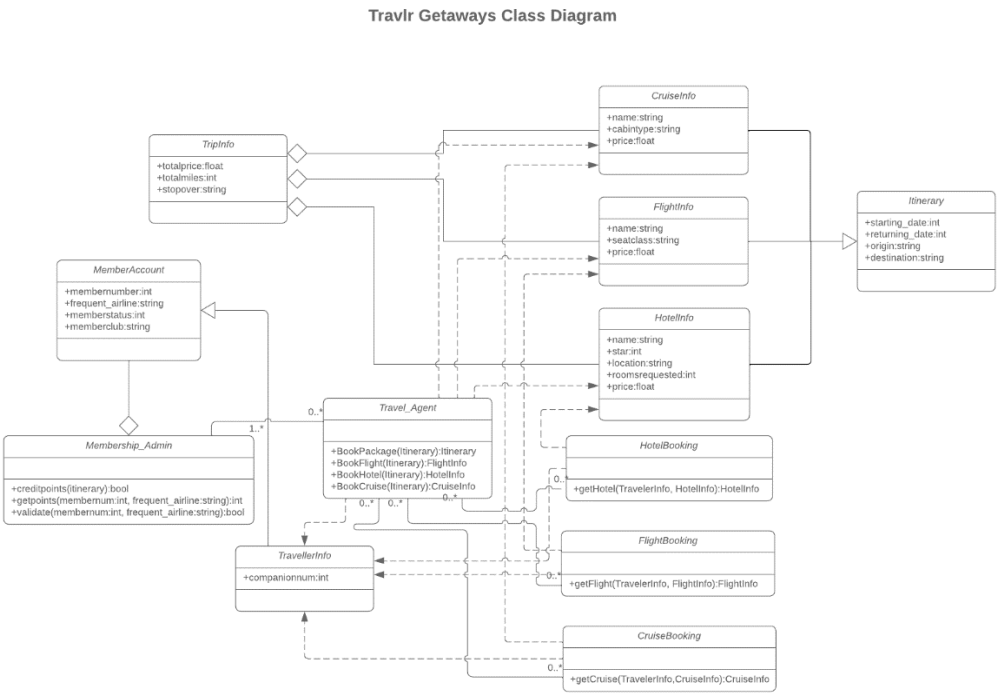
Sequence Flow

User (Actor)

- Initiates a request using a specific URL route, e.g, /singin, /trips, or /Admin.

1. Route
   - redirected to the appropriate view/template based on the URL
2. Browser /view / Template
   - Interacts with the Angular JS controller
3. Controller
   - make a call to service retrieve the necessary data such as user authentication data for sign in, trip details for trips or user management data.
4. HTTP
   - HTTP client in AngularJS sends an HTTP request to the server-side controller function
5. Route
   - Map URLs and HTTP methods to controller function
6. Controller / model
   - Model to interact with Database (Mogoose ODM)
7. MongoDB
   - Generating the response, process request.

Class Diagram



The class diagram for the Travlr Getaways web application outlines numerous JavaScript classes and their relationship, encapsulating the functionalities and data structure of information and operation.

JavaScript Classes Description:

1. TripInfo
    - Attributes: +totalprice:float +totalmiles:int +stopover:string
    - Description: This class holds general information of the trip, including cost, distance and stopovers.

2. MemberAccount
    - Attributes: +membernumber:int +frequent_airline:string +memberstatus:int +memberclub:string
    - Description: This class holds Member's account information such as member ID, the airline frequently used and current club and status

3. Membership_Admin
    - Methods: +creditpoints(itinerary):bool +getpoints(membernum:int, frequent_airline:string):int +validate(membernum:int, frequent_airline:string):bool
    - Description: This class inheriting from MemberAccount, manages administrative tasks such as credit points and validation.

4. Travel_Agent
    - Methods: +BookPackage(Itinerary):Itinerary +BookFlight(Itinerary):FlightInfo +BookHotel(Itinerary):HotelInfo +BookCruise(Itinerary):CruiseInfo
    - Description: This class handles booking packages, flights, hotels and cruises. This class interacts with Itinerary class

5. TravelerInfo
    - Attributes: +companionnum:int
    - Description: This class holds numbers of the Travelers' companions.

6. Itinerary
    - Attributes: +starting_date:int +returning_date:int +origin:string +destination:string
    - Description: This class holds a travel itineraray, details of the trips including start and return date, orgina and destionation.

7. CruiseInfo
    - Attributes: +name:string +cabintype:string +price:float
    - Description: This class holds cruise information, cruise name, cabin type and price

8. FlightInfo
    - Attributes: +name:string +seatclass:string +price:float
    - Description: This class holds flight information, flight name, seat class and price

9. HotelInfo
    - Attributes: +name:string +star:int +location:string +roomsrequested:int +price:float
    - Description: This class holds, hotel information, hotel name, star rating, location, rooms and prices

10. HotelBooking
    - Methods: +getHotel(TravelerInfo, HotelInfo):HotelInfo
    - Description: This class shows hotel booking information, based on traveler info and hotel info class

11. FlightBooking
    - Methods: +getFlight(TravelerInfo, FlightInfo):FlightInfo
    - Description: This class show flight booking information, based on traveler info and flight info class

12. CruiseBooking
    - Methods: +getCruise(TravelerInfo,CruiseInfo):CruiseInfo
    - Description: This class shows cruise booking information, based on traveler info and cruise info class

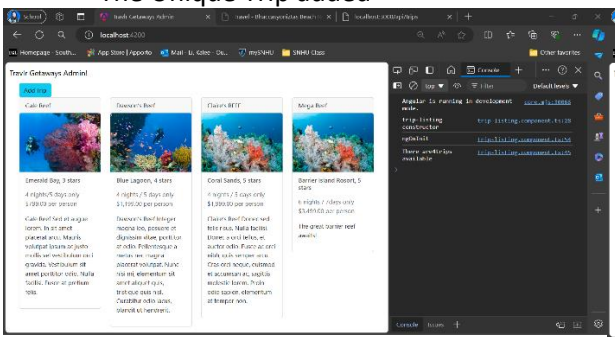
API Endpoints

The architecture of the standard of use of HTTP methods in REST API design to be used for creating a web application, includes GET, POST, PUT, PATCH and DELETE. These operations correspond to CRUD (Create, Read, Update, Delete) to manage data.

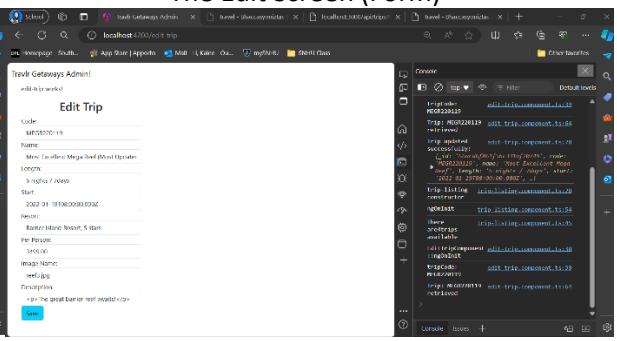
Method	Purpose	URL	Notes
GET	Retrieve list of trip	/api/trips	Returns all active trips
GET	Retrieve single trip	api/trips/trips_code	Returns single trip, identified by the trip code passed on the request URL
POST	Create a new trip Create a new credital Log in a user	/api/trips /api/register /api/login	Create a new trip with provided data Create a new credital with provided data Return JWT token with provided data
PUT	Update a single trip based on trip code	api/trips/trips_code	Update an existing trip, identified by the trip code passed on the request URL
DELETE	Remove a list of trips Remove a list of users	/api/trips /api/users	Remove all trips Remove all users
DELETE	Remove a single trip Remove a single user based on trip code	api/trips/trips_code /api/users/users_Id	Remove single trip, identified by the trip code passed on the request URL Remove single user, identified by the user ID passed on the request URL

The User Interface

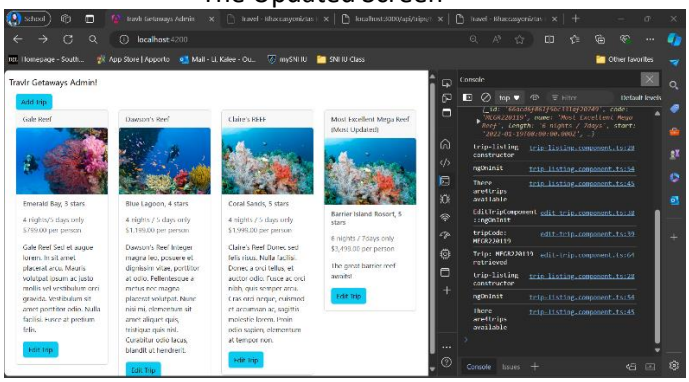
The Unique Trip added



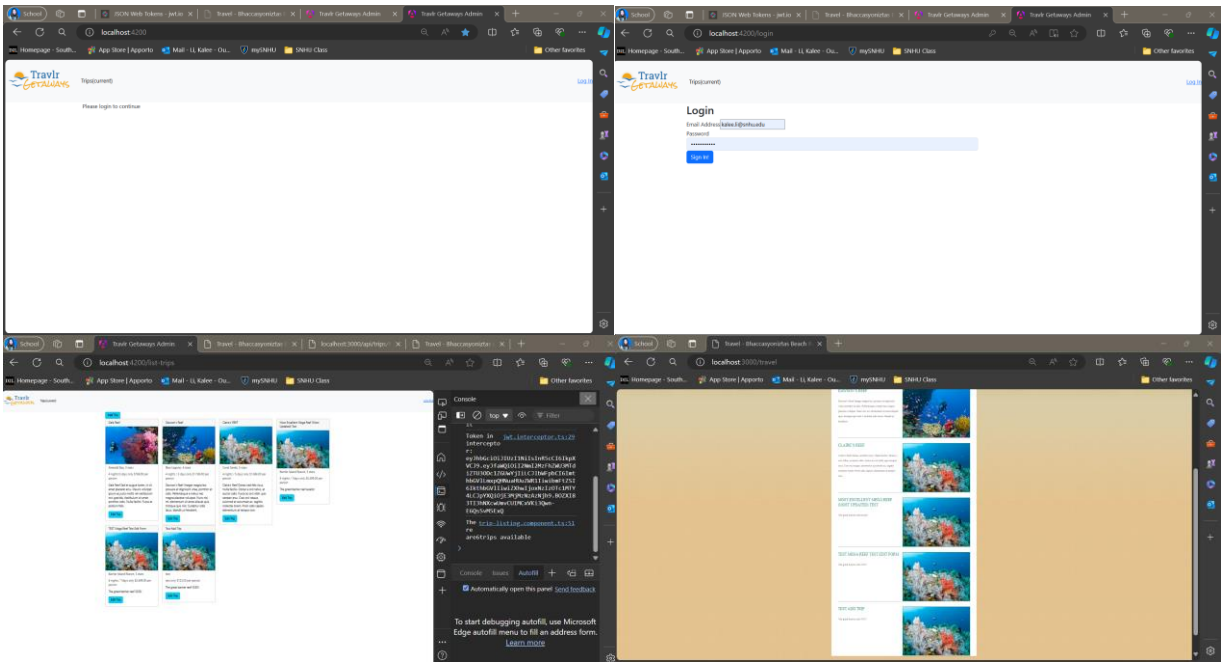
The Edit Screen (Form)



The Updated Screen

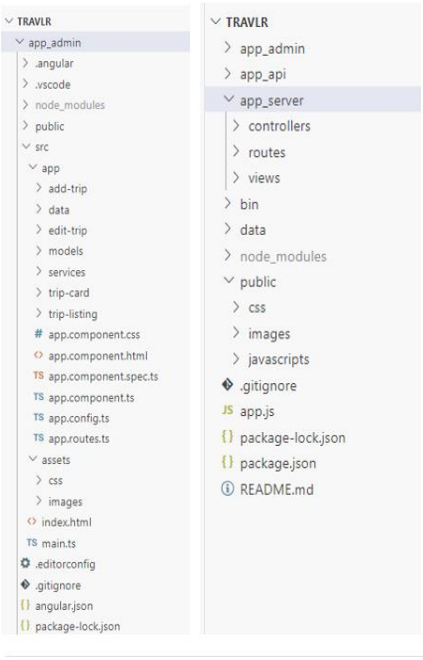


The additional screenshots for Authentication



Augular Project Structure VS Express Project Structure

While building Angular for module 6, I learned that Angular is more complicated than Express because of the initial setup and architecture. Separation of concerns (SoC) is one technique introduced in the full stack guide. Implementing SoC means dividing the application into multiple layers, interface, logic, data access, and components, then managing the interaction between these layers. For that reason, there are more files and modules to manage. Therefore, housekeeping is necessary to prepare the application structure, and frequent testing and practice are necessary. I took a screenshot of the architecture of Angular and Express to compare them structurally.



Angular Front End: app_admin/src

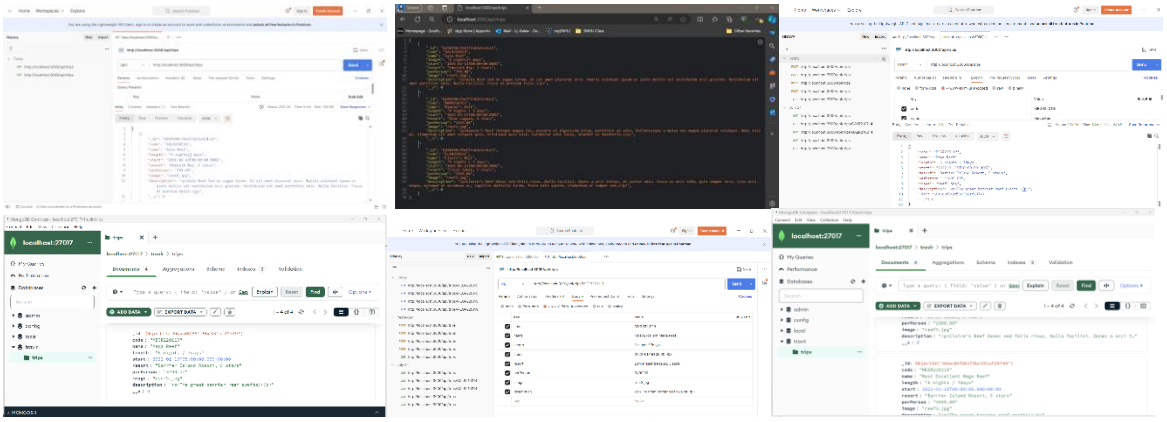
- Main app module and components – build and manage the user interface and interaction.
- Services – handle the logic request and fetching data to express
- Assets – image and style
- Other modules and components – include add-trip, edit-trip, trip-card, and trip-listing.
- Routing – Navigation between different views.

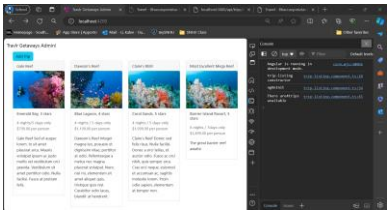
Express back end: app_server

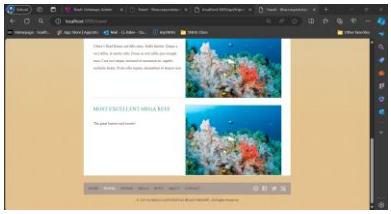
- Controllers – contain logic for handling requests and responses.
- Routes – define routes and handle requests.
- View – render dynamic HTML page template using handlebars.

Testing Process

There are many testing procedures and methods to ensure the SPA is working with the API HTTP methods. I used Postman, MongoDB compass, and manual for this project to ensure API HTTP methods were functioning. Postman is a powerful tool for testing API HTTP requests, and it was straightforward. To test with Postman, enter the endpoint URL, select the method from the drop-down menu, and click Send the request to inspect the response. Then, I would manually test using MongoDB compass and browser to ensure all methods work in the database. The stack guide provided clear and detailed instructions, so I didn't run into any technical errors, but I had a few syntax and typo mistakes. Therefore, frequent testing is essential to catch any error early. I attached some screenshots to show my testing process.







Reference:

Fix Search-Related JavaScript problems | Google Search Central | Documentation | Google for Developers. (n.d.). Google for Developers. https://developers.google.com/search/docs/crawling-indexing/javascript/fix-search-javascript

Harber, C., & Holmes, S. (n.d.). *O'Reilly Media - Technology and Business training*. O'Reilly Online Learning. https://learning.oreilly.com/library/view/getting-mean-with/9781617294754/?sso_link=yes&sso_link_from=SNHU