



Travlr Getaways Web Application
CS 465 Project Software Design Document
Version 1.0

Table of Contents

CS 465 Project Software Design Document	1
Table of Contents	2
Document Revision History	2
Instructions	2
Executive Summary	3
Design Constraints	3
System Architecture View	3
Component Diagram	3
Sequence Diagram	5
Class Diagram	7
API Endpoints	7
The User Interface	8

Document Revision History

Version	Date	Author	Comments
1.0	05/20/2025	John Bryson	Module 3 Submission
2.0	6/5/2025	John Bryson	Module 5 Submission
3.0	6/15/2025	John Bryson	Module 6 had some issues, I rectified them by doing a clean install and then continuing on in the development.
4.0	6/22/2025	John Bryson	Module 7 Set up the project structure with .env file, authentication layer to the app, app_api and app_admin. Implemented functionality with Registration, login and authentication to AddTrip, EditTrip, and added the final touches.

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

The Travlr Getaways web application is built to meet client expectations for a modern travel booking system. It uses a component-based architecture that ensures the app remains scalable and easy to maintain. The system is developed using the MEAN stack, which includes MongoDB, Express.js, Angular, and Node.js.

On the customer side, users can explore travel packages through an interactive and visually engaging interface. These packages are currently displayed using server-side Handlebars templates and populated with data from a JSON file. The admin interface is being developed as a secure Single Page Application using Angular, allowing internal staff to efficiently manage trips, content, and user accounts.

This development phase focuses on replacing static HTML with a flexible templating engine. By introducing Handlebars for dynamic content and organizing the backend with modular Express routes and controllers, the project sets the foundation for implementing full CRUD functionality and MongoDB integration in future phases.

Design Constraints

Several design constraints shape the development of the Travlr Getaways web application. First, the project must use the Node.js runtime along with the Express framework. Second, server-side rendering must be handled with Handlebars during the initial phase of development. Third, the admin portal is expected to evolve into a full Single Page Application built with Angular.

For now, trip data is retrieved from a local JSON file using `fs.readFileSync` to support fast prototyping. Although this file-based approach works for early testing, the system will later transition to MongoDB to enable persistent data storage and support for concurrent access.

These constraints influence how the application is developed, especially in the early stages. Reading from a file on every request limit performance and scalability, which is not ideal for production environments. In addition, because Handlebars rendering happens on the server, the current version of the application lacks the interactivity that Angular will bring in future phases.

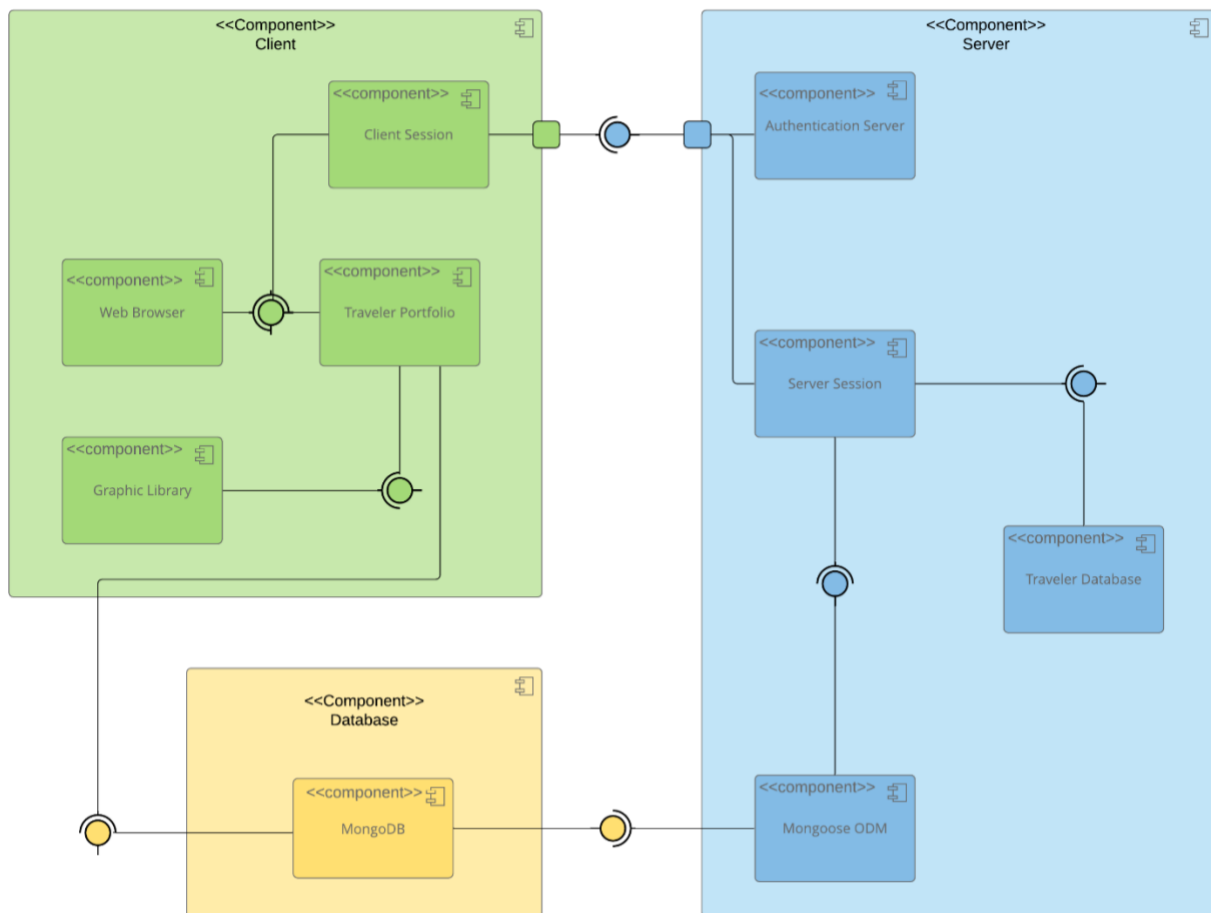
System Architecture View

Component Diagram

The Travlr Getaways system architecture is built on a layered model using the MEAN stack. It includes the following key components:

- **Client (Browser):** Serves as the user interface and sends HTTP requests to the server.
- **Express Web Server:** Manages routing and middleware logic for incoming requests.
- **Controllers:** Contain the application logic and handle data flow between the model and the view.
- **Views (Handlebars):** Render dynamic HTML content on the server side before sending it to the client.
- **Data Layer:** Currently uses a local JSON file to simulate trip data during development. This will eventually be replaced with a MongoDB database for persistent storage.
- **Node.js Runtime:** Runs the server-side code and manages input and output operations.

These components work together to create a modular and scalable web application. When a user sends a request, the Express server processes it, calls the appropriate controller, retrieves the necessary data, and renders the output using Handlebars templates. The resulting HTML is then sent back to the browser for display.



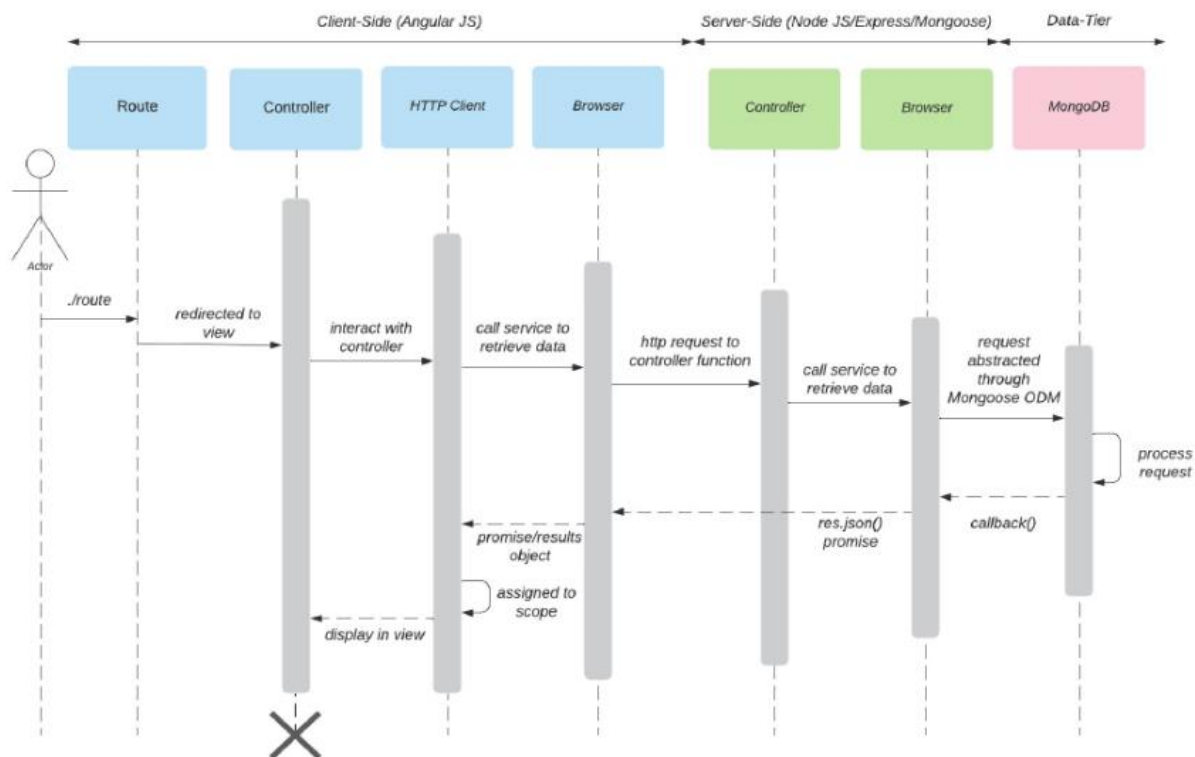
A text version of the component diagram is available: [CS 465 Full Stack Component Diagram Text Version](#).

The overall system architecture of the Travlr Getaways web application is made up of three main parts: the client side, the server side, and the database. The **client side** is what users see and interact with. It's built using a front-end framework like Angular and sends requests through HTTP to get or update data. It allows users to search vacation packages, make bookings, choose payment options, and more.

The **server side** is built with Node.js and Express. It works between the front end and the database. This part of the system handles things like checking user login information, keeping track of user sessions, and managing bookings. Express.js helps route each request to the correct function so the server knows what to do with the data.

The **database** is MongoDB. It stores all the important data like user profiles, login details, travel preferences, and vacation package information. It also keeps records of locations, descriptions, photos, prices, itineraries, and bookings. Because MongoDB is document-based, it can store all types of data in a flexible way, which is great for a travel app.

To sum it up, the main parts of the system include a user interface built with Angular or React, a server that uses Node.js and Express to handle requests, and a database like MongoDB that stores all the information the app needs.



Sequence Diagram

Step-by-Step Flow Explanation:

1. User Interaction (Actor)

The user starts the process by navigating to a route in the application (e.g., /route). This is captured by AngularJS on the **client side**.

2. Routing and View

The AngularJS **Route** component redirects the user to the correct view, and control is passed to the **Angular Controller**.

3. Controller and HTTP Request

The controller interacts with the **HTTP Client** to request data. This triggers a service call to retrieve that data, which is handled through an HTTP request sent to the server.

4. Server Controller Handling

On the **server side**, the Express **Controller** receives the HTTP request. It then uses a service to request the needed data, typically from the database.

5. Database Request

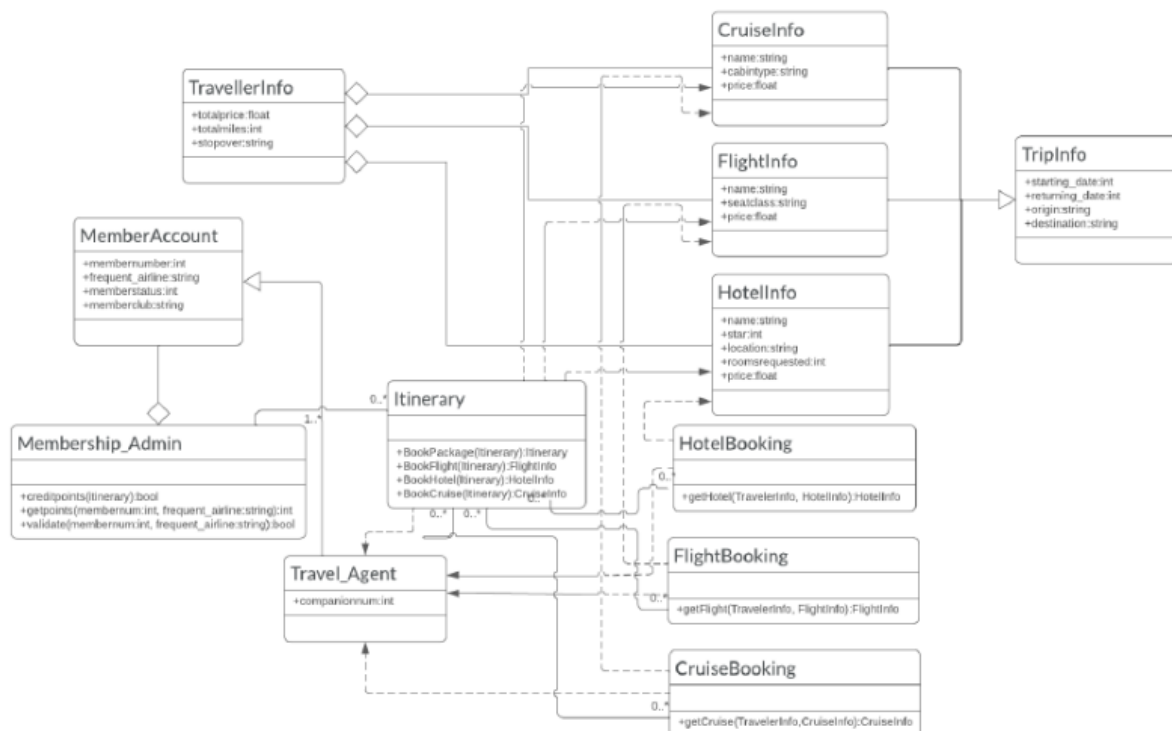
The service sends the request through **Mongoose**, an Object Data Modeling (ODM) tool that translates the request into MongoDB queries. The database processes the request and sends a **callback()** response.

6. Returning the Response

Once the server receives the data (via `res.json()`), it returns the result to the **client's browser**.

7. Displaying the Results

The **promise/results object** is assigned to the correct scope in the Angular controller, which then updates the **view** so the user can see the data.



The class diagram for the Travlr GetAways web application shows how travel bookings are managed. It includes classes that store information about individual travelers and the specific details for flights, cruises, and hotels. At the center is an **"Itinerary"** class that brings together all booking details for a single trip, which is managed by a **"Trip_Agent"** class. There is also a **"MemberAccount"** class that handles user-related information, and a **"Membership_Admin"** class that likely manages administrative access and functions.

Class Diagram

- **TravellerInfo**: Holds information about a traveler's booking such as total price, total miles, and stopover details.
- **TripInfo**: Contains the travel dates and destination details including origin, destination, start, and return dates.
- **FlightInfo, HotelInfo, CruiseInfo**: Each of these classes stores information about specific types of travel packages—such as flight seat class, hotel room type, or cruise cabin type and prices.
- **FlightBooking, HotelBooking, CruiseBooking**: These classes handle the booking process for each type of trip and include functions like `getFlight`, `getHotel`, and `getCruise` to retrieve available options based on traveler and package details.
- **Itinerary**: A central class that combines all parts of a trip. It includes methods like `BookFlight`, `BookHotel`, and `BookCruise`, and connects individual bookings under one organized plan.
- **Trip_Agent**: Helps manage bookings and may handle logic related to trip planning and companions.
- **MemberAccount**: Manages the traveler's login and membership data, such as member number, frequent flyer info, and membership status.
- **Membership_Admin**: Provides administrative control for member validation and reward point management. It includes methods for validating member data and managing loyalty rewards.

API Endpoints


<Exposing RESTful endpoints is a design approach to enable an application to participate in a larger ecosystem. Document each endpoint in the table below, including the HTTP method, purpose, URL, and notes.>

Method	Purpose	URL	Notes
GET	Retrieve a list of trips.	/api/things	Returns a list of all active trips.

GET	Retrieve a single trip.	/api/things/:thingId	Returns a single trip instance based on the trip ID passed on the URL.
POST	Creates a new list of trips.	/api/things	Creates a new list of trips.
POST	Creates a single trip.	/api/things/:thingId	Creates a single trip instance, identified by the trip ID in the URL.
PUT	Update the full trip list.	/api/things	Updates/replaces all trip records and overwrites existing data.
PUT	Update a single trip.	/api/things/:thingId	Updates/replaces a single trip using the ID in the URL.
PATCH	Modifies the full list of trips.	/api/things	Modifies all trip records with updated values.
PATCH	Modifies a single trip.	/api/things/:thingId	Modifies a single trip or updates values using the ID in the URL.
DELETE	Deletes all trips.	/api/things	Deletes all trip records.
DELETE	Deletes a single trip.	/api/things/:thingId	Deletes a single trip instance using the ID provided in the URL.

The User Interface

<Insert screenshots from the development of the SPA development to show the following: (1) a unique trip, added by you, (2) the Edit screen, and (3) the Update screen.>



Add Trip

Code:

Name:

Length:

Start:



Resort:





Per Person:

Image Name:

Description:

Save

+ Add Trip

<div>Gale Reef</div>  <div> Gale Reef Resort 5 days only \$1,200.00 per person Sed et augue lorem. In sit amet placerat arcu. Discover crystal-clear waters and diverse marine life. Relax in luxurious beachside resorts and enjoy the perfect getaway. </div> <div> Edit Trip Delete Trip </div>	<div>Dawson's Reef</div>  <div> Dawson's Reef Resort 7 days only \$1,500.00 per person Integer magna leo, posuere et dignissim vitae. Explore the beauty of coral gardens and colorful fish. Indulge in fine dining and thrilling water activities. </div> <div> Edit Trip Delete Trip </div>	<div>Claire's Reef</div>  <div> Claire's Reef Resort 4 days only \$1,500.00 per person Donec sed felis risus. Nulla facilisi. Enjoy breathtaking ocean views and sunset cruises. Experience world-class snorkeling and unwind in paradise. </div> <div> Edit Trip Delete Trip </div>	<div>Roseletha Mike</div>  <div> Roseletha Mike's Resort 8 days only \$1,100.00 per person Donec sed felis risus. Nulla facilisi. Enjoy breathtaking ocean views and sunset cruises. Experience world-class snorkeling and unwind in paradise. </div> <div> Edit Trip Delete Trip </div>
--	---	---	--

Edit Trip

Code:
GK101

Name:
Sam

Length:
3 days

Start:
11/02/2025

Resort:
Sam's Resort


Per Person:
400

Image Name:
kayak.jpg

Description:
kayak is beautiful river in the mountains of beautif

Save

+ Add Trip


Gale Reef


Gale Reef Resort

5 days only \$1,200.00 per person

Sed et augue lorem. In sit amet placerat arcu. Discover crystal-clear waters and diverse marine life. Relax in luxurious beachside resorts and enjoy the perfect getaway.

Edit Trip
Delete Trip


Dawson's Reef


Dawson's Reef Resort

7 days only \$1,500.00 per person

Integer magna leo, posuere et dignissim vitae. Explore the beauty of coral gardens and colorful fish. Indulge in fine dining and thrilling water activities.

Edit Trip
Delete Trip


Claire's Reef


Claire's Reef Resort

4 days only \$1,500.00 per person

Donec sed felis risus. Nulla facilisi. Enjoy breathtaking ocean views and sunset cruises. Experience world-class snorkeling and unwind in paradise.

Edit Trip
Delete Trip

Sam


Sam's Resort

3 days only \$400.00 per person

kayak is beautiful river in the mountains of beautiful country in solvadar, it is the hottest and beautiful place i have ever been.

Edit Trip
Delete Trip

Angular Project Structure

Angular is a front-end framework used to build single-page apps (SPAs). It's organized using components and modules, which helps keep things clean and easy to manage. Here's what the typical structure looks like:

1. **src/** – This is the main folder for all your code.
 - o **app/** – Holds everything for the app.
 - **components/** – Each folder here is for a different feature, like the navbar or login form.

- **services/** – Has code that helps with things like calling APIs (like `tripdata.service.ts` or `authentication.service.ts`).
- **models/** – Holds TypeScript files for your data, like `trip.ts` or `user.ts`.
- **app-routing.module.ts** – Handles page routing in the app.
- **assets/** – Holds images, fonts, or other files.
- **main.ts** – This is where the app starts running.
- **index.html** – The main HTML file that loads the app.
- **angular.json** – A config file for the Angular CLI.
- **package.json** – Lists all the app's dependencies and scripts.

Express Project Structure

Express is used on the backend to build APIs. Its structure is more focused on handling server-side requests. Here's what it usually looks like:

1. **app.js** – The main starting point of the server.
2. **routes/** – Where the route files live (like `trips.js` or `authentication.js`).
3. **controllers/** – Handles the logic for each route (like `tripsController.js` or `authController.js`).
4. **models/** – Where the database models are (like `trip.js` or `user.js`).
5. **config/** – Stores configuration files (like `passport.js` for handling login).
6. **package.json** – Lists dependencies and scripts, just like in Angular.

Single Page App (SPA) vs. Simple Web App

SPA (Single Page Application)

- **No Full Page Reloads** – SPAs load just one HTML page, and update it without reloading. This makes things smoother and faster.
- **Client-Side Routing** – Angular changes views on the client side, so clicking around the site feels instant.
- **Interactive UI** – Angular makes it easy to build cool features like pop-ups, forms, and live updates.
- **Works With APIs** – SPAs talk to backend APIs (like Express) to save or get data, handle login, and more.

Simple Web Application

- **Full Page Reloads** – Every time you click something, the page reloads, which can feel slow.
- **Server-Side Rendering** – The server sends a new HTML page every time you do something.
- **Not Very Interactive** – Simple apps usually don't have dynamic features or real-time updates.

Testing the SPA with the API

To make sure your Angular app and Express API work well together:

- Use a tool like **Postman** to test your API endpoints (like `/api/trips` or `/api/login`).
- Check if the responses are correct, like getting a 200 OK when things work, or 401 Unauthorized if access is denied.