

Mohamed Elmarzougui

# **CS 465 Project Software Design Document**

Version 3.0

## 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 | 05/22/25 | Mohamed Elmarzougui | <Brief description of changes in this revision> |
| 2.0 | 06/08/2025 | Mohamed Elmarzougui | <Charts and API Endpoints> |
| 3.0 | 06/22/25 | Mohamed Elmarzougui | <Final update (project completed)> |

## 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](#_heading=h.35nkun2)

Travlr Getaways is developing a full stack web application using the MEAN stack (MongoDB, Express.js, Angular, Node.js). This application will include a customer-facing single-page application (SPA) that allows users to browse, book, and interact with travel services quickly and easily. On the admin side, a separate SPA will enable staff to manage listings, bookings, and user data. The system architecture supports efficient data flow between the front end and back end, using Angular for the UI, Node.js and Express.js for the server logic, and MongoDB for secure data storage.

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

Developing the web-based Travlr Getaways application involves several constraints. Staying within the client’s budget may limit the number of developers or the scope of features we can include. Time is also a critical factor if the client expects a faster delivery than the current resources allow, we may need to adjust the timeline or prioritize key features. Balancing cost, time, and project requirements will be essential to delivering a successful application.

## [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 overall system architecture of the Travlr Getaways web application is divided into three main layers: Client, Server, and Database, as shown above in the diagram. The Client layer includes the Web Browser, Traveler Portfolio, Client Session, and Graphic Library components, which provide the user interface and session management. These components interact with the Server layer, which contains the Authentication Server, Server Session, and Traveler Database components. The Server manages user authentication, business logic, and communication with the database. The Database layer includes MongoDB, which stores all travel data. The components are connected through well-defined communication points, ensuring smooth interaction between the front end, back end, and data layer.

### Sequence Diagram

The sequence diagram begins with the actor (user) initiating a request by accessing a specific route in the application. The AngularJS frontend router interprets this route and redirects the user to the appropriate view/template.

The view interacts with a frontend controller, which is responsible for managing the view’s logic and data. The controller makes a call to a function in the HTTP service, which handles communication with the backend.

The HTTP service sends an HTTP request to a designated API endpoint on the server. On the server side, the Express router receives the request and invokes the corresponding backend controller function.

The backend controller then interacts with the MongoDB database using Mongoose, an Object Data Modeling (ODM) library. Mongoose abstracts the database queries and sends them to MongoDB.

MongoDB processes the request and sends the data back through the callback chain. The backend controller receives this data and sends a JSON response back to the frontend via the HTTP service.

Finally, the HTTP service passes the results to the frontend controller, which assigns the data to the view’s scope, allowing the updated information to be displayed to the user.

A diagram of a trade end

AI-generated content may be incorrect.

## Class Diagram

The CruiseInfo, FlightInfo, and HotelInfo classes each contain a name property along with additional fields unique to their respective travel types such as cabintype for cruises, seatclass for flights, and location for hotels. All three of these classes are associated with the TripInfo class, which stores shared trip details including starting\_date, returning\_date, origin, and destination.

The CruiseBooking, FlightBooking, and HotelBooking classes each have an association with both their respective Info class and the TravellerInfo class. These Booking classes are connected to the TravelAgent class through zero-to-many relationships in both directions, indicating that a travel agent may handle multiple bookings and vice versa.

The TravelAgent class is also associated with the CruiseInfo, FlightInfo, HotelInfo, and TravellerInfo classes, and it maintains a one-to-many relationship with the Membership\_Admin class.

The TravellerInfo class aggregates the TripInfo class and inherits from the MemberAccount class, which holds user-specific membership data such as membernumber, frequent\_airline, and memberstatus.

The Membership\_Admin class has an aggregate relationship with the MemberAccount class, meaning it manages one or more member accounts. The Itinerary class aggregates CruiseInfo, FlightInfo, and HotelInfo, indicating that a single itinerary may include multiple travel elements across different transport and accommodation types.

A diagram of a travel geograph

AI-generated content may be incorrect.

## [API](#_heading=h.2jxsxqh) 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** |
| --- | --- | --- | --- |
| **POST** | Login a user | /api/login | Authenticates a user |
| **GET** | Retrieve 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 single trip,  identified by the trip code at  end of URL |
| **PUT** | Update single trip | /api/trips/:tripCode | Updates single trip, identified by the trip code at end of URL |
| **DELETE** | Delete single trip | /api/trips/:tripCode | Deletes single trip, identified by the trip code at end of URL |
| **POST** | Login a user | /api/login | Takes an email and password to login a user |

## 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.>

A screenshot of a computer

AI-generated content may be incorrect.

Figure 1: User is logged In

A screenshot of a computer

AI-generated content may be incorrect.

Figure 2: Adding a Trip

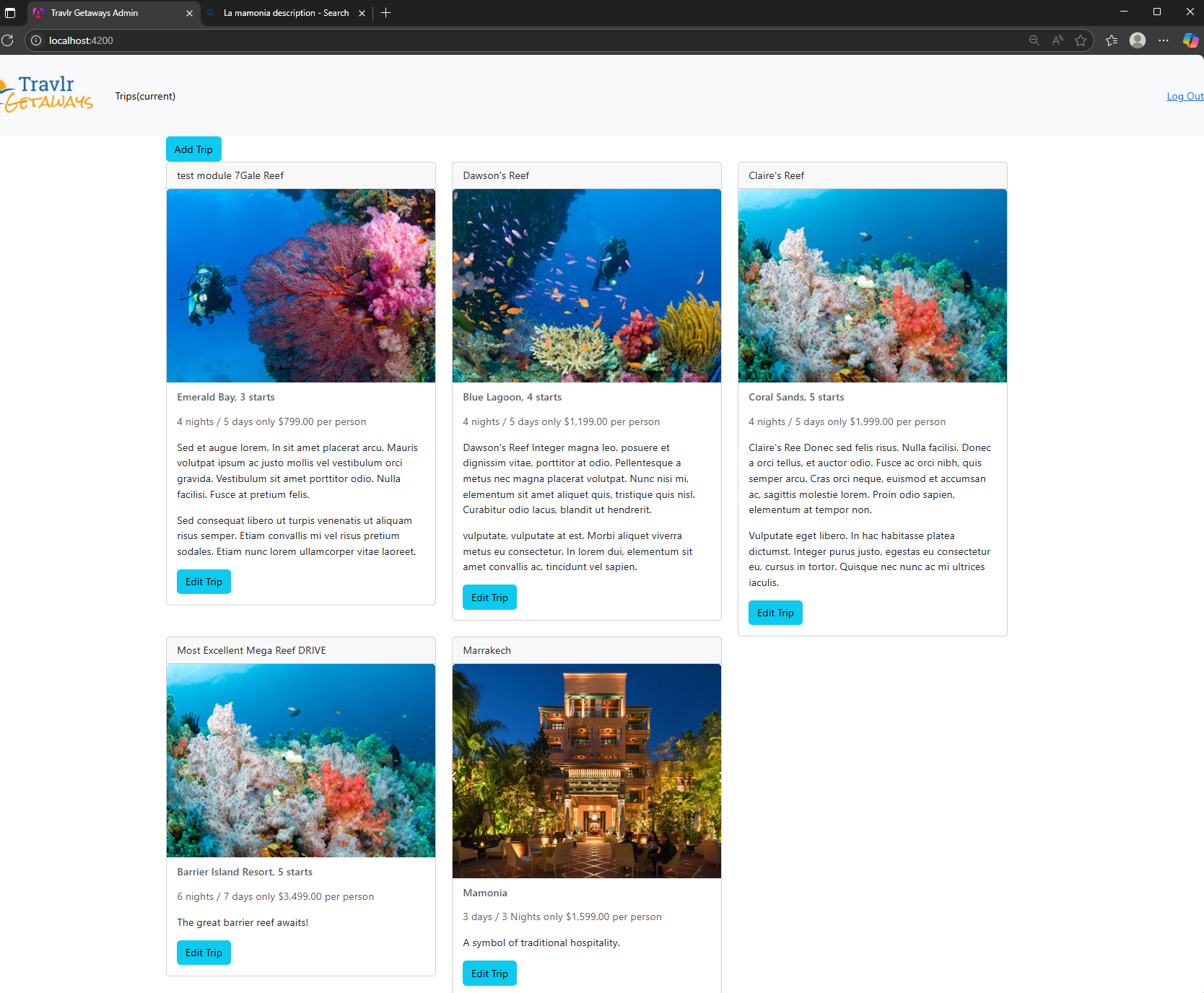


Figure 3: Trip added

A screenshot of a computer

AI-generated content may be incorrect.

Figure 4: Editing a Trip

A screenshot of a computer

AI-generated content may be incorrect.

Figure 5: Trip Edited Successfully

The project Travlr combines Angular(front-end), Express(back-end), and MongoDB(database) to create a full-stack Single Page Application (SPA) for managing travel trips. The Angular SPA provides a dynamic interface where admin users can view, add, and edit trips, all without full page reloads. This is achieved through components, services, routing, and an HTTP interceptor that automatically includes JWTs for secure API access.

Compared to Express, which typically uses separate routes and views for each page, Angular centralizes functionality in single HTML page and uses components to control different sections of the UI. Express handles the API endpoints and business logic on the server side, including authentication and database interactions using Mongoose models.

To test the full system, Postman was used to verify API routes like GET, PUT, and POST, ensuring proper responses and token requirements. Once confirmed, the Angular app was tested in the browser using developer tools to observe API calls and validate that trip data was correctly displayed and updated in the database.