

# **Travlr**

# **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 | 03/21/2025  04/05/2025  04/20/2025 | Sushmita Dhital  Sushmita Dhital  Sushmita Dhital | Works on Executive Summary, Design Constraints, and System Architecture View: Component Diagram.  System Architecture View: Sequence Diagram, System Architecture View: Class Diagram, and the API Endpoints table  work on the software development team for approval using a software design document |

## 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)

### ****System Architecture Overview****

The web application is built using the **MEAN stack**—a modern, full-stack JavaScript solution comprising **MongoDB**, **Express.js**, **Angular(not added yet)**, and **Node.js**. This architecture enables the development of a dynamic, scalable, and maintainable application that supports both end-users and administrators.

#### ****Customer-Facing Side****

The **front-end interface** for customers is implemented using **server-side rendering** with **Handlebars (hbs)** through **Express.js**.

Key technologies and structure:

* **Express.js** handles routing and server logic.
* **Handlebars (hbs)** renders dynamic HTML views using JSON data.
* **Static files** (CSS, images) are served from the /public directory.
* Data (e.g., travel destinations) is initially loaded from a structured JSON file and will be migrated to **MongoDB** for persistence in future phases.

#### ****Administrator SPA****

* Not developed yet

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

The development of the **Travlr Getaways** web application is guided by a set of design constraints

1. Use of MEAN stack
   1. Until now no mongo, nor Angular yet, but we should use them in the future
   2. Make data loading faster , right now they are being loaded from the json file,
      1. We should use Mongo and add caching layer to make data loading faster
   3. Include proper authentication system for the admin panel
   4. Include prober data isolation between customer bookings

## [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).

**1. Overview of System Architecture**

The system is divided into three main layers:

1. **Client (Front-End)**
   * Manages user interactions and displays content.
   * Handles user sessions and interactions with the **Traveler Portfolio**.
   * Utilizes a **Web Browser** to render the UI.
   * Uses a **Graphic Library** for UI enhancements.
2. **Server (Back-End)**
   * Manages authentication and session handling.
   * Processes user requests and connects with the database.
   * Stores and retrieves traveler data.
3. **Database (Persistence Layer)**
   * Stores traveler-related data in **MongoDB**.
   * Uses **Mongoose ODM** to interact with the database.

**2. Significant Components and Their Relationships**

The following are the major components in the system and how they interact:

**Client Components (Green Section)**

* **Web Browser**: The main interface where users interact with the application.
* **Traveler Portfolio**: Manages user-specific data, including travel itineraries, bookings, and preferences.
* **Graphic Library**: Handles UI rendering enhancements (e.g., images, animations).
* **Client Session**: Maintains session information for logged-in users.

**Client communicates with the server** to fetch data, authenticate users, and manage session-based interactions.

**Server Components (Blue Section)**

* **Authentication Server**: Handles user authentication and authorization.
* **Server Session**: Maintains user session information after authentication.
* **Traveler Database**: Stores traveler-related data for now it is a static json file.
* **Mongoose ODM**: Provides an abstraction layer to interact with **MongoDB**, making it easier to work with the NoSQL database.

**Server interacts with the database** using **Mongoose ODM** to read/write data in **MongoDB**.

**Database Component (Yellow Section)**

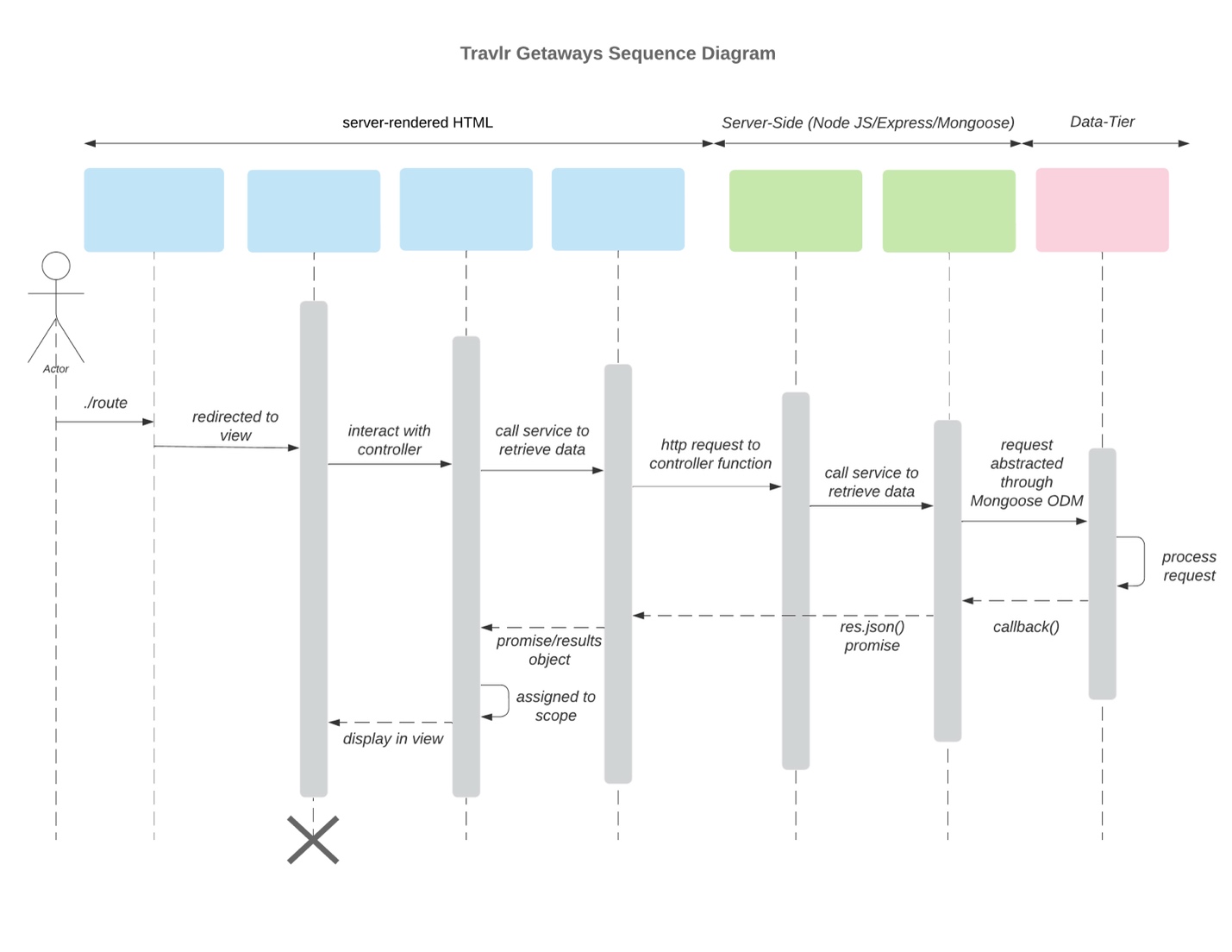
* **MongoDB**: The NoSQL database used to store structured traveler data.
* **Mongoose ODM**: Acts as an intermediary between the server and MongoDB, enabling data modeling and validation.

**MongoDB communicates with the server** via Mongoose, ensuring structured data retrieval and updates.

**3. Component Interactions**

1. **User Accesses the Application**
   * The **Web Browser** loads the customer-facing interface.
   * Users can browse travel destinations or log in to manage their **Traveler Portfolio**.
2. **Authentication Process**
   * The **Client Session** sends authentication requests to the **Authentication Server**.
   * If valid, a **Server Session** is created and maintained.
3. **Fetching and Storing Data**
   * User actions trigger requests from the **Traveler Portfolio** to the **Server**.
   * The **Server** interacts with **MongoDB** through **Mongoose ODM** to retrieve or store data.
   * The response is sent back to the **Client**, updating the UI.

### Sequence Diagram



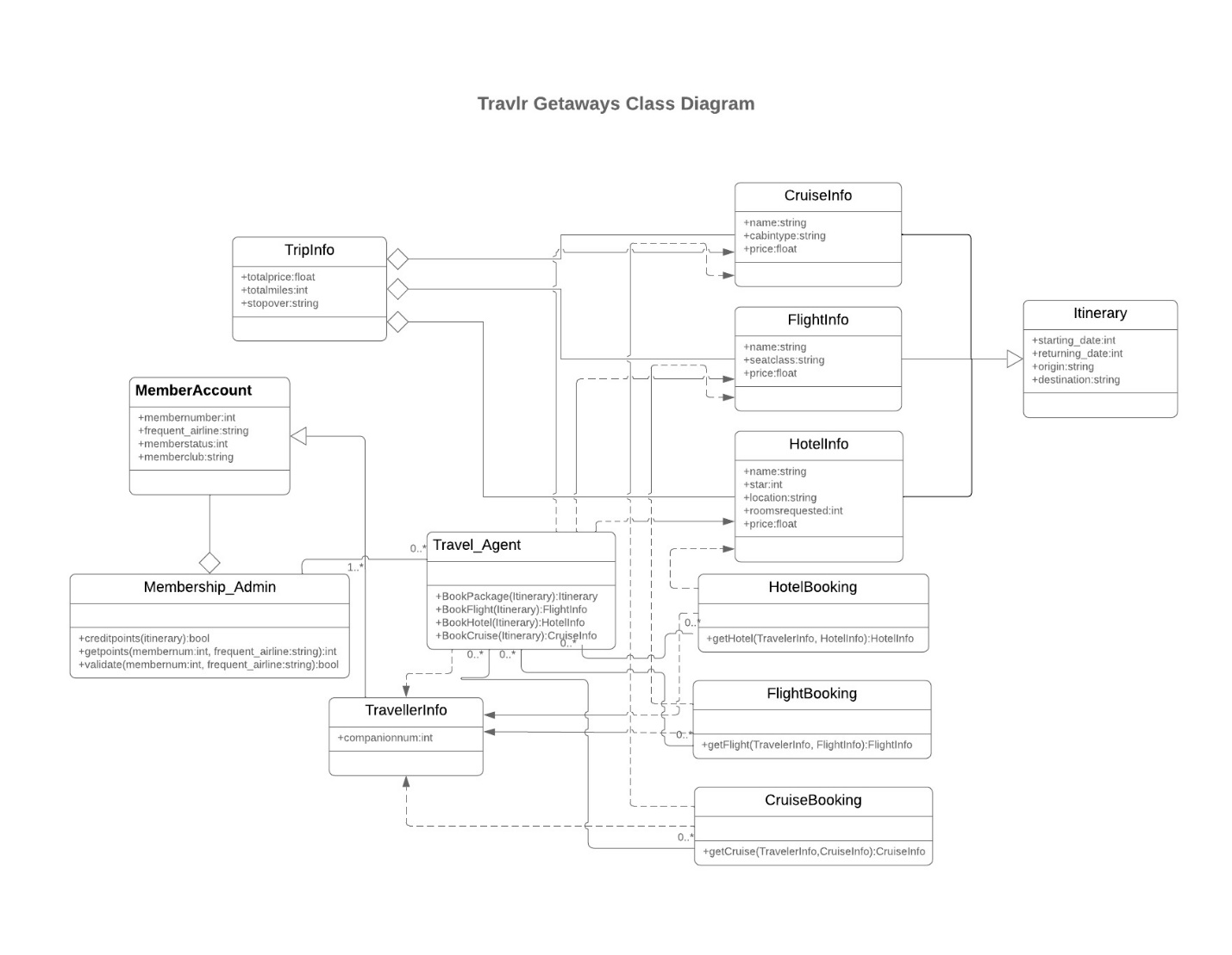
The Travlr Getaways web application follows a layered architecture: **client**, **server**, and **data tier**. When a user navigates to a route like /travel, the request is handled by an Express controller, which fetches trip data from the internal API (/api/trips).

This API calls a controller that queries MongoDB using Mongoose.

The returned data is passed to a Handlebars view and rendered as HTML back to the client.

This ensures organized, consistent communication across the full stack using the MEAN architecture.

## Class Diagram



* **Itinerary**: Stores trip details like start date, return date, origin, and destination.
* **CruiseInfo**: Holds cruise-specific data such as name, cabin type, and price.
* **FlightInfo**: Contains flight details including seat class and ticket price.
* **HotelInfo**: Represents hotel booking information like name, location, and number of rooms.
* **TripInfo**: Aggregates total price, distance, and stopover details of the trip.
* **MemberAccount**: Represents a user with frequent flyer information and membership status.
* **Membership\_Admin**: Manages member validation and rewards based on itinerary.
* **TravellerInfo**: Holds traveler details and links them to their bookings.
* **Travel\_Agent**: Handles booking logic for hotels, flights, and cruises.
* **HotelBooking / FlightBooking / CruiseBooking**: Provide methods to fetch and confirm bookings based on traveler and trip data.

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

| **Method** | **Purpose** | **URL** | **Notes** |
| --- | --- | --- | --- |
| **GET** | Retrieve list of trips | /api/trips | Returns all trips |
| **GET** | Retrieve single trip | /api/trips/:tripId | Returns a single trip instance, identified by the trip ID passed in the request URL |
| **POST** | Add new trip | /api/trips | Adds a new trip to the DB |
| **PUT** | Update new trip | /api/trips/:tripCode | Updates the record in the DB by ID |
| **DELETE** | Delete trip | /api/trips/:tripCode | Deletes the trip from the DB by ID |
| **POST** | Register user | /users/register | Register user data in the DB |
| **POST** | Login | /users/login | Authorize the user to validate its password |

## The User Interface

Listing trips for (guest users)

A screenshot of a computer

AI-generated content may be incorrect.

Listing trips for admin (Controls visible)

A screenshot of a computer

AI-generated content may be incorrect.

Editing trips (same as adding but with prefilled data)  
A screenshot of a computer

AI-generated content may be incorrect.

Login Page

A screenshot of a computer

AI-generated content may be incorrect.

Register page   
A screenshot of a computer

AI-generated content may be incorrect.

**Angular (Frontend SPA)**

* **Components-based architecture**: Organized into modular components like add-trip, edit-trip, trip-listing, login, register, and navbar
* **Services**: Contains trip-data. service.ts for API communication
* **Routing**: Defined in app.routes.ts for client-side navigation
* **Forms**: Reactive forms with validation (e.g., in add-trip.component.ts)
* **Interceptors**: JWT interceptor for authentication
* **Configuration**: Angular app config and browser storage management

**Express (Backend API)**

* **Controllers**: Handles business logic in trips.js and authentication.js
* **Models**: Mongoose schemas in travlr.js and user.js
* **Routes**: API endpoints defined in routes/index.js with JWT protection
* **Config**: Passport configuration for authentication
* **Database**: Connection management in db.js

**SPA Rich Functionality**

* **Single-page experience**: No page reloads between views
* **Client-side routing**: Instant navigation between components
* **Reactive forms**: Real-time validation and error handling
* **State management**: Maintains application state during navigation
* **JWT authentication**: Secure API access with token-based auth
* **Dynamic UI updates**: Updates content without server refreshes