

Travlr Getaways Application

# **CS 465 Project Software Design Document**

Version 1.2

## 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 | 07/19/2025 | Shawn Millin | Revision to Executive Summary, Design Constraints, and System Architecture View |
| 1.1 | 08/02/2025 | Shawn Millin | Revision to Sequence Diagram, Class Diagram, and API Endpoints |
| 1.2 | 08/14/2025 | Shawn Millin | Revision to User Interface section includes SPA |

## [Executive Summary](#_heading=h.35nkun2)

This project will present the full functioning web application of Travlr Getaways where users can book vacations. This project also allows administration to manage the data for the web application.

This web-based application called Travlr Getaways makes use of the MEAN stack development which stands for MongoDB, Express, Angular, and Node.js. These programs together allow for the development of a full functioning web application.

Administration will make use of Angular’s Single-Page Appliction(SPA) with dynamic updates as the user interacts with the page.

The Client-side of the web application will use JavaScript, HTML, and CSS to interact with the web application.

The Server-side of the web application will make use of the MongoDB for the database.

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

The constrains of this project include the complexity of the MEAN stack as this uses 4 different technologies to coordinate in order to work. Also the receiving technology needs to be able to run these programs on the local machine.

The SPA (Single Page Application) constrains the design in that there could be indexing problems as there tends to be a struggle to process dynamic content rendered by JavaScript.

Other constrains include: time fame and budgeting. The time frame of the projects will need to meet the set deadline. The budget limit will determine if the project can meet the time frame limit in having enough resources for staffing adequate people to complete and put together all parts of the project.

## [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).

**Explanation of Diagram:**

The System Architecture View is composed of three main components with several components combined within those.

The Client component is comprised of the client session, web browser, traveler portfolio, and graphic library components. This main component draws from the Database and the Server main components.

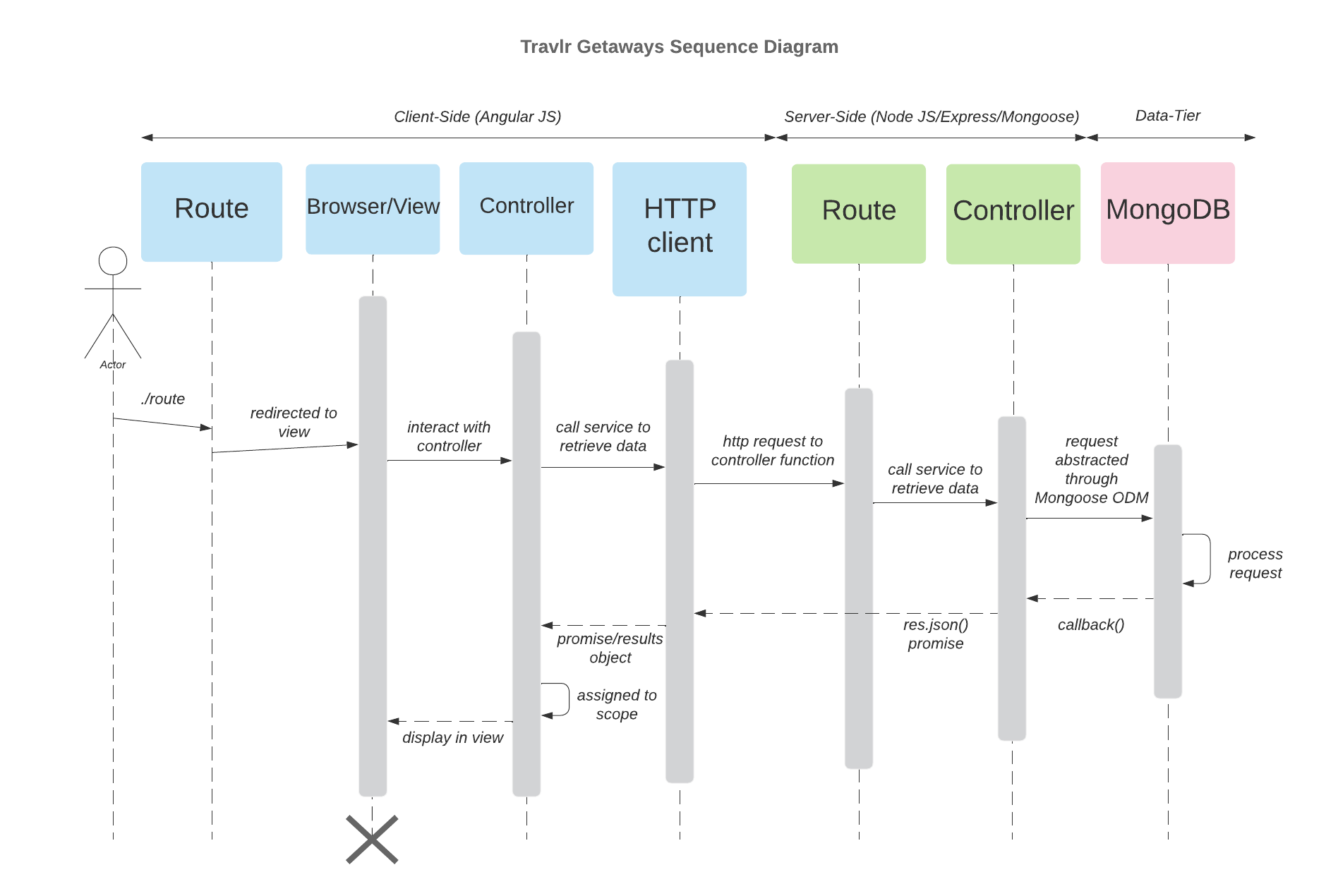
Web Browser allows users to interact with their information such as a profile, also interacting with the Client Session and Traveler Portfolio. Client Session manages the client-side session functions, interfaces with Authentication Server to authenticate and authorize users, as well as connects to Server Session for server-side session management. Traveler Portfolio uses the Graphics Library for graphical elements with the user interface and accesses the MongoDB for data storage and retrieval on traveler information. Graphic Library enhances the user experience by adding graphical elements the user interface.

The Server component is comprised of the authentication server, server session, traveler database, and Mongoose ODM. This main component draws data from the Database main component and allows authentication to the user to access the traveler database.

The Authentication Server handles authentication and authorizations to users for secure access an d interacts with Client Session for user session management. Server Session retrieves traveler data from the Traveler Database, Connects to Client Session for client-side session management, and uses Mongoose ODM to connect to MongoDB. Traveler Database stores and manages data in relation to travelers, as well as connection to Server Session for retrieving data. Mongoose ODM is the Object Data Model for MongoDB allowing interaction and data management between the application and the database.

The Database component is a single component of the MongoDB database. MongoDB database stores traveler information and the application’s data.

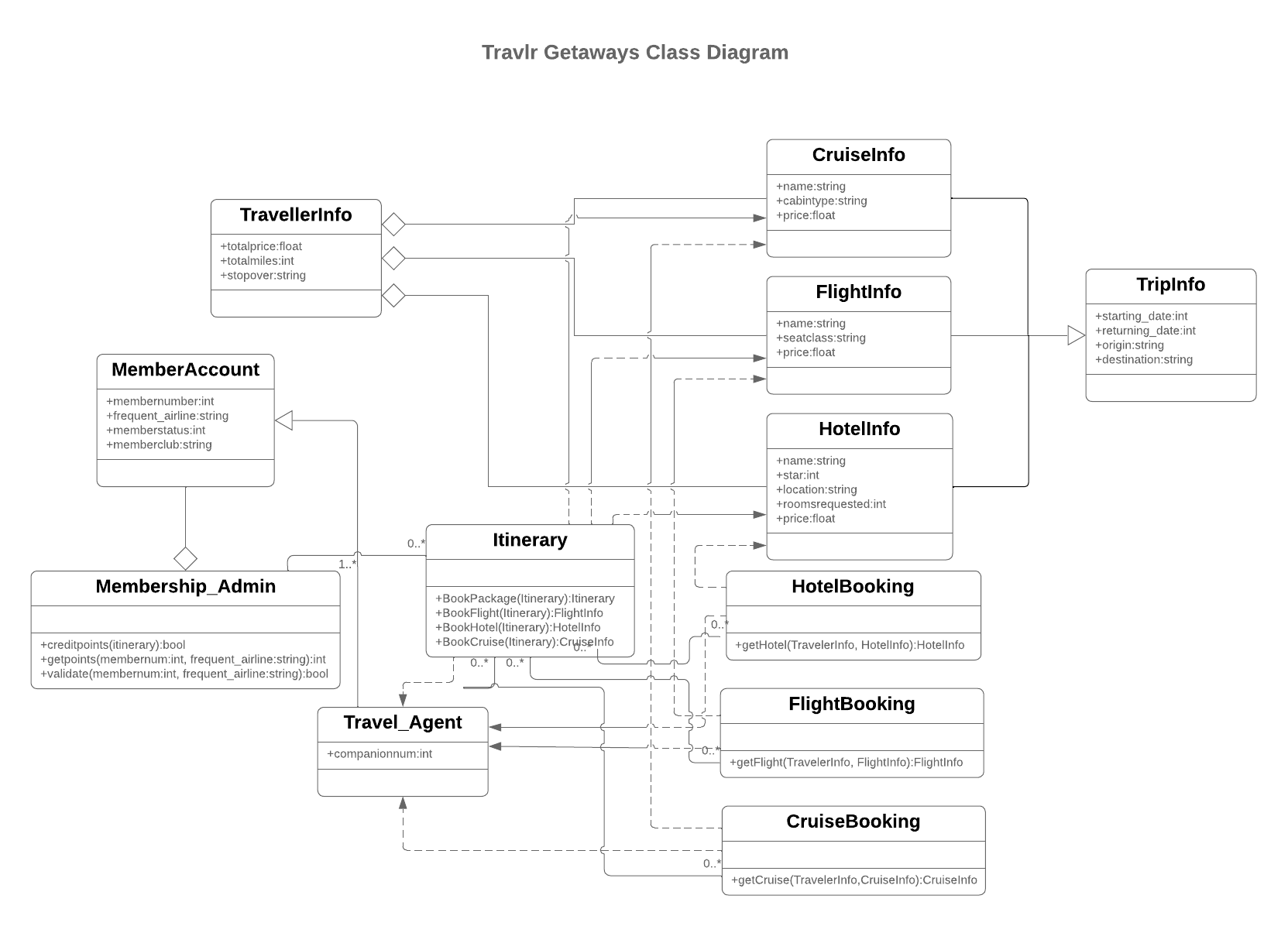
### Sequence Diagram



The above diagram shows the sequence of actions from a user’s interaction with the application. The major parts of the diagram are the client-side, server-side, and data-tier. After a user accesses the application the route brings the user to a view of the browser.

Information requested is from the controller which then communicates with the HTTP Client, then through the server-side which routes thought the controller to receive data from the database of MongoDB. This data is then sent back to the client-side controller and then to the browser view to display the data to the user.

## Class Diagram



Above is the class diagram for the traveler application. This diagram shows the different classes in the code for the application and the interactions with each other.

The TripInfo class holds the trip information with the methods of starting date, returning date, origin and destination. This class inherits the information from HotelInfo, FlightInfo, and CruiseInfo classes, each of which have information on that particular place. Methods for the HotelInfo class include the pricing of rooms, star rating, rooms requested, hotel name and location. Methods for the FlightInfo class includes the pricing, seat class, and name of airline. Methods in the CruiseInfo class includes the cruise name, pricing, and cabin type.

The FlightBooking, HotelBooking, and CruiseBooking classes associate with the HotleInfo, CruiseInfo, and FlightInfo classes which in turn aggregates with the TravelerInfo class. The TravelerInfo class hold the information of the traveler with the methods of total miles, total price, and stopovers for their trip.

The Itinerary class holds information on the Bookings of either the flights, cruises, hotels, or packages and is directly associated with the Travel\_Agent class. The Travel\_Agent class serves as a andler for the bookings of flights, cruises, and hotels.

MemberAccounts class inherits information from the Travel\_Agent class and aggregates with the MembershipAdmin class. The MemberAccounts class has the methods of member number, frequent airline, member status, and member club, this holds the account information of the traveler.

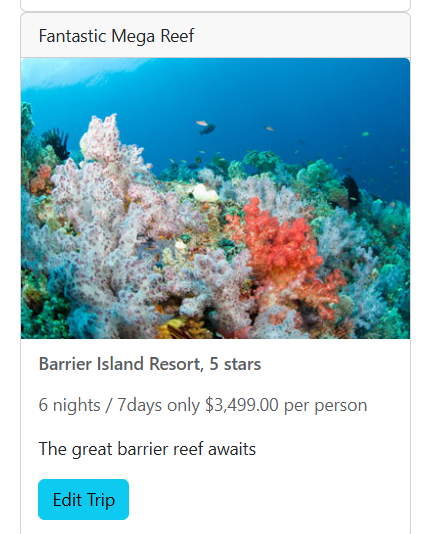
MemebershipAdmin class validates the account information and holds personal information that needs authorization to access.

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

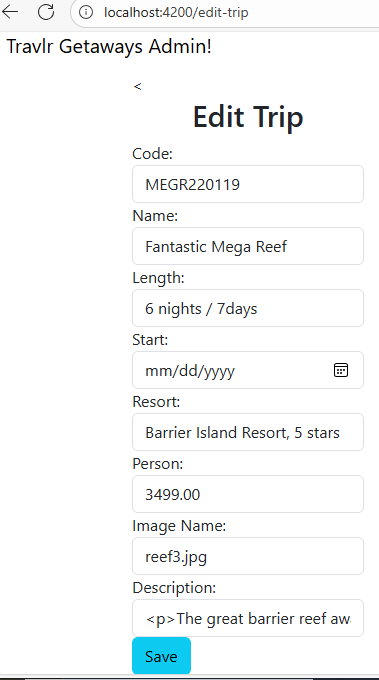
| **Method** | **Purpose** | **URL** | **Notes** |
| --- | --- | --- | --- |
| **GET** | <Retrieve list of trips> | </api/trips> | <Returns all active trips> |
| **GET** | <Retrieve single trip> | </api/trips/:tripId> | <Returns single trip instance, identified by the ttrip ID passed on the request URL> |
| **POST** | <Create a list of trips> | </api/trips> | <Creates a new list of trips> |
| **POST** | <Create a single trip> | </api/trips/:tripid> | <Creates a single trip instance, identified by the trip id passed on to the request URL> |
| **PUT** | <Update a list of trips> | </api/trips> | <Updates a new list of trips> |
| **PUT** | <Update a single trip> | </api/trips/:tripid> | <Updates a single trip instance, identified by the trip id passed on to the request URL> |
| **DELETE** | <Delete a list of trips> | </api/trips> | <Deletes a new list of trips> |
| **DELETE** | <Delete a single trip> | </api/trips/:tripid> | <Deletes a single trip instance, identified by the trip id passed on to the request URL> |
| **PATCH** | <Modify a list of trips> | </api/trips> | <Modifies a new list of trips> |
| **PATCH** | <Modify a single trip> | </api/trips/:tripid> | <Modifies a single trip instance, identified by the trip id passed on to the request URL> |

## The User Interface

A new unique trip was added, to the web application, of “Fantastic Mega Reef” using the ‘Add Trip’ button on the web application trip-card page. This addition includes an image, description, the resort, star rating, trip title, trip length, trip code, and trip cost per person. This addition is done on the front-end and adds the new trip into the database after submitting via the ‘Save’ button.



Each trip-card has the option to be edited via the ‘Edit Trip’ button on each trip-card. Each part of the trip’s information is editable, then updated into the database via the ‘Save’ button on the bottom of the page.



The unique trip added was updated with a new trip name of “Ultimate Reef Exploration” via the ‘Edit Trip ‘ button.



The Angular structure in this project differs from the Express.js in that Angular is more front-end focused and uses components to to handle functions and features while Express is a back-ended framework handling routing. These are not opposites of each other and can work in conjunction as the Express building the API endpoints that the Angular Single Page Application gets data from.

The Single Page Application(SPA) has the advantage of a better user experience with more responsiveness. This conjunct with content updating fast, which updating comes in real time as code is saved in the developers’ IDE with no full page reloading in the browser, gives a user a more interactive feel.

Additional SPA functionality includes a reduction in the number of server requests and data transferred which then has a lower server load and reduced bandwidth consumption. The offline functionality allows a user to access particular parts of the application by using local caching to store data locally.

The SPA does have difficulties such as the development that is more complex as the SPA is heavily reliant on JavaScript. This reliance on JavaScript could also exclude the full application experience users that have JavaScript disabled in their browser settings. The SPA may have security vulnerabilities when relying more heavily on the client-side rendering and SPA is more likely to have memory leaks.

During the development of the SPA testing can be done in the developer console of the browser which shows warnings, JavaScript errors, and network requests. GET an PUT verification of working with the API can also be done in the developer console in the network tab which will show the GET and PUT in the Method column. The use of MongoDB Compass allows a developer to check if changes on the client-side are changed in the database.