

<Name of Software Application>

# **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 | <05/26/24> | <Olivia Edey> | <3-2 Milestone> |

## 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)

< The Travlr Getaways app is designed using the MEAN stack, which includes Angular.js, Express.js, Node.js, and MongoDB. The front end of this system is a customer-facing website where users can browse travel packages, make bookings, and manage their accounts. The back end of the system is an admin single-page application (SPA) where Travlr Getaways staff manages travel packages, handle bookings, and oversee customer information. The front end of the application is built using Angular.js, while the back end utilizes Express.js and Node.js.>

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

< Budget plays a major role in projects; without the right amount of funds, the project can be delayed. Everything needs to work and be affordable to meet the budget. Another constraint that needs to be considered is usability. The application must be user-friendly and accessible on various devices. Performance is another constraint that needs focus. As the website gains popularity, it needs to handle thousands of users navigating the site and managing the amount of data being added. The main performance issues might arise during peak travel seasons.>

## [System Architecture View](#_heading=h.44sinio)

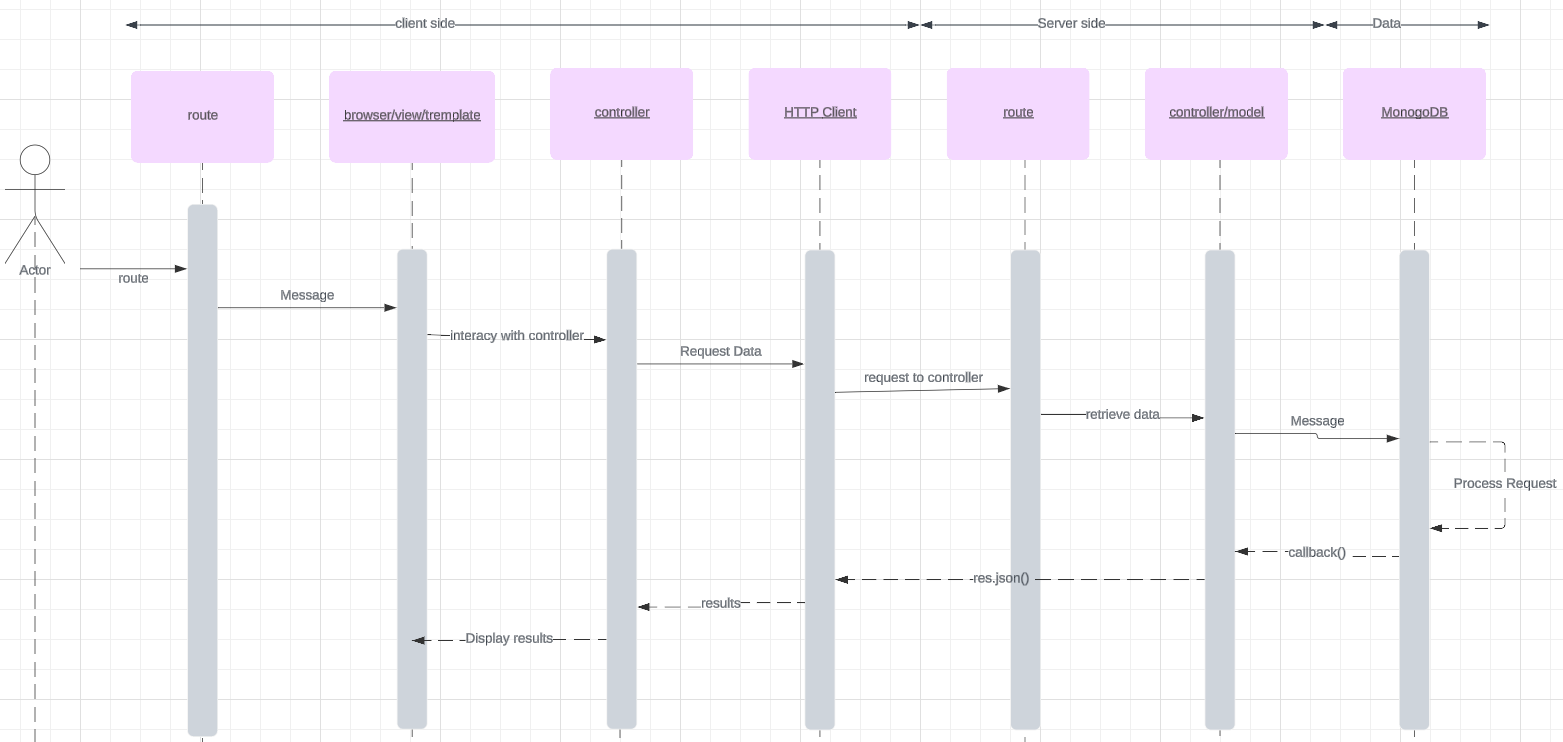
### Component Diagram



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 Travlr Getaways web application is designed using a component-based architecture, where the system is divided into three main component boxes: Client, Server, and Database. Each of these components are needed to work together to complete the system and operate efficiently. The client component box consists of client session, web browser, traveler portfolio and graphic library. The server component consists of authentication server, server session, traveler database and mongoose ODM. The database component consists of only mongoDB, which stores all the data for the application. The database stores the data until requested and feeds the server with the information, which can then be provided to the client. Before the client can receive the information, the server authenticates the information from the client.>

### Sequence Diagram

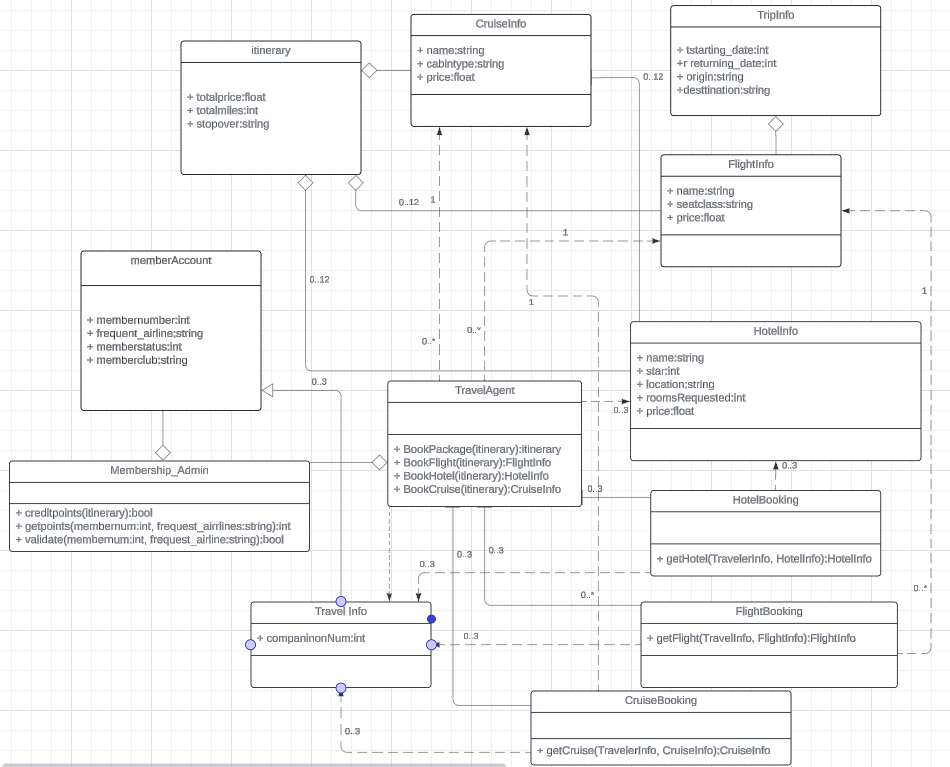


<This sequence diagram will showcase the interactions between different layers or tiers of the full stack application, including the Controller, Model, Route, Browser/View/Template, MongoDB, and HTTP Client. The user interacts with the Browser/View/Template layer, where they input their credentials. The input is sent to the Controller layer, which handles authentication. The Controller interacts with the Model layer to validate the user's credentials against stored data in MongoDB. If the credentials are valid, the user's dashboard is displayed. If the credentials aren’t valid then an error message is displayed.

If the browser makes request for data stored on the server these are handled by the HTTP Client.The HTTP Client sends a request via a second controller to the appropriate Route. The Controller interacts with the appropriate Model layer to fetch trip data from MongoDB or perform booking operations. Once the operation is completed, the Controller sends the updated data back to the Browser/View/Template layer for display.

An admin user interacts with the Browser/View/Template layer to access administrative functionalities. The Controller interacts with the appropriate Model layer to perform administrative tasks, such as managing user accounts or updating trip details in MongoDB. >

## Class Diagram



< A trip plan comprises essential components such as CruiseInfo, FlightInfo, and HotelInfo. Itinerary, serving as the parent class, can accommodate multiple instances of these child classes. Each child class can exist independently; for instance, a HotelInfo object may exist without an Itinerary object, but an Itinerary object cannot exist without at least one of CruiseInfo, FlightInfo, or HotelInfo.

The TravelAgent class encompasses various functionalities with respect to other classes. It can handle instances of CruiseInfo, FlightInfo, HotelInfo, or TravelerInfo, and is also associated with HotelBooking, FlightBooking, and CruiseBooking. This association allows for zero or multiple instances of these classes, recognizing that not every trip necessitates the involvement of a travel agent, nor does every trip entail multiple bookings for hotels, flights, or cruises.

TravelerInfo, a subclass of the superclass MemberAccount, signifies its role in managing member accounts within the system. This relationship ensures that membership administration is integrated with member accounts.>

## [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** |
| --- | --- | --- | --- |
| **GET** | <Retrieve list of things> | </api/trips> | <Returns all active things> |
| **GET** | <Retrieve single thing> | </api/trips/:tripId> | <Returns single thing instance, identified by the thing ID passed on the request URL> |
| **DELETE** | Deletes a trip from the database | </api/trips/:tripId> | <Deletes a single trip, using  the Admin site, found  using its “tripId”> |
| **POST** | Adds a trip to the database | </api/trips> | <Adds a single trip> |
| **PUT** | Updates existing trips in the database | </api/trips/:tripId> | <Updates info on a single trip, using the Admin site found using its “tripId”> |

## 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.>

<Summarize the Angular project structure and how it compares to the Express project structure. Be sure to describe the rich functionality provided by the SPA compared to a simple web application interaction. Describe the process of testing to make sure the SPA is working with the API to GET and PUT data in the database.>