

Travlr Getaways

# **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 | 07/15/2024 | Dominic Drury | Initial Commit of executive summary, design constraints, and system architecture view |
| 1.1 | 08/01/2024 | Dominic Drury | Added sequence diagram and class diagram as well as filled out API Endpoints table |
| 1.2 | 08/16/2024 | Dominic Drury | Finalized document, completed UI section |

## 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)

To create a travel booking site for customers, that allows them to create and use their account, search for travel packages, book reservations, and see their itineraries, and for administrators, to be able to maintain the customer base, available trip packages, and pricing, we have designed a web application using the MEAN stack. Mongo will handle the database for the users and trip data, Express will be used as the web application framework for Node.js, Angular will be the JavaScript framework that will create the interface for the web application, and Node will be the software platform that will hold the stack together.

The customer facing side will not be a single page application (SPA), this will make the website crawlable by search engines, increase first load time, and allow the travel information to be shareable. The admin side will be a SPA which will improve response times for actions and inputs and keep that side of the application private.

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

With this design there are few constraints. The database, REST API, and application can be separated into their own servers and even have multiple servers running each side depending on the future needs of the application. With this level of scalability, it should be easy to maintain a fast and responsive application for both administrators and customers.

For the application to meet the system requirements the system will need to load fast for customers to reduce bounce rates and accommodate a variety of mobile and PC devices. It will also need to respond quickly to admin inputs and allow for fast movement within the system.

Regarding the MEAN stack the constraints are that everything is built on Node.js so it is arguable irreplaceable, and MongoDB can suffer from response times if the data becomes too great so old data will need to be routinely scrubbed.

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

### Component Diagram

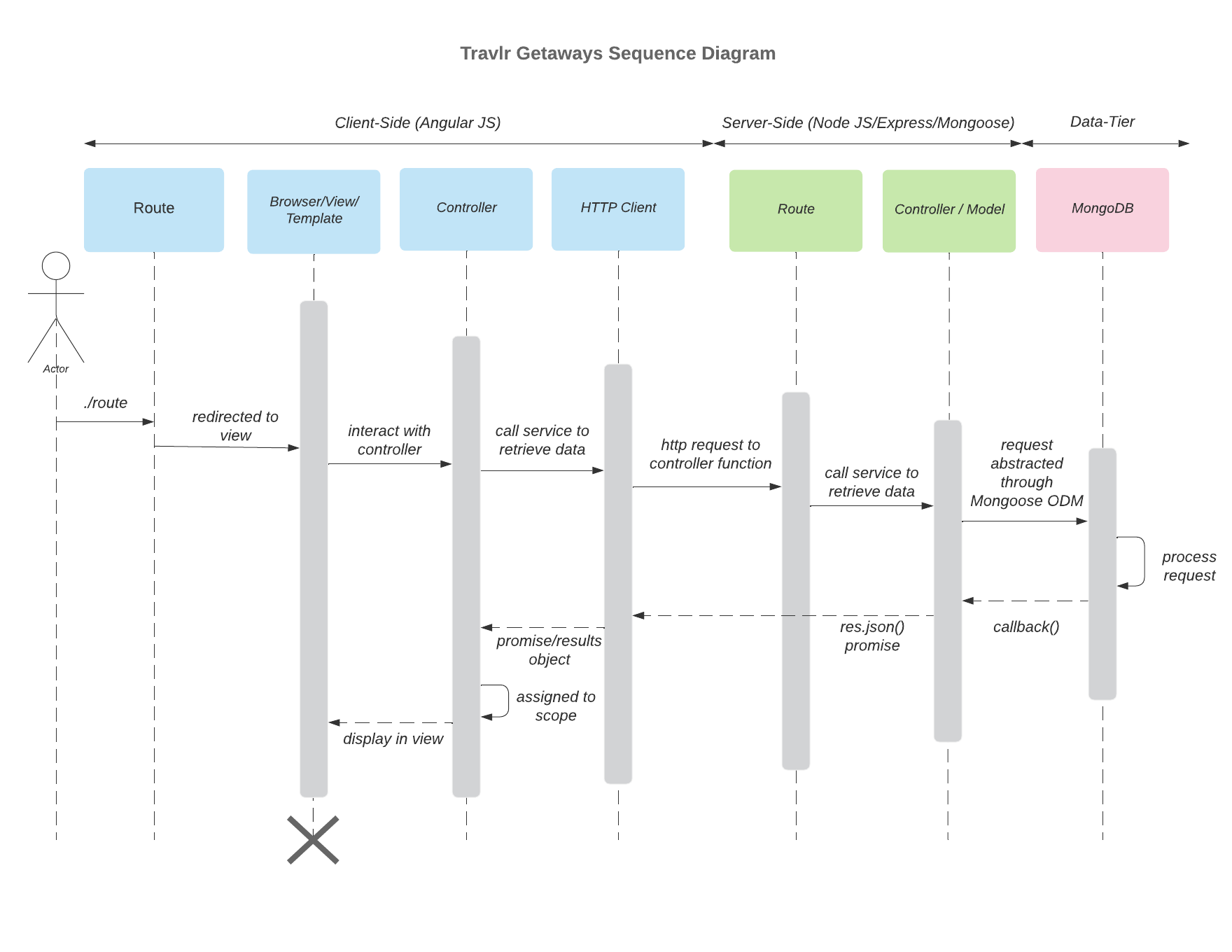


A text version of the component diagram is available: [CS 465 Full Stack Component Diagram Text Version](https://learn.snhu.edu/d2l/lor/viewer/view.d2l?ou=6606&loIdentId=24342).

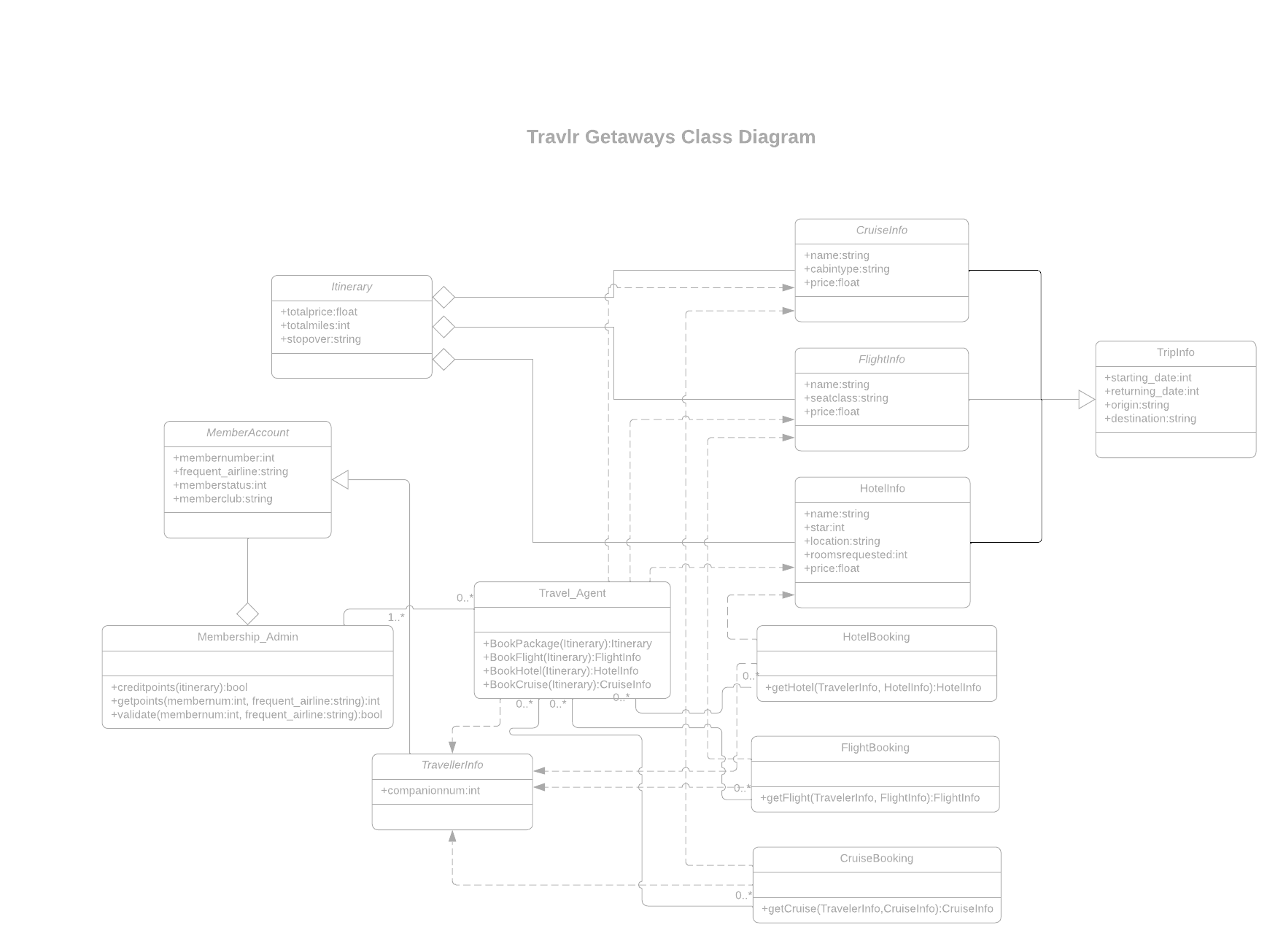
The overall system architecture is made up of three separate components, the client side, server side, and database side. Inside of the client side there is the client session, the web browser the client is using, their portfolio, and the graphic library. Inside the server side is the authentication server, the server session, the database of travelers, and Mongoose ODM. Finally, inside the database is the MongoDB database.

The database provides the data for both the client and server, the server provides administrative functionality and user authentication for the client and access to the database, and the client side handles input from users for authentication with the server and interaction with the data stored in the database.

### Sequence Diagram

The flow of logic for the Sequence Diagram above consists of three separate tiers that connect, the client facing side, the server facing side, and the database. A user signs into the site, their credentials are checked in the database and depending on if they are a customer or admin either the Trips or the Administration views will be returned to the user.

## Class Diagram

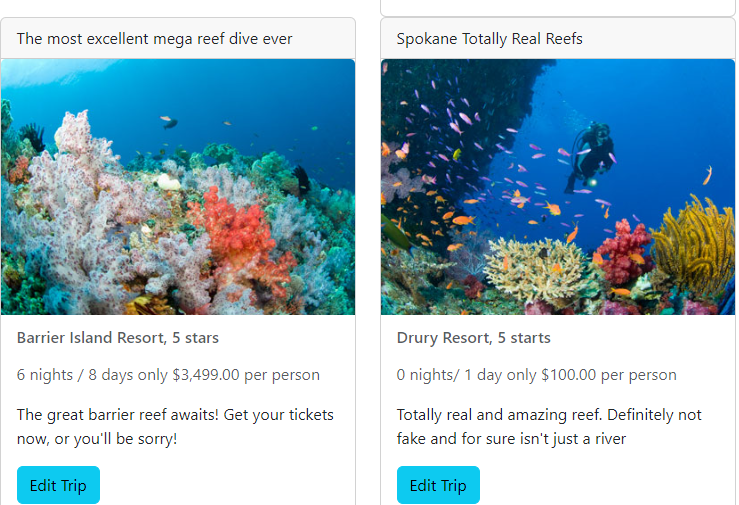


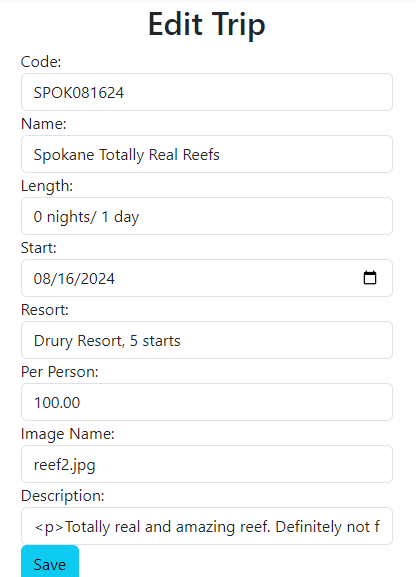
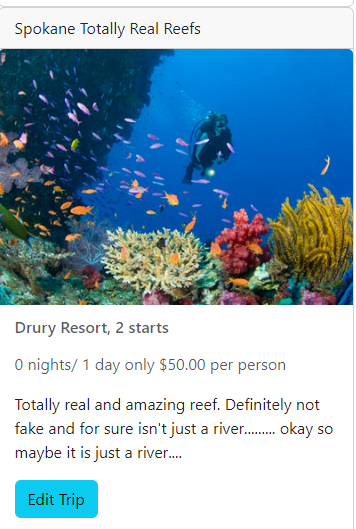
The TripInfo class contains public fields for the starting date, returning date, origin, and destination. CruisInfo, FlightInfo, and HotelInfo all inherit from the TripInfo and have a name and price field. CruiseInfo has a field for the cabin type, FlightInfo has a field for the seat class, and the HotelInfo has fields for the star rating, location, and rooms requested. The HotelBooking, FlightBooking, and CruiseBooking all have a function for getting their respective items and have a relationship between their respective Infos and the TravellerInfo. Travel\_Agent has a 0 to many relationship with the Booking classes and membership\_admin and has a relationship with the Info classes. Travel\_Agent also has functions for booking itineraries, flights, hotels, and cruises. TravellerInfo contains the field for the companion number and has a relationship with the booking classes and the Travel agent and inherits the MemberAccount class. Membership\_admin has a 1 to many relationship with Travel\_Agent. It has functions for crediting points, getting points, and validating member accounts. MemberAccount has fields for the member number, frequent airline, member status, and member club.

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

| **Method** | **Purpose** | **URL** | **Notes** |
| --- | --- | --- | --- |
| **POST** | Send Login Data | /api/login | Sends login data to database for authentication |
| **POST** | Send Registration info | /api/login/register | Sends information to database to register a new user |
| **GET** | Retrieve list of meals | /api/ meals | Returns all active meals |
| **GET** | Retrieve single meal | /api/ meals /:mealId | Returns single meal instance, identified by the meal ID passed on the request URL |
| **GET** | Retrieve list of news | /api/ news | Returns all active news |
| **GET** | Retrieve single news article | /api/news/:newsId | Returns single news article instance, identified by the news article ID passed on the request URL |
| **GET** | Retrieve list of rooms | /api/ rooms | Returns all active rooms |
| **GET** | Retrieve single room | /api/things/: roomId | Returns single room instance, identified by the room ID passed on the request URL |
| **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 trip ID passed on the request URL |
| **POST** | Add a trip | /api/trips/:tripId | Adds a trip to the list of active trips, identified by the trip ID passed on the URL |
| **PUT** | Update a trip | /api/trips/:tripId | Updates the trip information of a trip, identified by the trip ID |
| **DELETE** | Delete a trip | /api/trips/:tripId | Deletes a trip, identified by the trip ID |

## The User Interface



Unlike with the express project, the angular is much faster after the initial load and quickly updates since the screens are made up of fragments and not a full reload. Having the admin be a single page means that once the page has been loaded, everything is faster than if a simple web application was used. Testing the API for the SPA involves testing the connection to the database, testing the crud methods being utilized, and using an interceptor to make sure that only those with authorization can make changes to the application.