

Travlr Getaways Web Application

# **CS 465 Project Software Design Document**

Version 1.3

## 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 | 03/22/2025 | Braedan French | Completed the Executive Summary, Design Constraints, and Component Diagram sections. |
| 1.1 | 04/03/2025 | Braedan French | Completed the Sequence Diagram, Class Diagram, and API Endpoints sections. |
| 1.2 | 04/19/2025 | Braedan French | Added explanation about the differences between the Angular and Express project structures under the User Interface section. Also added information about SPA functionality, its advantages, disadvantages, and how it compares to simple web interactions. And the testing process of the SPA with its API was also included. |
| 1.3 | 04/20/2025 | Braedan French | Added screenshots of the current page results for the login, home, and add trip pages. |

## 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 project is a full stack web application that utilizes the MEAN stack. MEAN is an acronym for the stack tools that will be incorporated from the front end to the back end of the project. These include MongoDB, Express, Angular, and Node.js. The application will contain a static portion that displays the content of the Travlr Getaways website, while the admin portion will manage the website and its processes.

The customer facing side of the Travlr Getaways web application will utilize MongoDB, Express, and Handlebars. MongoDB is a NoSQL database management system that is open source and provides the application with a database to store static content. The static content will then be displayed within Handlebars templates. And Express will be used to serve the static content contained within MongoDB and the Handlebars templates.

The admin facing side of the Travlr Getaways web application is achieved through the completion of a client single page application (SPA) using Angular. This will add the ability for admin privileges, such as the option to manipulate static content displayed on the webpage to customers. To gain access to these privileges, administrators will need authenticated credentials to login to the single page application.

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

Some of the design constraints the project is faced with include the need to use Handlebars templates to render static content as JSON data. Travlr Getaways will also need to be an Express web application with various routes, controllers, views, and data models to be in line with the MEAN stack for proper functionality. A single page application will also be utilized as an admin portal and the method to manage the web application’s static content. Other general constraints include ease of use when browsing on the customer side of the web application and responsiveness to updates when on the admin side. And since it is a web-based application, the Travlr Getaways webpage will need to be capable of displaying properly on different devices that have varied specifications or screen sizes.

Looking at the MEAN stack itself, one disadvantage is the reliance on MongoDB as the database management system. MongoDB performs well and is highly scalable but can be memory intensive if datasets become too large. And MongoDB’s aptitude for scalability can backfire if sharding is not properly handled, leading to more performance issues. Regarding Express, Angular, and Node.js, each one of these tools are based in JavaScript. Meaning that extensive knowledge of the language will be needed to utilize these tools to their fullest potential.

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

### Component Diagram

Referencing the diagram below, the Travlr Getaways web application consists of larger components. These include Client, Server, and Database components that control different aspects of the application. The Client component contains the Client Session, Web Browser, Traveler Portfolio, and Graphic Library components. Based on how each client component is connected in the diagram and their name scheme, the Web browser and Graphic library components provide the interface for the client to engage with for the Travlr Getaways application. A browser will then display the contents of the Graphic Library when a user navigates to the web application to start a session. And since Travlr Getaways is a web-based application, it needs the Web Browser component to interface with the contents of the application, which is contained under the Traveler Portfolio component. Considering that the main content of the application is within the Traveler Portfolio component, it requires the Graphic Library to provide its interface to correctly display the web page. As well as the MongoDB component in which the data for the Traveler Portfolio is stored before it gets called for display in the web application. All of which eventually leads to the Client Session component, as the Web Browser and Traveler Portfolio provide the content of the application and its interface during a session of the application. The Client Session component is then connected to a port on the Client component to interact with the Server and its components.

The four components of the larger Server component are the Authentication Server, Server Session, Traveler Database, and Mongoose ODM. The Server Session and Mongoose ODM provide an interface to which the Traveler Database will require the Server Session components interface. The Server Session component will require the usage of the Mongoose ODM component’s interface, while the Mongoose ODM component utilizes the interface of the Database component. The Authentication Server and Server Session components alike engage with the port of the Server component then provide interface for the Client component.

The last of the larger components is the Database component. This contains only a single component, that being the MongoDB component. The MongoDB component lends its interface to both the Client and Server components with the Traveler Portfolio and Mongoose ODM components respectively.



### Sequence Diagram

Using the diagram below, the logic sequence of the web application is as follows. Beginning with the actor or user, the actor enters the initial route to be redirected to a view/template of the website. Once the view is accessed by the actor, the view will make requests to the controller to populate and render the website for display. The client-side controller then calls upon the HTTP Client to retrieve information that will then be sent back to the controller. The HTTP Client is also what connects the client-side to the server-side components of the web application. The HTTP Client achieves this by calling API service to different routes of the application. Depending on route used, the appropriate server-side controller will be used to retrieve the requested data. Once selected, the controller will make a request for the data contained within the application’s MongoDB database using Mongoose. When the data is retrieved from the database, it is returned to the controller so that it may send it to the HTTP Client at the frontend. The HTTP Client can then forward the data to the client-side controller to assign it to the scope of the application, which then allows it to be displayed to the web application’s template.

A diagram of a diagram

AI-generated content may be incorrect.

## Class Diagram

As the name describes, the TripInfo class contains variables related to an existing trip’s information. This includes variables for the starting date of a trip, the end date of a trip, the origin location where the trip begins, and the last destination when the trip ends. Three other classes inherit the TripInfo class, these being the CruiseInfo, FlightInfo, and HotelInfo. These classes contain information related to the various modes of travel that the trip will incorporate. Each one contains variables for the names and prices of the travel method and hotel, with some unique variables that pertain to the type of transportation and hotel information. An example from each class would be the cabin type under CruiseInfo, the class of seating within the FlightInfo class, and the star rating of the hotel in the HotelInfo class. The CruiseInfo, FlightInfo, and HotelInfo classes also have corresponding Booking classes that they are associated with. These include the CruiseBooking, FlightBooking, and HotelBooking classes, which are associated with the TravellerInfo Class. Under the Booking classes are GET methods to obtain the data from their respective Info classes that allow users access for the purpose of booking cruises, hotels, and flights on their chosen trip. The Booking classes also contain zero-to-many relationships with the Travel\_Agent class at both ends. The Travel\_Agent class contains methods to arrange the booking of hotels, flights, and cruises with regards to the Itinerary class to determine pricing, mileage, and stopover time within a trip. Going back to the Info classes, they each have association with the Travel\_Agent class with the inclusion of TravellerInfo class and exclusion of TripInfo. TravellerInfo contains the information related to the user or users attempting to book a trip with data related to the number of companions that will be apart of the trip. Besides its relationships with the Booking and association with the Info classes, the Travel\_Agent class has a one-to-many relationship with the Membership\_Admin class. The job of the Membership\_Admin class is to handle discounts with the inclusion of credit points for frequently used airliners when travelling. The TravellerInfo class is connected to the MemberAccount class as it inherits it. MemberAccount contains information related to a user’s account on the Travlr Getaways website. This includes their account status, what clubs they are a part of, their favorite airliner, and their account number. MemberAccount is then connected to Membership\_Admin through an aggregate relationship. Lastly, the Itinerary class has aggregate relationships with the Info classes, excluding TripInfo and Travellerinfo. As the name suggests, the Itinerary class contains details about the trip. This includes a trip’s planned cost, miles traveled, and the stopover times between destinations.

**A diagram of a travel application

AI-generated content may be incorrect.**

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

| **Method** | **Purpose** | **URL** | **Notes** |
| --- | --- | --- | --- |
| **POST** | Login a user | /api/login | Authenticates user account and returns JWT |
| **POST** | Register a user | /api/register | Adds a new user account to the database and returns JWT |
| **GET** | Retrieve list of trips | /api/trips | Returns all available trip entries |
| **GET** | Retrieve single trip listing | /api/trips/:tripCode | Returns a single trip entry with its trip code in the URL |
| **GET** | Retrieve list of rooms | /api/rooms | Returns all available room entries |
| **GET** | Retrieve single room listing | /api/rooms/:roomCode | Returns a single room entry with its room code in the URL |
| **GET** | Retrieve news list | /api/news | Returns all news entries |
| **GET** | Retrieve single piece of news | /api/news/:newsCode | Returns a single news entry with its news code in the URL |
| **GET** | Retrieve list of meals | /api/meals | Returns all meal entries |
| **GET** | Retrieve single meal listing | /api/meals/:mealCode | Returns a single meal entry with its meal code in the URL |
| **POST** | Add a trip | /api/trips/:tripCode | Adds a new trip to the database. Its trip code can be found within the URL |
| **PUT** | Update a trip | /api/trips/:tripCode | Updates an existing trip within the database. Its trip code can be found within the URL |
| **DELETE** | Delete a trip | /api/trips/:tripCode | Deletes an existing trip within the database. Its trip code can be found within the URL. |

## The User Interface

**Current Login Screen Results**

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.**Current Home Page**

**Current Page for Adding New Trips**

A screenshot of a computer

AI-generated content may be incorrect.

In terms of their project structure, Angular and Express contain different aspects to achieve frontend and backend frameworks respectively. Within Angular, the key aspects are the components and modules that organize different frontend functions. The components are made up of three different files, including a TypeScript class, an HTML template, and a CSS style sheet. Editing these files will change the appearance of an application generally and add to it . In the case of the previous SPA assignment, multiple components were incorporated to add buttons and trip cards to the admin page of the Travlr Getaways web application. This was done so that the page was more visually appealing, as well as allowing authenticated users to add and edit new trips. Within each component, there is also a module that organizes component functions and relations. A module is where code can be written for the component to perform an action that can be interacted with on the web application. Such as giving the buttons added within the add and edit trip components the ability to add and edit trips. In comparison, an Express project tends to follow what is known as an MVC structure. MVC stands the key aspects of an Express application, which are models, views, and controllers. Models are what define data structures and the logic of the web application. Views are HTML templates that are used to display information to a user. And Controllers are what handle requests made to the application's server by the user, returning the correct views for display and models for proper functionality. In combination, the MVC structure allows for a flexible backend framework that can handle data and server operations.

Advantages of SPA functionality include quicker loading times, a more seamless user experience, and they consume less bandwidth. SPA functionality allows for quicker loading times because only one page needs to be loaded at the first request to do so. Other web applications that have multiple pages have longer loading times in comparison since each page must be loaded with every subsequent request made by the user. SPAs also create seamless user experience since loading is quick. Content dynamically changes on within an SPA as well since there are no other pages to load with different content. And since there is a single page, an SPA will consume less bandwidth since there is no need to load multiple pages within the application at one time. This helps greatly in areas with poor connection quality or for users with slower internet speed, enabling users to interact with an SPA whenever convenient. As for disadvantages, SPAs have trouble with search engines, consuming lots of browser resources, and general security concerns. With search engines, ranking can be based off different factors including the page count of a website or web application. For an SPA, this means that it can be hidden away behind multiple search results, lowering user traffic. Browsers perform a lot of the heavy lifting within SPAs, meaning that resources can become strained quickly when browsing. And cross-site scripting attacks are a common threat to SPAs, allowing malicious users to gain access to the application's code and possibly confidential user data. As for the additional functionality and SPA provides over a simple web interaction, SPAs are much more dynamic as the UI can update without needing a new page to load or refresh. Simple web interactions on the other hand do require multiple pages to update the UI with full refreshes. Overall SPAs are quicker with smooth transitions between displays of content, while simple web interactions require more loading and multiple pages to get a full user experience.

Using Postman, tests can be performed with database with the different GET, PUT, and POST API endpoints. The correct localhost URL of the web application's database must be entered within the input field first, with the correct endpoint selected beside it. From there, each key-value pair must be given an proper input for it to be entered into the database using the PUT endpoint. Once entered, the request can be made and the data entered within the MongoDB database. To check if it has successfully been entered, switching to the GET endpoint and adding the data's unique identifier to the URL. This will then search the database for the corresponding database entry and return each key-value pair. At the time of writing, I have not gotten this part of the SPA to function as somewhere along the way things began breaking. I tried following the instructions to the letter, but I must have missed something and will have to double back. If I had to guess where things went wrong, it could have been a naming error between a key value within one of the components. This would then cause error within the console and cause the page not to render properly.