

Travlr Getaways Full Stack Web 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 | 09/28/2025 | Wesley Van Wie | Implemented the Executive Summary, Design Constraints, and Component Diagram. |
| 1.1 | 10/12/2025 | Wesley Van Wie | Added Sequence and Class diagrams. Implemented descriptions for the API endpoints |
| 1.2 | 10/25/2025 | Wesley Van Wie | Implemented the User Interface as well as the Summary. |

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

Travlr Getaway’s web application architecture uses the MEAN stack. The MEAN stack is represented by **MongoDB**: The NoSQL DB that stores data as JSON

**Express.js**: The backend framework for Node.js that handles and processes server-side logic from requests made

**Angular:** The frontend framework that renders the UI, sends requests to APIs, and used to build both DPAs and SPAs.

**Node.js:** The runtime environment that allows JavaScript to run.

The customer-facing side of the application contains the static content of the web app. This content is served using Handlebars (a templating view that allows for separation of HTML from application logic). The client side contains the formatted web app that the customer will interact with. Its functionality is to make requests to the backend where Express will handle the server-sided logic and format the response back to the client side where Angular will perform the rendering and display for the UI that the customer will interact with.

The administrator single page application will also use Angular to allow for common administrative use. The functionalities that will be included in the SPA administrator are authentication and content management (i.e. CRUD methods) that can alter the content displayed by the web application as well as update the database.

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

The MEAN stack comes with challenges and design constraints that are both similar and different to other stacks. Since MongoDB (a non-relational DB) is used. Non-relational databases have constraints like enforcement of data constraints on the application rather than the DB server. The query tools are less sophisticated. In a MEAN developed application, scalability is not as simple since the DB used is non-relational.

The web-based Travlr Getaways application contains a few Design Constraints that must be considered throughout the application development. Updating static content should not require the website to be redeployed. The customer-facing side of the application should follow UI/UX best practices and should not feel frustrating to use. Clickable elements must be responsive and display the content in a timely manner. The admin side of the client should ensure proper authentication as well as the necessary admin functionality needed.

## [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 for the Travlr web application contains three major components:

Client, Server, and Database. The client component contains the Web Browser (the user interface), a managed client session that is authenticated by the server as well as the Graphic Library that is used for communication of rendering data from the traveler portfolio. The server side authenticates user sessions and server sessions. Upon authentication, the client can make requests through the server to the database which is contained by MongoDB. Content is gathered from the DB, formatted and sent back to the client where it is rendered to view and interact with. This system architecture follows a best practice like separation of concerns. The client is used to interact, display, and request data -> the server authenticates, handles client requests, and serves client requests -> the database persists the content and responds to queries from authenticated requests.

### Sequence Diagram

A diagram of a diagram

AI-generated content may be incorrect.

The Sequence Diagram begins with the user entering a route through an address in their browser. Angular which is handling the Routing, takes the address sent from the user and routes it to the proper controller to handle the HTTP request. Once the client has sent a valid HTTP request to the server-side, Express takes the request and moves it to the servers Controller functions. The Controller functions on the Server-Side handles the request by accessing the necessary service / model function to get the data from the database. When accessing the database, Mongoose is used to send the query request where MongoDB processes and returns the queried data to the server’s controller. The Controller formats the data into JSON and finalizes the request by responding to the client with the JSON data. Angular then handles the response by updating the view with the acquired data.

## Class Diagram

A diagram of a travel application

AI-generated content may be incorrect.  
The Travlr Getaways Class Diagram contains the classes and dependencies that allow data to be passed along to other classes for the overall functionality of Travlr Getaways. The TripInfo class is abstract and contains relevant information that other classes like CruiseInfo, FlightInfo, and HotelInfo are used for specializing in their specific trip type. Each of these classes all contain public members that are shared and depended on by other classes. Each of the Info classes has Booking classes that are specific to the type of Info class. The relationship between the Itinerary class and the Info classes shows that an Itinerary can have and use multiple Info classes but are not required. The TravelAgent class sits in the center and has public methods that depend on information from the Info and Itinerary classes. These methods in the TravelAgent class are used for booking Packages, Flights, Hotels, and Cruises. On the left, we have the TravelerInfo class. The Booking and TravelAgent classes depend on this class to pass the number of companions for further details of the trip. The MemberAccount holds public attributes related to a member’s information, like their member number, frequently used airlines, status, and club. The MemberAccount has a Membership\_Admin that contains information related to a member’s reward points.

API ENDPOINTS

| **Method** | **Purpose** | **URL** | **Notes** |
| --- | --- | --- | --- |
| **GET** | To GET all trips stored in the database. | /api/trips | Retrieves an Array containing all trips stored in the DB formatted into JSON |
| **GET** | To GET a single trip by its trip code | /api/trips/:tripCode | Retrieve a single trip specified by id of the trip’s code. Returns a JSON response of all information that is related to the trip code specified. |
| **POST** | To POST a user into the database | /api/register | Registers a user into the user database, the user is given a JWT token |
| **POST** | To POST an active user to login into the frontend | /api/login | Sends a request with user credentials, if the user is authorized, they are given access to login functions of the frontend. Responds with a JWT token |
| **PUT** | To PUT an edit of a current trip | /trips/:tripCode | Sends a request to update an existing trip with a new field. Returns a JSON response with a body containing the updated trip. |

## 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.>

A screenshot of a cellphone

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**Summary:**

The Angular project structure serves as the frontend (client side) of the project. Angular is considered standalone meaning that the components are independent of each other (for maintainability purpose). Angular handles the frontend routing, displaying of UI, and requests to APIs. The Express project structure is the backend (server) side of the application, Express handles incoming requests and routes those requests the proper methods to handle them. Express delegates to Mongoose for queries to MongoDB. Express then sends the data that was requested back to Angular so it can be formatted and displayed for the client. Both Express and Angular can set up security proxies to protect the transmission of data as well as authorization of requests.

SPA’s provide rich functionality that enhances interaction of the web application. Without the need for reloading and rendering for every single page a user visits, an SPA can dynamically adjust data within the same page providing performant and enjoyable web apps.

To ensure that an SPA is working properly, we can implement testing of the application. There are multiple methods that can be used, but for this project, console logging, Postman, and manual testing was performed. Console logging provides direct feedback and is especially useful when implementing button functionality. Postman was used to ensure that a request to the API received an expected response. Manual testing was performed as well which included interacting with all elements of the SPA and checking MongoDB to ensure CRUD methods functioned properly.