

<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 | 01/30/25 | Sean Born | Completed the following sections:   * Executive Summary * Design Constraints * System Architecture View: Component Diagram |
| 1.1 | 02/06/2025 | Sean Born | Completed the following sections:   * Sequence Diagram * Class Diagram * API Endpoints |

## 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)

Travlr Getaways is wanting a web application that is aimed at improving their overall customer service as well as making the administrative processes more streamlined. This application uses MEAN stack, which is working with MongoDB, Express.js, Angular, and Node.js. This allows for the building process of the web application to be dynamic and responsive. MEAN stack is used because of compatibility functions, performance abilities, and its ease of scalability. MongoDB is the NoSQL database which provides data storage. Express,js and Node.js are what control the server-side and API routing. This provides a strong and secure connection between various parts of the application. Angular is used for the front end of the application and allows the updates performed to be dynamic and responsive.

The customer-facing application is designed so that it is engaging and intuitive for the users. It is built using Angular. This allows for updates to be dynamic and responsive to the user’s experience. It will allow customers to look-up, book, and manage travel plans. With the use of in-house APIs, it is able to interact with the backend.

The admin single-page application (SPA) is built using Angular. This allows for admin to have a user-friendly interface that allows them to manage the application, the users, and the database. Functions that are within the application include user management, control of booking, and system configuration. The SPA communicates with the backend by secure APIs. This ensures that the application has the ability to be managed efficiently.

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

Some of the design constraints that are present within Travlt Getaways include scalability, security, and performance.

In terms of scalability, the application needs to be able to be scaled for many users that are booking, purchasing, or even just using the application all at one time. This will require the database queries to be efficient, the code to be optimized, and for the server infrastructure to be strong enough to handle the load.

In terms of security, the users data must be secure as it is very important. Especially since users are entering sensitive personal information. This requires the application to implement authentication and authorization systems that are strong enough to ensure the data is safe, encrypting data, and an API that is secure to protect that information.

In terms of performance, it is a priority for customer-facing application and the SPA. Requirements include fast loading speeds, interfaces that are responsive, and the downtime is kept to a minimum when there is a issue. It must also be able to balance the load across various server clusters.

## [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 of the web application includes the client side, the database, and the server side. The client side includes components called client session, web browser, traveler portfolio, and graphic library. On the client side, the client session interacts with web browser and the traveler portfolio. It also interacts with the Server side. The web browser interacts with the traveler portfolio. The traveler portfolio interacts with the graphic library as well as MongoDB in the database. The database side includes MongoDB which is what runs the database to store the data. This interacts with the traveler portfolio as well as Mongoose ODM in the server side. Within the server side the components include authentication server, server session, Traveler database, and Mongoose ODM. The server session interacts with the Traveler database and Mongoose ODM. Mongoose ODM interacts with MongoDB within the database.

### Sequence Diagram

A diagram of a trade end

AI-generated content may be incorrect.

The flow of logic for the web application represented in this sequence diagram is as follows. It starts with a actor or user. The actor will enter a route and will be directed to the browser/view/templates for the site. The view will call on the corresponding controller which will ultimately show the view that will be displayed to the actor. The client side controller will call functions within the HTTP client to get the information. The results are passed back to the controller. The HTTP client connects the client side to the server side by making API calls to certain routes. This then allows the appropriate server side controller to be called. Then the controller is able to make a call to the database in MongoDB. MongoDB will receive and process the result to be returned. When that information is returned from MongoDB is passes the results back to the HTTP client.

## Class Diagram

A diagram of a travel geysers class diagram

AI-generated content may be incorrect.

In the class diagram above each mode of travel has a class that has a name property as well as other properties that are specific to that mode of travel. These classes are CruiseInfo, FlightInfor, and HotelInfo. Each of these classes will also inherit the properties from the TripInfo class. Each mode of travel also has separate classes for booking which are CruiseBooking, FlightBooking, and HotelBooking. Each of these are also associated with the corresponding Info class for that mode as well as the TravellerInfo Class. The Travel\_Agent class is associated with CruiseInfo, FlightInfo, HotelInfo, and TravellerInfo classes. There is also a one-to-many relationship with the Membership\_Admin class. The TravellerInfo class will inherit the MemberAccount class attributes. The Membership\_Admin class has ab aggregate relationship with the MemberAccount class. Lastly, there is also an aggregate relationship between the CruiseInfo, FlightInfo and HotelInfo classes with the Itinerary class.

## [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/things> | <Returns all active things> |
| **GET** | <Retrieve single thing> | </api/things/:thingId> | <Returns single thing instance, identified by the thing ID passed on the request URL> |
| **POST** | Create new list of things | /api/things | Creates a new list of things |
| **POST** | Create single thing | /api/things/:thingId | Adds new users to database, returns JWT |
| **PUT** | Update and replace a full list of things | /api/things | Returns all meals |
| **PUT** | Update single thing | /api/things/:thingId | Updates and replaces  single thing instance,  identified by the thing ID  passed on the request  URL |
| **PATCH** | Updates a full list of things | /api/things | Updates and modifies full  list of things |
| **PATCH** | Update ad change single thing | /api/things/:thingId | Updates and modifies  single thing instance,  identified by the thing ID  passed on the request  URL |
| **DELETE** | Delete full list of things | /api/things | Delete full list |
| **DELETE** | Delete single thing | /api/things/:thingId | Deletes single thing  instance, identified by the  thing ID passed on the  request URL |

## The User Interface

A screenshot of a computer screen

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a website

AI-generated content may be incorrect.

Angular follows a modular, component-based architecture that organizes the application into reusable components, services for data handling, and modules for functionality grouping, making it ideal for large-scale single-page applications (SPAs). It also has a more complex setup with specific folders for assets, environments, and testing, making it suitable for scalable and maintainable code. While, an Express HTML customer-facing page is typically a simpler, server-side rendered structure with routes, views (HTML templates), and static assets for minimal client-side interaction. Express focuses on handling HTTP requests, while Angular is designed to manage the UI and interactions on the client side.

Testing to make sure the SPA is working with the API to GET and PUT data in the database involved verifying on MongoDB Compass when changes where made using postman. Some errors that I ran into were syntax error within my code that were fixed after going through the code to make sure all spelling and characters used were correct. Expected error that could come up may be spelling when entering the information or using the wrong format.