

Travlr Getaways 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 | 9/15/2023 | Ryan St George | Module 3 Milestone Revisions / Setup |
| 1.1 | 9/30/2023 | Ryan St George | Sequence & Class Diagram |
| 1.2 | 10/15/2023 | Ryan St George | Finish Document / Final Touches |

## 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)

In today’s world there is a need for an intuitive, robust, and secure web applications. Travlr Getaways new website delivers both a smooth user experience and an efficient administrative platform for managing the services they offer. In using the MEAN stack approach, we are using a NoSQL database that offers flexibility and scalability, crucial for managing a variety of data. Along with this we are using Node.js for both the front end and back end to enable the seamless flow of data and for quicker development. Alongside (M and N) we have Express.js to simplify tasks of writing server code for fast and robust server responses. And AngularJS for the foundation of the front-end, promoting dynamic web pages and a responsive design. By adopting the proposed architecture and the MEAN stack for development, we are not only adhering to industry standards best practices, but we are also ensuring an efficient solution for Travlr Getaways.

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

The application must function seamlessly across various platforms, including Windows, macOS, and mobile devices like Android and iOS. Along with ensuring browser capability to the major browsers like Safari, Chrome, and Edge. Depending on what we will need to store we need to ensure we have the correct amount of data storage for this information. From personal to financial information. We also want to ensure good load time to web pages and payment processing systems. All while ensuring, we are remaining within budget for the customer. And lastly, we want to ensure this project gets completed within the correct amount of time using some Agile framework techniques to ensure we are getting the correct and proper feedback while continuously developing the webpage.

## [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).

As illustrated in the component diagram above we can see this follows a modular approach. Emphasizing separation of concerns and scalability. By dividing the application into discrete components, we ensure that each part functions independently, facilitating easier updates, debugging, and maintenance. From the client side seeing with the AngularJS application, we can see the interactive, responsive interface the client uses. The server side which contains Node.js, Express.js, routes, controllers, middleware, where we have access to the request and response of objects. And lastly the database using MongoDB the chosen NoSQL database to store users’ data, booking details, and travel destinations.

As shown the relationship between components is crucial. The AngularJS application makes requests to the server, primarily via the services player which then communicates with the Express.js API routes. And upon reaching the server, each request goes through various middleware functions depending on what was needed. For instance, authentication, logging, or data validation. The routes then direct the request to the appropriate Controller which contains the logic to process the request and interact with the database through Mongoose. And lastly the processed response travels back through the Express.js framework, to the AngularJS frontend, where the data gets displayed to the user. Or sometimes requesting further operations from the user. This is a comprehensive view of the Travlr Getaways architecture with a modular design means that will have minimal disruption to the entire system and allow for easier modifications and scalability.

### Sequence Diagram

A diagram of a diagram

Description automatically generated

Starting at the front end of the application the user initiates an action such as logging in or asking to view a list of travel destinations. The front-end layer of the web application receives the request. The back-end layer which will manage the business logic and data storage for the program, receives a request from the front-end layer. This request could contain for instance a travel destination search. The front-end layer then modifies the user interface to show the required information after receiving a response from the back-end layer. The back-end layer for instance if it was a log on may verify the users’ information/credentials against the database of users. It may then return a response if it was successful, or for travel destinations it will return them from the database.

## Class Diagram

A diagram of a flight

Description automatically generated

The JavaScript classes provided above are a basic representation of the classes that can be used in the Travlr Getaways web application. Each class represents a specific part of the application and contains properties and methods that are related to the entity. Looking at the diagram above we can see FlightInfo which displays the name, the seat class such as economy or 1st class, and the price. Putting that all into the class allows for ease of use throughout the application and really breaks down what is required. And each class has its own parameters which we can see above.

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

| **Method** | **Purpose** | **URL** | **Notes** |
| --- | --- | --- | --- |
| **POST** | Login for user | /api/login | Authenticates a user and returns a JWT |
| **POST** | Register a user | /api/register | Add a new user to the database and return JWT |
| **POST** | Add a trip | /api/trips | Add a new trip into the DB |
| **PUT** | Update a trip | /api/trips/:tripCode | Updates a single trip defined by the trip code |
| **GET** | Retrieve list of rooms | /api/rooms/:roomCode | Retrieves all rooms |
| **GET** | Retrieves single room | /api/rooms/:roomCode | Returns single room |
| **GET** | Retrieves list of trips | /api/trips | Retrieves all trips |
| **GET** | Retrieves single trip | /api/trips/:tripCode | Returns single trip |
| **DELETE** | Delete single trip | /api/trips/:tripcode | Deletes single trip |
| **GET** | Retrieves all flights | /api/flights | Returns single room |
| **POST** | Add a new flight record | /api/flights | Adds a new flight |

## The User Interface

Angular follows a component-based architecture where each component represents a view with its own logic and data. Whereas Express, is organized around middleware and routes. In Angular the modules are a group of related components and services and in Express it doesn’t have this same modularization. Also, Angular offers two-way data binding allowing for automatic updates between the modules and views. Express does not have this feature.

SPA in Angular provides a seamless user experience by loading content dynamically without refreshing the entire page. This contrasts with simple web application interactions where each page request reloads the entire page from the server. And we can test the SPA with API by setting up automated integration testing with tools like Jasmine and Karma within Angular. Which can be much more efficient than manual testing. We can then simulate many API responses to test the SPA’s ability in handling them. Also, with testing the GET and PUT requests to ensure they are correctly sent and saved within the database all while monitoring the network within chrome tools. This is an excellent way of troubleshooting.