

# **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 | 01/25/24  02/11/2024  02/24/2024 | Zoe Domagalski | Updated Executive Summary, Design Constraints, and System Architecture View: Component Diagram.  Added the diagrams and their descriptions.  Added the user interface. |

## 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 aims to have a strong web application that integrates customer-facing features with an efficient administrator single-page application (SPA). The approach involves utilizing the MEAN stack (MongoDB, Express.js, Angular, and Node.js) to deliver a scalable and modern web experience. A unified JavaScript stack streamlines development, making it efficient and easier to maintain.

The web application will consist of two main components:

* **Customer-Facing Website:**
  + Developed using Angular on the front end for a dynamic and responsive user interface. Utilizes Express.js on the back end to handle server-side logic and API interactions. Data is stored in MongoDB, providing flexibility and scalability for customer-related information.
* **Admin Single-Page Application (SPA):**
  + Angular is utilized for the admin interface, making sure there is consistent user experience. Express.js manages server-side functionalities and communicates with MongoDB for admin-related data storage.

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

* **Technology Stack Limitation:**
  + The choice of the MEAN stack (MongoDB, Express.js, AngularJS, Node.js) is fixed by requirement or preference. This limits the use of alternative technologies and frameworks for development, as you must adhere to the components of the MEAN stack for consistency, integration, and compatibility reasons.
* **Scalability Considerations:**
  + The web app must support thousands of concurrent connections, leading to potential scalability challenges. This will require careful optimization and performance testing to handle it efficiently.
* **Single Page Application (SPA) Architecture:**
  + AngularJS, the front-end framework in the MEAN stack, promotes the development of Single Page Applications (SPAs) where most of the application's UI and logic are loaded initially and subsequent interactions are handled dynamically without full page reloads.
* **NoSQL Database:**
  + Developers need to understand the principles of NoSQL databases, such as schema flexibility and document-based storage, to successfully design and interact with the database layer of the application.

## [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 architecture of the Travlr Getaways web application is characterized by its client-side, server-side, and database components.

On the client-side, components such as the Web Browser, Client Session, Traveler Portfolio, and Graphic Library are responsible for managing user interactions, session management, travel information, and graphical rendering.

The server-side components handle backend functionalities. The Authentication Server confirms secure access to the application, while the Server Session manages session state and interacts with the Traveler Database to retrieve and update user data. Additionally, the Mongoose ODM allows communication between the MongoDB database and the server-side components.

### Sequence Diagram

A diagram of a diagram

Description automatically generated

**Routing and View Rendering:**

* The flow begins with an incoming request to the web application.
* The retrieved data is then passed to the view layer for rendering, where HTML templates are generated dynamically based on the received data.
* The rendered view is sent back to the client (web browser) for display.

**User Interactions:**

* Upon viewing the rendered page, users interact with the application by performing actions such as viewing the home page and clicking through the options for travel, rooms, food, etc.

**Controller Logic:**

* When the server receives requests from the client, the corresponding controller functions handle the incoming requests.
* For example, if a user clicks the travel tab, the travel controller function processes the request.

**Service Layer Interactions:**

* When receiving requests, controller functions interact with the service layer to perform logic and data manipulation tasks.
* For example, when a user requests to view trip information, the Trips controller function may interact with the Trips service to fetch trip data from the database.
* When fetching the trip data, the Trips service uses ODM (Object-Document Mapping) tool within Mongoose to interact with the database (MongoDB).
* Additionally, the admin service can interact with the database to perform operations such as user management and updating settings.

**Response Generation and Delivery:**

* Once data manipulation and logic tasks are completed, the server generates the HTML pages and JSON data, then sends it back to the client.
* The client receives the response and updates the user interface to show any changes or new data.

## Class Diagram

A diagram of a travel company

Description automatically generated

**TripInfo:** Represents information about a trip, including total price, total miles, and stopover information.

**MemberAccount:** Stores details about a member's account such as their member number, frequent flyer airline, member status, and club affiliation.

**Membership\_Admin:** Manages membership-related operations such as crediting points, retrieving points for a member, and validating membership.

**TravellerInfo:** Contains information about the number of companions.

**Travel\_Agent:** Responsible for booking various travel components like packages, flights, hotels, and cruises.

**CruiseInfo:** Holds details about a cruise, such as its name, cabin type, and price.

**FlightInfo:** Stores information about a flight, including its name, seat class, and price.

**HotelInfo:** Contains details about a hotel, including its name, star rating, location, rooms requested, and price.

**HotelBooking:** Handles operations related to hotel bookings, such as retrieving hotel information.

**FlightBooking:** Deals with operations regarding flight bookings, such as retrieving flight information.

**CruiseBooking:** Manages operations related to cruise bookings, such as retrieving cruise information.

**Itinerary:** Central starting point. Represents a travel itinerary, including details like starting date, return date, origin, and destination.

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

| **Method** | **Purpose** | **URL** | **Notes** |
| --- | --- | --- | --- |
| **GET** | Retrieve list of trips | /api/trips | Returns a list of all the trips stored in the database |
| **GET** | Retrieve single trip | /api/trips/:code | Returns a single trip based off the code store for a specific trip. For example code: GALR210214 |
| **POST** | Create a new trip | /api/add | Adds a new trip listing to the list of trips. |
| **PUT** | Update a trip | /api/edit | Updates an existing trip that has been edited. |
| **DELETE** | Delete a tip | /api/delete | Deletes a trip from the list of trips. |

## The User Interface

**Angular Project Structure:**

* **src:** This is the main folder containing all the source code for the Angular application.
* **app:** This folder contains the core functionality of the application, including components, services, modules, etc.
* **trips folders:** These folders contained html and the components for all the trip information, such as listings, cards, adding trips, and editing a preexisting trip.
* **Login:** This folder contains the html for login screen and its component for handling the login application.
* **Home:** This folder is the home screen for the website, noting the user needs to login before viewing any trip information or adding or editing trips.
* **services:** This folder contains services that handle data retrieval, manipulation, and interaction with APIs. This folder also contains authentication for login.
* **assets:** This folder contains static files such as images, fonts, and other resources used in the application.
* **angular.json:** This file is the main configuration file for Angular projects, including settings for build and development.
* **package.json:** This file contains metadata about the project as well as dependencies and scripts for building, testing, and running the application.
* **tsconfig.json:** This file contains TypeScript compiler options.

**Express Project Structure:**

* **src:** This is the main folder containing all the source code for the Express application.
* **routes:** This folder contains route handlers for different endpoints of the API.
* **controllers:** This folder contains controller functions that handle the business logic of the application.
* **config:** This folder contains configuration files for the application.
* **public:** This folder contains static assets such as HTML, CSS, and client-side JavaScript files.
* **app.js:** This is the main entry point for the Express application.

**Comparing Angular and Express Project Structures:**

* + Angular follows a modular structure that highlights component-based architecture, making it easier to organize and maintain large-scale applications. While Express projects typically follow a simpler structure focused on handling server-side logic and API endpoints.
  + Angular projects include features like routing and dependency injection, providing functionality for building single-page applications (SPAs). On the other hand, Express projects are used to serve static content, providing simpler web application interactions compared to SPAs.

**Testing the SPA with API Integration:**

To ensure that the Single Page Application (SPA) is effectively interacting with the API to retrieve (GET) and update (PUT) data in the database, following an organized testing approach primarily using Postman will work.

* **Setting Up Postman:**
  + Organize the API endpoints into collections in Postman. Each collection had requests for getting data (GET) and updating data (PUT).
* **Checking GET Requests:**
  + The first test gets data from the database by sending GET requests to the API.
  + This helps see if the website could fetch things like login info and trips.
* **Verifying Response Data:**
  + When the API responds, check the data to make sure it is right. Look at the status codes and the actual information returned.
* **Testing PUT Requests:**
  + After that, update data in the database using PUT requests.
  + Send requests from Postman with the new information, like updating a trip.
* **Handling Errors:**
  + Check how the website and API handled errors. For example, sending wrong requests, to see if everything worked as expected.
* **Watching Logs and Requests:**
  + While testing the website, watching the server logs and network requests can help to see any problems or errors happening behind the scenes.
* **Fixing Issues:**
  + It is a process of trying things out, seeing what went wrong, and then making improvements until everything works smoothly.

By accurately testing the SPA's interaction with the API using Postman, monitoring terminal logs, and validating network requests, this ensures that the application could seamlessly retrieve and update data in the database.