

Travlr Getaways Web Application Development

# **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 | 01/25/2024 | Brandon Goller | <Brief description of changes in this revision> |

## 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)

**Architecture Overview**: The Travlr Getaways web application is designed using the MEAN stack - MongoDB, Express.js, Angular, and Node.js, ensuring a robust, full-stack development framework. This architecture supports both the customer-facing website and the administrator single-page application (SPA). The customer-facing side offers users an intuitive interface for exploring and booking travel packages, while the admin SPA allows for efficient management of these packages and user data.

**MEAN Stack Utilization**: The MEAN stack provides a cohesive environment where each technology complements the others. MongoDB offers a flexible, scalable database for storing travel package details and customer information. Express.js and Node.js create a reliable server-side platform, managing requests and responses between the database and the front end. Angular is used to develop the SPA, providing a dynamic and responsive user experience.

**Customer-Facing Application and SPA**: The customer side of the application focuses on user interaction, presenting travel packages, and managing bookings. The admin SPA is geared towards administrative tasks, like updating travel package details, managing bookings, and analyzing user data for business insights.

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

The development of Travlr Getaways is subject to several constraints that influence its design and implementation:

1. **Technology Stack**: Being committed to the MEAN stack defines the tools and languages used, impacting how features are implemented.
2. **Scalability and Performance**: The application must handle fluctuating user loads, requiring efficient code and database design.
3. **Security and Data Privacy**: Given the nature of the application, handling user data securely and complying with data protection regulations is crucial.

**Implications of Constraints**: These constraints necessitate a careful planning approach. The choice of the MEAN stack influences the application's scalability and performance. Security considerations require implementing robust authentication and data encryption methods.

## [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 structured into three primary components: Client, Server, and Database. Each of these components contains sub-components that interact with one another to provide a seamless user experience and efficient data management.

1. **Client Component**:
   * **Web Browser**: The entry point for the user, interfacing with the Traveler Portfolio for information display.
   * **Client Session**: Manages user sessions and interacts with the server session for data synchronization and authentication purposes.
   * **Traveler Portfolio**: Serves as the user interface where travelers interact with the application, and it connects to the Graphic Library for visual elements.
   * **Graphic Library**: Supplies the graphical elements to the Traveler Portfolio, enhancing the user interface.
2. **Server Component**:
   * **Authentication Server**: Responsible for user authentication, communicating with the Client Session.
   * **Server Session**: Acts as an intermediary between the Client Session and the Traveler Database, ensuring session data is consistent and secure.
   * **Traveler Database**: Stores user data and travel information, accessed and managed by the Server Session.
   * **Mongoose ODM (Object Document Mapping)**: Interfaces between the Traveler Database and MongoDB, facilitating data storage and retrieval.
3. **Database Component**:
   * **MongoDB**: The primary database for storing all application data, connected to the Mongoose ODM for data manipulation and query handling.

**Component Interactions**:

* **Web Browser to Traveler Portfolio**: The Web Browser requires information from the Traveler Portfolio to display travel packages and user-specific data.
* **Traveler Portfolio to Graphic Library**: The Graphic Library is crucial for providing graphical elements to the Traveler Portfolio, enhancing user interaction.
* **Traveler Portfolio to MongoDB**: This connection signifies the direct interaction between the user interface and the database for data retrieval and storage.
* **MongoDB to Mongoose ODM**: Mongoose ODM acts as a layer between MongoDB and the Server, managing the object-document mapping for the application.
* **Mongoose ODM to Server Session**: This connection is vital for translating database queries into usable data for the server.
* **Server Session to Traveler Database**: Ensures secure and efficient data transactions between the server and the database.
* **Server Session to Client Session**: Maintains synchronization between client and server sessions, essential for user authentication and data consistency.
* **Client Session to Authentication Server**: Manages user authentication processes, providing security and access control.

### Sequence Diagram

<Illustrate the flow of logic in a web application by completing a sequence diagram. Insert an image of the sequence diagram here.>

<Describe the flow of logic in the web application based on the sequence diagram. Be sure to describe the interactions between the layers, or tiers, of the full stack application. It will be helpful to include significant processes such as Sign In, Trips, and Admin interactions when referring to the sequence diagram.>

## Class Diagram

<Illustrate the JavaScript classes of the web application by completing a class diagram for the web application. Insert an image of the class diagram here.>

<Describe the JavaScript classes of the web application based on the class diagram.>

## [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> |

## 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.>

<Summarize the Angular project structure and how it compares to the Express project structure. Be sure to describe the rich functionality provided by the SPA compared to a simple web application interaction. Describe the process of testing to make sure the SPA is working with the API to GET and PUT data in the database.>