

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 | 12/9/24 | Veronica Guzman |  |

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

The web application for Travlr Getaways will be designed using the MEAN stack, ensuring a modern, scalable, and robust solution for the client. The architecture will comprise a customer-facing website, which allows users to explore and book getaways, and an administrator single-page application (SPA) for managing the site’s content and user data. The MEAN stack (MongoDB, Express, Angular, Node.js) will be used for its flexibility, performance, and strong community support, enabling both the front-end and back-end to communicate seamlessly. The customer-facing site will be a dynamic interface allowing easy navigation, while the admin SPA will enable efficient management of content with a modern user experience.

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

The development of the Travlr Getaways application is subject to certain constraints. First, the application must handle potentially high traffic, especially during peak booking periods. This requires the system to be scalable, ensuring it can handle thousands of simultaneous users. Another constraint is ensuring that data security measures are in place to protect sensitive user information, including personal data and payment details. The user interface (UI) should be simple and intuitive, with accessibility in mind. These constraints will guide the development of the web application and its components, ensuring it is efficient, secure, and user-friendly.

## [System Architecture View](#_heading=h.44sinio)

### Component Diagram



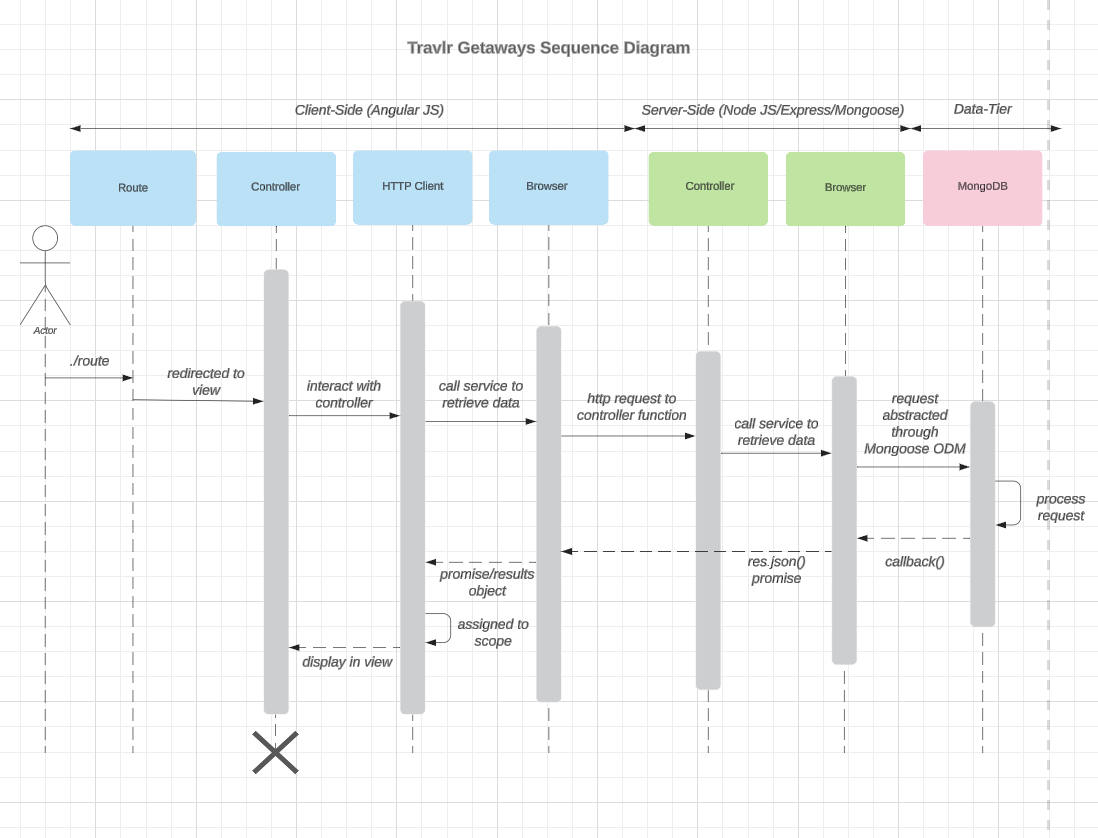
The system will be composed of several key components:

* Frontend (Angular SPA): The customer-facing website and the administrator interface will be built using Angular. Angular will communicate with the server-side Node.js API to retrieve and send data.
* Backend (Node.js and Express): The server will handle API requests, ensuring smooth communication between the front-end and the database.
* Database (MongoDB): MongoDB will store user data, booking information, and other application data, providing a flexible and scalable database solution.
* Third-party APIs: These will be integrated for payment processing and possibly other external data (e.g., maps, travel information).

The system architecture follows a three-tier model:

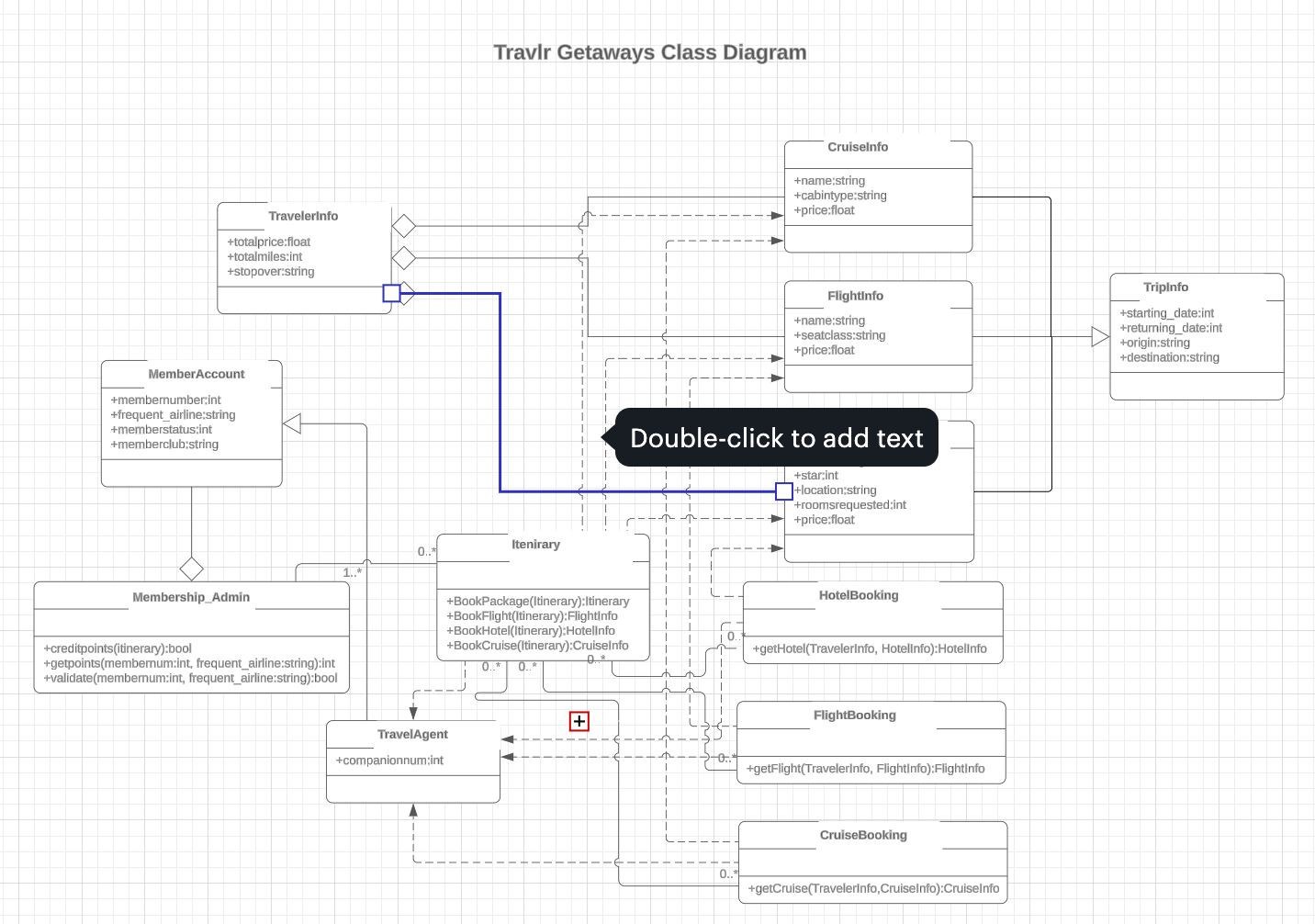
1. Presentation Layer (Frontend): Angular SPA that interacts with users.
2. Business Logic Layer (Backend): Node.js and Express to handle the API logic.
3. Data Layer (Database): MongoDB to store and manage data.

### Sequence Diagram



The diagram illustrates three primary components: the client, the database, and the server. The process begins with the user's computer, transitioning to the view. The user logs in to gain access. On the server side, a request is made to the website using MongoDB, establishing a connection with the Travlr website. A scope is then assigned, and a view is presented. Finally, the data delivers the HTTP response back to the user.

## Class Diagram

****

The class diagram illustrates the various relationships between the classes. Every user begins with a standard account but has the option to advance to higher positions and roles. For instance, the standard account role is designated for travel agents. Travel agents can use the itinerary feature to view and interact with different items. Information related to flights, cruises, and hotels is stored within the TravellerInfo class. This data interacts with the website, allowing it to display relevant information to the user during their interaction.

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

| **Method** | **Purpose** | **URL** | **Notes** |
| --- | --- | --- | --- |
| **GET** | Retrieve list of trips | /api/trips | Returns all active trips in the system |
| **GET** | Retrieve single trip | /api/trips/ | Returns a single trip by ID |

## The User Interface

An Angular project is structured to support modularity and maintainability, with its key components organized within the src/app directory. This directory contains modules for grouping related features, components for building reusable UI elements, and services for handling business logic like API interactions. Static assets such as images and styles are stored in the assets folder, while the environments folder provides configuration files for different deployment settings. The main.ts and index.html files serve as the bootstrap entry point and container for Angular’s dynamic rendering. In contrast, an Express project has a simpler structure tailored to backend development. It typically includes folders for routes, controllers, and models, where routes define API endpoints, controllers handle logic, and models define schemas for database operations. Static files are hosted in the public folder, and the main server configuration is handled in a file like app.js or server.js.

Angular Single Page Applications (SPAs) offer a richer user experience compared to simple web applications. SPAs provide dynamic, real-time updates without requiring full-page reloads, thanks to Angular's data binding and component-based architecture. This makes SPAs well-suited for interactive and complex applications like dashboards or client portals. In contrast, simple web applications rendered on the server side, such as those built with Express, are effective for static or minimally interactive sites but lack the seamless, responsive experience of an SPA.

Testing the integration of an Angular SPA with an Express API involves several steps. Unit testing in Angular can be achieved using the HttpClientTestingModule, which simulates API interactions within services, and tools like Jasmine and Karma to verify that GET and PUT requests function correctly. For example, tests can ensure a GET request fetches the expected data and that a PUT request updates it accurately. Integration testing involves mocking API responses with tools like JSON Server or Postman Mock Server to confirm that the SPA correctly displays fetched data and updates the UI after PUT requests. End-to-end testing, using tools such as Protractor or Cypress, simulates real user interactions to validate the application's behavior in a browser, ensuring data fetches and updates work as expected. Additionally, independent API testing with tools like Postman ensures the backend endpoints are reliable, returning accurate responses with appropriate status codes. This thorough testing process ensures seamless data flow between the SPA and API, creating a robust and user-friendly application.