

Travlr Web Application

# **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 | 01/28/2024 | Dan Taylor | Updated the Executive Summary, Design Constraints, and System Architecture View sections of the document. |
| 1.1 | 02/11/2024 | Dan Taylor | Updated Sequence Diagram, Class Diagram, and API Endpoints. |
| 1.2 | 02/25/2024 | Dan Taylor | Updated Executive Summary, System Architecture View, and User Interface sections of the document. |

## 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, a web application facilitating secure trip bookings for travelers exploring diverse destinations, incorporates a comprehensive structure comprising a user interface, backend logic, and a database. To develop this application, we've opted for the MEAN stack framework, incorporating MongoDB, Express JS, Angular JS, and Node JS.

In the MEAN stack, MongoDB serves as the database, Express JS functions as the web framework, Angular JS handles the client-side framework, and Node JS serves as the premier JavaScript web server.

The client-side employs the single-page application technique, where Angular JS utilizes a single HTML page and selectively updates sections based on user interactions. This approach results in quicker load times and a more responsive user experience. Additionally, Angular JS extends HTML tags with metadata, utilizing the MVC architecture for functionalities such as validation, localization, and communication with the backend.

On the backend, an Express JS framework operates on a Node JS server, managing incoming URLs by matching them with server functions. These functions may leverage MongoDB's Node JS drivers to access and update data on the MongoDB database. MongoDB consists of JSON documents that can be transmitted to the Express JS server, where they undergo validation, processing, and become accessible to the user, or vice versa.

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

The initial challenge revolved around the absence of specific general JS coding guidelines, which might initially seem like a benefit but, in hindsight, is a constraint. While creating high-quality code necessitates adherence to best practices, achieving top-quality code universally interpretable by coders globally demands adherence to specific guidelines. Scouring the internet for diverse code snippets to establish a standardized structure can be both draining and often fruitless, leading to time wastage and emphasizing the importance of time constraints in a project.

Time allocation is crucial for project completion, and projects frequently surpass their initially allocated timeframes. In our case, crafting the project's structure required ample time to achieve perfection, considering the myriad ways to arrange different components. Mere adherence to time constraints might result in a suboptimal web application.

Moreover, technological challenges emerged, particularly with JavaScript. Despite its excellence, it can burden websites, causing delays in loading. It became imperative to design the code efficiently to minimize unnecessary overhead time. Rather than viewing design constraints as hindrances, it is more beneficial to consider them as cues.

By addressing these constraints, a connection emerged between problem-solving, available resources, and criteria, preventing wastage of time and energy on irrelevant design ideas. Every client being unique, spending time researching and understanding the audience before initiating a design is crucial. Budget constraints must also be considered, as having the necessary budget is vital to meet website requirements without resorting to low-budget integration.

Additionally, adhering to timelines is crucial in the development process, where deadlines for each step may or may not be met. A flexible timeline, achieved by breaking the project into manageable pieces, helps avoid rushing when the project unexpectedly becomes a top priority.

## [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).

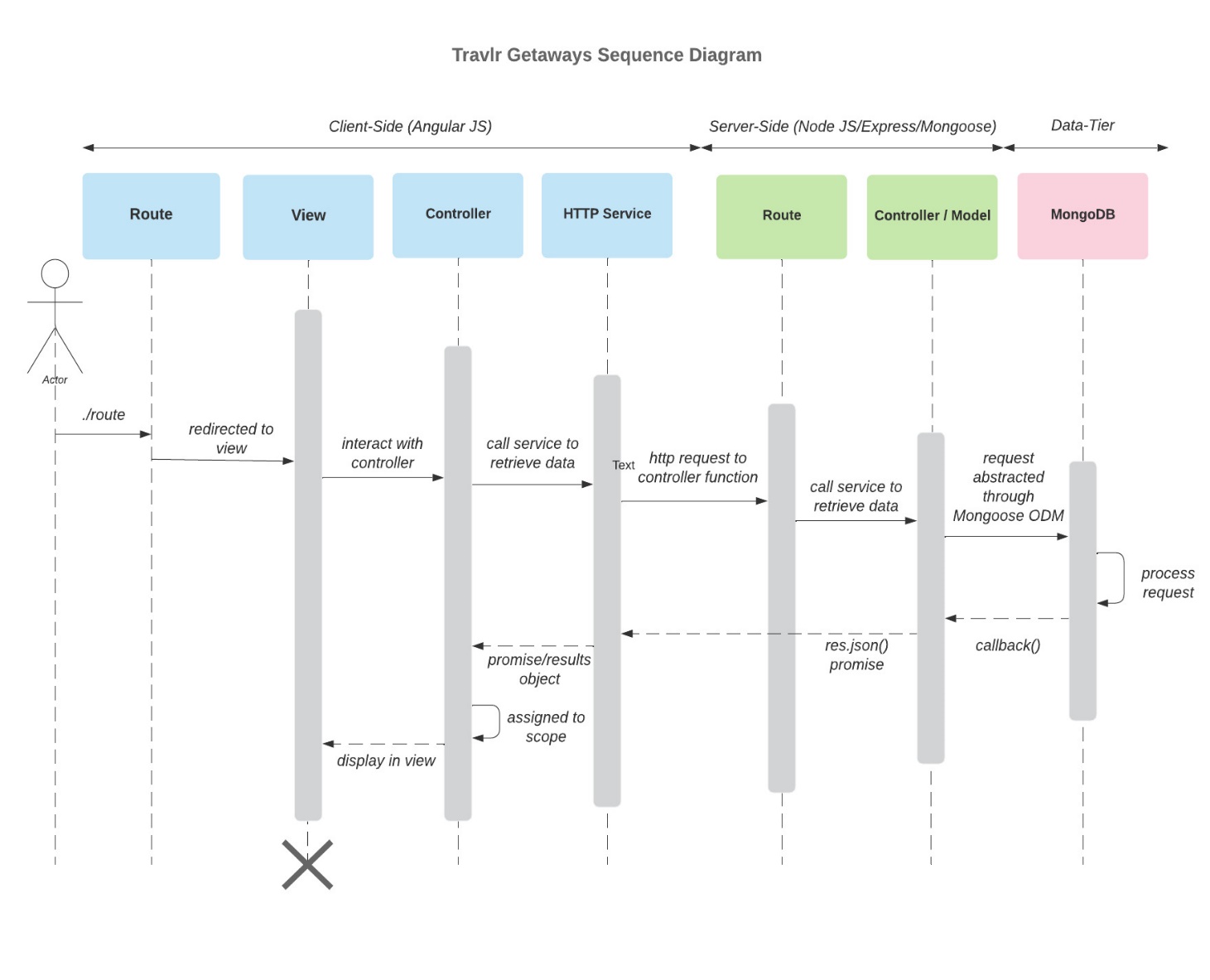
In examining the component diagram provided, the web application follows a three-tier architecture, a prevalent structure in most full-stack web applications. The three primary components include the Client Component, the Server Component, and the Database Component.

The Client Component is connected to the server through a designated port, facilitating interaction. This component engages with the server through an interface provided by the server and essential for the client-side operations. This connection serves purposes such as authentication and session control, managed by the server-side. Users, or clients, access content visually via a web browser and interact with the server through this established connection.

Moving to the Server Component, its subcomponents collaboratively handle session management, authentication, and database queries. Crucially, the business logic that offers services to users is embedded here. The authentication and session sub-components communicate with the Travlr database, utilizing the mongoose ODM to authenticate users, manage sessions, and fulfill user requests by querying the database.

Lastly, the Database Component serves as the data storage unit for the entire application. It features an interface to establish connections and houses MongoDB, a NoSQL database integral to the MEAN stack. The synergy and interactions between these components bring the entire application to fruition, highlighting the significance of their relationships in the system architecture.

### Sequence Diagram



The depicted diagram illustrates a recurring cycle involving three pivotal components: the server, client, and database. The interaction among these key elements unfolds in the following sequence. Initially, the user logs in or signs up to secure access. Subsequently, the server initiates loading and establishes a connection with the "travlr" database. Following this, the mongoose Object-Document Mapping (ODM) utilizes the obtained information to request models from the database. Finally, the database responds by presenting a user-accessible library for the individual to express their opinions.

## Class Diagram

A diagram of a travel geoways class diagram

Description automatically generated

The depicted class diagram illustrates the flow of activities within the website's code execution. Initially, the user undergoes a login process, and based on the provided credentials, the website dynamically transitions into either a member or admin state. As the user explores various itineraries, the website fetches data from the database. For instance, when the user searches for a specific trip, such as "XYZ," the relevant details regarding the cruise, flight, hotel, and bookings are retrieved from the database and dynamically displayed on the screen for the user's perusal.

## [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** | Returns all trips within the database | </api/trips> | The app\_api's trips controller utilizes a MongoDB find() function to retrieve information and returns it as a JSON object named "trips." |
| **GET** | Returns single trip specified by the trip ID parameter on the URL | </api/trips/:tripId> | The parameterized route's trips controller utilizes the parameter provided by the request to locate the corresponding trip code in the database and then returns the specific JSON data associated with that object. |
| **POST** | Register a user | </api/register> | Add a new user to database and return a token |
| **POST** | Login a user | </api/login> | Login a user to the database and return token |
| **POST** | Add a trip | </api/trips> | Add a new trip to the database |
| **PUT** | Edit a trip | </api/trips> | Edit a trip and save to the database |
|  |  |  |  |

## The User Interface

Below are some screenshots of the trip listing, and edit trip screens.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

The Angular project structure is markedly different from that of the Express HTML customer-facing page. Angular follows a modular structure, dividing the application into modules, components, services, and other Angular-specific entities. This modular approach enhances code organization and maintainability. In contrast, Express HTML projects are often more straightforward, with the server and client-side code blended together. This distinction emphasizes Angular's emphasis on modularity and separation of concerns.

Advantages and Disadvantages of SPA Functionality:

Advantages:

* Enhanced User Experience: SPAs provide a smoother and more responsive user experience by loading resources dynamically, avoiding full page reloads.
* Efficient Resource Utilization: SPAs can reduce server load as they fetch only the necessary data, minimizing bandwidth usage.
* Easy Navigation: SPAs enable seamless navigation between different views without the need for complete page reloads.

Disadvantages:

* Initial Load Time: SPAs may have a slightly longer initial load time as they need to fetch the entire application code upfront.
* SEO Challenges: Search engine optimization can be a challenge since search engines traditionally rely on the initial HTML content.

Compared to a simple web application, SPAs offer dynamic content loading, real-time updates, and improved interactivity. They enable the development of responsive and feature-rich applications without sacrificing user experience.

Testing the SPA's interaction with the API involves unit tests for individual components, integration tests for checking the collaboration of components, and end-to-end tests to ensure the entire application functions as expected.