

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 | 5/26/25 | April Nixon | Filled out the Executive Summary, Design Constraints, and System Architecture View sections |
| 1.0 | 6/8/2024 | April Nixon | Filled out Sequence Diagram, Class Diagram, and the API Endpoints table sections |
| 1.0 | 6/22/2024 | April Nixon | Filled out The Usage Interface section |

## 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)

The Travlr Getaways web application will be developed entirely in JavaScript using the MEAN stack framework. This comprehensive approach utilizes MongoDB as the database for storing and managing all application data, allowing for seamless editing and entry additions by users. The backend structure and functionality will be built on Express.js, while Angular will be employed to enhance the application's front-end features and provide a streamlined administrative interface through a Single Page Application (SPA). Finally, the entire application will run on the Node.js server environment, ensuring efficient and scalable performance.

Or (in other words)

The Travlr Getaways project aims to develop a comprehensive full-stack web application using the MEAN stack (MongoDB, Express.js, Angular, Node.js) to manage travel-related data and interactions. This application will feature two main components: a customer-facing website and an admin single-page application (SPA).

The customer-facing website will allow users to browse and book travel packages, view travel itineraries, and access various travel-related services. The admin SPA will provide administrators with tools to manage travel packages, update travel information, and handle customer queries efficiently. This dual-component architecture ensures that both end-users and administrators have tailored interfaces to interact with the system effectively.

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

When developing the Travlr Getaways web application, it's important to consider key design constraints. First, understanding customer preferences and feedback is important, as their approval will serve as a metric for design decisions. Second, careful scheduling is necessary to ensure the development process runs smoothly, avoids disrupting existing operations, and stays on track in terms of both timeline and budget.

Or (in other words)

Developing the Travlr Getaways application involves several design constraints that must be addressed to ensure a successful implementation. Firstly, scalability is an important concern as the application must handle a high volume of concurrent users, especially during peak travel seasons. This necessitates efficient database design and load balancing strategies to distribute traffic and prevent bottlenecks. Security is equally important, with a focus on protecting user data and ensuring secure transactions through authentication and authorization mechanisms to safeguard sensitive information.

Also, the application must deliver a seamless user experience with fast loading times and responsive interactions, requiring optimization of both frontend and backend performance. Compatibility across various devices and browsers is also essential to provide a consistent user experience regardless of the platform. These constraints will influence the development process by necessitating careful planning, comprehensive testing, and the implementation of best practices to meet the stringent performance, security, and scalability requirements.

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

The overall system architecture of the Travlr Getaways web application consists of several interconnected components, which include the client-side, server-side, and database layers. On the client-side, the Angular application provides the user interface for both customers and administrators, facilitating interaction with the backend through RESTful APIs. This application runs in the user's browser, rendering the interface and enabling a smooth user experience. On the server-side, a Node.js server handles HTTP requests, serves the Angular application, and acts as the middle layer between the client-side and the database. The Express.js framework within the server aids in creating RESTful APIs, managing routing, and handling various backend functionalities.

The database layer employs MongoDB, a NoSQL database that stores essential travel-related data, including user information, travel packages, and bookings. This database offers flexibility in handling dynamic data structures, making it suitable for the application's needs. The Angular application communicates with the Node.js server via HTTP requests, which the server processes and uses to interact with the MongoDB database for data retrieval and updates. By leveraging the MEAN stack, the Travlr Getaways application enjoys a unified JavaScript-based development environment, ensuring seamless integration between the frontend and backend. This approach guarantees a robust, scalable, and high-performance application capable of meeting the needs of both end-users and administrators.

**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).

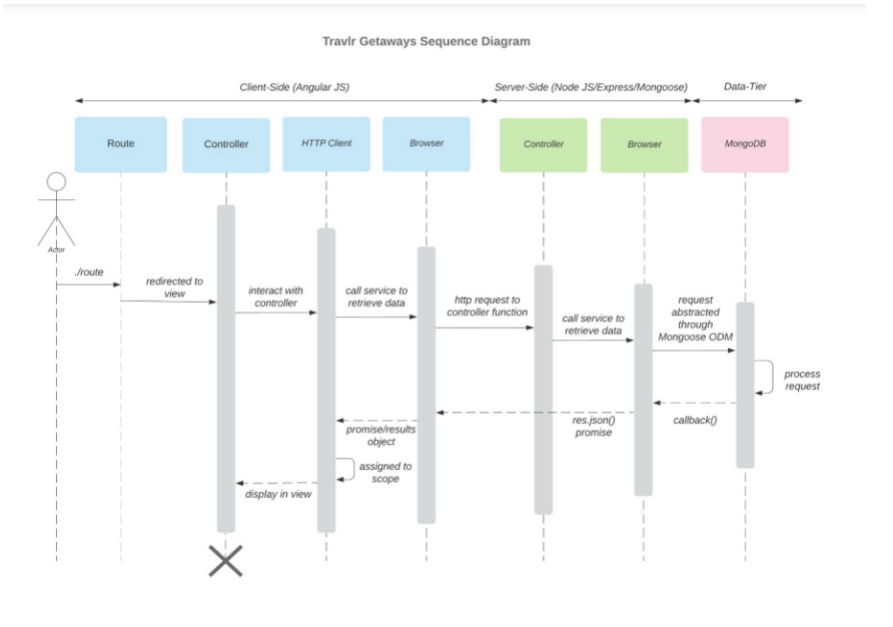
The Travlr Getaways web application is built upon three core components: the client-side, server-side, and database. The database, specifically MongoDB, serves as the central repository for all application data, allowing for efficient storage, retrieval, modification, and deletion of information as needed. The client-side and server-side components work in tandem to deliver a seamless user experience.

The client-side, interacting with the user through a visually appealing and intuitive graphical interface, facilitates the browsing and consumption of information presented by the web application. It establishes a connection with the server-side through designated ports, enabling the exchange of data and requests.

On the server-side, the application leverages the power of Mongoose ODM (Object Data Modeling) to seamlessly communicate with the MongoDB database. This interaction is crucial for verifying user credentials, ensuring the security and integrity of data, and executing various operations on the stored information. The server-side acts as the bridge between the client-side and the database, processing requests, retrieving relevant data, and returning it to the client-side for display.

This modular architecture ensures a clear separation of concerns, making the application easier to maintain, scale, and update. The client-side focuses on delivering a user-friendly experience, while the server-side handles the complex logic of data management and interaction with the database.

### Sequence Diagram



Above the diagram is showing the three key components, the client, the database and the

server. Beginning with the users’s computer we can then move to the view. The user would log in and

have access. The server side would then call to the website using MongoDB and it will connect with the

Travlr website. From there a scope is assigned and a view is displayed. The data will then deliver the

HTTP to the user.

Above the diagram is showing the three key components, the client, the database and the

server. Beginning with the users’s computer we can then move to the view. The user would log in and

have access. The server side would then call to the website using MongoDB and it will connect with the

Travlr website. From there a scope is assigned and a view is displayed. The data will then deliver the

HTTP to the user.

Above the diagram is showing the three key components, the client, the database and the

server. Beginning with the users’s computer we can then move to the view. The user would log in and

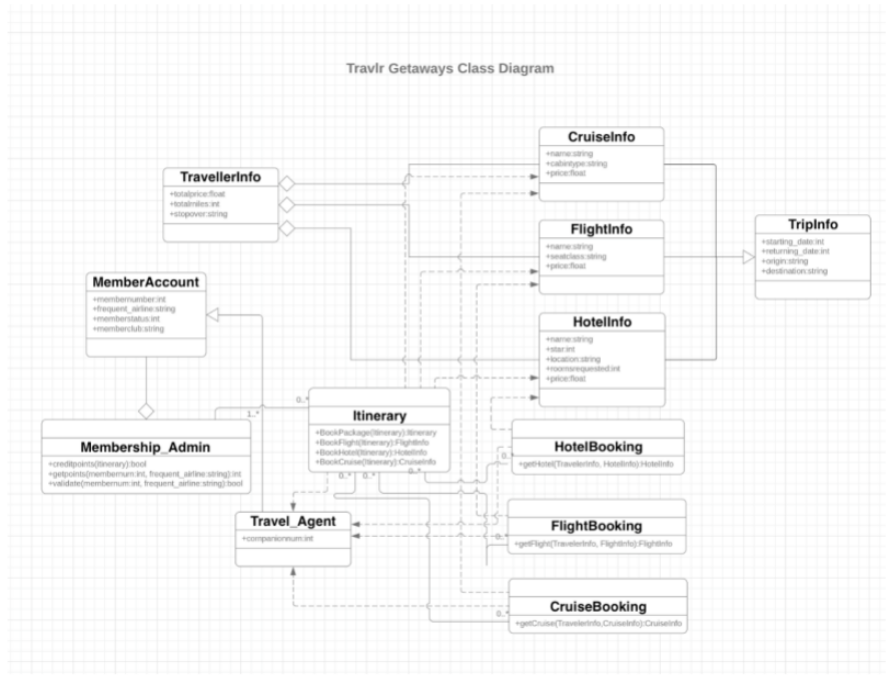
have access. The server side would then call to the website using MongoDB and it will connect with the

Travlr website. From there a scope is assigned and a view is displayed. The data will then deliver the

HTTP to the user.

Above the diagram shows the three key components, the client, the database and the server. Beginning with the users’ computer we can then move to the view. The user would log in and have access. The server side would then call to the website using MongoDB and it will connect with the Travlr website. From there a scope is assigned and a view is displayed. The data will then be delivered to the HTTP to the user.

## Class Diagram



The class diagram shows us the different relationships between the classes. Each user will start out with a standard account but can get higher positions and higher roles within. The standard account role, for example, would be travel agents. The travel agents could use the itinerary to view the different items and interact with each of them.

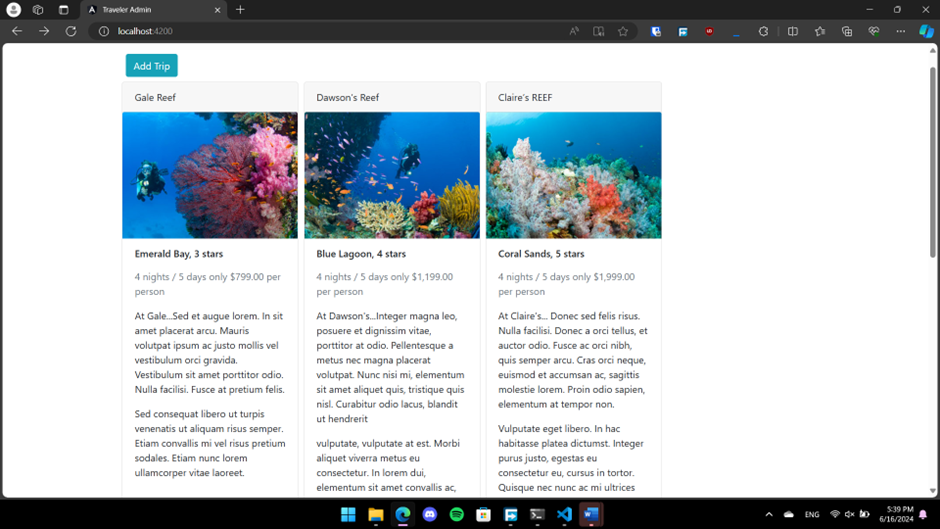
The data for Flightlnfo, Cruiselnfo and Hotellnfo's data is stored within the Travellerlnfo. The data will then interact with the website and the website will show the site's information to the user when the user is interacting with the website.

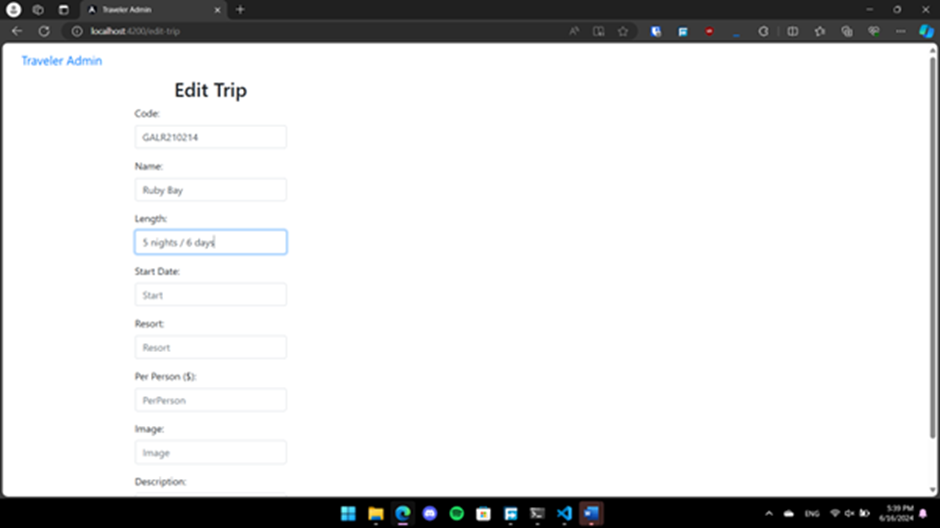
## [API](#_heading=h.2jxsxqh) Endpoints

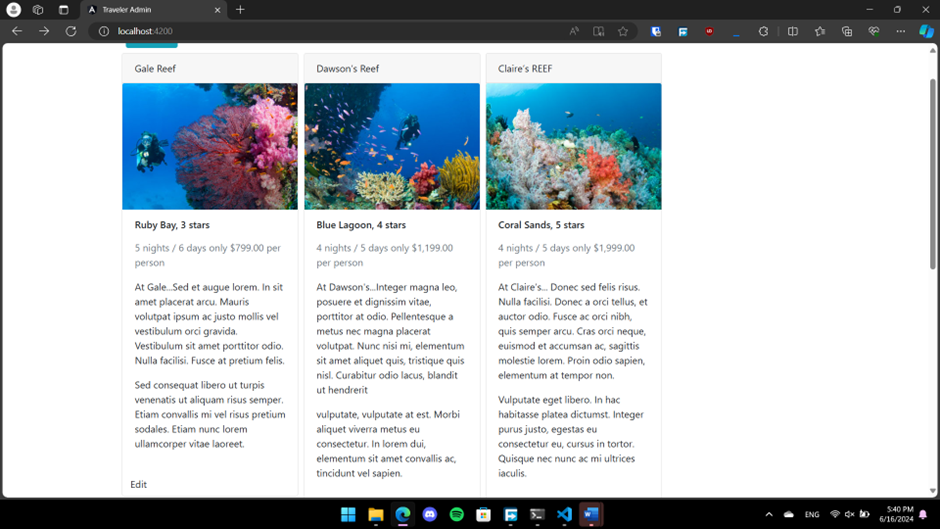
| **Method** | **Purpose** | **URL** | **Notes** |
| --- | --- | --- | --- |
| **GET** | Retrieve list of trips | /api/trips | Returns all trips in database |
| **GET** | Retrieves a single trip | /api/trips/:tripCode | Returns a single trip that matches the code entered in the URL as a parameter |
| **POST** | Creates a new single trip | /api/trips | Creates a single new trip and adds it to the database |
| **PUT** | Updates a single existing trip's information | /api/trips:tripcode | Updates the information in the database of the trip specified by the code entered as a parameter |
| **DELETE** | Deletes a single existing trip | /api/trips:tripcode | Deletes the trip information in the database of the trip that matched the code passed as a parameter |
| **POST** | Creates a new user account | /api/register | Creates a single new user account based on the parameters passed through the form |
| **POST** | Creates a new JWT token for the user's session | /api/login | Creates a single new JWT authentication token for the user's session based on the parameters passed through the form and validation of the credentials |

## 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.>







The primary distinction between standard customer-facing Express HTML and Angular projects lies in how they're loaded. Express HTML loads only the currently viewed section, fetching new HTML from the server for each change. In contrast, an Angular SPA loads the entire application at once, allowing seamless navigation between sections without server communication for each change.

Imagine Express HTML as being in a one-room building, where changing rooms require rebuilding around you. An SPA is like a multi-room building, where you can instantly teleport between rooms.

SPAs offer faster loading times, smoother user experiences, and lower bandwidth usage. However, they may not be ideal for search engine optimization due to their single-page nature, and they can be more demanding on browser resources.

Testing SPAs usually involves checking API responses or using end-to-end testing. A key question is whether SPAs can coexist effectively with multi-page applications.

OR

Angular and Express are two popular frameworks used in web development, each with their own project structure and capabilities. Angular, designed for building Single Page Applications (SPAs), follows a structured approach with directories for end-to-end testing (e2e), installed npm packages (node\_modules), and source code (src). Within the src directory, the app folder contains the main application code, including components, services, and modules. The assets folder holds static assets like images and styles, while the environments folder contains environment-specific configuration files. The index.html file serves as the main HTML file, and the main.ts file is the entry point for the Angular application. Also, global styles are defined in the styles.css file. Configuration files such as angular.json and tsconfig.json play important roles in managing the Angular project setup and TypeScript configuration.

Express, a web application framework for Node.js, has a simpler and less opinionated structure compared to Angular. The main application file (app.js) initializes and starts the server. The bin directory contains executable scripts, while the data directory is used for data storage and database files. Public assets like images, CSS, and client-side JavaScript are stored in the public directory. The routes directory defines the application’s endpoints, and the views directory contains templates for server-side rendering. Similar to Angular, the package.json file lists project dependencies and scripts, and package-lock.json locks the versions of these dependencies.

Angular projects are centered around components and services, promoting a modular and maintainable codebase suitable for dynamic and responsive user interfaces. The functionality of SPAs built with Angular includes dynamic content updates without full page reloads, client-side routing for fast navigation, and a component-based architecture that enhances reusability and maintainability. Angular’s two-way data binding synchronizes the model and the view, simplifying the management of user input and updates. In contrast, simple web applications built with Express often follow a server-side rendering approach, where each user interaction can trigger a full page reload, resulting in slower performance and less dynamic user experiences.

Making sure the SPA works correctly with the API to GET and PUT data in the database involves several steps. First, unit tests should be written for individual components and services using frameworks like Jasmine and Karma for Angular. Integration testing verifies the interaction between the Angular frontend and the Express backend, with tools like Protractor used for end-to-end testing. API testing with tools like Postman helps manually verify that GET and PUT requests return the expected responses. Automated testing frameworks such as Mocha or Jest can be used for running tests on the Express backend, while Angular’s TestBed facilitates testing Angular components and services. Also, using mock services to simulate API responses during testing can help isolate frontend tests from the backend, making sure the frontend logic is correct. These steps make sure that the SPA interacts correctly with the API, retrieving and updating data as expected.