

Travlr Getaways Web Application

# **CS 465 Project Software Design Document**

Version 1.2

## Table of Contents

[**CS 465 Project Software Design Document** 1](#_Toc174876029)

[Table of Contents 2](#_Toc174876030)

[Document Revision History 2](#_Toc174876031)

[Executive Summary 3](#_Toc174876032)

[Design Constraints 3](#_Toc174876033)

[System Architecture View 4](#_Toc174876034)

[Component Diagram 4](#_Toc174876035)

[Sequence Diagram 5](#_Toc174876036)

[Class Diagram 6](#_Toc174876037)

[API Endpoints 7](#_Toc174876038)

[The User Interface 7](#_Toc174876039)

[Angular vs Express Structure 10](#_Toc174876040)

[Testing 11](#_Toc174876041)

## [Document Revision History](#_heading=h.lnxbz9)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 07/17/2024 | Mary Cigliola | Completion of Executive Summary, Design Constraints, and System Architecture View sections |
| 1.1 | 08/04/2024 | Mary Cigliola | Completion of Sequence Diagram, Class Diagram |
| 1.2 | 08/18/2024 | Mary Cigliola | Completion of User Interface section |

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

The Travlr Getaways web application will be built using the MEAN technology stack, which includes MongoDB, Express.js, Angular, and Node.js. Each of these are powerful tools and together they allow for the design of a web application that is flexible, responsive, and easy to maintain.

* MongoDB is a scalable schema-less database, ideal for handling large volumes of data for high-traffic applications. Data is stored in JSON-like documents, which allows for improved query performance and horizontal scaling.
* Express.js is a web app framework for Node.js. It supports the use of templating engines, like Handlebars, which simplify development and add flexibility.
* Angular is a framework for developing dynamic, single-page applications (SPA) for the front end.
* Node.js is a runtime environment that enables execution of server-side JavaScript, supporting the entire application. It uses event-driven architecture which makes it appropriate for real-time applications.

The Travlr Getaways application will have two main components:

* User-Facing Side: Built with Express.js and Handlebars, this side will generate dynamic content views.
* Admin-Facing Side: Implemented as a SPA built with Angular, this side will provide a responsive administrative interface for managing bookings, user accounts, and travel offerings.

Both interfaces will interact with MongoDB to retrieve and manage data. With this architecture, the Travlr Getaways application will be robust, scalable, and capable of providing excellent user experience.

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

To reach new potential clients, the application must support Search Engine Optimization (SEO) and compatibility with multiple platforms. Ensuring cross-platform compatibility requires using responsive design techniques, such as CSS media queries and flexible grid layouts. Extensive testing across a variety of browsers, devices, and screen sizes will be critical.

Supporting SEO allows for improved visibility in search engines, which will bring new traffic to the application. SEO is achieved by ensuring crawlers can read HTML and meta tags to index content effectively. The choice to use Express.js for the user-facing side, rather than an Angular SPA, provides crawlers with the necessary information for SEO.

Integration with third-party services, such as payment gateways, is also a requirement. The use of carefully chosen APIs, such as Stripe for secure payment processing, will facilitate this integration. Other potential integrations include Google Maps for location services and OAuth for social media login. Proper error handling mechanisms to manage interruptions to service will help create a smooth customer experience. This will involve implementing logging and monitoring tools to resolve issues.

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

### Component Diagram

The diagram below represents the overall system architecture of the Travlr Getaways web application. There are three main components: the Client, Server, and Database components.

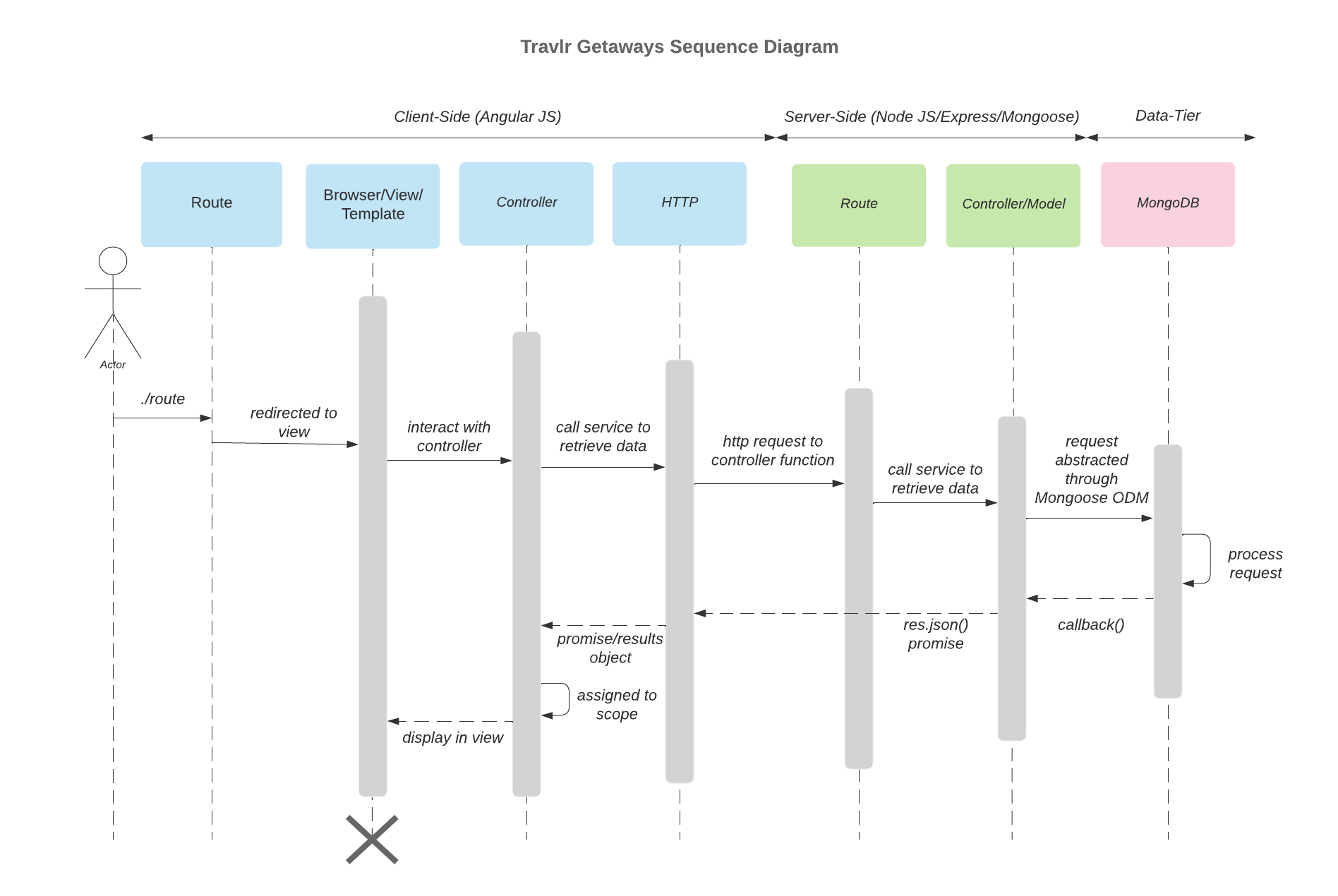
The Client component includes the web browser, client session, traveler portfolio, and graphic library. The web browser is the interface through with the user interacts with the application. The client session keeps track of the user’s state and manages the session on the client side. The traveler portfolio handles user interactions, utilizing the graphic library for rendering images, maps, and charts. The graphic library provides the graphic functionalities that the traveler portfolio uses to render images and other visual elements.

The Server component consists of the authentication server, server session, traveler database, and Mongoose ODM. The role of the authentication server is to ensure that only authorized users can access certain parts of the application. The server session maintains the user state and manages interactions between the client and server. Traveler-related data is retrieved through the traveler database. Mongoose is an Object Data Modeling (ODM) library that facilitates interaction with MongoDB, ensuring data validation through schemas.

Finally, the Server component is MongoDB. This is a NoSQL database where all data is stored.



### Sequence Diagram



The diagram above shows the sequence of interactions between the layers in the MEAN stack supporting the Travlr Getaways web application. To understand the role of each layer, it is helpful to look at the sequence through the lens of specific processes. Signing in to the application and viewing trip offerings are similar, while there are slight differences when using the admin interface.

**Signing In/Viewing Trips**

The actor in the diagram is the user who enters a route for a page they wish to access, such as the sign-in page or the trips page. They might enter the URL directly or click on a link to the route. The Express.js router handles the route and renders the Handlbars view in the browser. Next, the user submits their login credentials to sign in or initiates a request for trip details. The view calls the appropriate controller, which in turn sends the request through HTTP.

The HTTP GET request is sent to the corresponding route in the Express.js server to fetch the required data. After receiving the GET request, the route directs it to the correct controller, such as the trips controller for a request to view trips. The controller uses Mongoose ODM to query the MongoDB database. MongoDB will return this data through Mongoose to the controller. In the case of a user signing in, if the credentials are valid (i.e. the database query was successful), the controller will create a token. In the case of a request to view trips, the controller will format the data returned from the database. Then this token/data is returned through the route to the through HTTP to the client-side controller. The view will update to a successful sign-in message or will display the trip data.

**Admin Interface**

The primary difference between the sequence of signing in or viewing trips and using the admin interface is that the client side is managed by Angular, rather than Express.js. Thus, the actor is the admin user who navigates to the admin SPA via the Angular router. The Angular controller requests data from the server side through the HTTP service. The Express.js route receives the request and forwards it to the admin controller, which uses Mongoose to query the database. MongoDB processes the operation, returning the results through Mongoose to the controller. The controller formats the results and sends them back to Angular through the HTTP service. The client-side controller receives the results and updates the view to display the data to the admin.

## Class Diagram

A diagram of a travel geoways class diagram

Description automatically generated

The Travlr Getaways web app is composed of interrelated classes, each of which has a distinct purpose in the travel booking process. The ‘MemberAccount’ and ‘MembershipAdmin’ classes handle membership details, including frequent flyer information and membership status. These classes ensure that the users’ membership credentials and points are validated and managed correctly.

The ‘TravelerInfo’ class holds information about the travelers, while the ‘Itinerary’ class holds the details of the travel itinerary, such as price and stopover information. These two classes work closely with the ‘TravelAgent’ class, which contains methods for booking travel components, like flights and hotels. The ‘TravelAgent’ class interacts with ‘FlightInfo’, ‘HotelInfo’, and ‘CruiseInfo’ to manage specific details for those components.

The actual booking process is done through the ‘HotelBooking’, ‘FlightBooking’, and ‘CruiseBooking’ classes. These classes retrieve the necessary information to finalize reservations based on the traveler’s requirements. They interact with the ‘TravelAgent’ class to facilitate the booking process. The ‘TripInfo’ class provides key details, such as start and return dates, origin, and destination. It ties together the various components of the trip.

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

| **Method** | **Purpose** | **URL** | **Notes** |
| --- | --- | --- | --- |
| **GET** | Retrieve list of trips | /api/trips | Returns all active trips |
| **GET** | Retrieve single trip | /api/trips/:tripCode | Returns single trip, identified by the trip code passed on the request URL |
| **POST** | Add a trip | /api/trips | Adds a new trip to the database |
| **PUT** | Update a trip | /api/trips/:tripCode | Updates a single trip, identified by the trip code passed on the request URL |
| **DELETE** | Deletes a trip | /api/trips/:tripCode | Deletes a single trip from the database, identified by the trip code passed on the request URL |
| **GET** | Retrieve list of rooms | /api/rooms | Returns all rooms |
| **GET** | Retrieve a single room | /api/trips/:roomCode | Returns a single room, identified by the code passed on the request URL |
| **GET** | Retrieve list of meals | /api/meals | Returns all meals |
| **GET** | Retrieve news | /api/news | Returns all news articles |
| **POST** | User login | /api/login | Authenticates user against the database |

## The User Interface

The following screenshots demonstrate how the admin-facing SPA functions. The implementation of security allows the “Add Trip” and “Edit Trip” buttons to be hidden until the user successfully logs in.

A screenshot of a computer screen

Description automatically generated

After logging in, the user can utilize the add and edit features to update the available trips in the database. This screenshot shows the “Add Trip” screen with information for a unique trip to demonstrate:

A screenshot of a computer

Description automatically generated

This new trip is now visible in the list of trips:

A screenshot of a website

Description automatically generated

Similarly, the “Edit Trip” screen allows the user to update an existing trip listing:

A screenshot of a computer

Description automatically generated

This change is now visible in the interface:

A screenshot of a computer

Description automatically generated

### Angular vs Express Structure

The Angular project structure and Express project structure differ significantly due to their distinct roles in the application. The Express side contains the API that handles both server-side routing and rendering static HTML pages for the customer-facing site, following a more traditional web application architecture. The server processes requests and sends complete HTML pages back to the client for the customer-facing site. The key components of the Express structure are the ‘app\_server’, which includes the controllers, routes, and views for rendering HTML pages, and the ‘app\_api’, which defines the RESTful API with models, routes, and controllers that interact with the database.

In contrast, the Angular structure manages the SPA on the client-side and interacts with the server-side only to retrieve from or update the database. The SPA’s modular structure supports dynamic updates and complex client-side logic. It consists of:

* Components – these are the individual user interface elements that can work independently or together to create complex interactions without loading the page,
* Services – these handle the business logic, security, and interaction with the API, and
* Models – these define the structure of the objects (e.g. Trip, User) that are used in the application.

While the initial load time of the SPA may be longer than that of the static HTML page, the SPA offers better responsiveness for user interactions because it does not need to completely reload the page each time. In fact, the SPA loads once and then handles any necessary updates on the client-side by fetching only the necessary data from the server, which reduces the load on the server and creates a more fluid user experience.

### Testing

The Angular SPA must interact with the Express API to be fully functional. The first step in testing that the program works as expected was to ensure that the API was responding to requests such as GET and PUT for the MongoDB data. The Postman tool was used to manually send HTTP requests to the API endpoints. After implementing the user authentication service, Postman was again used to verify that tokens were issued for successful login and that the API endpoints requiring user authentication functioned correctly with a token and denied access without.

The second step was end-to-end testing with the Angular SPA. This involved accessing the API endpoints directly through the SPA by logging in and attempting to make changes to the data. This step verified that the Angular services were correctly integrated with the API. During this process, the browser’s developer tools were referenced frequently to monitor network requests and console logs to detect and address any issues.

A common issue during testing was receiving 404 errors. However, these were typically resolved by restarting the server to ensure that all changes were correctly loaded. Monitoring logs and responses helped in quickly identifying problems related to API calls to expedite debugging.