

Travlr Getaways Website

# **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)

[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 | 09/20/2024 | Isaac Jang | The Travlr Getaways application is a full-stack web app that serves a customer-facing website for browsing travel options and manages travel data through backend routes and controllers using the MEAN stack. |

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

Travlr Getaways is a web application with the MEAN stack, which includes MongoDB, Express.js, Angular, and Node.js. This architecture enables us to create a strong, scalable system that meets both customer and administrative requirements. For your customers, the front-facing website will provide a simple and intuitive way to browse and book vacation packages. On the administrative side, we'll use a single-page application (SPA) to streamline content administration, allowing your team to easily update travel listings, manage bookings, and monitor customer interactions. This strategy ensures that the application is flexible, easy to use, and capable of accommodating development as your organization grows.

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

The Travlr Getaways online application is being developed under many important design limitations. First, the MEAN stack (MongoDB, Express.js, Angular, and Node.js) allows for a more efficient development process using a single programming language, but also requires careful control of JavaScript's asynchronous nature to maintain seamless functionality. Furthermore, the application must be responsive, which means it must look and function properly across a variety of platforms, including computers, tablets, and smartphones, influencing design decisions and testing. Security is also an important consideration, since client data must be safeguarded throughout the booking process, necessitating more time to install secure authentication and encryption mechanisms. Finally, the application must be scalable to accommodate an increasing number of users, which means the development team must prioritize maximizing both database and server performance to ensure the system can grow with the business.

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

### Component Diagram



The Travlr Getaways online application has a simple and well-organized system architecture, consisting of three major components: the Client, Server, and Database. The Client component provides a Web Browser that allows users to interact with the application, as well as session management via the Client Session and trip-related data display via the Traveler Portfolio. The Graphic Library handles visual elements, resulting in a seamless and beautiful user interface. On the server side, several components work together to manage user authentication and data flow. The Authentication Server secures user logins, while the Server Session and Traveler Database handle user and trip data, respectively. The Mongoose ODM connects the Server and the MongoDB database, allowing for quick data management and retrieval.

MongoDB, the database component, maintains all essential traveler and booking data. The system architecture is intended to allow for smooth interaction between components, with each piece connected by explicit interfaces, ensuring that the application stays responsive, safe, and scalable as users research trips, book services, and manage accounts. This architecture ensures that all levels function together seamlessly, providing a smooth experience for both customers and administrators.

### Sequence Diagram

A diagram of a diagram

Description automatically generated

The Travlr Getaways web application's logic begins with client-side user interactions, which are handled using AngularJS. The Browser/View/Template receives the initial input from the user and then communicates with the Controller when the user initiates an action, such as browsing to a certain page or making a request. After processing this input, the Controller calls a service to retrieve or update data from the server via the HTTP Client. The request is subsequently transmitted to the server-side, where it is forwarded via a Route that designates the proper Controller/Model function on the server-side to respond to the request. Using MongoDB and Mongoose ODM, the Controller/Model interfaces with the data layer to manage the essential business logic.

After receiving the request, the Controller/Model uses Mongoose to query the MongoDB database in order to obtain or update the required data. After the database has processed the request, Mongoose is used to send the Controller/Model a response. After formatting into a JSON object, this response is then sent back to the HTTP client on the client-side via the Route. The data is processed by the HTTP Client and delivered as a promise/results object. It is then assigned to the relevant scope within the Controller, which updates the Browser, View, and Template. When the user sees the finished data, the client and server activities flow together seamlessly. This method ensures that users may engage with the program in real-time, with data being retrieved, updated, and shown promptly.

## Class DiagramA diagram of a travel company Description automatically generated

The Travlr Getaways web application's JavaScript classes are set up to handle various facets of booking and organizing vacations. The fundamental information about a trip itinerary, such as the total cost, the number of miles, and any stops, is stored in the Itinerary class. This becomes the basis for organizing and designing travel packages. By managing particular trip details like the start and return dates, the origin, and the destination, the TripInfo class enhances the Itinerary. Together, these two classes provide a thorough framework for a traveler's itinerary.

Information on cruises, airlines, and hotels is managed by the CruiseInfo, FlightInfo, and HotelInfo classes. This information includes characteristics such as cabin types, seat classes, star ratings, and costs. The Travel\_Agent class provides methods such as BookPackage, BookFlight, BookHotel, and BookCruise to generate reservations for various travel services. These methods enable booking functions. The HotelBooking, FlightBooking, and CruiseBooking classes tackle the specifics and procedures needed to complete reservations, further refining these booking activities. The TravellerInfo class stores information about travelers, including the number of companions, to ensure that the application provides a personalized travel experience for each user. The MemberAccount and Membership\_Admin classes manage user membership details, including status, loyalty points, and frequent traveler programs. When combined, these classes allow the Travlr Getaways app to handle reservations and trip information with ease.

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

By making RESTful API endpoints accessible, the Travlr Getaways program can effectively communicate with various system components, guaranteeing a seamless data transfer between the client and the server. Key functionality for creating, retrieving, updating, and deleting data is provided by these endpoints, guaranteeing that administrators and customers may access the required data.

| **Method** | **Purpose** | **URL** | **Notes** |
| --- | --- | --- | --- |
| **GET** | Retrieve list of trips | /api/trips | Return all available trips for customers |
| **GET** | Retrieve Single Trip | /api/trips/:tripId | Returns JSON of a specific trip, identified by the trip Id in the URL |
| **POST** | Create a new trip booking | /api/bookings | Allows a cuomer to book a trip, creating a new booking in the system |
| **GET** | Retrieve booking details | /api/booking/:bookingId | Returns the details of a specific booking, identified by booking ID. |
| **PUT** | Update booking | /api/booking/:bookingId | Allows customers or admins to update the details of an existing booking. |
| **DELETE** | Delete booking | /api/booking/:bookingId | Cancels a specific booking by removing it from the system. |
| **GET** | Retrieve User profile | /api/users/:userId | Returns user information including bookings and membership status. |
| **PUT** | Update User profile | /api/users/:userId | Allows users to update their profile information. |
| **POST** | Register a new User | /api/users/:userId | Creates a new user account in the system. |
| **POST** | Authenticate user login | /api/users/:userId | Validates user credentials and initiates a session. |

## The User Interface

The Travlr Getaways application has a user interface that is easy to use and intuitive. Customers can view comprehensive details about individual journeys in the single-page application (SPA), and administrators can quickly manage these trips by making screen edits and updates. Through the admin dashboard, for instance, users may choose a particular trip, view all pertinent facts, and easily change or amend the data. The screenshots that follow highlight these features. They include a journey that was added during development, as well as the edit and update panels that show how real-time system modifications are done.

The front-end experience is handled by the Angular project structure, which is arranged around services and components that collaborate to produce a dynamic user interface. On the other hand, the back-end Express project structure concentrates on routes, controllers, and middleware to manage database connectivity and server-side logic. When compared to a regular online application, the Angular SPA offers far more capability. The SPA offers a faster, more seamless experience by dynamically updating the material on the screen without requiring a page refresh, as opposed to loading new pages for every interaction. Thorough testing is done to verify that data can be updated (by PUT requests) and retrieved (via GET requests) in the database, ensuring the SPA and API operate together flawlessly. This guarantees a dependable and effective experience for users in every interaction between the front-end and back-end.