



Travlr Getaways  
**CS 465 Project Software Design Document**  
Version 1.2

## Table of Contents

<b>CS 465 Project Software Design Document</b>	<b>1</b>
Table of Contents	2
Document Revision History	2
Instructions	<b>Error! Bookmark not defined.</b>
Executive Summary	3
Design Constraints	3
System Architecture View	4
Component Diagram	4
Sequence Diagram	5
Class Diagram	6
API Endpoints	7
The User Interface	7

## Document Revision History

Version	Date	Author	Comments
1.0	11/16/24	Jonathan Warner	Milestone 1 Submission
1.2	12/1/2024	Jonathan Warner	Milestone 2 Submission
2.0	12/15/2024	Jonathan Warner	Project 2 Submission

## Executive Summary

The Travlr Getaways web application is built utilizing the MEAN stack, which is an open-source full-stack solution for developing quality web applications through the integration of MongoDB, Express.js, Angular, and Node.js. MongoDB serves as the database component for this application, where it holds user information, travel data, and administrative records. The NoSQL structure of this database enables flexibility by allowing for easy scaling and the efficient management of data. In this implementation, MongoDB's document-oriented approach is perfect for handling the dynamic user travel data to ensure quick data retrieval.

Acting as the backend framework, Express.js handles server-side operations and creates RESTful API endpoints to allow for the gap between the client and the database to be easily bridged. The Express framework greatly improves the development process thanks to its middleware functionality and how it simplifies routing. This provides us with the capability to easily modify request and response objects before forwarding them to other middleware functions and easily scale up functionality through the use of template engines, like in this case, Handlebars.

The customer-facing side of the application, in addition to its administrative functionality, are both handled by the web development framework Angular. Thanks to the utilization of Angular, the Travlr Getaways web application is a single-page experience with seamless page loading and interaction. This side of the application provides support for browsing destinations, managing travel plans, and an overall, user-friendly experience. The Travlr admin tools are designed through Angular as a single-page application (SPA). These will allow our administrators to effectively manage content and bookings once they have logged in and have been securely authenticated. Once at the helm of the interface, our admins will have the power to add, edit, and delete entries related to travel offerings on our various pages.

Serving as the server's runtime environment, Node.js enables the use of JavaScript on both client and server sides of the Travlr application. This ensures consistency in our development process and also provides us with the ability to access real-time, event-driven server responses thanks to the environment's event-driven architecture. Node.js in combination with Express.js creates a strong foundation for handling concurrent client requests in an efficient manner, minimizing the total lines of code needed to achieve the desired output. Overall, our implementation of the MEAN stack development process ensures that the Travlr Getaways application provides customers with a high-performance and seamless environment that facilitates smooth interaction between client, server, and database.

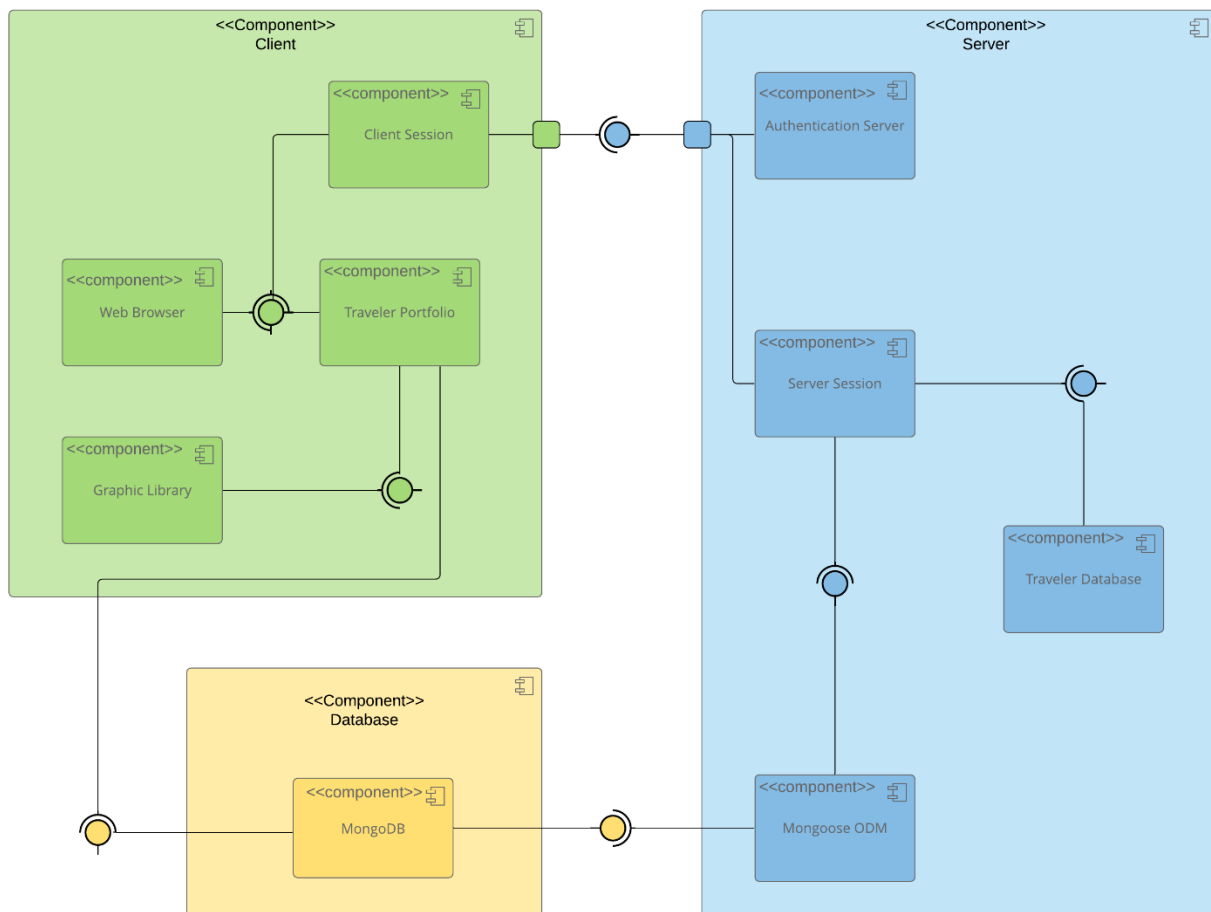
## Design Constraints

Developing the web-based Travlr Getaways application involves identifying several design constraints that should be considered going forward, including security, performance optimization, and scalability. Due to the fact that our application will deal with sensitive data, including user financial information and travel plans, our security implementation must be robust and shore up any weaknesses or vulnerabilities inherent to the MEAN stack. This necessitates the implementation of reliable authentication and properly authorized roles. Our application must be able to handle high user traffic in an efficient manner without resulting in regular performance hits. To combat this, our application should implement optimized database queries and proper server-side resource management through Node.js and Express.js. Finally, scalability must be considered to support the growth of our user base. As such, the

implementation of some sort of horizontal scaling should be considered that allows for increasing numbers of traffic. Addressing these constraints must be a consistent consideration throughout the development of the application to ensure they are all properly accounted for.

## System Architecture View

### Component Diagram



The Travlr Getaways web application consists of three main components: Client, Server, and Database, each with their own internal critical components.

### Client

On the client side, the Web Browser serves as the user interface, which allows users to interact with the site. The Client Session manages the session state to enable consistency in the user experience. The Traveler Portfolio serves as a personalized space for users where they are able to manage their travel plans, which is further supported by the Graphic Library for the implementation of enhanced visuals. The Client component interacts with the Server component through the Client Session and Authentication Server.

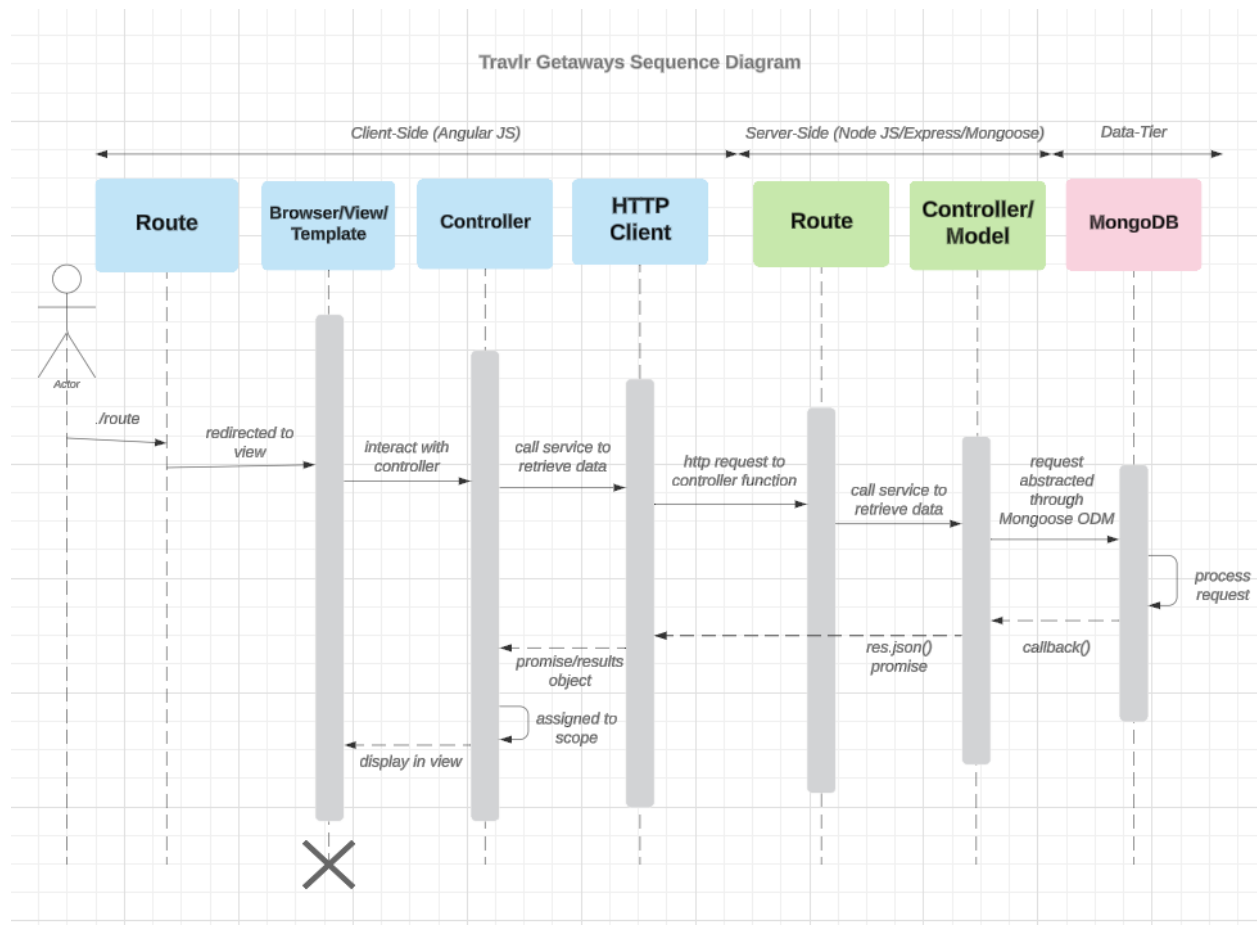
### Server

Users are able to connect to the Server component by logging in through the Authentication Server, which handles authentication and access control. A Server Session component is utilized to maintain session data and facilitate communication with each of the internal components. The Traveler Database is where essential user information is stored, and the Mongoose ODM aids the interaction between the server and MongoDB, allowing for more efficient queries.

## Database

The Database component of our application utilizes MongoDB as the primary data storage, efficiently accommodating user data, travel information, and administrative records thanks to its document-oriented NoSQL structure. The Database component interacts with both the Server and Client components by allowing them to query for data. The Client component can request the data for Traveler Portfolios, while the Server is able to query the database through the use of Mongoose.

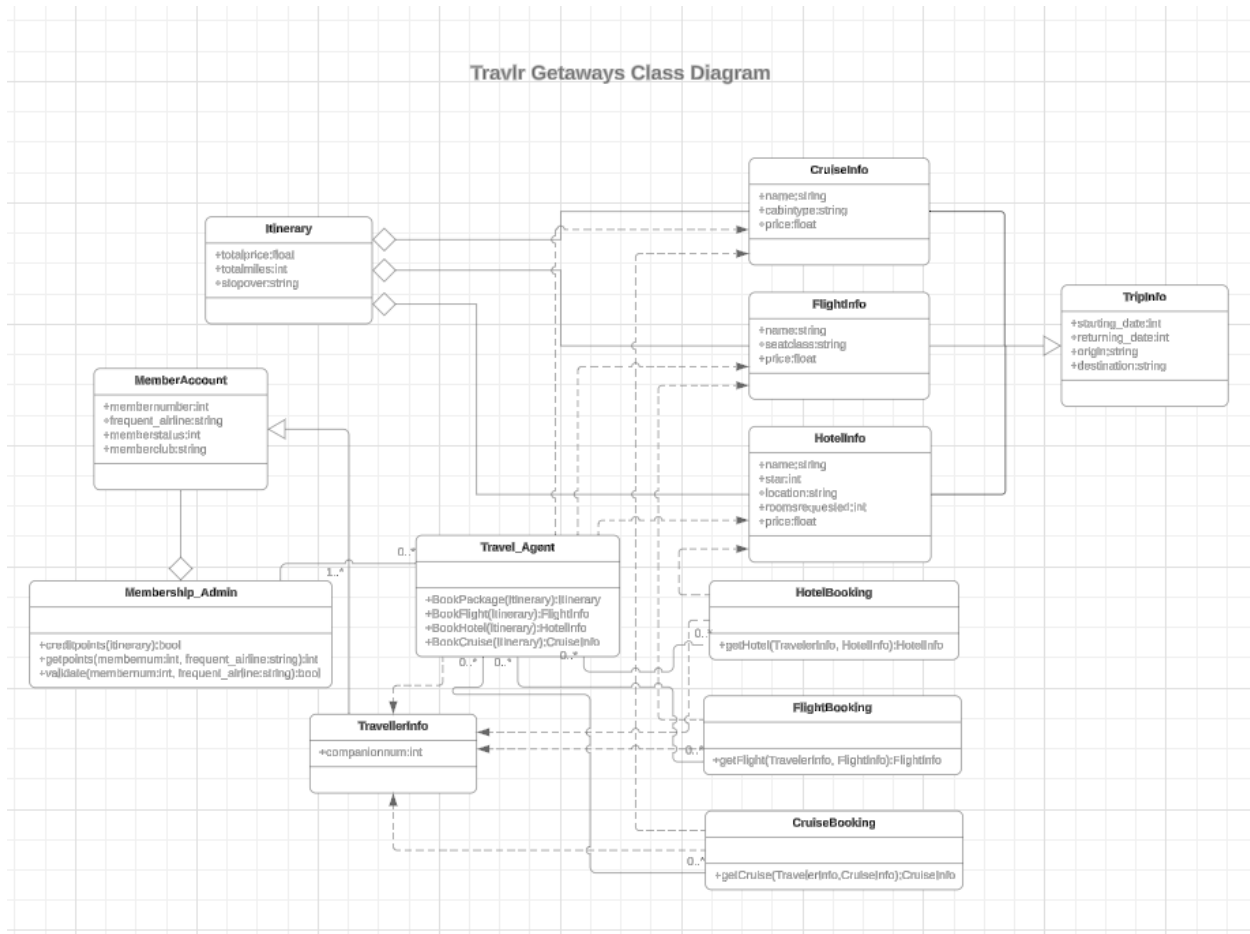
## Sequence Diagram



The above sequence diagram illustrates the workflow of the Travlr Getaways web application and the interactions that occur between its three core components: the client-side, server-side, and data-tier. The sequence begins with a user, or actor, initiating a request via a specific route, redirecting them to the associated AngularJS-built view template on the client's side. The view template interacts with the associated controller that can then call an HTTP client service to retrieve data. This request is sent from the HTTP client to the server-side, where it is routed by Node.js working with Express.js. From here, a service request for data is sent to the controller/model function, which is able to interact with the object

data modeling (ODM) library Mongoose. Mongoose abstracts the request to the database and processes it. The result is then passed back through the various layers as either a JSON response or promise object. This updates the client-side to display the data within the user interface, completing the sequence for a typical Travlr Getaways request.

## Class Diagram



The above class diagram is a visual representation of the system architecture and its various internal relationships for the Travlr Getaways web application. At the core of the system is the **Travel\_Agent** class, which allows users to access methods that can book hotels, flights, and cruises. The **Travel\_Agent** handles various bookings by creating an **Itinerary** object for users that aggregates data from the three Info classes (**Cruise**, **Flight**, and **Hotel**) and stores a total price, total miles, and stopovers. Each of the Info classes has details specific to their various booking types and inherits general information, like a starting date, returning date, and destination, from the abstract **TripInfo** class. The Info classes are accessed via the Booking classes, which in turn provide access to the **Travel\_Agent**. User account information is stored within the **MemberAccount** class, which can be manipulated via the aggregating **Membership\_Admin** class. Finally, the **TravellerInfo** class contains information about how many companions the user is bringing for booking calculations.


## API Endpoints

Method	Purpose	URL	Notes
GET	Retrieve all trips	<api/trips>	Returns all trips currently stored within the database.
GET	Retrieve a specific trip	</api/trips/:tripcode>	Returns a single trip based on which tripcode is used as an argument.
POST	Register a new user into the database	</api/register>	Takes a username, email, and password and then registers the user into the database. The password is hashed prior to storage.
POST	Login a user	</api/login>	Takes a username and password and then compares them to users in the database. If a matching user with a matching password exists, log the user in.
POST	Add a trip	</api/trips>	Creates a new trip and adds it to the database.
PUT	Edit a trip	<api/trips/:tripcode>	Updates an already existing trip.

The User Interface

Custom Trip

Tropical Hideaway Retreat



Jungle Breeze Resort, 4.9 stars

3 nights / 4 days only \$799.00 per person

Private bungalow with a canopy bed and panoramic jungle views.

Edit Trip

Edit Trip

Add Trip

Add Trip

Code:

Code

Name:

Name

Length:

Length

Start:

mm/dd/yyyy

Resort:

Resort

Per Person:

Perperson

Image Name:

Image

Description:

Description

Save

Edit Trip

Code:

GALR210214

Name:

Name

Length:

Length

Start:

mm/dd/yyyy

Resort:

Resort

Per Person:

Perperson

Image Name:

Image

Description:

Description

Save



Developed in Angular, the Travlr Getaways web application's front end employs a modular and component-based architecture, in contrast to more traditional Express-based HTML pages. Encapsulating each piece of functionality within its own component helps keep the codebase more easily readable and maintainable. This modular design is flexible and facilitates future growth of the application. This approach allows for the implementation of a single-page application (SPA), allowing users of the Travlr application to enjoy dynamic interactions and a responsive user experience without needing to reload entire pages. SPAs do come with challenges, such as being more complicated to debug and SEO limitations with regard to client-side rendering, but the benefits greatly outweigh the drawbacks. Testing Travlr's SPA implementation with the Express-based API for GET and PUT requests involved extensive use of console logging, endpoint testing with Postman, and confirming database changes with MongoDB Compass. These combined efforts work towards a seamless SPA and API integration that allows the Travlr Getaways application to provide a smooth interface.