

Elizabeth Marticello

Travlr Getaways

# **CS 465 Project Software Design Document**

Final Version

## 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 – Sections Added |
| --- | --- | --- | --- |
| 1.0 | 09/28/25 | Elizabeth Marticello | Milestone One: Executive Summary, Design Constraints, System Architecture View added. |
| 2.0 | 10/05/25 | Elizabeth Marticello | Sequence Diagram, Class Diagram, API Endpoints |
| 3.0 | 10/25/25 | Elizabeth Marticello | Fixed Sequence Diagram, The User Interface |

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

As the software developer responsible for creating the travel booking website for Travlr Getaways, I have been tasked with building the application using the MEAN stack which consists of MongoDB, Express, Angular, and Node.js. This technology stack was chosen because it covers the entirety of the full stack Development process while providing a modern, scalable, and maintainable solution. The architecture is planned to meet the client’s requirement for a responsive and user-friendly web application that supports both customer interactions and administrative management of travel content.

The customer-facing side of the application will focus on providing travelers with an intuitive and appealing interface. Customers will be able to create an account, log in, and interact with the website to search and filter trips, book reservations, and view upcoming itineraries. In the current stage of development, the customer site is being built with Express and Handlebars templates to load server-rendered pages that display destination names, images, and descriptions. As the application matures, the customer-facing website will be supported by RESTful endpoints connected to MongoDB so that data is rendered dynamically. This allows for more robust search features and booking functionality.

Once the customer front-end development and back-end architecture is completed the administrative functions can be added to manage the database. The administrative experience is implemented as a single-page application (SPA) built with Angular. This interface can be used by Travlr Getaways staff to maintain the system by adding, editing, and removing customer accounts, updating trip packages, and adjusting pricing. The SPA communicates with the Node.js and Express API by sending and receiving JSON data which is stored in MongoDB.

By separating the application into two different user experiences, the architecture provides the experience required for both customers and administrators. The MEAN stack provides the skeleton for both user experiences and provides the option for future scalability.

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

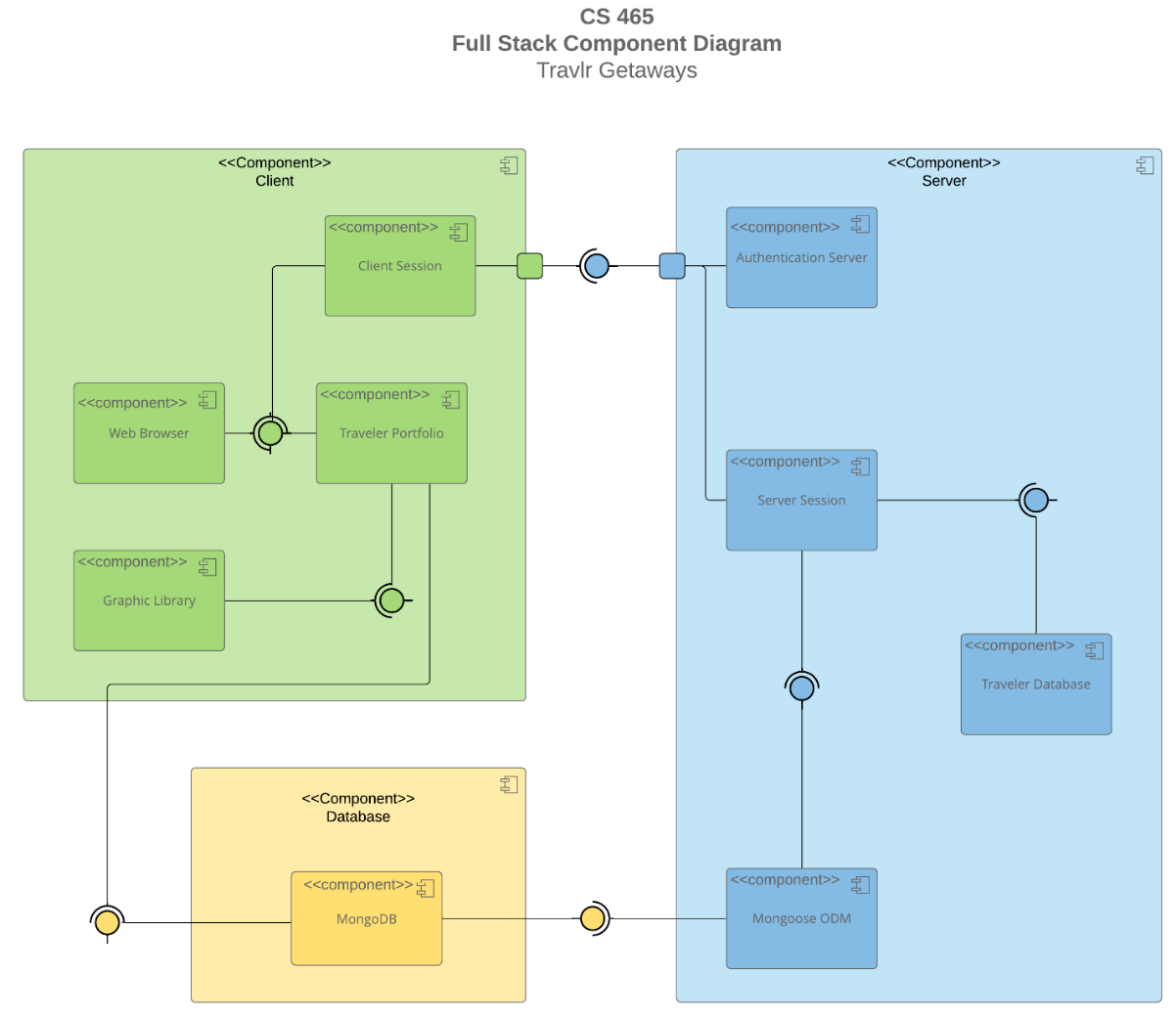
A design constraint for this application is the requirement to use the MEAN stack. This requirement may provide consistency but locks the project into being built around JavaScript. So far in the development process I have used Node.js and Express with Handlebars templates for customer-facing pages to lay the foundation for later Angular and MongoDB integration.

Another constraint would be the requirement for two user experiences, split between customers and administrators. Customers need to browse trips and book trips, while admins need tools to manage users, trips, and pricing. Express routes and controllers have already been created that will soon expand into REST endpoints.

The entire MEAN stack is considered from the very beginning of this development process. I’m anticipating the integration of MongoDB and creating authentication layers.

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

### Component Diagram



The Travlr Getaway system uses a three-part structure made up of the client, the server, and the database. On the client side, the significant components include the web browser which displays the traveler portfolio interface, and the client session, which tracks user activity. This layer is responsible for showing trips and content to users, and in the current stage it is supported by Express and Handlebars to render pages.

The server layer is made up of components that handle authentication, sessions, and application logic. The authentication server verifies user credentials, while the server session manages active logins. The Express server then uses the Mongoose Object Data Model (ODM) as a bridge to the database. These components work together to ensure that only valid requests are processes and that all data passes between the client and the database is properly structed.

The database layer is run with MongoDB, which serves as the traveler database. It stores customer accounts, trip packages, and itineraries. The Mongoose ODM communicates with MongoDB while the server retrieves and updates this information before sending results back to the client. These relationships allow the process to start with the client sending requests, the server handles logic and validation, and finally the database securely stores and returns the necessary data.

### 

### Sequence Diagram

A diagram of a software project

AI-generated content may be incorrect.A diagram of a diagram

AI-generated content may be incorrect.

The Travlr Getaways web app follows a flow of logic. The above Sequence Diagram illustrates this flow. As seen above, when a user visits a route like “Trips/Travel” or “Log In”, Angular loads the proper view and controller to handle input and send an HTTP request. That request goes to the Node.js/Express server, where the controller processes it and asks the model for data. The model uses Mongoose to communicate with MongoDB, retrieves the needed information and sends it back. The response then returns through the same layers, and Angular updates the view with the results. This structure keeps each layer focused on its role to provide a secure and smooth experience while navigating through the web app.

## 

## A diagram of a travel application AI-generated content may be incorrect.Class Diagram

The Travlr Getaways app uses JavaScript classes to organize travel and booking information. TravellerInfo and MemberAccount store traveler and membership details, while Membership\_Admin handles points and account validation. The Itinery class connects to Hotel, Flight, and CruiseInfo to build complete travel plans. Each booking type, hotel, flight, and cruise, manages its own reservations. TripInfo tracks trip dates and destinations. Travel\_Agent helps create and manage itineraries. Together, these classes work to manage users, travel options, and bookings in a clear and manageable way.

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

| **Method** | **Purpose** | **URL** | **Notes** |
| --- | --- | --- | --- |
| ***GET*** | *<Retrieve list of things>* | *</api/things>* | *<Returns all active things>* |
| ***GET*** | *<Retrieve single thing>* | *</api/things/:thingId>* | *<Returns single thing instance, identified by the thing ID passed on the request URL>* |
| **GET** | Retrieve a list of all trips | /api/trips | Returns all available trip records from the database |
| **GET** | Retrieve a single trip by its unique code | /api/trips/:tripcode | Returns the details for one trip, using the tripcode parameter in the URL |

## The User Interface

***A screenshot of a computer

AI-generated content may be incorrect.Unique Trip By Liz***

***A screenshot of a computer program

AI-generated content may be incorrect.A screenshot of a trip form

AI-generated content may be incorrect.Edit and Update Screen***

**Postman After Edit**

A computer screen shot of text

AI-generated content may be incorrect.

***Comparing Angular Project Structure and Express HTML Customer-Facing Page***

Two major differences between Angular and Express project structure are where the code runs and how routes are handled. Express executes its code and manages routes on the server, while Angular does so in the browser. By handling routing and much of the logic client-side, Angular can reduce the load on the server and provide a smoother, more responsive experience for users. What are some advantages and disadvantages of the SPA functionality? What additional functionality is provided by a SPA compared to a simple web application interaction?

***Advantages, Disadvantages, and Additional Functionality of SPA***

SPA offers faster, smoother user experiences by loading the main framework once and dynamically updating content through API calls instead of reloading entire pages. This reduces server load and creates a more interactive, app-like feel. Some disadvantages of  SPAs can be having longer initial load times and being more complex to debug. When comparing a SPA to a simple web application, the SPA provides added functionality such as client-side routing, real-time data updates through APIs, and instant, seamless navigation between views without refreshing the page.

***Process of Testing SPA to Ensure Database Data is Added from GET and PUT Calls***

After implementing the GET and PUT on the backend and registering the routes I used Postman to test the endpoint. This was done by setting the query type to “post” and the api endpoint to [http://localhost:3000/api/trips](http://ocalhost:3000/api/trips). When the upper ‘Body’ tab is selected, ‘x-www-form-urlencoded’ can be set as the data type. Here I added example key-value pairs for each field of the record being tested. Once I sent that query against the backend api endpoint, I was able to see a successful call that returned a JSON object with the example record I created. A result also appeared in the Powershell Window that was running my Express server, and the record appeared on the front-end web app. Once I implemented an “Edit Trip” button, I was able to use the web app to edit any value in the example and see it reflect across the project.

***Afterthoughts***

Something I appreciate about this course is how each week builds toward a final project. However, with a project of this size being spread out across multiple weeks, I can sometimes lose track of the order in which I tackled each step. For future projects, I think I would benefit from keeping the overall objective in mind throughout each week. There were some weeks when I approached my assignments as standalone tasks rather than as interconnected parts of a larger whole. Maintaining a clear focus on the final goal will help me better understand how each component contributes to the overall structure of the MEAN project.

Top of Form