

Travlr Getaways

# **CS 465 Project Software Design Document**

Version 1.2

## 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 | 7/21/2025 | Carter Williams | Module 3 revision – Software Design Document: Executive Summary, Design Constraints, and System Architecture View: Component Diagram |
| 1.1 | 8/3/2025 | Carter Williams | Milestone 2 – System Architecture Views and API Endpoints completed. |
| 1.2 | 8/18/2025 | Carter Williams | Project Submission – Added views of the Angular page development and final explanation on Angular and Express page structures. |

## 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 MEAN stack is used by the Travlr Getaways web app to make a whole travel booking site. That means with this method, we can easily make a website for customers and an admin interface that operate well together. The customer website, which was made with Express.js and Handlebars templates, is fantastic for optimization because it loads quickly and helps passengers look through packages and make accounts while also allowing them to book trips. The angular-based admin panel, on the other hand, gives Travlr Getaways personnel a more modern and responsive interface that lets them manage customers and edit vacation packages in real time.

Using JavaScript across the whole stack makes development easier, while the MongoDB database stores things like trip data, customer information, or possible bookings in a way that is easy to change. This type of design makes sure that both parts of the application can expand on their own, just as the business does.

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

The thing is, there are quite a few critical limitations that affect how we would create the Travlr Getaways app. Because we want to use the MEAN stack – which keeps everything in JavaScript – but limits the technologies we can use. We also need to find a balance between two very different user experiences: one being a public website that needs to load quickly and rank well in search engines, the other is an admin panel that needs to be updated in real time and has a lot of different managing features. Security is very important here because we deal with client information and payment information, meaning we need strong authentication and data protection conceptions. The system needs to be able to account for several users booking trips at the same time without any problems; the database and transaction management need to be very well thought out. These types of limits let us make judgments about the architecture and make sure we build a strong platform that is scalable and works well for both clients and the Travlr Getaways personnel.

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

### 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).

From the component diagram, you can see that it shows how the Travlr Getaways app is divided into three separate layers that work together like an organized group. The client side (green section) is where users actually use the app. Their web browser connects to both the *Client Session* for logging in and keeping logged in, and the *Traveler Portfolio*, where they can look at and manage their trip bookings. There's also an added a *Graphic Library* part to take care of all the visual parts that make looking through vacation packages fun and easy to understand. The blue part of the server side is like the brain of the operation. It has an *Authentication Server* that keeps track of who is logged in, a *Server Session* component that handles all the requests from users, and our *Mongoose ODM* that connects the JavaScript code to the database. Everything here connects to our *MongoDB* database (yellow), where we keep all the customer information, booking details, and vacation packages. The structure is fairly ideal since each layer has its own function to accomplish, but they all work together through clear connection points. It really makes the whole system dependable and easy to keep up with as Travlr Getaways grows.

### Sequence Diagram

A diagram of a diagram

AI-generated content may be incorrect.

The sequence diagram above shows how the Travlr Getaways app works, starting with when an actor goes to a route that takes them to the right view template. The browser/view then talks to the controller, which uses HTTP requests to the server-side controller/model layer to get data. The server handles these requests by performing abstracted calls to MongoDB through Mongoose ODM. It then returns JSON promises that are shown in the user's view.

## Class Diagram

A diagram of a travel company

AI-generated content may be incorrect.

The class diagram here shows the JavaScript classes that the Travlr Getaways app uses, where the essential travel components (*TravellerInfo, CruiseInfo, FlightInfo*, and *HotelInfo*) provide the basic information for trips. The main *Itinerary* class manages booking packages, and the specialist booking classes (*HotelBooking, FlightBooking, CruiseBooking*) take care of certain types of reservations. The *MemberAccount* and *Membership\_Admin* classes take care of user management, and the *Travel\_Agent* class acts as a middleman for the overall booking process.

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

| **Method** | **Purpose** | **URL** | **Notes** |
| --- | --- | --- | --- |
| **GET** | Retrieve all trips | /api/trips | Returns all available travel packages |
| **GET** | Retrieve one trip | /api/trips/:tripId | Returns specific trip details by ID |
| **PUT** | Update existing trip | /api/trips/:tripId | Admins – modifies trip details |
| **DELETE** | Remove trip | /api/trips/:tripId | Admin – removes travel package |
| **POST** | User authentication | /api/auth/login | Validates user credentials |
| **POST** | User registration | /api/auth/register | Create new user |
| **GET** | User profile | /api/users/:userId | Returns user info. |
| **POST** | Create booking | /api/bookings | Creates a new trip reservation |
| **PUT** | Retrieve user bookings | /api/bookings/user/:userId | Returns all bookings of one user |

## The User Interface

Added Trip:

A screenshot of a computer

AI-generated content may be incorrect.

Main Screen (after login):

A screenshot of a computer

AI-generated content may be incorrect.

“Add Trip” Screen:

A screenshot of a computer

AI-generated content may be incorrect.

“Edit Trip” Screen:

A screenshot of a computer

AI-generated content may be incorrect.

The **Express** backend follows a straightforward MVC structure with separate folders for models, controllers, and routes – it's pretty linear where requests flow through routes to controllers to the database. Now, **Angular** takes a different approach with its component-based architecture, as it organizes everything around reusable UI components; they work together with services to handle data and business logic. The Angular structure definitely feels more modular since components can be mixed and matched, while Express is more about processing requests in sequence.

What makes the Angular SPA really shine compared to a basic web app is the user experience - everything happens without page refreshes and buttons appear and disappear based on login status. Plus, forms provide instant feedback, which is really nice. Testing the integration between the SPA and API includes checking that the GET requests properly display trip data on the main page and verifying that only logged-in admin users can see edit buttons – you should also make sure the PUT requests actually update trip information in the database when you modify something through the edit form. The JWT interceptor we built automatically handles authentication for protected endpoints, so testing becomes a matter of trying to edit trips both when logged in and logged out to make sure the security works like it should.