

Travlr Getaways Fill Stack Application

# **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)

[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 | 05/24/2024 | Dalton Short | Filled out Executive summary, Design constraints, and system architecture view sections. |
| 1.1 | 6/3/2024 | Dalton Short | Completed sequence diagram – API endpoints table. |

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

Travlr Getaways is seeking to develop a comprehensive web application to enhance their service offerings and streamline administrative operations. This project involves creating an integrated system that includes a customer-facing website and an administrator single-page application (SPA), both supported by a robust database. The following executive summary outlines the proposed solution, architecture, and critical information necessary for the project's success.

**Architecture overview:**

The current plan for the web application is a typical MEAN stack, this includes four main software, MongoDB, Express.js, Angular, and Node.js. This provides many benefits, namely using Javascript for with the entire application which simplifies and expedites the entire development processes.

MongoDB:

NoSQL databases offer flexibility for handling diverse data types and

Has high scalability.

Express.js:

Web framework for Node.js that simplifies server-side code and makes it

easier to build robust APIs.

Angular:

Front end web application framework by google which excels in its modularity.

It is easy to maintain applications with it and is helpful for Single Page

Applications like the one planned for administrator page.

Node.js:

Runtime environment that allows for the execution of JavaScript on the server-

side. It facilitates development and increases scalability with high performance applications.

**Customer facing side:**

Using Angular, the website will be built to provide a dynamic user experience. Key features follow

User registration and authentication:

Secure user account and login functions.

Travel package and browsing:

Interface for browsing travel packages.

Booking Management:

Simple and easy to use forms for reservations and itineraries.

Compatibility:

Ensuring access from various devices such as desktops, tablets, and phones.

**Administrator Single Page Application (SPA):**

Angular also provides a streamlined process for the SPA adding ease of use for functionality like content management, user management, data analytics, and security access control.

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

1. Performance constraints:
   1. Low latency and scalability to at least industry standard.
   2. Will require more resources for optimization.
   3. Use of monitoring tools will assist in mitigating this constraint and optimizing performance
2. Security constraints:
   1. Protecting user data and implementing access control.
   2. Will require more testing and complexity.
   3. For the user side, there are security practices for writing code as well as many software Applications that will help. For backend, regular audits will help maintain a secure access point control.
3. Technology constraints:
   1. It is designed using the MEAN stack, so our technology is limited to compatible software within the MEAN stack.
   2. Could potentially limit technology choices.
   3. Using MEAN to its fullest extent will mitigate this. The MEAN stack is very versatile so this should not be a huge concern. It was chosen for that reason.
4. Time constraints:
   1. Each phase so far has been planned to take one week.
   2. Could compromise quality or limit the capabilities.
   3. Clear agile methodology, plans, and schedules will help to make efficient use of time.
5. Budget constraints:
   1. There will always be a budget.
   2. It could effect time working on the project, technology used, and the application’s scope of functionality
   3. Similar top time constraints, planning it all ahead of time will be helpful, but surprises often come up so being able to pivot and dynamically allocate funds is important.
6. Usability constraints:
   1. This encompasses any required functionality given by the client.
   2. Provides high priority targets that must be achieved.
   3. Clear communication back and forth with the client will make these targets clearer and completed with more confidence that the client will be satisfied.
7. Legal/compliance constraints:
   1. Any relevant laws will need to be adhered to for relevant locations.
   2. This will make the development process much more complex and will be different in each area. For example, America and all of Europe have very different laws when it comes to data privacy.
   3. Again, extensive planning is critical to mitigate the effects. It may be beneficial to roll out certain regions one at a time.

## [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).

1. **Client layer (green):**
   1. Web browser:
      1. Interface in which users interact with the application, rendering UI and communicating with the server.
         1. **Web browser interactions:**
            1. Communicates with authentication server. Once that is successful, it establishes a user client session.
   2. Client session:
      1. Manages user sessions client side. It maintains state information such as authentication, tokens (if applicable), and user preferences.
         1. **Client session interactions:**
            1. Interacts with travel portfolio to manage user-specific data.
   3. Traveler portfolio:
      1. Represents user profile and personal data. This includes travel plans and booking history.
         1. **Traveler portfolio interactions:**
            1. Uses graphic library to render user interface components dynamically.
   4. Graphic library:
      1. This component will handle graphical and UI elements to ensure a smooth user experience.
2. **Server Layer (blue):**
   1. Authentication server:
      1. Handles user authentication and authorization server side.
      2. Some examples would be proper user permissions and a secure login session.
         1. **Authentication server interactions:**
            1. Interacts with server session to maintain session information as well as user states across interactions.
   2. Server session:
      1. Manages server-side sessions. Similar to client session but for the server itself.
         1. **Server session interactions:**
            1. Communicates with database interactions to fetch, update, and store user and travel-related data.
            2. It is mediated by Mongoose ODM.
   3. Database interactions:
      1. Interacts with the MongoDB database’s data repository.
         1. **Interactions:**
            1. Direct interfacing with database with CRUD functionality.
      2. The fourth component is tied to MongoDB too, so it is grouped here
         1. Mongoose ODM
            1. Object data modelling library for MongoDB. Simplifies many different interactions with the data.
3. **Database Layer (yellow):**
   1. Database:
      1. NoSQL database to hold all application data. This includes user data, travel packages, bookings, and administrative data. MongoDB was specifically chosen for its robust ability to handle both structured and unstructured data.
         1. **Database interactions:**
            1. Communicates with the server layer components to manipulate and communicate data.

### Sequence Diagram

A diagram of a trade end

Description automatically generated

1. Client-Side Interaction (Browser/View/Template)
   1. User interacts with the application via the browser, e.g., clicking "Sign In" or viewing trips.
2. Routing (route)
   1. User's action triggers a route, redirecting to the appropriate view.
3. Controller Interaction (Controller)
   1. The view interacts with the controller to handle application logic.
4. HTTP Request Handling (HTTP Client)
   1. The controller sends an HTTP request to the server for data retrieval or updates.
5. Server-Side Processing (Controller/Model)
   1. The server-side controller processes the request and interacts with the service layer.
6. Database Interaction (MongoDB)
   1. Mongoose ODM abstracts database interactions with MongoDB to fetch or update data.
7. Data Processing and Response (MongoDB)
   1. MongoDB processes the request and returns the data to the server-side controller.
8. Server-Side Response (Controller/Model)
   1. The controller sends a response back to the client-side.
9. Client-Side Update (Browser/View/Template)
   1. The client-side updates the view based on the server's response.

## Class Diagram

A diagram of a travel getaway class diagram

Description automatically generated

1. Itinerary
   1. Manages trip schedules and locations with attributes like starting\_date, returning\_date, origin, and destination.
2. TripInfo
   1. Contains summary details such as totalprice, totalmiles, and stopover.
3. Membership\_Admin
   1. Manages member details with attributes like membernumber, frequent\_airline, memberstatus, and memberclub.
4. FlightInfo
   1. Represents flight details with name, seatclass, and price.
5. HotelInfo
   1. Represents hotel details with name, location, roomsrequested, and price.
6. CruiseInfo
   1. Represents cruise details with name, cabin\_type, and price.
7. MemberAccount
   1. Handles member management with attributes and methods like creditpoints(itinerary), getpoints(membernum, frequent\_airline), and validate(membernum, frequent\_airline).
8. TravellerInfo
   1. Contains traveler information, including companionnum.
9. Travel\_Agent
   1. Provides booking methods: BookPackage(itinerary), BookFlight(itinerary: FlightInfo), BookHotel(itinerary: HotelInfo), and BookCruise(itinerary: CruiseInfo).
10. GetHotel
    1. Retrieves hotel details using getHotel(traveller: TravellerInfo, itinerary: HotelInfo).
11. GetFlight
    1. Retrieves flight details using getFlight(traveller: TravellerInfo, itinerary: FlightInfo).
12. GetCruise
    1. Retrieves cruise details using getCruise(traveller: TravellerInfo, itinerary: CruiseInfo).

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

| **Method** | **Purpose** | **URL** | **Notes** |
| --- | --- | --- | --- |
| **GET** | Retrieve homepage | app\_server\routes\index.js | Directs user to home page |
| **GET** | Retrieve users listing | app\_server\routes\users.js |  |
| **GET** | Retrieve travel page | app\_server\routes\travel.js | Routes to travel page |
| **GET** | Retrieve Api router | app\_api\routes\index.js | Routes to api trips page |
|  |  |  |  |

## The User Interface

Angular vs. Express HTML:

* Angular organizes code into modules, components, and services, while Express serves HTML pages with server-side rendering.
* Angular uses TypeScript and follows a modular structure, while Express is more flexible and typically follows a simpler directory structure.

Advantages and Disadvantages of SPA:

* Advantages: Enhanced user experience, faster navigation, reduced server load.
* Disadvantages: SEO challenges, longer initial load time.
  + Additional SPA Functionality:
    - Enhanced user interactivity, client-side routing, state management.

Testing SPA with API:

1. Unit testing for components and services.
2. Integration testing for component-service interactions.
3. End-to-end testing for API interactions.
4. specific to this project, postman and mongo compass were used.
5. Potential Errors:
   1. CORS issues, authentication errors, data consistency issues.

Questions for Future SPA Builds:

1. Lazy loading implementation.
2. State management best practices.
3. Authentication and authorization strategies.
4. syntax about most of the sofware used in vs code.