

**Augustus Mendy**

# **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 | 05/25/2025 | Augustus Mendy | Enhancing the Executive Summary, System Architecture View, and Design Constraints |
| 1.2 | 6/08/2025 | Augustus Mendy | Filled out the System and Class Diagrams, and the API Endpoints sections |
| 2.0 | 6/25/2025 | Augustus Mendy | The User Interface updated, Document completed, submitted |

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

The purpose of the application is to allow users to make reservations for vacation packages through the website. In order to do this, the website must allow visitors to register, search for vacation packages with prices and places presented, and then book the package of their choice. The website must include both a client side and an admin side so that administrators may access the data they require to collect information or make adjustments.

In order for the Travlr Getaways website to be created in accordance with the specifications, the MEAN stack will be utilized. The acronym for MongoDB, Express.js, Angular.js, and Node.js is MEAN. The creation of a website that functions properly and efficiently is made possible by the usage of MEAN. Since the database side of the website is created using MongoDB, MEAN uses each acronym to achieve a specific objective. This contains the website's data, which Angular.js uses to manage the application's Web side. Express.js handles this. This is for the server side with Node.js, and it also uses the Express.js framework to retrieve data.

The application's customer-facing side lets users see travel packages. They are able to view and reserve the packages here. The user can search for their stay using pricing or location on the customer-facing portion of the application. There are several sites on the website where users can view information about the various getaways.

The administrator’s single-page application will be a single page that contains all of the website's data and is only accessible by administrators. This application makes it simple to edit the content on the website and was developed using the Angular.js framework. For instance, adding or removing posts and other site-related data.

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

When creating a web application with the MEAN tech stack for Travel Getaways, there are a number of architectural limitations to take into account. It must be simple to update the content on the site's static sections without requiring a whole website relaunch. In order to offer a positive user experience on any kind of device, the design must be responsive. A single page application (SPA) that can dynamically show the many components required to enable the administrator to operate the online application must be the admin portal. Regarding the components of the MEAN tech stack, the database utilized in the MEAN stack is MongoDB. Although MongoDB is generally a good fit for online applications, if the dataset grows too big, it may have performance problems. Node.js, Angular, and Express work well together to finish the MEAN technology, but they can also produce a sizable codebase that may be challenging to manage. It can also be challenging to go back to the original website after converting an existing one to use the MEAN tech stack.

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

### Component Diagram



Considering it depicts the client side, server side, and database, the diagram displays all three sides of the application. There are four sub-components from the client side: the Web browser, Graphic Library, Traveler Portfolio, and Client Session. A web browser is launched by the user and loaded into a client session when the client application launches. This enables the client session to communicate with the authentication server, ensuring that the user is not logged in via a connecting port. The user will be able to view all of the application's information once the Traveler Portfolio has been completed. It then compels the Graphic Library to launch concurrently with the main website.

The database then consists of just one subcomponent, MongoDB. after the Traveler Portfolio has accessed the webpage. To reflect the modifications made to the MongoDB database, the website will be updated by the MongoDB component. The database facilitates the website's correct operation.

The Traveler Database, Mongoose ODM, Server Session, and Authentication Server are the final four subcomponents of the server side. MongoDB is necessary on the server side for Mongoose ODM to function correctly. With Mongoose ODM, code may be written more quickly and efficiently. The proper operation of the Server Session requires Mongoose ODM. This permits the admin and users to log in for the Server Session after verifying the user details against the user database. The Authentication Server then receives this data over the Server Session to confirm that the user is a site user. After sending the Client Session the authentication data, the cycle is restarted.

### Sequence Diagram

A diagram of a software sequence diagram

AI-generated content may be incorrect.

A route is entered by the user into the browser bar as the page address.  
The request is sent to the relevant controller by the browser when it has read the route.  To let the browser know which routes are open for use and where to find them, the controllers are specified in the application's app.js file. The route sends the browser to the relevant controller, which on the server side retrieves the requested data from the models. The controller generates an HTML page that shows the desired information (in this case, a generic web page) and sends it back to the browser for the user to read.

If a browser requests information stored in the database, a second controller for the API is utilized to retrieve and return the information. The process of displaying the HTML page goes as follows, for example, if the page in question displays data from the database in its view. But while the page is being rendered, a second controller is called, and this controller makes use of a middleware device to help retrieve the needed data.

Data from the MongoDB is retrieved using the middleware "mongoose" in the diagram above. The API controller, which is the second controller in this case, receives this data as one or more JSON strings, if any. After the HTTP client receives this answer from the API controller, it sends it back up the chain to the first controller, which renders it for the browser together with the HTML website data.

## Class Diagram

A diagram of a company

AI-generated content may be incorrect.

Each class—Cruise, Flight, and Hotel-Info—should include the name, cost, and particular  characteristics that are available from outside the class as variables for the specified kind of reservation. These classes are generated from the superclass "TripInfo," since it makes sense that an itinerary would contain details on the hotel, flight, and/or cruise a traveler is taking. Ideally, the TripInfo class would also contain the start and end dates as well as other crucial basic details about the entire journey. "Itinerary" is a collection of all the "\_\_\_\_Info" classes since they are all a portion of the information about a journey.

Every "\_\_\_\_\_Booking" class (located on the lower right-hand side) has the methods required to "book" the corresponding item (Cruse, Flight, and Hotel). The "\_\_\_\_Info" classes above and the TravelerInfo class can each implement these classes in various ways.

The center's TravelAgent class will have access to all other classes, enabling it to make reservations and retrieve the data required to examine the trip details.  
A collection of all MemberAccount classes makes up the Membership\_Admin class (far left). The one that   
  
The TravelerInfo class's superclass, the MemberAccount class, is where the number of traveling companions is accessible.

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

| **Method** | **Purpose** | **URL** | **Notes** |
| --- | --- | --- | --- |
| **GET** | Retrieve a single trip using the code | /api/trip/:tripcode | Returns one trip object from the database, referencing the trip to retrieve with the tripcode. |
| **GET** | Retrieve all trips | /api/trips | Brings up every journey in the database. |
| **POST** | Adds a trip to the database | /api/trips | Enables a database trip |
| **PUT** | Updates a single trip in the database | /api/trip/:tripcode | The Angular application uses an HTML form to update a single trip. |
| **DELETE** | Deletes a single trip in the database | /api/trip/:tripcode | Removes a chosen trip code from the list. |
| **POST** | Register a new user | /api/register | Uses a user's name, email address, and password to register them in the database and using hashing to secure their password. |
| **POST** | Login a user | /api/login | Gives back a JWT after receiving a password and email address that have been established for a specific user. |

## The User Interface

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

From what I had previously observed, the angular project structure looks completely different from express. It had a pleasing appearance, and the trips were much larger on the screen than on the client-side express website. In contrast, the angular project structure has a relatively plain visual representation and shows smaller travels. In addition to having an angular structure, the application had a single page, which meant that all of the data was contained on it. Without viewing the remainder of the page, the user can create, amend, and remove trips in the database using this page. On the other hand, the customer-side express software is a multi-page MPA program. When changes are made to SPA, they can be viewed on MPA because both SPA and MPA use the same database to obtain their data.

Receiving and entering data into the database is necessary to test the SPA and make sure it is functioning. That is visible on MongoDB and Postman. However, because my program would not load the page, I was unable to input any data in my situation. However, after reading the reading and reviewing the PDF, I discovered that in order to add the trip to the database, you would first need to press the "add" button and then enter the required data. You can see it on MongoDB after it has been added, and you may then remove it from the database. Posting the URL in Postman is necessary for this to function, as it will add the new journey. Following this, you would choose PUT to edit the trip; this would require updating the URL as well. After that, you GET the database data, which will update the data that was sent in the body.

**References**

“Holmes, S., & Harber, C. (2019). Getting MEAN with Mongo, Express, Angular, and Node (2nd ed.). Manning Publications. <https://learning.oreilly.com/library/view/getting-mean-with/9781617294754>.”