

Travlr Getaways

# **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 | 09/12/23 | Zac McBride | Setting up routes controllers and views for App Server |
| 1.1 | 09/29/23 | Zac McBride | Setting up MongoDB and RestAPI |
| 1.2 | 10/10/23 | Zac McBride | Finishing SPA Admin application. |

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

The full stack for Travlr Getaways will use the MEAN stack architecture. A NoSQL database, MongoDB will be used to store information about the travel destinations and user information, including both clients and admins. The front end will use an SPA for a single-page application for a fast and responsive user experience. This will be built using Angular and is especially useful on the admin side of the application. The application will use node.js to connect the front end and backend together using the Express framework. To simplify the build process Express will also use Handlebars templates which will help minimize duplicate code in the application. One main benefit with the MEAN stack is to have the entire application using JavaScript.

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

Using Angular can make it difficult for search engines to crawl and index. Search engines are getting better at overcoming this and there are workarounds using the MEAN stack where certain aspects for the application would only use Express, Node.js and MongoDB. Ultimately this needs to be prioritized based on the value crawlable HTML would add to the application.

Another thing to consider when using MEAN is that it is not ideal for large-scale applications. MongoDB is not designed for very large datasets meaning the application would experience slow down. During heavy loads it may even see a loss of records written by MongoDB. Since this would not happen until millions of documents were reached it is unlikely Travlr Getaways will run into this problem.

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

Below there are several diagrams to help illustrate the overall structure of the application. As well as how users will interact with it.

### 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).

The main components of the application will have a client-side built using Angular SPA, a database using MongoDB, and the server- side being built with Express running on a node.js server.

The client-side is run in a web browser, where the user will log in and view the application with a personalized experience. The Angular SPA uses metadata to extend HTML tags which will make the application more dynamic and interactive. It reduces the amount of loads the application will need, creating a faster experience for the user.

The server-side handles the communication between the front end (the clients view) and the back end (the database and node server). The user’s session starts after validation through an authentication process. The Express framework supports and hosts the node.js application. It also controls the routes from the application. This part of the application gets information from the database and communicates what to display in the front end.

Lastly, the database is run using MongoDB and Mongoose middleware. It is a NoSQL database, meaning it stores JSON documents without having to use rigid relational schemas. The documents are passed from the MongoDB database using Express and then passed to the view templates.

**Sequence Diagram**

A screenshot of a computer screen

Description automatically generated

The sequence for accessing the application begins with the user entering a route in the URL. This is done originally by accessing the website’s homepage and then using links to enter other routes. The route will then be forwarded to the app server controller which pulls the appropriate view and template. The view will most likely need information from the database. This is requested through the HTTP client. This requests the data through the Rest API route for accessing the database. The route interacts with the API controller and model. Mongoose interacts with the database to request the data from MongoDB.

After the database processes the request, the data is sent back through the controller to the route which sends the results to the view. The data is processed by the app server controller and puts the appropriate data into the view using the handlebars template.

**Class Diagram**

A computer screen shot of a diagram

Description automatically generated

As this is a travel site, the first class to look at is the TravelAgent class. This class Is used to book the trips. It can book a hotel, flight, and/or cruise. This is extended by the HotelBooking, FlightBooking, and CruiseBooking classes. These classes get and assign the information for the CruiseInfo, FlightInfo, and HotelInfo classes. These are aggregated by the Itinerary class. With all this info the TripInfo class is assigned values for the information about the entire trip. The TravelAgent class also interacts with the Membership\_Admin class which is used to verify and get points for the MemberAccount.

All these classes are used to store the information necessary to let a user plan and book a vacation. The MemberAccount class is used to keep track of which user is booking a vacation. A member number is used to differentiate each member. This class also has information about the user’s membership plans and is used with the Membership\_Admin class to keep track of how many member points they have as well as use these points toward a vacation.

The rest of the classes in this diagram are used to keep track of information about a specific vacation. The user will need to access information about each part of the trip including Hotel, Flight and/or Cruise information. The most important part in all these classes is the price information. Using this, the Itinerary class will tell the user how much the trip will cost in total. The TravelAgent class uses the Itinerary data to book the trip. The TripInfo class uses the rest of the data in these classes to let the user know the dates of the trip and the airport or city where they will begin their trip as well as the destination.

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

| **Method** | **Purpose** | **URL** | **Notes** |
| --- | --- | --- | --- |
| **GET** | Retrieve a list of trips in database | /api/trips | Returns a list of all trips from the database |
| **GET** | Retrieve a trip by using the trip code | /api/trips/:tripCode | Returns one trip from the database using the trip code to reference which specific trip |
| **POST** | Add a new trip to the database | /api/trips | Adds a trip with all appropriate information about the trip including the name and trip code |
| **PUT** | Updates a trip already in the database using the trip code | /api/trips/:tripCode | Updates the trip by taking information about the trip and changing what is different |
| **DELETE** | Deletes a trip from the database using the trip code | /api/trips/:tripCode | Removes the trip with the trip code for reference |

## The User Interface

The customer facing application is an Express application using Express and Node.js to pull data from the database which is then displayed in the views for the user. Having the application set up in this way means any changes happening to the data in the database can be displayed immediately to the customers who are viewing and booking trips. The admin side of the application uses the same API created to pull data from the database, but it also uses Angular to create an SPA which gives a better experience for the admin and their needs.

An SPA means everything is loaded into the browser at once. Because the admin will have to use more information to perform their tasks, an SPA helps give them a better experience. They may have to wait longer initially to load the page but after this they will be able to work with larger amounts of data without having to wait for everything to load for every little task.

The customer facing application only uses GET requests to the database to view the information stored there. The admin tasks require POST, PULL, and DELETE requests as well. Using these allows the Angular SPA application to use the custom API to view data as well as add, update, and delete from the database. Once the user logs in to the admin side of the application and navigates to the trips list, they will be able to see cards showing each trip in the database with all the information about the trips.

A screenshot of a website

Description automatically generated

Image of trips list including Mega Reef and Kayak Cove which were added by the admin user.

The admin user can use the Edit button at the bottom of each trip card to update the information for a given trip. Clicking this will take them to the Edit trip screen.

A screenshot of a computer

Description automatically generated

Edit trip screen preloaded with current information about the trip.

The user can then change what information they need to change to update the trip data. At this point they can save using the save button at the bottom of the edit trip form. Using a PUT request, the API will update the information for this trip in the database.

A screenshot of a phone

Description automatically generated

Update trip card for Kayak Cove.

These screen shots show how the user goes through the process of updating a trip. This is also the process of testing how the Angular SPA works using the custom API to get, add, change, and delete information from the database.