

Travlr GetAways Webpage Application

# **CS 465 Project Software Design Document**

Version 3.0

## 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 | 01/27/2024 | Jessica Duft | Added text to Executive Summary, Design Constraints, and System Architecture View |
| 2.0 | 2/10/2024 | Jessica Duft | Added text to System Architect View, Sequence Diagram and API Endpoints |
| 3.0 | 2/25/2024 | Jessica Duft | Added text to API Endpoints and The User Interface |

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

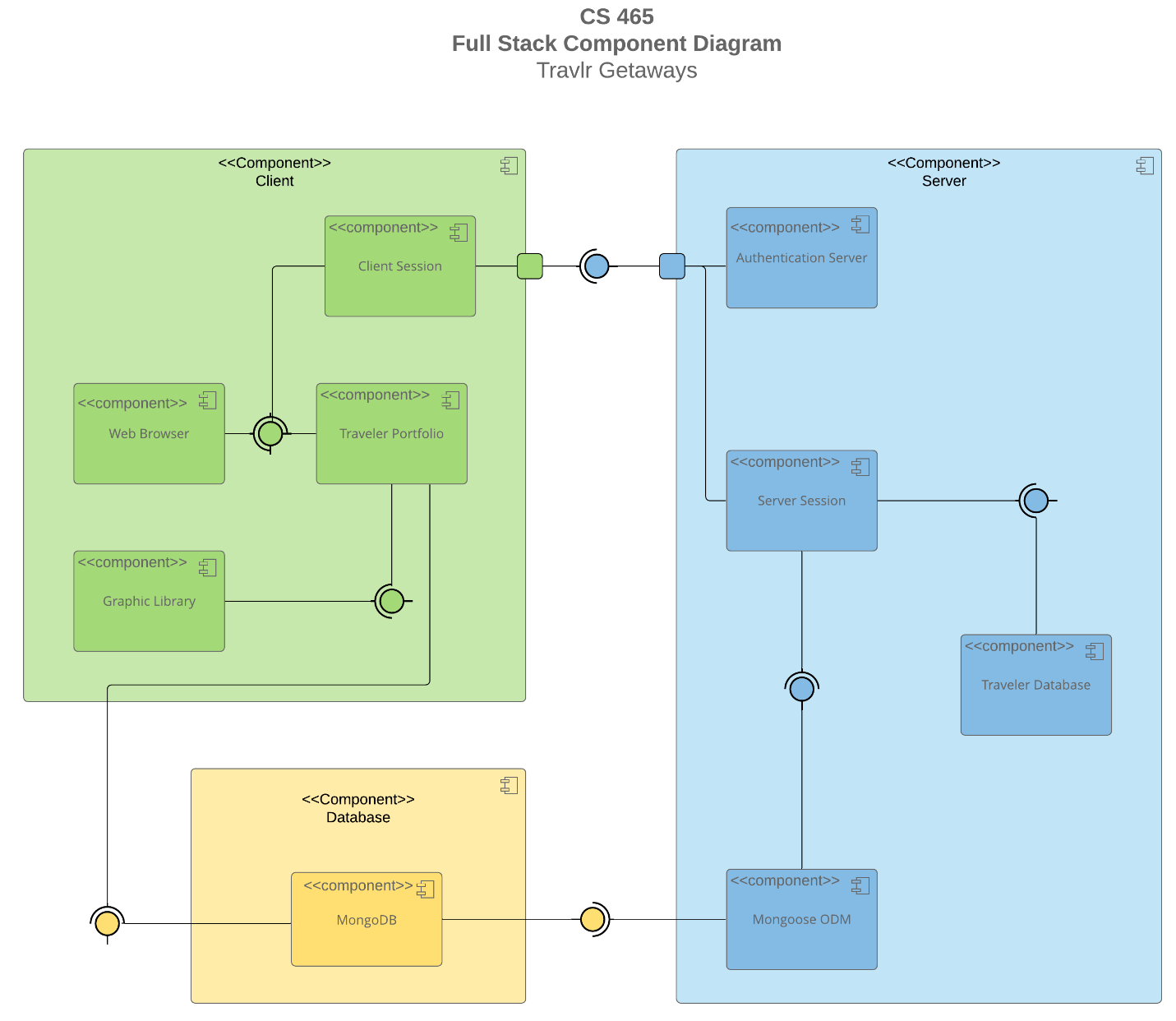
The company, Travlr Getaways, needs to develop an easy to navigate and intuitive web based application that allows users to browse and book vacations. Specifically the company needs a customer-facing web page, a database, and an admin page that can all be integrated and work together. The main issue is getting the three to work together because notoriously, integration can be 'buggy'. In order to ensure flawless integration, MEAN stack development practices will be implemented in order to create compatible and attractive web pages for Travlr Getaways. This means we'll be using MONGODB for the database, Express for back-end services, Angular for front-end services and Node.js to run the code in real time on a browser. All technologies are increasingly popular, especially in cloud-based applications. All technologies use JavaScript and are easily integrated resulting in an easy to use, easy to customize, and intuitive end product. The customer facing side and the admin-based side of the application will use Angular to create a single-page application (SPA). This means that instead of loading whole new pages when a user or admin navigates to a new page, the application will dynamically pull certain features (such as headers, or footers that remain the same on each page) in order to decrease load time. MEAN stack development is perfect for Travlr Getaways because this method will allow the client to create the web application of their dreams in a cost effective and resourceful way. Additionally, if the client would like to make changes it would be easy and quick to do so.

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

One major design constraint that comes to mind when considering the MEAN stack development method for this project is the lack of ability to support large-scale applications and a possible loss of data when attempting to transport large records into the Mongo database. Neither of which should be problematic for a company like Travlr GetAways, and if the company were to expand so much that either of these did become a problem : additional actions can be taken at that time. For the foreseeable future, Travlr GetAways should be able to operate with these technologies with very limited issues.

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

### Component Diagram

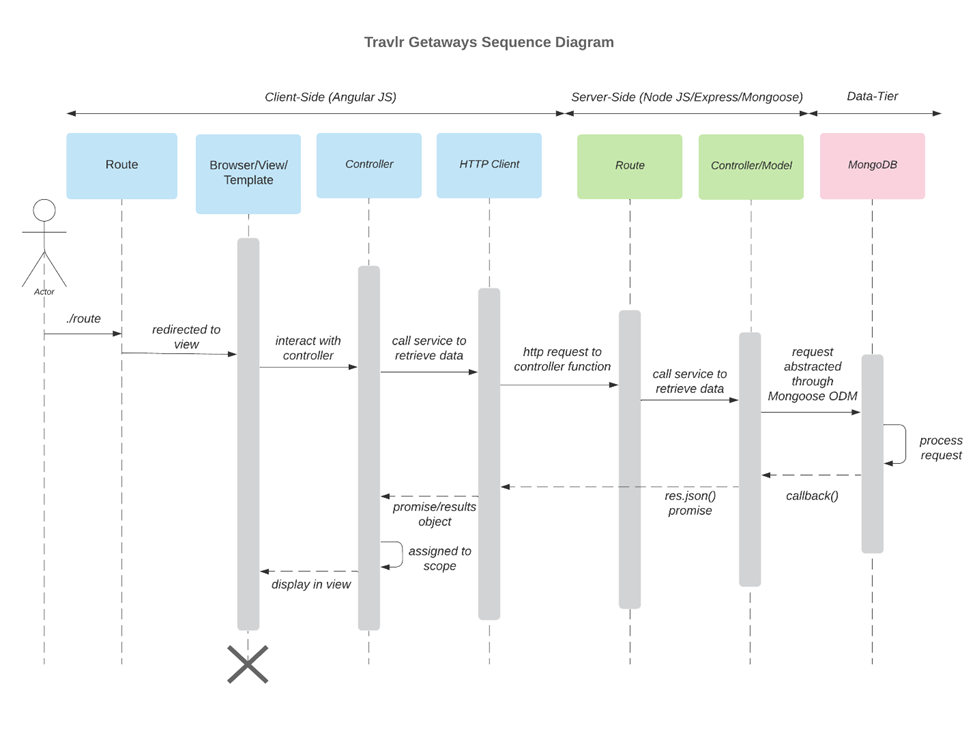


A text version of the component diagram is available: [CS 465 Full Stack Component Diagram Text Version](https://learn.snhu.edu/d2l/lor/viewer/view.d2l?ou=6606&loIdentId=24342).

As far as the web application architecture goes, the diagram above shows a detailed view of the interactions between the client, server, and database ends.

The architecture will work something like this: A user will click or type the link to Travlr GetAways's web application. As soon as the user presses go or enter, the server sends the appropriate data and files to the web browser. The web browser loads the page, and the user is able to interact and browse. As soon as a user clicks a link within the web page to go to a different page (Think user went from home -> trips) the server again sends the appropriate files to the web browser and the new page loads. The database is a component in and of itself, but the server has access to that database and allows for things like authenticating users and saving new user information. The client side also has limited access to the database, but not to sensitive information. The client side is able to access the Traveler Portfolio database so that they are able to view trips, rooms, etc.. quickly and seamlessly. All three components work together to create a seamless user end experience.

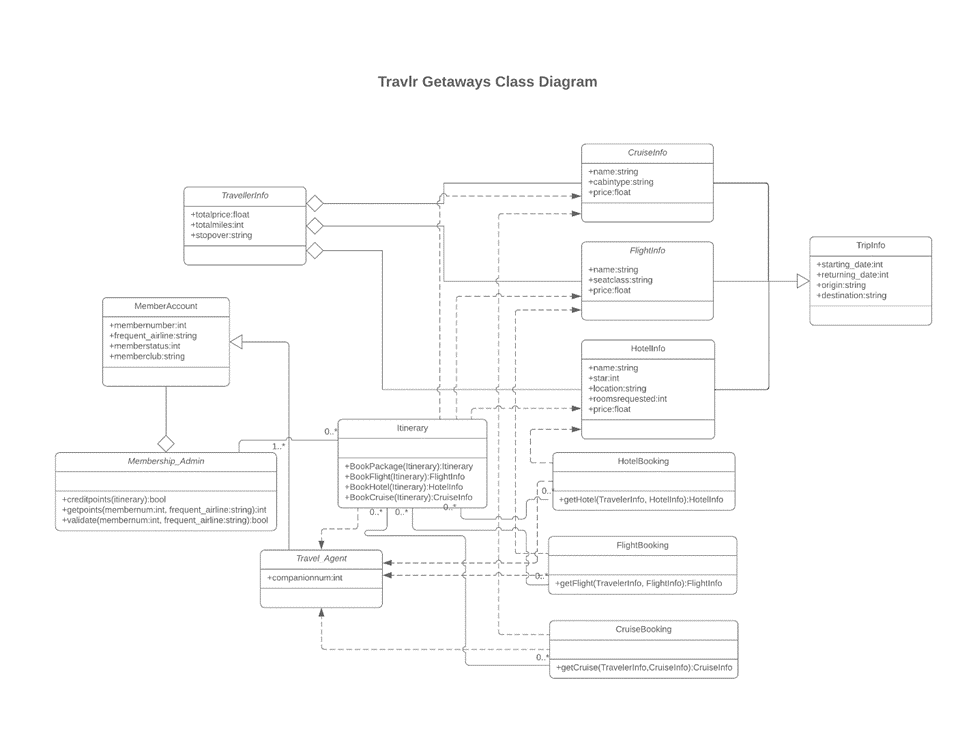
### Sequence Diagram



The user routes to the web page by typing in the appropriate URL into the address bar in their browser. The browser then deciphers the route and forwards the request to the appropriate controller. The controller then gets the data requested from the models on the server side of the system. The controller creates a page that displays the data and returns it back to the browser so that the user can view it. This repeats each time a user clicks a new link or anchor on the web page. Data collected from the database works in a similar manner. Mongoose will get the data from MongoDB and return that data to the API controller which returns it back to the web browser.

So for example let’s say a user types in the web page URL and enters their login information. The web page will forward the request to the appropriate server, which will then forward the request to Mongoose to retrieve the data from the MongoDB (to validate credentials). This information is sent back to the server side and the controller returns the appropriate data back to the webpage; in this scenario the controller will either allow the user to login with their credentials and display the requested page or it will display a page requesting the user to login again (if the username and password were invalid).

## Class Diagram

  
TripInfo is our superclass which includes informations about cruises (CruiseInfo), flights (FlightInfo), and hotels(HotelInfo). The superclass has all of the information listed above plus start date, return date, origin and destination.

Right Below the CruiseInfo, FlightInfo, and HotelInfo classes we have HotelBooking, FlightBooking, and CruiseBooking which all contain booking information for the cruises, flights and hotels in the classes mentioned above. The itenirary class allows a member to book a combination of the three as a package.

Next we have the TravelAgent class which is the only class that has access to all other classes. It is set up this way so that a travel agent can assist with booking and planning a trip. In order to do that, they would need access to flight info, destination info, etc..

Lastly we have the MemberAccount class and MemberShip\_Admin accounts. This just relates to members and allows admins to retrieve points earned and adjust as necessary.

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

| **Method** | **Purpose** | **URL** | **Notes** |
| --- | --- | --- | --- |
| **GET** | <Retrieve all trips> | </api/trips> | <Returns all active trips> |
| **GET** | <Retrieve trip by trip code> | </api/trip/:tripId> | <Returns trip, identified by the trip ID passed on the request URL> |
| **DELETE** | <Delete a trip from the database> | </api/trips> | <Deletes a single trip> |
| **POST** | <Add a trip to the database> | </api/trips> | <Adds a single trip> |
| **PUT** | <Edit a trip in the database> | </api/trips> | <Edits/Updates a single trip> |

## The User Interface

Let’s first take a quick look at the customer facing web application. The customer facing page is an Express application. When a customer navigates to the corresponding URL, they will see the following :

A person lying on a beach

Description automatically generated

From here a user has the choice to view travel choices, room options, meal options, news, the about page, and contact information. When a user clicks the “travel” link, the trips will be displayed as below:

A screenshot of a computer

Description automatically generated

You can also view trip data using the API endpoint /trips. See below :

A screenshot of a computer

Description automatically generated

When development is complete potential customers will be able to successfully view, and book trips, rooms, and more on the customer facing Travlr web application.

The admin site for Travlr is designed as a single page Angular application (which we will refer to as an SPA in this article). Using the SPA accounts with authenticated admin access are easily able to add, delete, and edit data in the Mongoose database. This is an easy-to-use, intuitive , and cost effective resource for the company. See below how an authorized administrative account holder is able to easily add, edit, and delete data within the database.

Here is a look at the admin site prior to logging in and making any changes

A screenshot of a computer

Description automatically generated

First we log in, these are credentials that have been registered as admin credentials using POSTMAN :

A screenshot of a computer

Description automatically generated

See POSTMAN verification of credentials below:

A screenshot of a computer

Description automatically generated

Next we load the trip-cards by clicking the “Trips” link on the nav-bar. The admin user will see a page that looks like the image below. Notice the three original trips that are currently displayed.

A screenshot of a computer

Description automatically generated

And here is the same data in PostMan :

A screenshot of a computer

Description automatically generated

Now let’s test our SPA and confirm that when we add, edit, and delete data within the application, that those changes are actually taking place. First we click the Add Trip button located right underneath the nav-bar component.

For testing purposes I have entered the “test trip” information shown below: A screenshot of a computer

Description automatically generated

Not shown in this photo is the “SAVE” button located at the bottom of the page. Upon clicking save and returning to the trip-list page, you can see that our test trip has successfully been added to the SPA.

A screenshot of a computer

Description automatically generated

See that the data for the test trip is now also shown in PostMan, this verifies that the data was successfully added to the database.

A screenshot of a computer

Description automatically generated

The data is also now displayed in our API , localhost300:/api/trips , as shown below:

A screenshot of a computer

Description automatically generated

Let’s go back to our trip list and delete the test trip we just added. To do this I will simply just click the “Delete” button located on the bottom of the test trip trip-card.

A screenshot of a phone

Description automatically generated

Now we return to the trip-list page and you can see that only the original three trips are shown on the web application.

A screenshot of a computer

Description automatically generated

We can confirm the data was successfully removed from the database by checking the trip data in PostMan. As shown below the delete method was successful.

A screenshot of a computer

Description automatically generated

Now let’s confirm that our EDIT method works. To edit a trip the admin user simply will click the EDIT button located at the bottom of the trip-card. For this example we will be editing the price of the Emerald Bay (Gale Reef) trip.

A screenshot of a website

Description automatically generated

Upon clicking the EDIT button, the following page will render :

A screenshot of a computer

Description automatically generated

Let’s change the price from $799.00 to $699.00, maybe there’s a promotional offer on this trip or some other applicable discount. All the admin user has to do is edit the price field and click the SAVE button shown below:

A screenshot of a computer

Description automatically generated

Now we return to the trip listing screen and as you can see the price was successfully changed for the Emerald Bay (Gale Reef) trip.

A screenshot of a website

Description automatically generated

The data is also changed in PostMan, which further verifies that our EDIT method works.

A screenshot of a computer

Description automatically generated

The admin-site for Travlr’s web application will prove to be a valuable tool for maintaining and updating the Travlr database.

Now that we’ve discussed what the SPA is and how it works, let’s talk a little bit about the structure of the admin web page. Our Angular, SPA admin page, uses a component-based structure wherease the Express, user-end page, follows a Model-View-Controller(MVC)based. The MVC structure constantly needs to communicate with the server to load data, which is done mainly through API end points. The SPA, component-based structure, does not require the constant ‘back and forth’ communication with the server. Once the page loads initially, it does not have to be reloaded unless the session ends or the user refreshes the page. The SPA helps to relieve some of the stress put on the server by the user-end MVC structure. This will be particularly important as Travlr GetAway’s database grows. The SPA allows admins to quickly and easily maintain their data without affecting the user end application (Our Express) load time or abilities to interact. The SPA created for Travlr will allow admins to easily maintain their web page data (add trips, delete trips, etc..) without interrupting the flow of business on the customer-facing web application. Both the SPA Angular application and the MVC Express application work hand in hand to create a system that meet all of Travlr GetAway’s business requirements.