

Travlr Web Application

# **CS 465 Project Software Design Document**

Version 2.0

## 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 5](#_Toc36198470)

[Class Diagram 6](#_Toc36198471)

[API Endpoints 7](#_Toc36198472)

[The User Interface 8](#_Toc36198473)

## [Document Revision History](#_heading=h.lnxbz9)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 05/22/2020 | Lesley Potts-Langdon | Initial Executive Summary, Design Constraints and System Architecture View Component Diagram explanations. |
| 1.5 | 06/16/2020 | Lesley Potts-Langdon | Added Class Diagram and API endpoint. Needs Diagram explanation and additional endpoints. |
| 2.0 | 07/17/2020 | Lesley Potts-Langdon | Added Class Diagram description, API endpoints and explained the User Interface |

## 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 Travlr web application has three components. The front end is the user interface. This is not just esthetics; it is the client experience and how the various pages fit together. The back end is the administrator side. That is a single page that enables the administrator to interact with the application. It isn’t as pretty as the client side, but it must be efficient. These two sides connect not only with each other, but also with the database. To keep all these parts connected and communicating we are using a system known as the MEAN stack. This is a group of very different software applications that work in harmony to build the different components. The M stands for MongoDB which is the database part of your application. The web framework is Express, the E. Express reduces the amount of coding we have to do making the process faster and reducing potential errors. Angular is the front-end framework and the A in MEAN. We choose angular because it works well with the other components and it will make your website run faster. Finally, the N, Node will be our server application. This is how we get your web application components talking to each other and how we get the full application on the internet for the world to see.

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

One key design constraint is the need to update the site frequently. Travel options and trends change constantly and to keep up with those changing trends, the application needs to be easy to maintain. By linking the front-end user experience to the database we can set up an application where changes to the database are seamlessly reflected on the web page.

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

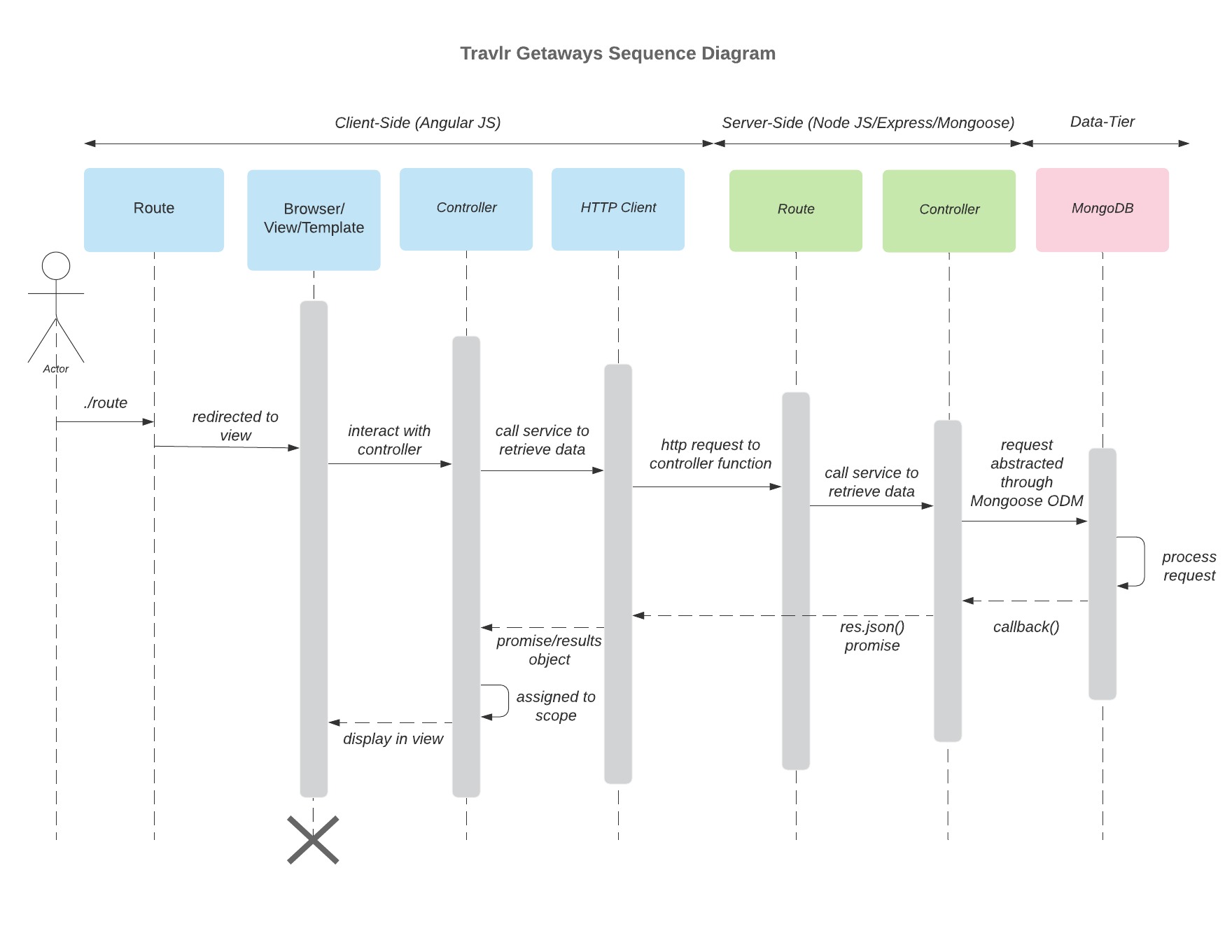
### Component Diagram

If you look at the diagram that follows, you will see the three components we talked about in the executive summary; the client, the server and the database. The diagram shows how the three components link and work together. When a user opens the web application, that request is authenticated by the server and a server session begins. The web browser is how the user interacts with the server. As the user is viewing content on the application, that content is being pulled from the database. The database also links with the server side so the administrator can update the data being fed to the application.



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).

### Sequence Diagram



When a user enters a URL that is the route to the page they are asking to view, the page will show in the browser window. When the user interacts with the web page it is the controller that takes their requests and displays the response. The control does fulfill the request by making an HTTP request to the server side. The route leads to the controller which will communicate with MongoDB when needed. The requested information than works its way back through the controller which passes data back through the HTTP client and client side controller to finally be displayed in the browser window.

## Class Diagram

A close up of text on a white background

Description automatically generated

The class diagram shows all the classes of the Travlr Getaways website and how they interact with each other.

The Travel\_Agent class connects with all other classes either directly or indirectly. It is the hub for booking each aspect of a trip.

CruiseInfo/CruiseBooking, FlightInfo/FlightBooking, and HotelInfo/HotelBooking are three similar pairs of classes. The Info class holds all the information about the cruise, hotel or flight and the Booking class uses a get method to access that information in order to book the trip.

The TripInfo class holds some information that will be consistent each time the same trip is offered while the Itinerary is instance specific including dates.

The MemberAccount class contains information needed for traveler rewards and is administered from the Membership\_Admin class.

Finally, the TravelerInfo class is used to establish if there is a second traveler.

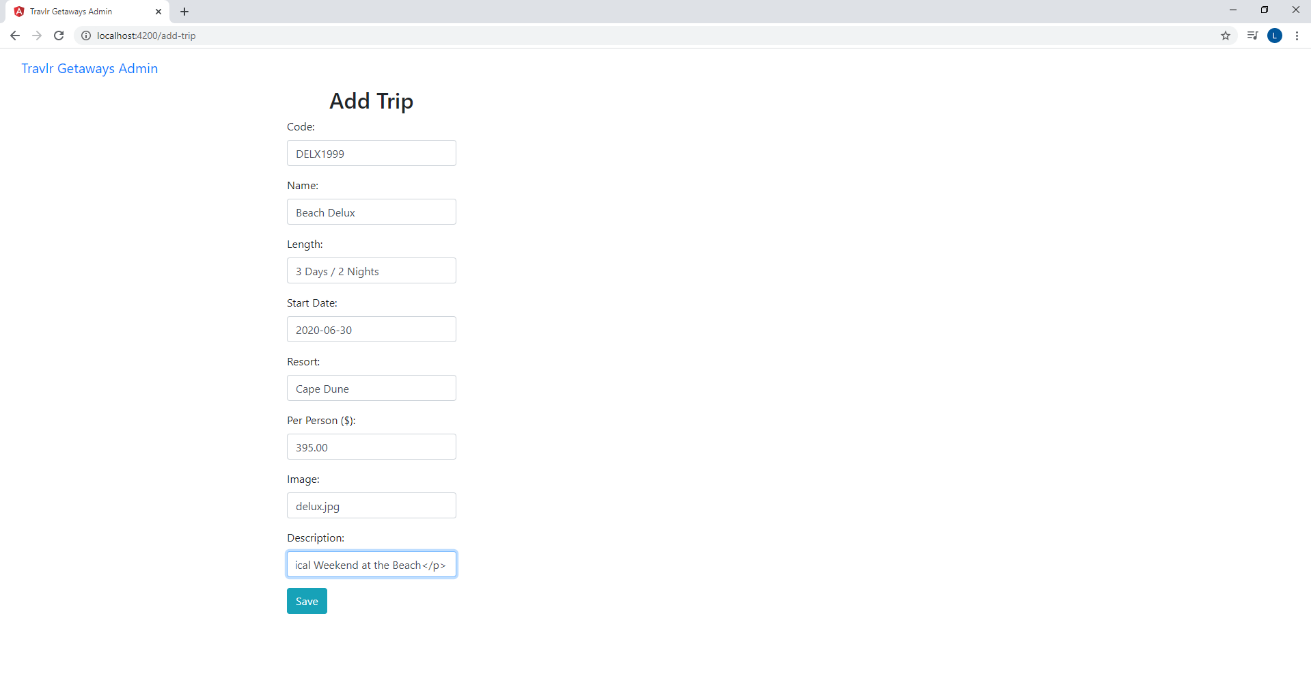
## [API](#_heading=h.2jxsxqh) Endpoints

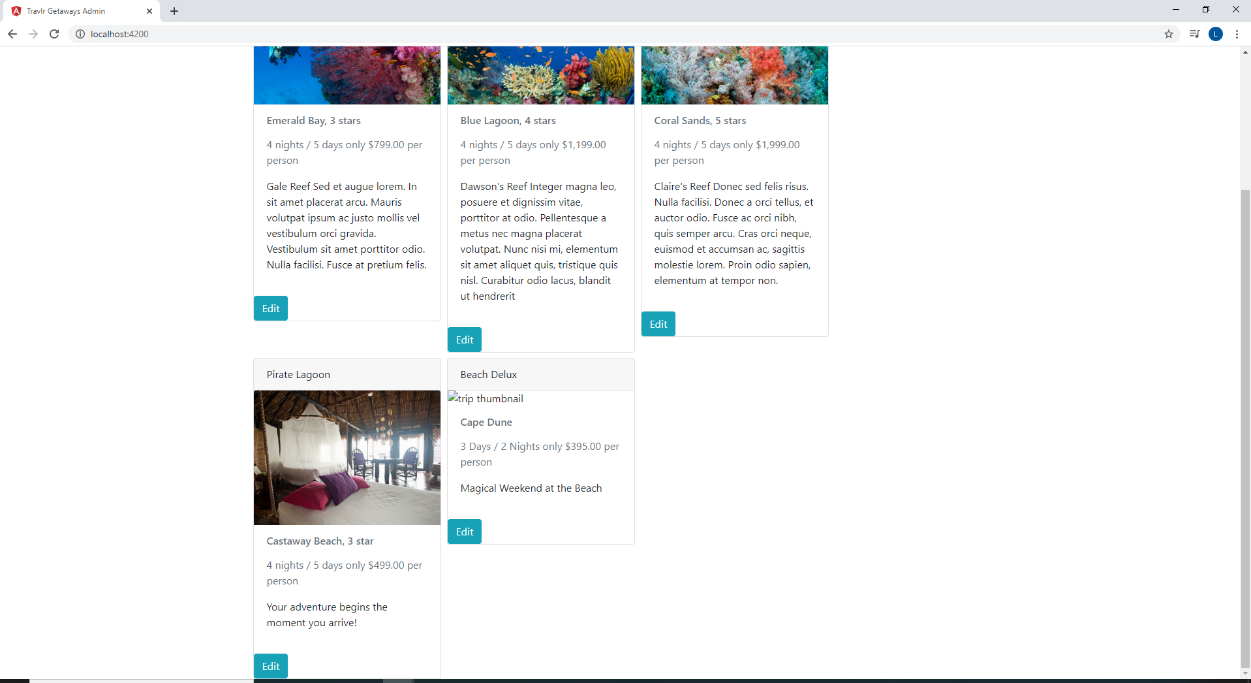
| **Method** | **Purpose** | **URL** | **Notes** |
| --- | --- | --- | --- |
| **GET** | Retrieve a list of trips | </api/trips> | Returns all trips |
| **POST** | Add a trip | </api/trips> | Checks for authorized user, Stores trip info in db |
| **GET** | Retrieve individual trip info | </api/trips/:tripCode> | Pulls up trip info for editing |
| **POST** | Update trip | </api/trips/:tripCode> | Checks for authorized user, Stores new trip info in db |
| **POST** | Login in a user | </api/login> | Checks authorization and logs in user |
| **POST** | Add a user | </api/register> | Registers a new user including name, email and password. |

## 

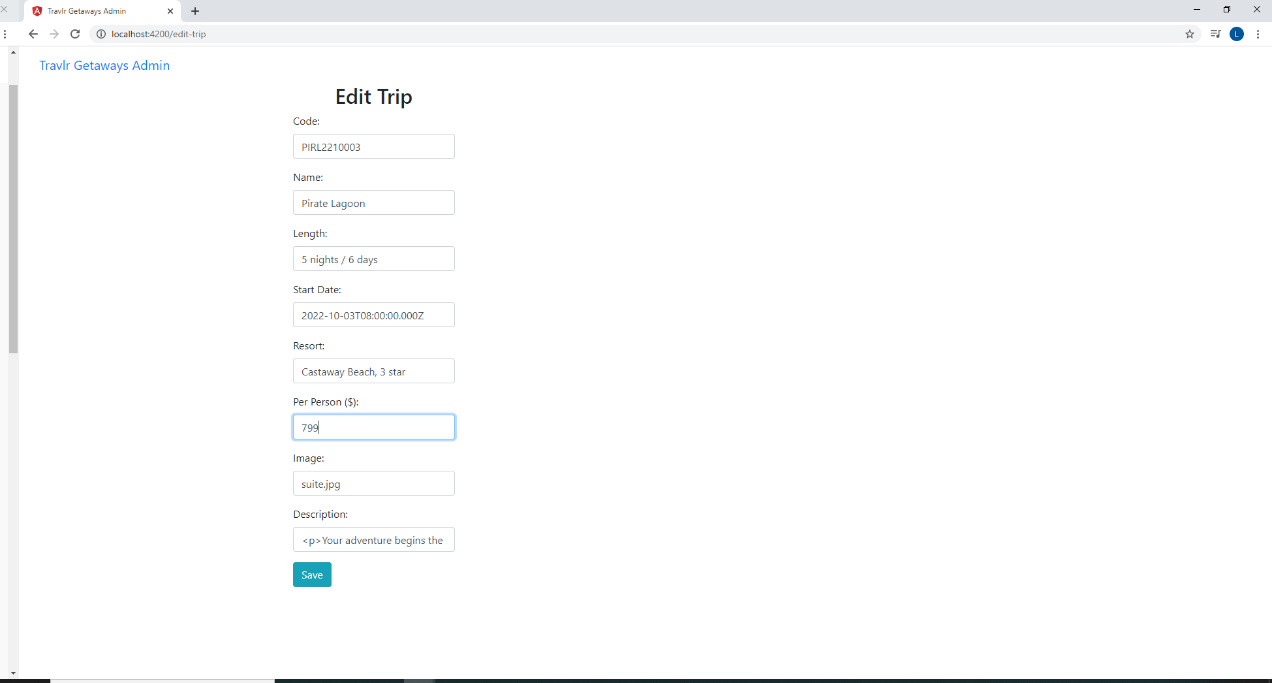
## The User Interface

<Insert screenshots from the development of the SPA development to show the following: (1) a unique trip, added by you, (2) the Edit screen, and (3) the Update screen.>

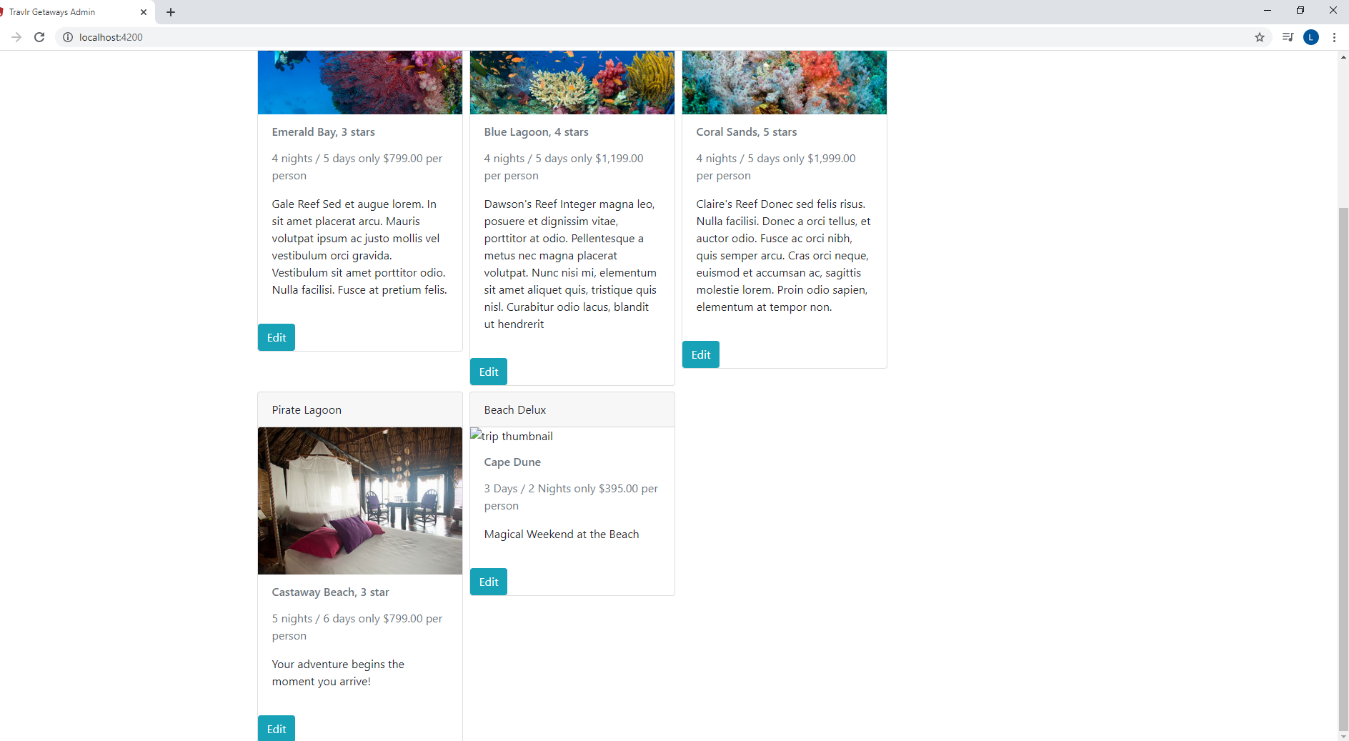


The Add Trip Screen

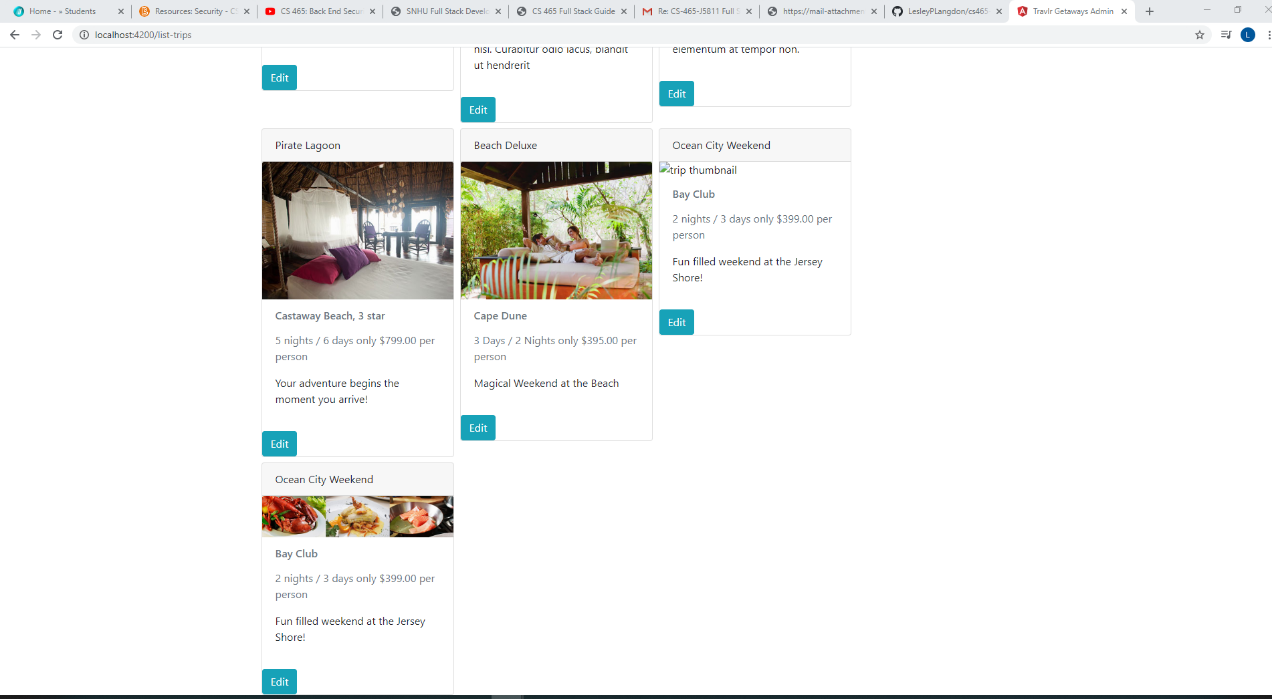
The Trip has been added



The Edit Trip Screen



The Trip has been edited and updated



Another trip added after security features implemented

The significant structural difference between the Angular project structure and the Express project structure is which side carries the larger load. When we use express for our whole application there is a lot of back and forth communication between the server and the client-side browser. Angular shifts much of the load onto the browser by loading most of what is needed as a single page application (SPA) at start up. The only communication between client and server with Angular is to send and receive data.

In deciding on which approach to take we weighed the advantages and disadvantages of a single page application (SPA). One of the most noticeable differences from the user’s perspective is load time. A SPA loads more in the beginning and therefore takes longer. If you have a large application that people are not familiar with that could be a real problem. If you are trying to drive traffic to your site to introduce your product and the site takes a long time to load, you could lose people before they even see what you have to offer. If you have a smaller application with limited server capacity, you would benefit from shifting the load to the user’s browser and they would likely not notice the delay in loading. When compared to an amazon or google, Travlr Getaways is a smaller application. We also anticipate that once someone is on the site, they will spend some time reading details on each trip. Travlr Getaways can benefit from sharing the server load with the browser. For these reasons, a SPA is a good option.

We used a few tools to test the SPA to make sure it was communicating with the database and working properly. We used Robo 3T, the terminal and Postman to test the POST, GET and PUT methods. We first used postman to POST a new trip, we could see that the POST showed in the terminal. We then went into Robo 3T to see that the new trip data was included in the MongoDB database. When we call or post something from within the site, we can see the GET and POST in the terminal. We also used postman to test the PUT method. Postman enables us to test the API separate from the front end application which is critical to efficient project management.