

Travlr Getaways Website Build-out

# **CS 465 Project Software Design Document**

Version 1.2

## Table of Contents

[**CS 465 Project Software Design Document** 1](#_Toc142059558)

[Table of Contents 2](#_Toc142059559)

[Document Revision History 2](#_Toc142059560)

[Executive Summary 3](#_Toc142059561)

[Design Constraints 3](#_Toc142059562)

[System Architecture View 3](#_Toc142059563)

[Component Diagram 3](#_Toc142059564)

[Sequence Diagram 4](#_Toc142059565)

[Class Diagram 6](#_Toc142059566)

[API Endpoints 7](#_Toc142059567)

[The Admin Site User Interface 7](#_Toc142059568)

[SPA Framework vs. Express 8](#_Toc142059569)

[Testing the Admin Site 9](#_Toc142059570)

[Admin Security 12](#_Toc142059571)

[References 15](#_Toc142059572)

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

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | June 30, 2023 | Brandon Hobbs | Module 3 Updates and Initial Release |
| 1.1 | July 19, 2023 | Brandon Hobbs | Adding Sequence, Class Diagrams and API |
| 1.2 | August 1, 2023 | Brandon Hobbs | Adding SPA User Interface |

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

The *Travlr Getaways* website is based upon the established MEAN technology stack. This stack, using MongoDB, Express, Angular, and Node.js, “…pulls together some of the “best-of-breed” modern web technologies into a powerful, flexible stack” (Holmes & Harber, 2019).

This toolset is also valuable to your end users especially as the user interface becomes richer. The user experience will be rendered quickly with cross-platform support (unlike older technologies such as Silverlight).

This stack allows rapid transition from prototyping to commercial availability while also providing a free, streamlined, open-source architecture. Using a single-page application (SPA) for the administration tasks allows admins to add, remove, or update site content easily. These features allow for a cost-effective application with reduced on-going support costs.

Moreover, if a smartphone app is envisioned the MEAN stack is isomorphic so limited changes are needed between versions (Vivim, 2019).

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

All design and architectures come with compromise. For the SPA and MEAN architecture there are a few noteworthy.

MongoDB, the M in MEAN, which will be storing the raw files has a few limitations. Firstly, MongoDB does not support large documents. The maximum size of a single document, e.g., picture, text file, etc., may only be 16 Mb. Moreover, MongoDB assumes the server will have a lot of storage, hard disk. This can lead to slightly larger cloud costs over time.

One other constraint of creating the site as a SPA limits search engine optimization (SEO). “If your server-generated HTML is deemed to be too different from the SPA content, your site will be penalized” (Holmes & Harber, 2019). However, because only the administrative tools, and not the customer facing side, will be built as an SPA no penalty should occur.

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

Within the next three sections the individual components, the sequence of messages passed between objects, Sequence Diagram, and the Class structure, Class Diagram, from which the Objects are created are discussed and shown.

### Component Diagram

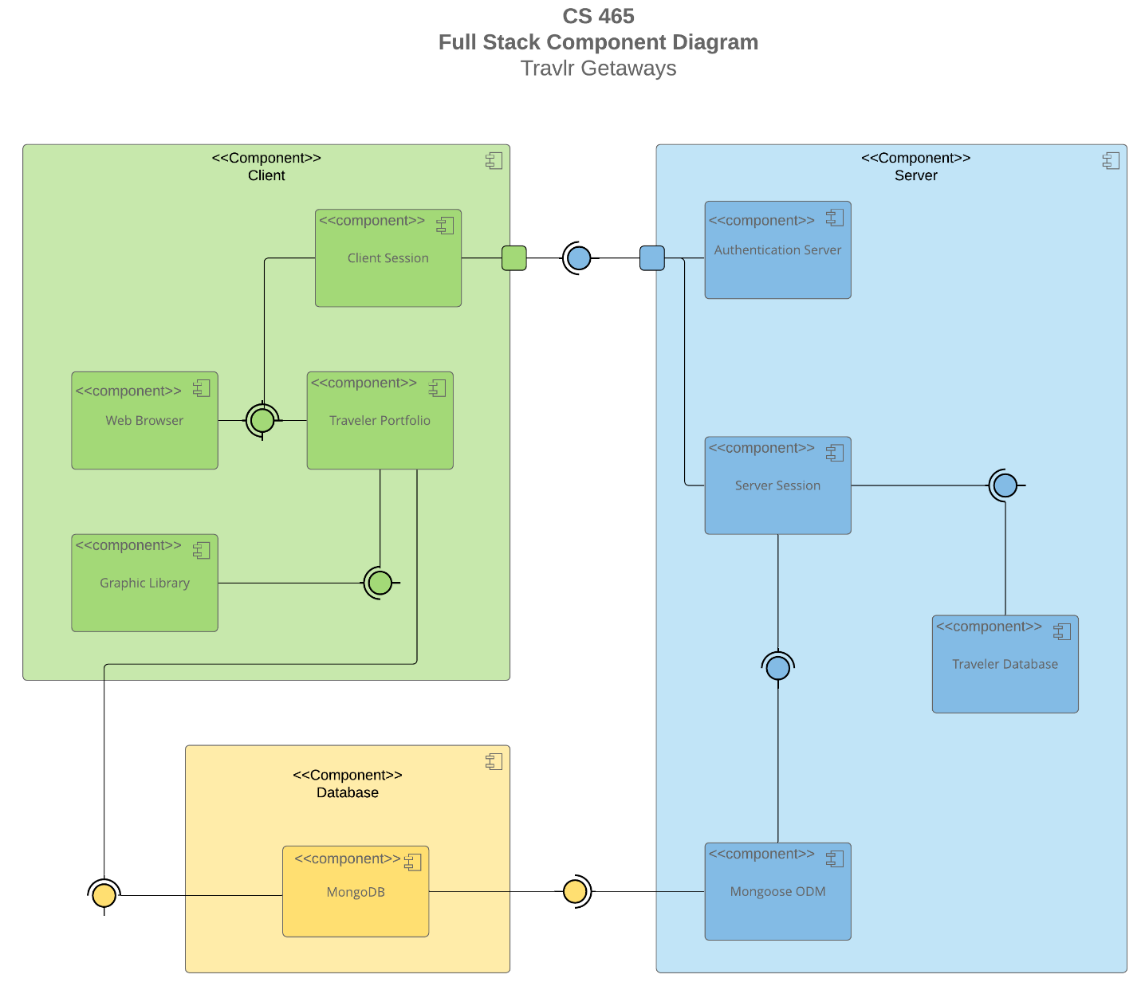
**Figure 1** shows the overall Travlr site’s component diagram. The client-side, customer’s browser, commonly called the front-end and shown in green, is running Angular.js. The client-side will contain multiple smaller components enabled by the MEAN stack, i.e., the graphics library and the ability to store sessions (tailor customer interactions or configure entitlements and persist them). The client-side connects to the Server and Database components through required interfaces. For example, the *Client Session* requires a user’s entitlements stored on the *Server* component *Authentication* server – which it does through one of these interfaces.

The *Server* component, commonly called the back-end, contains the tools needed to authenticate a user and handle the communication between the MongoDB and serve the information to the front-end. MongoDB will be interfaced with via Mongoose middleware – which allows for data model schema enforcement at the application layer.

The Database component physically exists on the server but is external to that component. Moreover, it can independently connect to the Client or Server. The MongoDB will store the information files needed for the site, e.g., images, descriptions of packages (stored as a JSON file). The documents stored in MongoDB can be served either to the backend or to the front-end which depends on the type of call made.

The three components will be connected through REST APIs. A REST API (Representational State Transfer Application Programming Interface) is a type of web service that allows systems to communicate and interact with each other over the internet. REST APIs are commonly used to enable communication between different software applications or services. They provide a standardized way for clients to request and manipulate data from a server. The server, in turn, responds to these requests and provides the requested data in a structured format, typically in JSON (JavaScript Object Notation) or XML (eXtensible Markup Language).

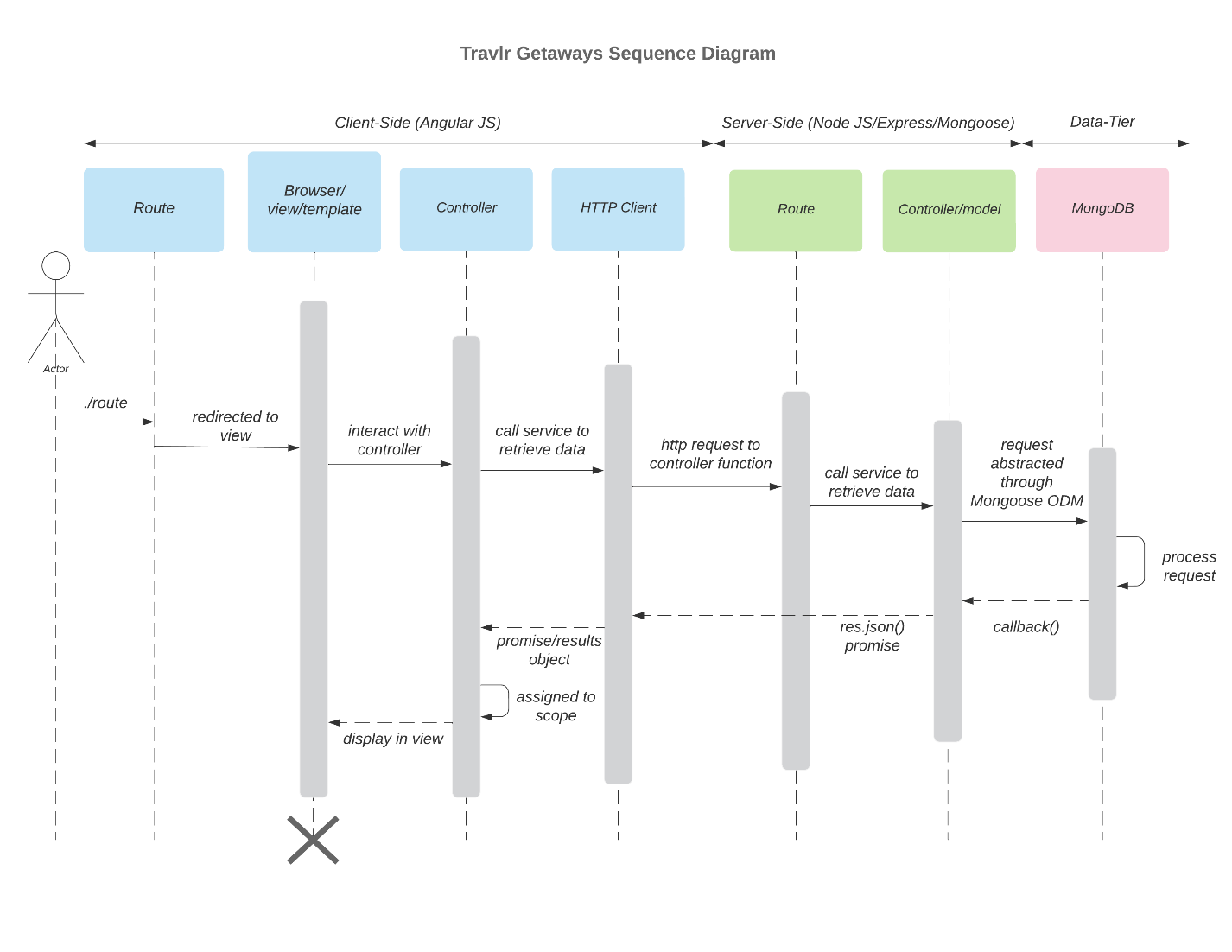
All calls between the components will be made using these APIs.



**Figure 1: Travlr Website Component Diagram**

### Sequence Diagram

**Figure 2**, below, shows the Sequence Diagram for the Travlr Getaways site.



**Figure 2: Travlr Getaways Sequence Diagram**

The Sequence Diagram is a graphical representation of how messages flow between objects arranged in time (“Sequence Diagram”, 2023). On the Client-side of the architecture, messages begin to flow with a user initiating an action on the website, e.g., clicking a link or typing a page’s address. These actions are intercepted by the *Routes* (router methods) and directed to the view.

The view is able to interact with the *Controller*, which are listed in app.js, to obtain the data needed for the view. The browser is routed to the correct controller via a page’s web address.

The *Controller* requests, makes a call to the appropriate service, to retrieve the data, and any associated model, from the server side. The *Controller* then creates and HTML page and returns the data to be viewed.

As previously stated, and shown in **Figure 2**, not all of the messages stay within the Client-side. If the browser makes request for data stored on the server these are handled by the *HTTP Client*.

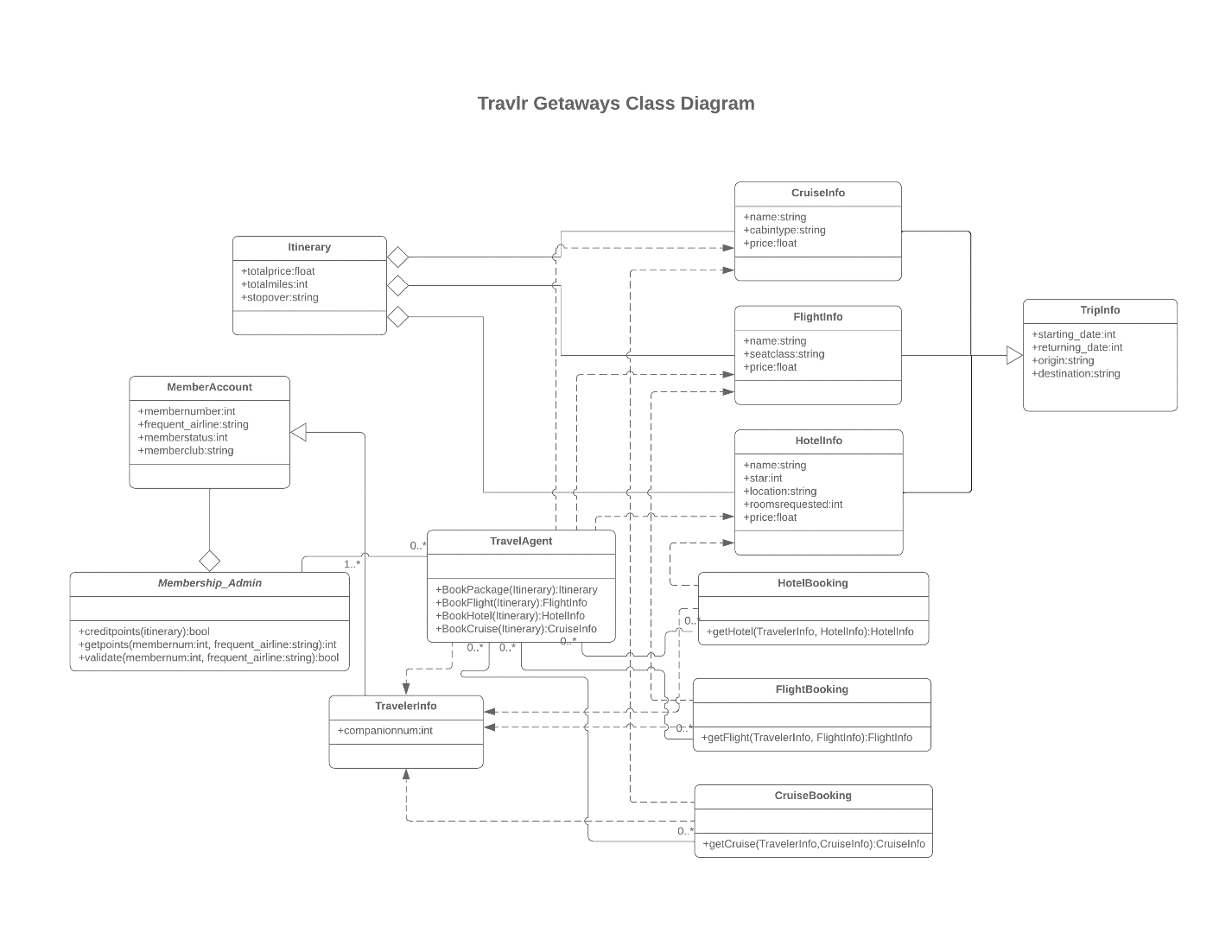
The *HTTP Client* sends a request via a second controller to the appropriate *Route* (server router class). This interface between the Client- and Server-side is done using a RESTful API. For example, if the page is requesting all travel packages this is routed through a different controller than if the user is requesting information on a single package.

The Server-side *Controller* does not talk directly to MongoDB but uses a middleware layer called Mongoose that abstracts and applies Object Document Mapping (ODM). Mongoose acts as the MongoDB driver and passes commands to MongoDB where they are processed.

Once MongoDB processes the requests the data, formatted as a JSON, is returned through the controllers and routers back to the page, *View*, that initiated the request.

### Class Diagram

**Figure 3**, below, illustrates the classes used in the Travlr Getaways website and their interactions.



**Figure 3:Travlr Getaways Class Diagram**

*CruiseInfo*, *FlightInfo*, and *HotelInfo* are all aggregations of *Itinerary.* This means that *Itinerary*, the parent class, can contain multiples of the child classes. This makes since as these children would be pieced together to make the trip itinerary. As these classes are associated through aggregation, as opposed to composition, they all may have different lifetimes. Moreover, any of the child classes can be instantiated by themselves, i.e., a *HotelInfo* object may be created without an *Itinerary* object but an *Itinerary* object can never be created without *CruiseInfo*, *FlightInfo*, or *HotelInfo.*

*CruiseInfo*, *FlightInfo*, and *HotelInfo* are also subclasses of *TripInfo*, the super class. This allows each of the subclasses to inherit the attribute of *TripInfo*.

The *TravelAgent* class has a complex relationship with other classes. It is capable of implementing *CruiseInfo*, *FlightInfo*, *HotelInfo,* or *TravelerInfo.* It is also associated with *HotelBooking*, *FlightBooking*, and *CruiseBooking*. **Figure 3** indicates there may be 0 or multiple of any of these classes. Which makes since not every trip will be made with a travel agent nor must they have multiple hotels, flights, or cruises they are wanting to booking.

*HotelBooking*, *FlightBooking*, and *CruiseBooking* may also implement *CruiseInfo*, *FlightInfo*, *HotelInfo,* or *TravelerInfo*. Moreover, these *Booking* classes contain the methods needed for an agent to make the actual bookings.

*TravelerInfo* is the subclass of the super class *MemberAccount*.

*MemberAccount* is an aggregation of *Membership\_Admin*. This allows the MemberAccount to exist without *Membership\_Admin* but not to allow *Membership\_Admin* to exist without *MemberAccount*.

*Member\_Admin* is also associated to the *TravelAgent* class. In this association there must be at least one *Member\_Admin* object but not necessarily a *TravelAgent* object.

## API Endpoints

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Purpose** | **URL** | **Notes** |
| GET | Retrieve all trips | /api/trips | Retrieves, via the GET verb, all of the entries within the MongoDB. Data is returned as a JSON |
| GET | Retrieve a single trip by its code | /api/trips/:tripcode | Retrieves, via the GET verb, a single trip from the MongoDB using its “tripcode” |
| DELETE | Delete a trip within the database | /api/trips/:tripcode | Deletes a single trip, using the delete methods on the Admin site, identified using its “tripcode |
| POST | Add a trip to the database | /api/trips | Adds a trip to the database via the POST verb |
| PUT | Update an existing trip within the database | /api/trips/:tripcode | Updates a single trip, using the edit methods on the Admin site, identified using its “tripcode” |

## The Admin Site User Interface

The Admin site is designed as a Single Page Application (SPA). Using an SPA for the administration tasks allows admins to add, remove, or update site content easily. These features allow for a cost-effective application with reduced on-going support costs.

### SPA Framework vs. Express

The SPA framework has a project structure designed to organize code for a modular, component-based architecture. The project structure for an Express HTML customer-facing page is typically simpler compared to an Angular application.

The Angular structure has this form:

1. sac: contains the code to make the SPA function
   * app: The app folder holds the main application components, services, and modules.
     + components: individual reusable components, i.e., trip-card, delete-trip, etc.
     + services: services for data handling and business logic
     + models: contains the data model
   * assets: contains static files like images
   * index.html: main HTML file that acts as the entry point of the application
2. node\_modules: dependencies installed via npm
3. package.json: project's dependencies and scripts

The Express HTML project has this structure:

1. public: folder containing static assets that will be served by Express
2. views: folder holding the server-side templates used to generate HTML pages
3. routes: route handlers and business logic
4. app.js or index.js: main entry point of the Express application
5. package.json: Defines the project's dependencies and scripts

The key differences are that Angular is primarily a client-side framework, while Express is a server-side framework. Angular handles the presentation and user interactions on the client-side, whereas Express deals with server-side logic and rendering of HTML pages.

In an Angular project, the majority of the code resides in the src folder, while an Express project typically has a simpler structure with folders like public, views, and routes.

Angular projects follow a component-based architecture, while Express projects typically follow a more traditional server-side MVC (Model-View-Controller) pattern.

Angular projects have extensive configuration files like angular.json and tsconfig.json for managing the build and development process. Express projects usually have a simpler configuration.

Angular projects have a dependency on node modules, but Express projects also have dependencies, though they are typically more focused on server-side functionality.

One of the main advantages of the SPA is a more responsive UI as the entire page never needs to be reloaded. In our case that will be advantageous once there are more trips in the db. SPAs can also reduce the server load as they offload some processing tasks to the client-side. The server primarily focuses on providing data through APIs, which can lead to better server performance. SPAs can navigate between different views without waiting for full page reloads, resulting in a more fluid and efficient interaction.

As far as disadvantages the initial load time of SPAs can be slower than traditional multi-page applications, as the entire application code, frameworks, and dependencies need to be loaded upfront. Search engine optimization can be more complex in SPAs because search engines typically rely on static HTML content for indexing, but this is not an issue since only our ADMIN site, and not the landing page, is an SPA. The SPAs code was more complex. Handling client-side routing, state management, and asynchronous data fetching was a new challenge.

### Testing the Admin Site

To test the Admin site, the seeded database, **Figure 4**, was taken as the baseline.

A screenshot of a computer

Description automatically generated

**Figure 4: SPA Site Seeded with Three Trips**

From this state the *Add* button was clicked. A successful test will add the test trip to the database and a new trip-card will be created reflecting the user entered data, shown in **Figure 5**.

A screenshot of a computer

Description automatically generated

**Figure 5: SPA Site Showing the Addition of a new Trip after Clicking Add; MongoDB also Showing the Same Data**

The next test was to validate the *Delete* methods. To perform this test a sacrificial test was created via the *Add* method, **Figure 6**. This added trip was then deleted, **Figure 7**.

A screenshot of a computer

Description automatically generated

**Figure 6: Trip-cards with the Sacrificial Trip**

A screenshot of a computer

Description automatically generated

**Figure 7: Trip-cards after Clicking Delete on the Sacrificial Trip; API Response**

After successfully deleting a trip, the next method to exercise is the *Edit* functionality. To test this functionality a record was added so that it could be edited, **Figure 8**. The name of the trip, *EDIT ME*, is then changed to *I HAVE EDITED THIS*, **Figures 9** and **10**.

A screenshot of a computer

Description automatically generated

**Figure 8: Trip Added, EDIT ME, to Validate Edit Functionality**

A computer screen with a white background

Description automatically generated

**Figure 9: Edit Form Showing the Data After Clicking Edit**

A screenshot of a computer

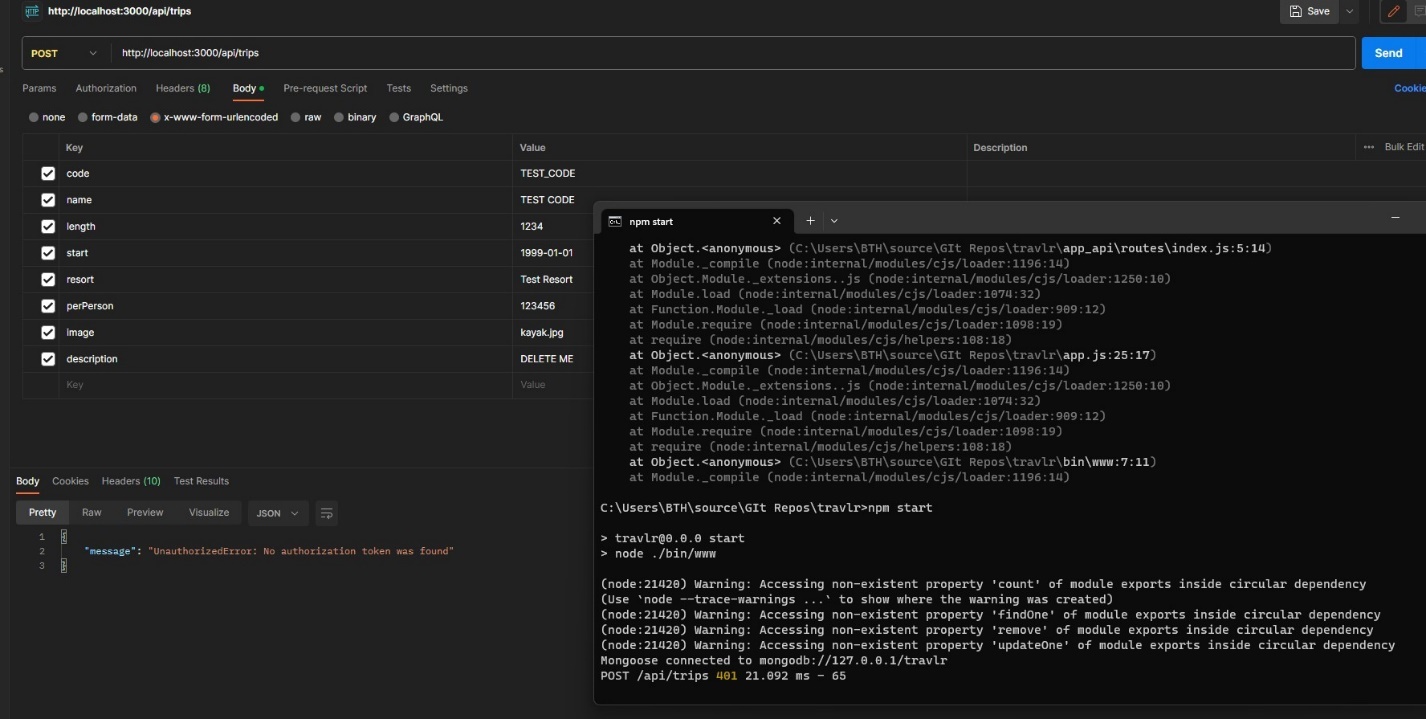
Description automatically generated

**Figure 10: Trip Added in Figure 8 with the Name Changed**

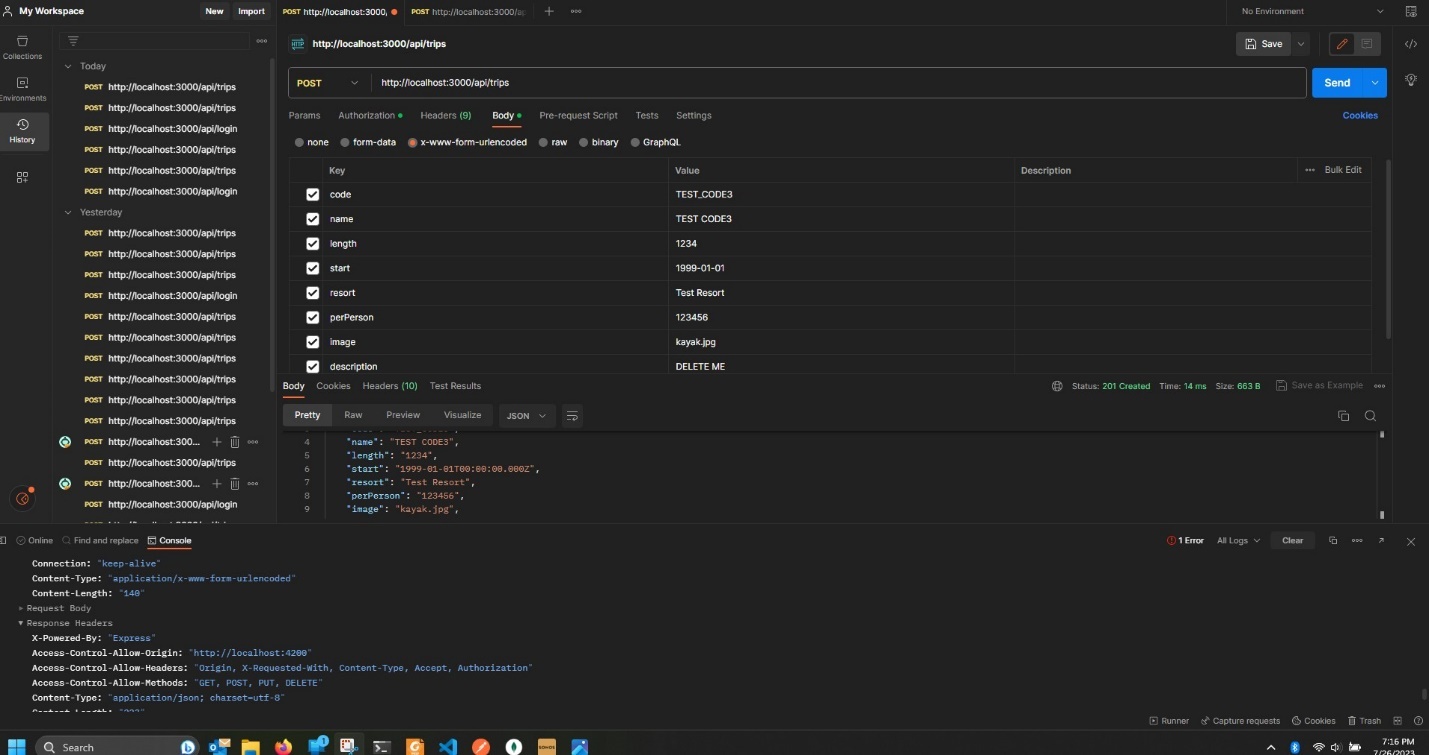
### Admin Security

Because the Admin site can add, delete, and edit the trip entries and is available through the internet, security through anonymity is not enough protection. Therefore, a user must login before they are allowed to make changes. Once a user is logged in a hashed token is saved to local storage. The API expects this token in the headers or any POST, PUT, DELETE calls will be rejected.

Testing of the security components was carried out in two phases. Phase one testing isolated the API by using Postman. **Figure 11** shows the API rejecting any POST calls without the token. **Figure 12** shows the API accepting the POST after passing the token.



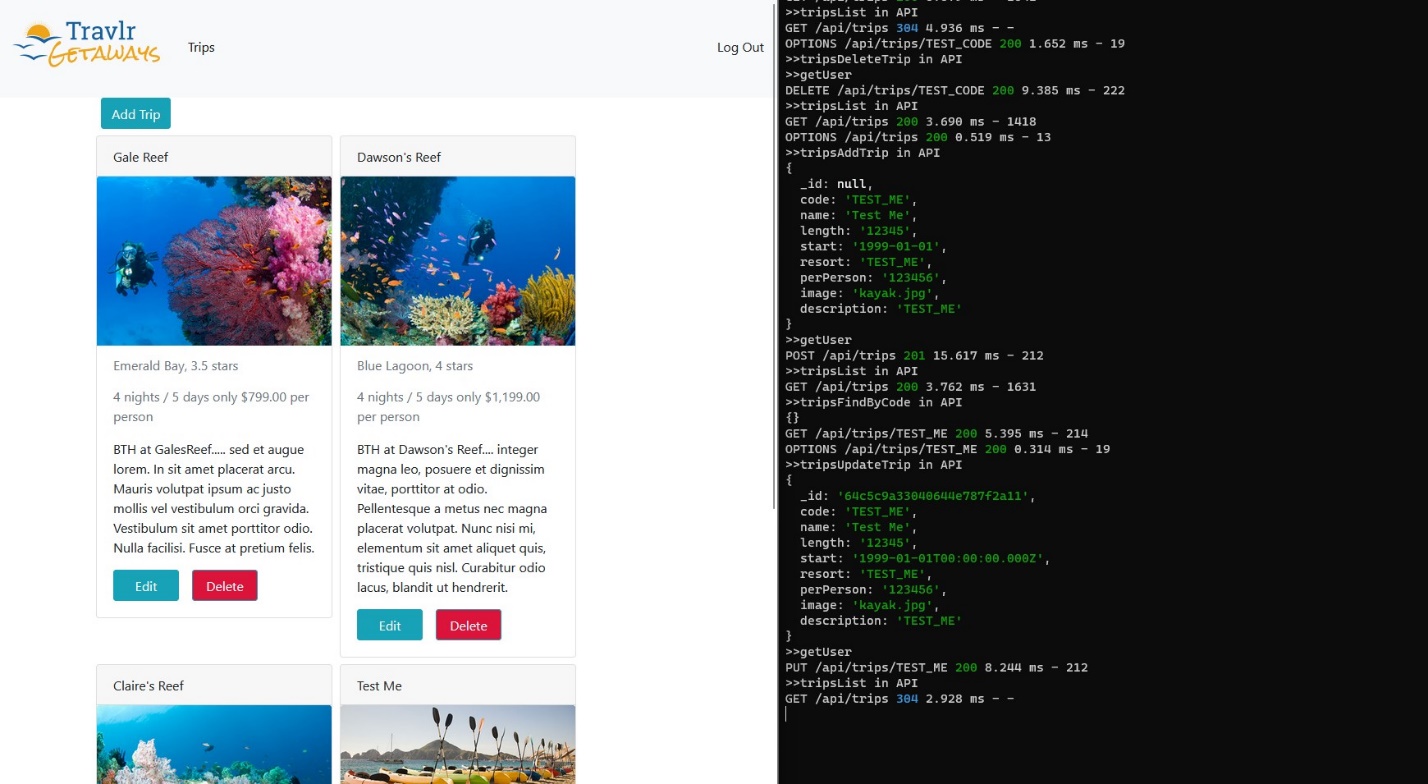
**Figure 11: Postman Testing the API with a POST Command, no Token Passed (Note the 401 Status Returned)**



**Figure 12: Postman Testing the API with a POST Command, Token Passed (Note the 201 Status Returned)**

Once it was confirmed that the API was only allowing some commands if the appropriate token was passed, the same testing was conducted with the Admin’s frontend.

**Figure 13** shows the status in the console after editing a trip.



**Figure 13: Admin Site and Console Output after Logging In (Note the 200 Status on PUT)**

If a user logs out then the Add, Edit, and Delete buttons are removed and only become visible again upon logging back into the site. **Figure 14** shows the Admin site after logging out.

A screenshot of a computer

Description automatically generated

**Figure 14: Admin Site After a User Logs Out (Notice Add, Edit, Delete buttons are hidden).**

## References

AWS. (n.d.) *What is RESTful API*? https://aws.amazon.com/what-is/restful-api

BasuMallick, C. (2022, October 18). *Pros and cons of single-page applications*. https://www.spiceworks.com/tech/devops/articles/what-is-single-page-application

Brehm, S. (2013, Nov. 11). *Isomorphic JavaScript: The future of web apps.* The Airbnb Tech Blog - Medium. https://medium.com/airbnb-engineering/isomorphic-javascript-the-future-of-web-apps-10882b7a2ebc

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

Kaur, A. (2023). *What is single page application (SPA)? pros and cons with examples*. https://www.netsolutions.com/insights/single-page-application

Knowledge Nile (n.d.) U*nderstanding the pros and cons of MongoDB.* https://www.knowledgenile.com/

blogs/pros-and-cons-of-mongodb/#MongoDBHighMemoryUsage

Sequence Diagram. (2023, May 9). In *Wikipedia*. https://en.wikipedia.org/w/index.php?

title=Sequence\_diagram&oldid=1153944336

Vivim. (2019, April 11). *6 benefits of using mean stack development.* https://vivim.net/blog/6-benefits-of-using-mean-stack-development