

Travlr Getaways 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 | 03/19/2023 | Gabriel Sanchez | Updated the Executive Summary, Design Constraint, and System Architecture View sections. |
| 2.0 | 04/02/2023 | Gabriel Sanchez | Updated Sequence Diagram, Class Diagram, and API Endpoints Table |
| 3.0 | 04/16/2023 | Gabriel Sanchez | Updated Sequence Diagram, and the User interface Section. |

## 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)

We have a client Travlr Getaways, we are tasked with creating a full stack web application taking advantage of the Mean Stack which include MongoDB, Express. By creating the application through MEAN stack we are able to incorporate multiple frameworks that allows for the administration ability to improve the overall speed of different procedures, and this provides the framework in order to provide the required aspects of the Travlr getaways, it would allow for the creation of accounts, the searching of the travel packages, and the creation of orders, the tracking of order history.

Single-Page applications or SPA will be used for the administrators of the travlr site this is because SPA’s are client -side rendered and will need to fetch data after they load, essentially allows for the browser to download the code but not the data, therefore it will need to request the data from the server after it loads and this is done through the RESTful API we will also create. The database will be created using MongoDB this is required in order to store information and to store user profiles, with AngularJS client for the frontend as it uses HTML to define the user interface and ExpressJs for the backend as it provides plugins and middleware packages that aid in more efficient web development and it being a service side framework build for Node.js.

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

The constraints would include at first time in order to meet the requirements by Travlr Getaways optimal use of time is ideal as they will need to adequate enough time to create all the aspects of the application, and then have adequate time for proper app testing which is highly influence by the overall time for the project, more time to test usually provides a better initial end product, there is also the cost limitations the application needs to provide the expected requirements while also staying withing the allocated budged this will have an impact on the resources used and the labor cost and the speed of the program.

There could also be some scope limitations whether due to the before mentioned time or budget constraint therefore there needs to be a properly defined scope that is clearly explained and outlined for all shareholders this will ensure that the application will not only be to the specifications but that al resources are maximized for the expected use cases, this provides the necessary clear scope definition in order to properly create an adequate app and in order to plan accordingly for the time necessary for both creation, implementation, security , and testing.

## [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).

We have two ports illustrated as a small square, as we can see above they are one at the server end and one at the client end, this indicates that the server component uses a port in order to delegate it’s Api to the client class.

We see through the use of required interface icons which are the half circles and the full circles which are provided interface, with this in mind when we look at the above diagram in green we can see the relationship the client section component is connected by a line to the web browser and the traveler portfolio component, this tells us that a user interface is required in order to have the web browser and traveler portfolio interact, with the same logic we see that the Traveler portfolio is connected to the graphic library and as we see this indicates that the traveler portfolio relies on the graphic library.

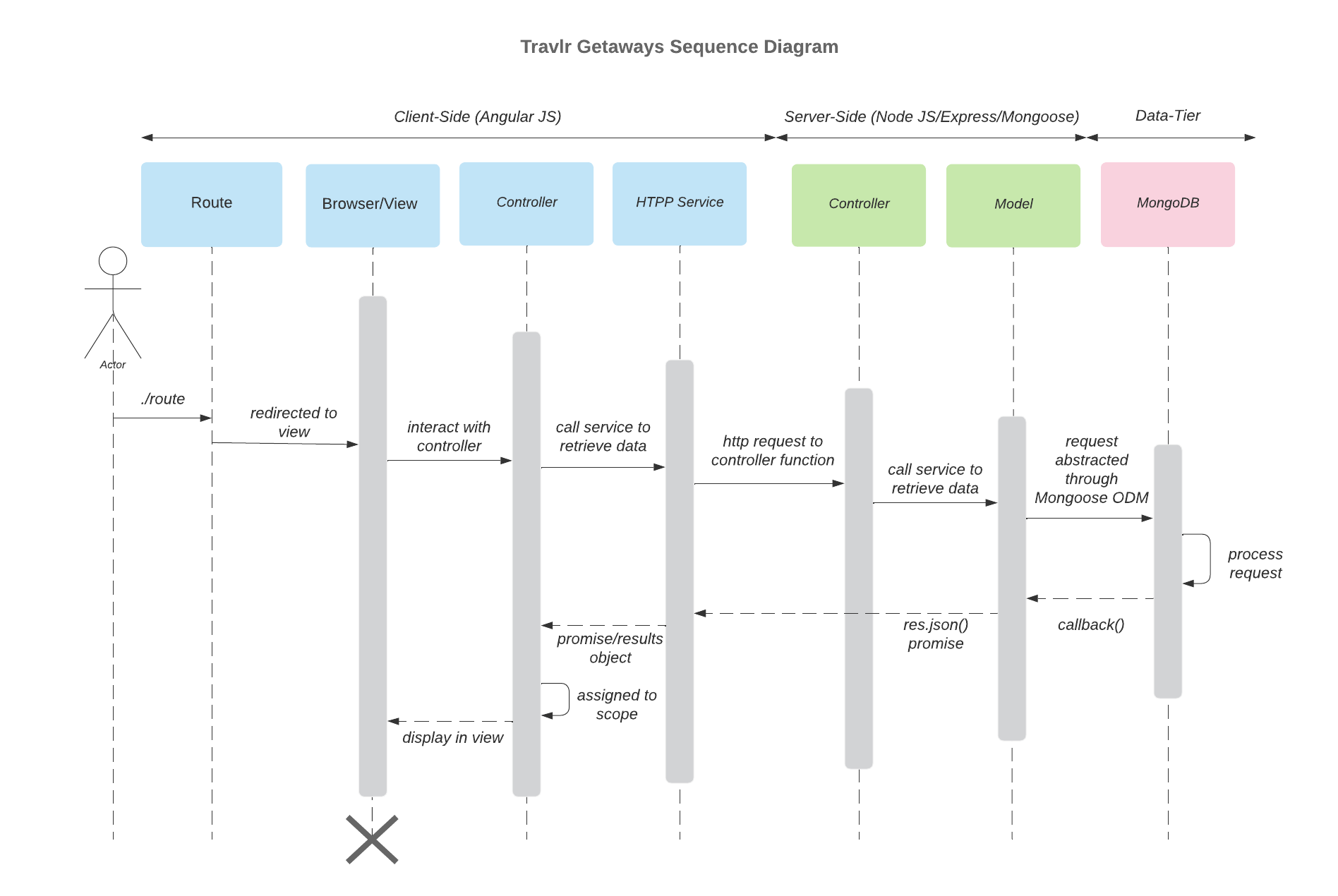
For the one in blue we see that there is a lien connecting the authentication server and server location

we have a line connecting the MongoDb component database with the traveler portfolio component and then the server section is connected to the Mongoose ODM which by the symbols provides the interface that the server session requires, and on the other side of the server session we see that it is connected to the travelers database and this in turn provides the interface that server session requires.

Now when looking at the final yellow diagram we see that MongoDb is connected on one side to the traveler portfolio and the traveler portfolio requires the interface that the MongoDb provides and on the other side of the MongoDB component we see that the MongooseODM requires the interface provided by the MongoDB.

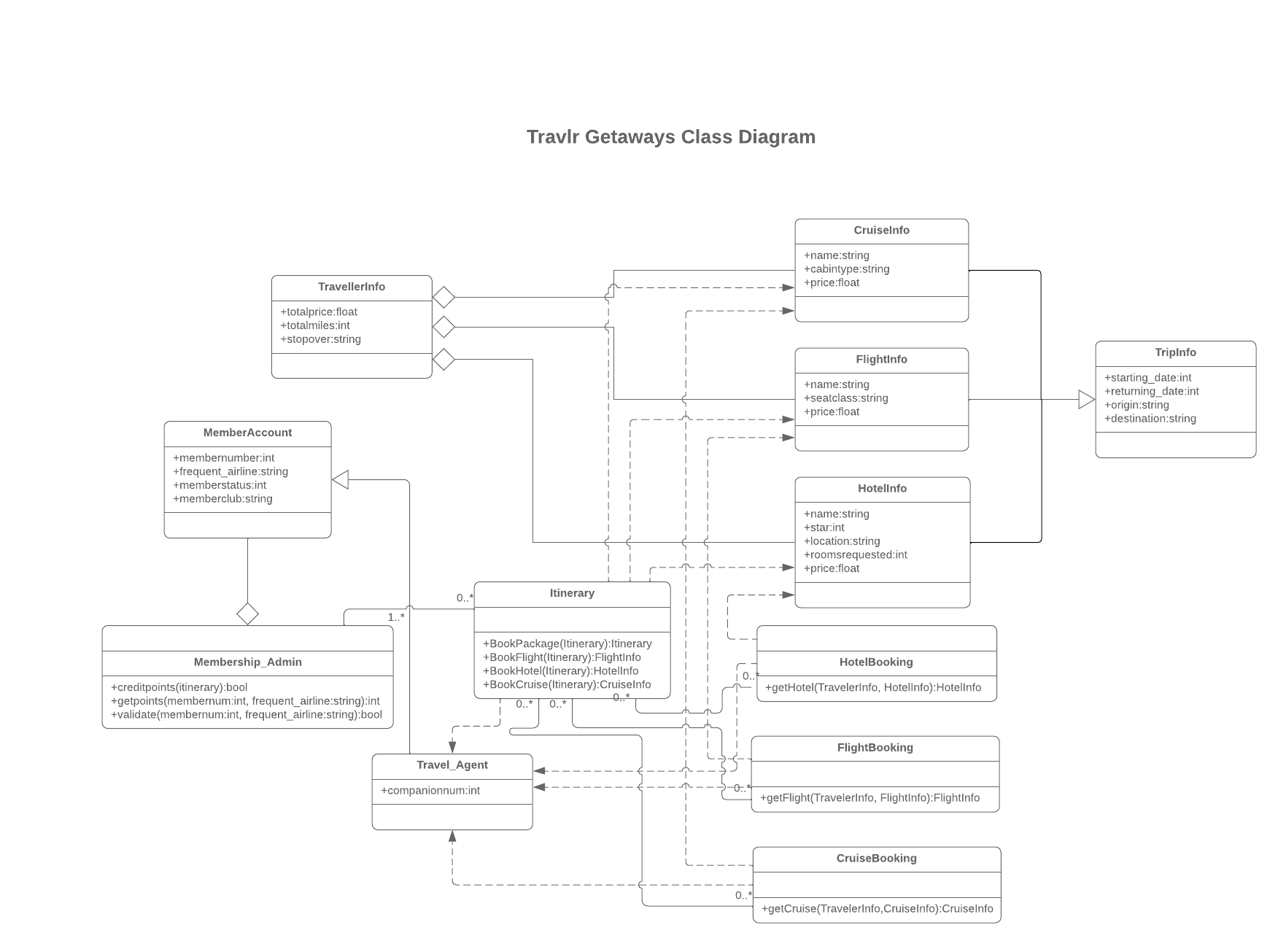
### 

### Sequence Diagram



The Sequence Diagram displayed above showcases the interactions involved when a user/actor interacts with the applications, first the user would type the route(URL) on a browser, then the system would take them to the correct views that’s associated with the selected Route, the system would then interact with the user by accepting the users inputs and submit the data back to the server, then an HPTP request is sent to the controller, the browser then helps translate user requests and retrieves the appropriate data, MongoDB is used to store the data and process the requests and filter them back when displayed on the screen for the user.

## Class Diagram



In the Class Diagram above we see a depiction of many of the classes needed in order for our application to perform the required tasks for example with the TravelerInfo, CruiseInfo, FlightInfo, HotelInfo, TripInfo, these classes have a similar setup In which they create a set of constructors I which they store information, and these will be needed for the users as they will need to have their information like the price of certain things like the flight, hotel, and the important dates for the different trips, we also have classes like HotelBooking, FlightBooking, CruiseBooking, these utilize the information from the information classes in order to interact with the user and allow them to book the services for their future trip as their name implies for their role whether it be for booking hotel/flight/cruise etc.,

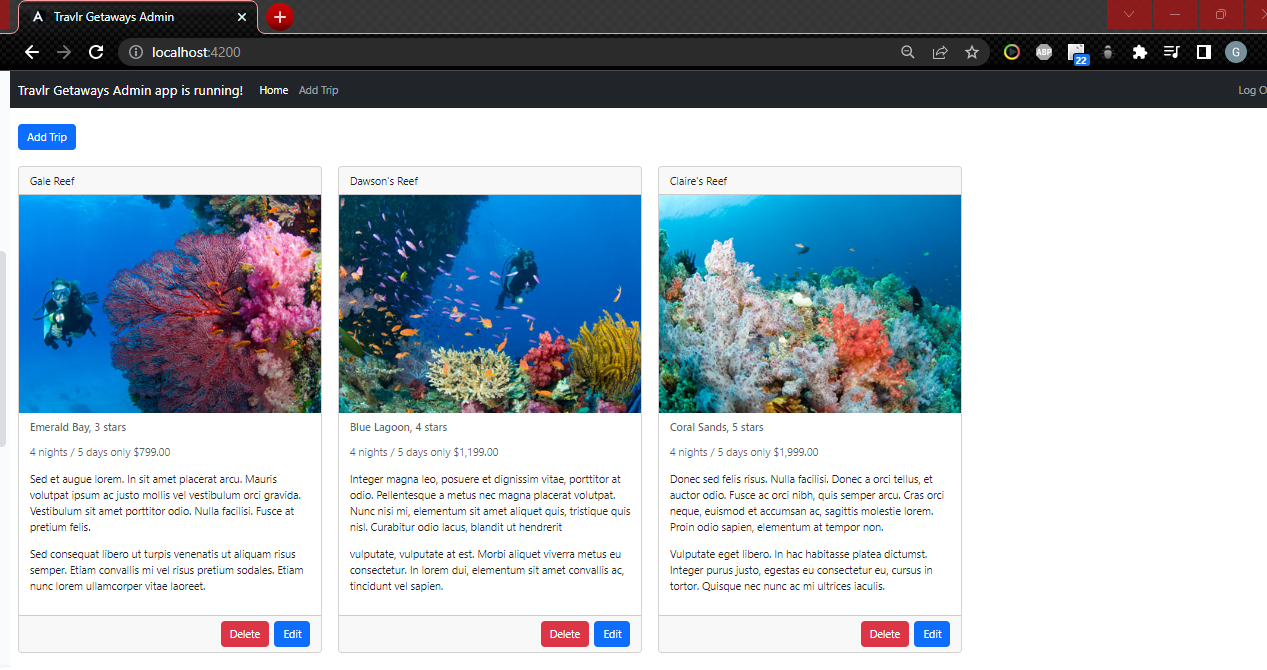
there also needs to be a way to store information and display the information this is done with the MemberAccount and the Itinerary classes, and then we also have a class for a traveler aid with the agent in order to aid in their process of purchasing a package TravelAgent class, and then for the administrative aspects we have the Membership\_admin this one allows for the creating and getting of points as well as validating membership.

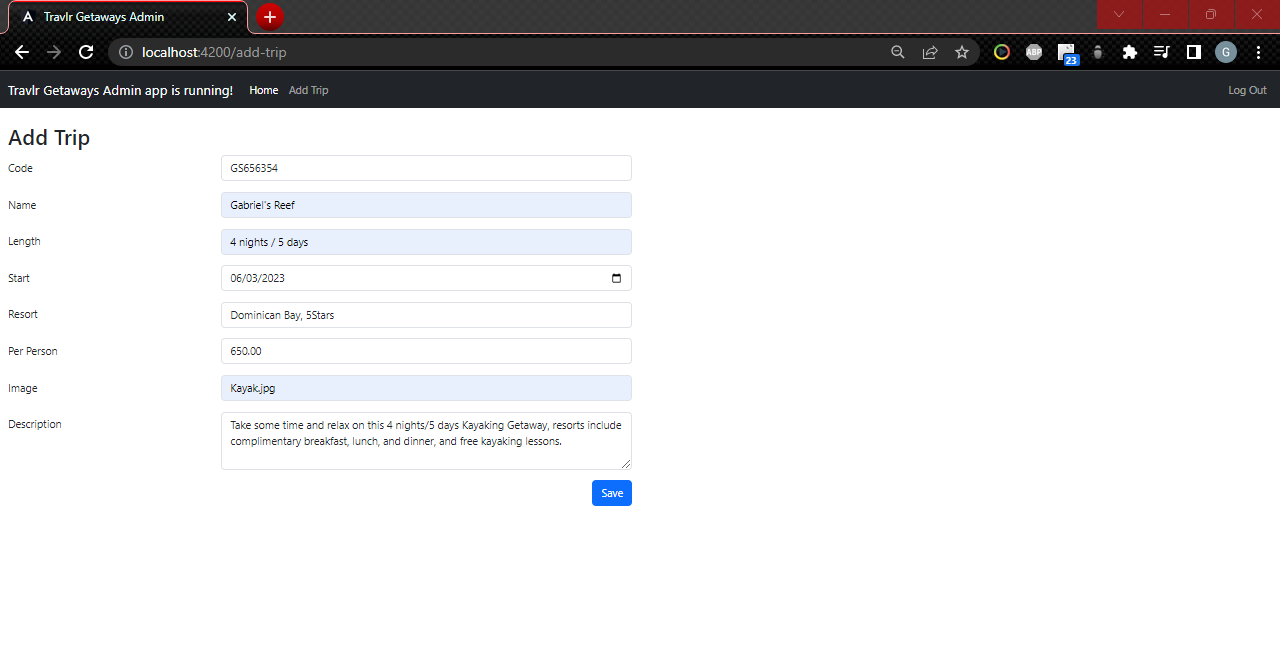
## [API](#_heading=h.2jxsxqh) Endpoints

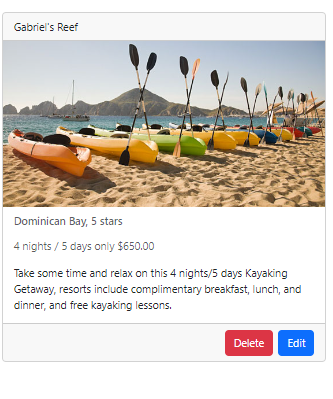
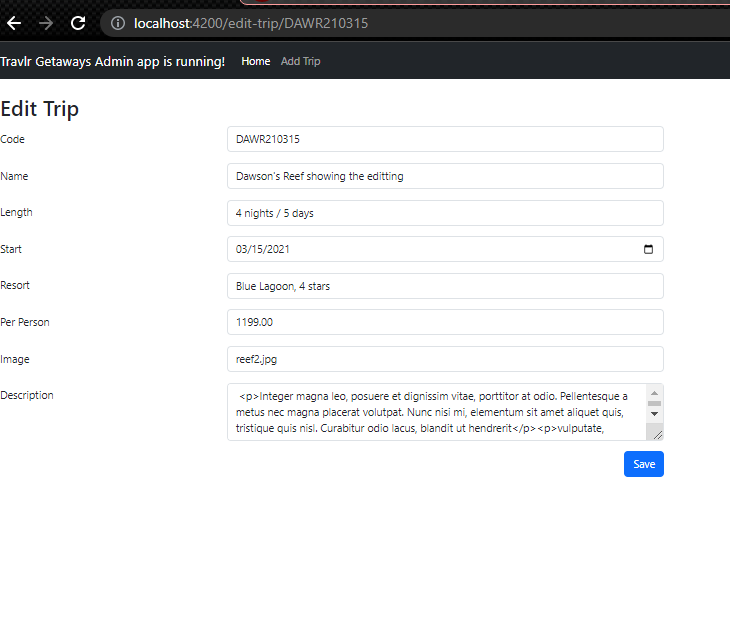
| **Method** | **Purpose** | **URL** | **Notes** |
| --- | --- | --- | --- |
| **GET** | <Retrieve list of things> | </api/things> | <Returns all active things> |
| **GET** | <Retrieve single thing> | </api/things/:thingId> | <Returns single thing instance, identified by the thing ID passed on the request URL> |
| **GET** | <Returns the resource> | </api/trips> | <returns all active things> |
| **POST** | <Creates a list of things thing> | </api/trips> | <Creates a list of things>. |
| **POST** | Creates a single thing | </api/trips/tripsID> | <Creates single thing instance, identified by the trips ID passed on the request URL> |
| **DELETE** | <Deletes a list of things> | </api/trips> | <Deletes a list of things> |
| **DELETE** | <Deletes a resource > | </api/trips/tripsID> | <Creates a single thing instance, identified by the trips ID passed on the request URL> |
| **PUT** | <Update/replace a list of resources> | </api/trips> | <This will not only update all the records but will also override the data.> |
| **PUT** | <Update/replace a single resource> | </api/trips/tripsID> | <updates/replaces a single thing instance, identified by the trips ID passed on the request URL> |

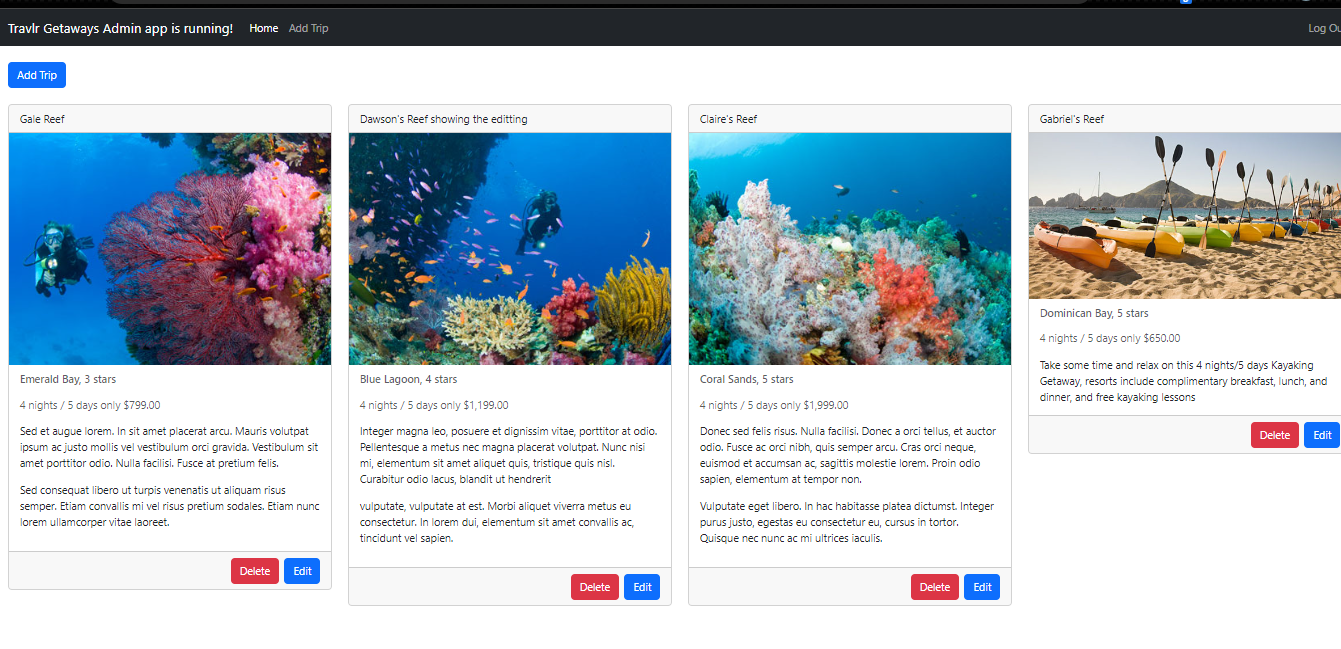
## 

## The User Interface







AngularJS client is for the frontend as it uses HTML to define the user interface and it’s commonly used for the frontend abilities and helps create a more dynamic approach by creating scalable web applications while ExpressJs is used for the backend as it provides plugins and middleware packages that aid in more efficient web development and it being a server side framework build for Node.js.

The advantages of SPA when compared to Multi-page Applications (MPA) is they can increase the speed, lower the amount of bandwidth required and lower the load on the server, this is due to SPA executing most of the code on the clients machine as it works like a desktop application but in the users browser, this allows the user to receive instant feedback, some of the disadvantages would be in terms of poor link sharing and security, a benefit of SPA is the are able to be paired with many different frameworks that provide many functions like routing navigation, security, dependency injection for various components, and provide integration options with other common web technologies.

We could test routes using tools like Postman in which we would send HTTP requests and process HTTP responses to aid in our testing, for example to test DELETE we would hit Get /api/trips endpoints this would give us a list of the trips in our database, we could then use the trip codes returned in a request in order to DELETE them, and we could expect codes like “200 Ok response”, we could also test using tools like chrome developer tools, and we could check when testing in order to receive the right functionality as most of the errors would be network errors, request errors, and if a trip can’t be found we have error codes to signal like the “404 NOT FOUND” ERROR.