



Travlr Getaways  
**CS 465 Project Software Design Document**  
Version 1.5

## Table of Contents

<b>CS 465 Project Software Design Document</b>	<b>1</b>
Table of Contents	2
Document Revision History	2
Executive Summary	3
Design Constraints	3
System Architecture View	5
Component Diagram	5
Sequence Diagram	6
Class Diagram	6
API Endpoints	9
The User Interface	10

## Document Revision History

Version	Date	Author	Comments
1.0	01/22/2024	Christian Henshaw	Updated Executive Summary, Design Constraints, and System Architecture View sections.
1.1	02/07/2024	Christian Henshaw	Updated System Architecture View (Sequence Diagram and Class Diagram) and API Endpoints sections.
1.2	02/08/2024	Christian Henshaw	Added additional Design Constraints relevant to the MEAN stack.
1.3	02/12/2024	Christian Henshaw	Updated the Server-Side Section in the Sequence Diagram.
1.4	02/13/2024	Christian Henshaw	Added additional API Endpoints.
1.5	02/21/2024	Christian Henshaw	Updated User Interface sections. Included various figures to depict the user experience.

## **Executive Summary**

The client, Travlr Getaways, requires a travel booking website that will service customers and administrators in separate ways. The primary purpose of the travel booking website is to enable customers to book available travel packages. To do so, customers must be able to create an account, search for travel packages by various search factors, book reservations, and regularly visit the website for information and itineraries. Furthermore, administrators must have access to an admin-only site to maintain the customer base, available travel packages, and pricing.

The appropriate architecture of the web application, based on these software requirements, is the MongoDB, Express, Angular, and Node.js (MEAN) stack while leveraging the Model-View-Controller (MVC) architecture. A notable benefit of utilizing the MEAN stack for this project is the use of JavaScript throughout the stack, easing development. MongoDB will serve as the database for the travel booking website to optimize the storage and retrieval of pertinent information such as user accounts, travel package details, and customer reservations. Express is the framework for the travel booking website, which easily redirects incoming Uniform Resource Locator (URL) requests to associated code and abstracts development difficulties by performing common tasks. Angular forms the front-end framework to create the website interface and offload processing and logic to the customer's browser. Finally, Node.js is the software platform enabling the creation of the webserver and increasing development control over implementation. Regarding the MVC, Angular represents the view, Express and Node.js represent the controller, and MongoDB represents the model. Customers and administrators will interact directly with Angular, which will pass requests to Express that leverage applicable code blocks to interact with the stored information in MongoDB. Upon obtaining the necessary information from MongoDB, Express will modify the interface and information in Angular, which will be displayed to the customer.

The customer-facing side of the travel booking website is comprised of a collection of screens containing navigation hyperlinks and relevant information. The home screen serves as the landing page for customers and will provide preliminary information on the purpose of the company, important blogs, and testimonials. The travel screen will display destinations and supplemental travel information. Similarly, the room screen displays the housing and amenities available in a travel package. The meals screen describes food accommodations that are available in a travel package. The news screen provides customers with critical news and insights on travel information. The about screen serves to increase the visibility and transparency of the company, staff, and communities. Finally, the contact screen provides contact information and a way to directly communicate with the company. Alternatively, the administrator single-page application (SPA) enables administrators to create reservations, adjust travel pricing, maintain customer records, and navigate the site through hyperlinks.

## **Design Constraints**

There are some design constraints that must be considered in developing the Travlr Getaways website application. Travlr Getaways' timeline for the website to be completed is a critical factor that must be considered throughout development. Adequate time must be allocated to appropriately implement the necessary features and thoroughly test the functionality to ensure

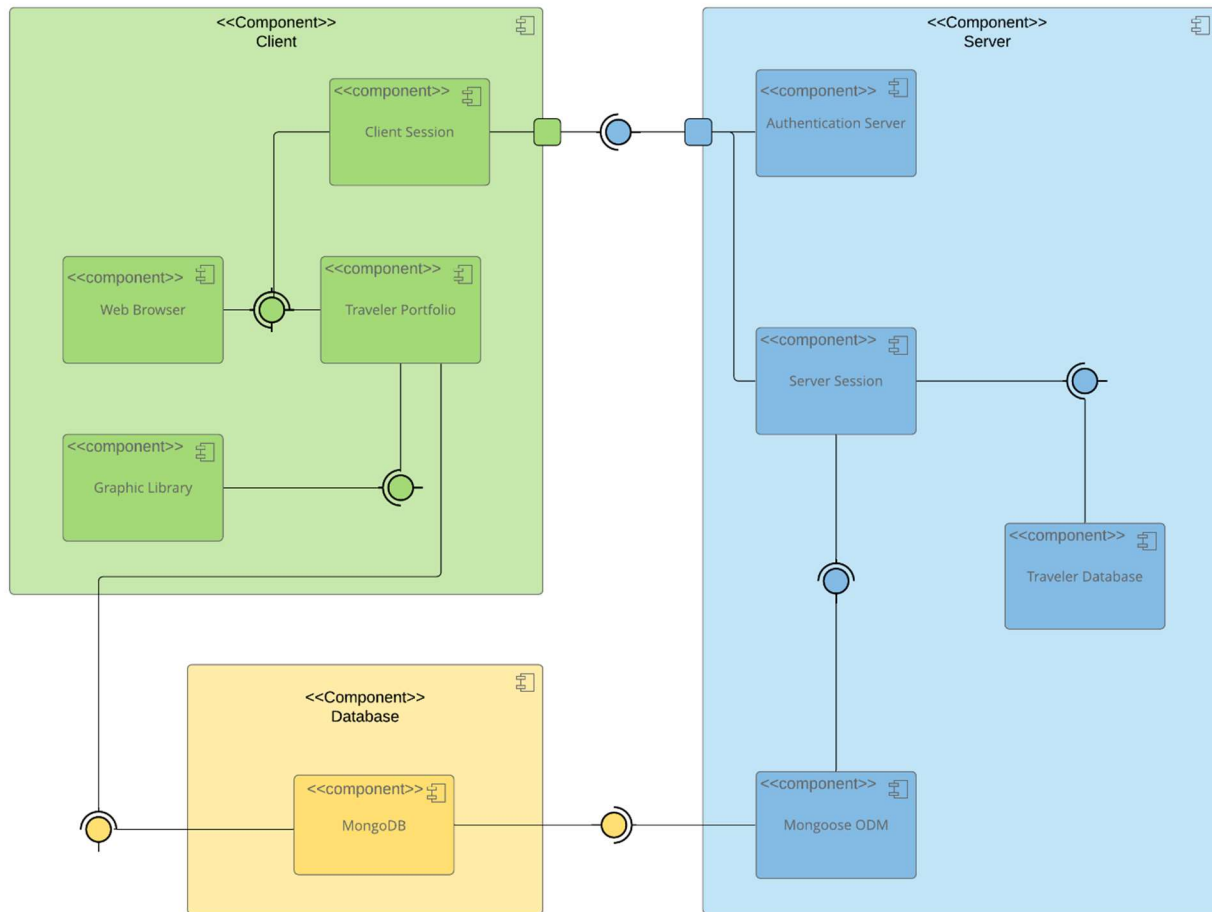
the software requirements have been met. Additionally, the resources available to the development staff should be evaluated. This involves the number of personnel actively involved in the development process, expertise in development tools, and time committed to the project. A critical design constraint is the requirement to comply with government regulations if the travel booking website involves customer payment options. Due to the severity of noncompliance, appropriate time and resources must be allocated to ensure applicable regulations are met. Finally, the tools and software used to develop the travel booking website must be considered. As previously mentioned, this project will leverage the MEAN stack and MVC architecture.

It is also necessary to analyze the implications of the design constraints on application development. The timeline must be appropriate for the software requirements. If the timeline to complete the website is too short, the development staff will likely not be able to fulfill each requirement fully. Subsequently, thorough testing may not be performed, which can lead to uncaught errors, vulnerabilities, or other unintended consequences. Alternatively, if the timeline to complete the website is too long, the client would invest more money into development than necessary. Similarly, the allocated resources must be appropriate based on the timeline and desired software requirements. If not enough resources, such as personnel or a lack of knowledge of utilized tools, are used, some software requirements may not be met sufficiently, or the timeline may be exceeded. Applicable government regulations, such as proper security measures for customer banking information, have severe repercussions for noncompliance. If these are not met, regardless of the reason, the client would be placed at significant risk for fines or other government intervention. Finally, as the tools and software utilized in development have already been defined, it is necessary for development staff to be familiar with them. By doing so, the development process can be made more efficient, software requirements can be more comprehensively implemented, and the product will be significantly enhanced.

Finally, there are additional design constraints inherent in the use of the MEAN stack. Notably, MongoDB is a NoSQL database, which means it is flexible in its requirements for data to be stored. As there is no rigid schema to follow, there is an additional need to sanitize and validate user data before it is introduced into the database. However, MongoDB and NoSQL databases are well suited for regular query use, which is a key staple of Travlr Getaways functionality. Additionally, the MEAN stack uses multiple key software tools that all share the same programming language. While this makes for easier development, troubleshooting, and testing, utilizing each tool for its intended purpose—Angular for the front-end or MongoDB for the database—can significantly expand the amount of code present. This can hinder maintenance efforts by convoluting some functionality and points of communication. Finally, the use of a NoSQL database can introduce significant difficulties in refactoring to an SQL database should the development requirements change.

## System Architecture View

### Component Diagram



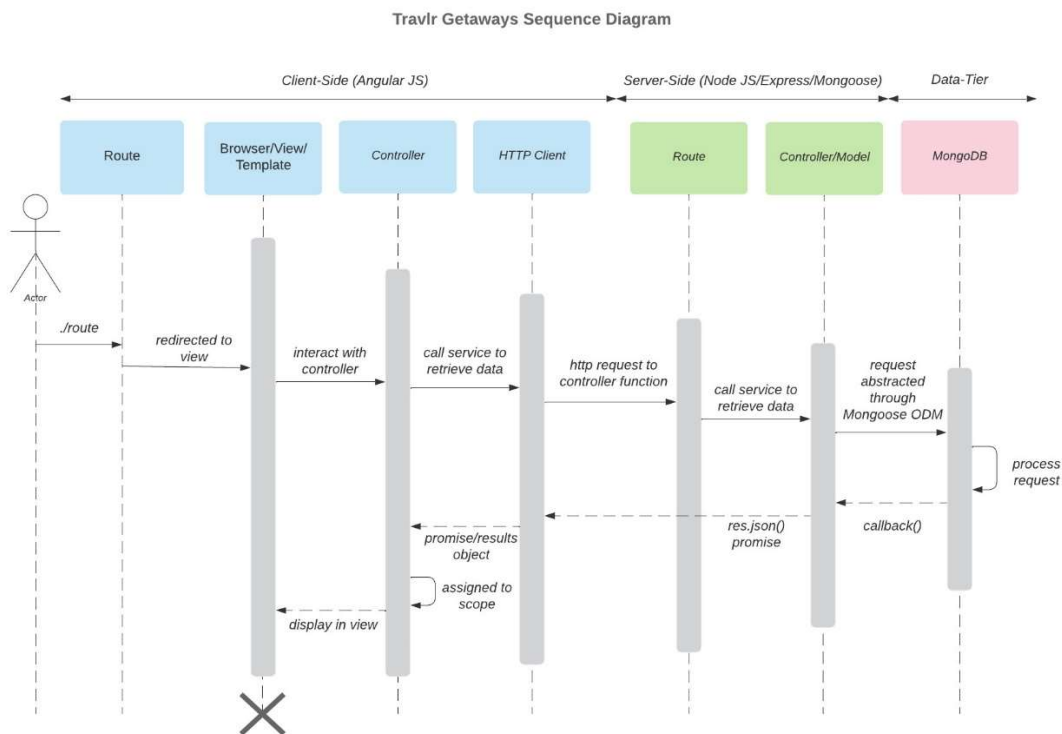
The overall system architecture of the web application is depicted in the above component diagram. The system architecture of the web application is broken down into three main components, comprised of subcomponents and various channels to communicate. The server component is comprised of the authentication server, server session, traveler database, and Mongoose ODM subcomponents. The database component is comprised of the MongoDB subcomponent. The client component is comprised of the client session, web browser, traveler portfolio, and graphic library subcomponents. The server and client component edges each have a port, which enables a separate interface to communicate between the two components.

Much of the data originates from the MongoDB subcomponent in the database component. MongoDB data flows to the server component from the Mongoose ODM subcomponent. This data and data from the traveler database subcomponent are sent to the server session subcomponent. The server session and authentication server subcomponents pass data to the client session subcomponent through the previously mentioned ports. Additionally, the MongoDB subcomponent passes data to the traveler portfolio subcomponent, which then provides data to the graphic library, web browser, and client session subcomponents. In doing so,

the component diagram describes where data originates from and how each subcomponent is connected.

While each of these components and subcomponents serves a unique purpose, there are a few key and significant subcomponents that are essential to the web application. These include the MongoDB, server session, authentication server, client session, and traveler portfolio subcomponents. As previously mentioned, MongoDB supplies much of the data throughout the web application and communicates with the server session indirectly and the traveler portfolio directly. The traveler portfolio contains key information about the client, or user, which is then passed to the client session. The client session directly communicates with the server session. In doing so, MongoDB provides a significant amount of data to the server and client components, which enables the user to perform tasks, schedule reservations, and communicate with the company, to name a few, through the client session. While the other subcomponents provide supplemental functionality or abstract complex functionality, these subcomponents are essential for the web application to function and to enable the customer to accomplish their goals.

## Sequence Diagram

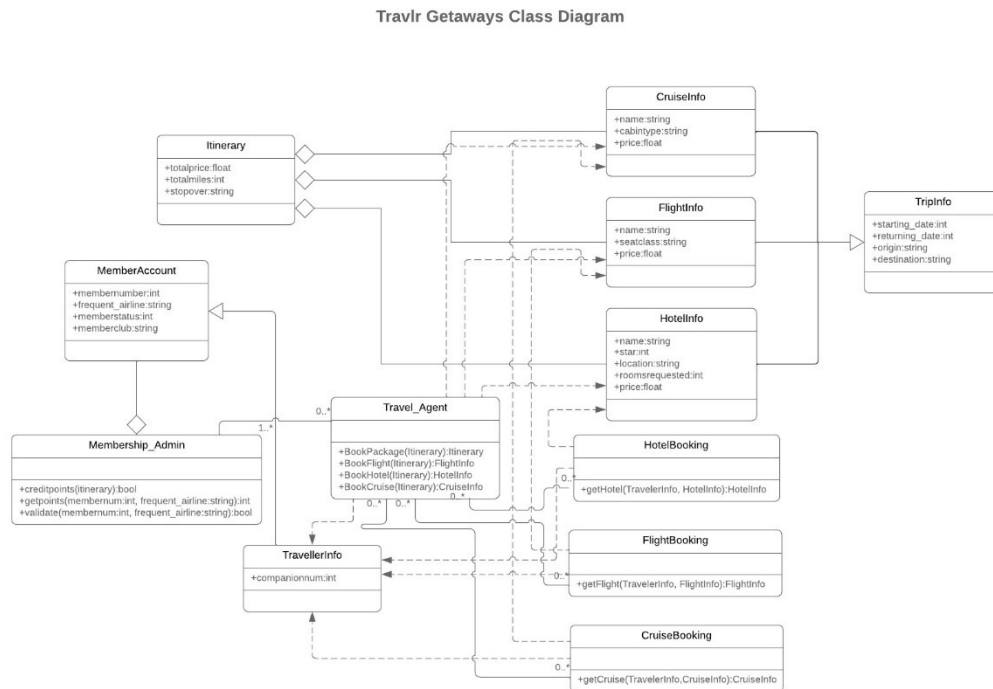


Above is the Unified Modeling Language (UML) sequence diagram, which demonstrates the sequence of events that will occur when users utilize the Travl'r Getaways website. This process starts with the user inputting a route to navigate to the website. Based on the route provided, the user will be redirected to a given page in the Angular client-side user interface (UI), or view.

Upon user interaction, the view will interact with the associated client-side controller, which will call the Hypertext Transfer Protocol (HTTP) client with a request to retrieve data specified by the user. The HTTP client will perform an HTTP request to the server-side application. The HTTP request will be routed to the applicable server-side controller, which will interact with Mongoose ODM to expose the model and request the data from the database, MongoDB. MongoDB will process the request, and the data will be provided to Mongoose ODM through a callback function. Then, the server-side controller will pass the response as a JavaScript Object Notation (JSON) promise to the HTTP client that originally made the HTTP request on the client side. The HTTP client passes the results of the promise, as a JSON object, to the client-side controller. The client-side controller will assign the data to the applicable scope and update the view. Finally, the view is updated in the user's browser with the requested data displayed.

To further demonstrate this process, it is helpful to examine the sign-in process. In doing so, the user's goals are to navigate to the Travlr Getaways website and login. As before, the user first inputs a route to navigate to the website and is routed to the view pertaining to logging in. The user inputs their login credentials into applicable sections of the UI and presses the submit button. Once the button is pressed, the credentials are given to the client-side controller, which calls the HTTP client to request to verify the user's information. The HTTP client creates an HTTP request to the server-side application to engage with the user information database. The HTTP request is routed to the associated server-side controller that handles login requests. The server-side controller calls Mongoose ODM to expose the MongoDB database, enabling the authentication of the user credentials. MongoDB processes the request, ultimately either authenticating or rejecting the request, and returns data in the callback function to Mongoose ODM. This information is then passed back to the HTTP client in a response as a JSON promise. The HTTP client provides the associated client-side controller with the JSON object from the promise. The client-side controller then assigns the information to the scope and updates the view with information pertaining to authentication or rejection. Finally, the view is displayed to the user and describes the outcome of the login request.

## Class Diagram



Above is the UML class diagram, which describes how the data is modeled, its functionality, and how data is passed throughout the Travlr Getaways website. Information about any reserved trip starts in the TripInfo class, which contains basic trip information such as starting and end dates as well as the origin and ending locations. The CruiseInfo, FlightInfo, and HotelInfo classes inherit from the TripInfo class and implement additional attributes that are relevant to a mode of travel, such as cabin type for a cruise. These three classes also have an aggregation relationship with the Itinerary class. In practical terms, this means that CruiseInfo, FlightInfo, and HotelInfo can each individually form part of the Itinerary class to create a more complex object for a travel package. Furthermore, CruiseInfo, FlightInfo, and HotelInfo are related to and realized by the CruiseBooking, FlightBooking, and HotelBooking classes, respectively. These Booking classes implement a means to obtain specific information from each Info class. Additionally, the Booking classes share zero-to-many relationships with the Travel\_Agent class, which implements functionality to book packages, flights, hotels, and cruises. This means the Travel\_Agent class can support any type or number of bookings. Similarly, CruiseInfo, FlightInfo, and HotelInfo are also realized by the Travel\_Agent class, which is used to set data in the Info classes.

Information about any user starts in the MemberAccount class, which contains basic user information such as member number, frequent airline, member status, and member clubs. The TravelerInfo class inherits from the MemberAccount class and implements the companion

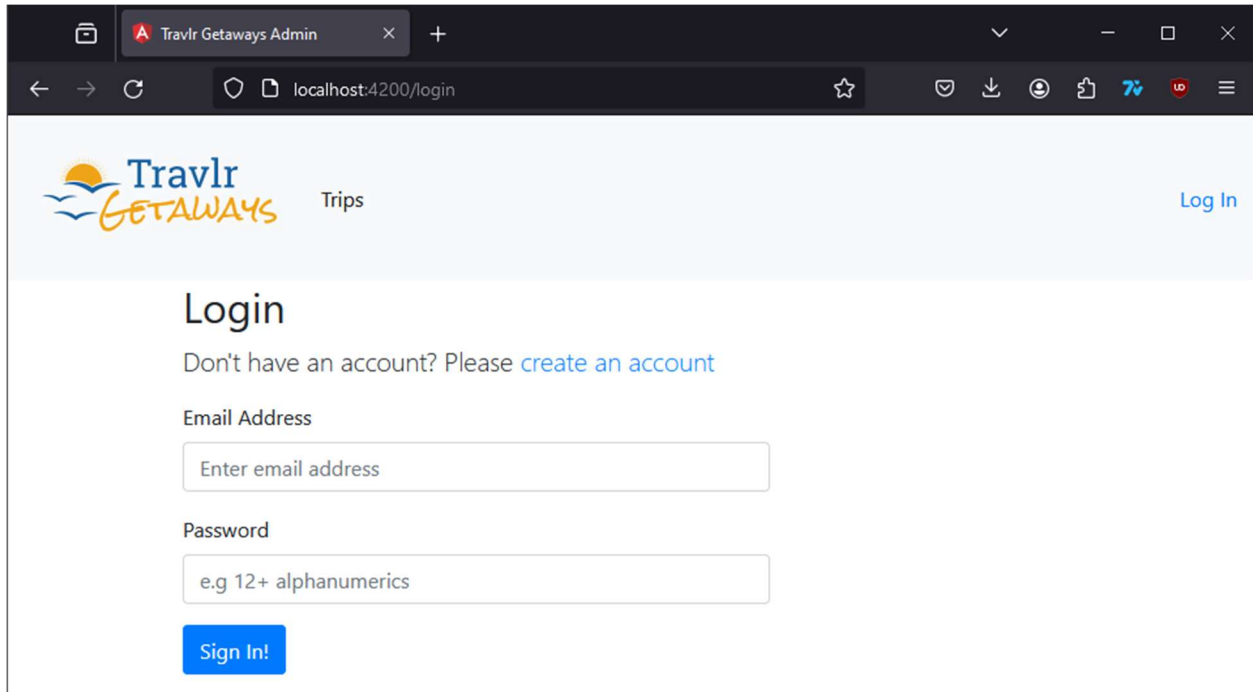


number attribute. Additionally, the MemberAccount class has an aggregation relationship with the Membership\_Admin class. In doing so, the Membership\_Admin class forms a part of the MemberAccount class to create more complex objects, such as a user account with a certain number of travel credit points. Finally, the TravelerInfo class is realized by the Travel\_Agent, CruiseBooking, FlightBooking, and HotelBooking classes. This enables additional functionality to be incorporated into the TravelerInfo class by booking travel packages or retrieving travel package information. The TravelerInfo class is the key point that facilitates the communication of travel and user data together to create user travel reservations.

## API Endpoints

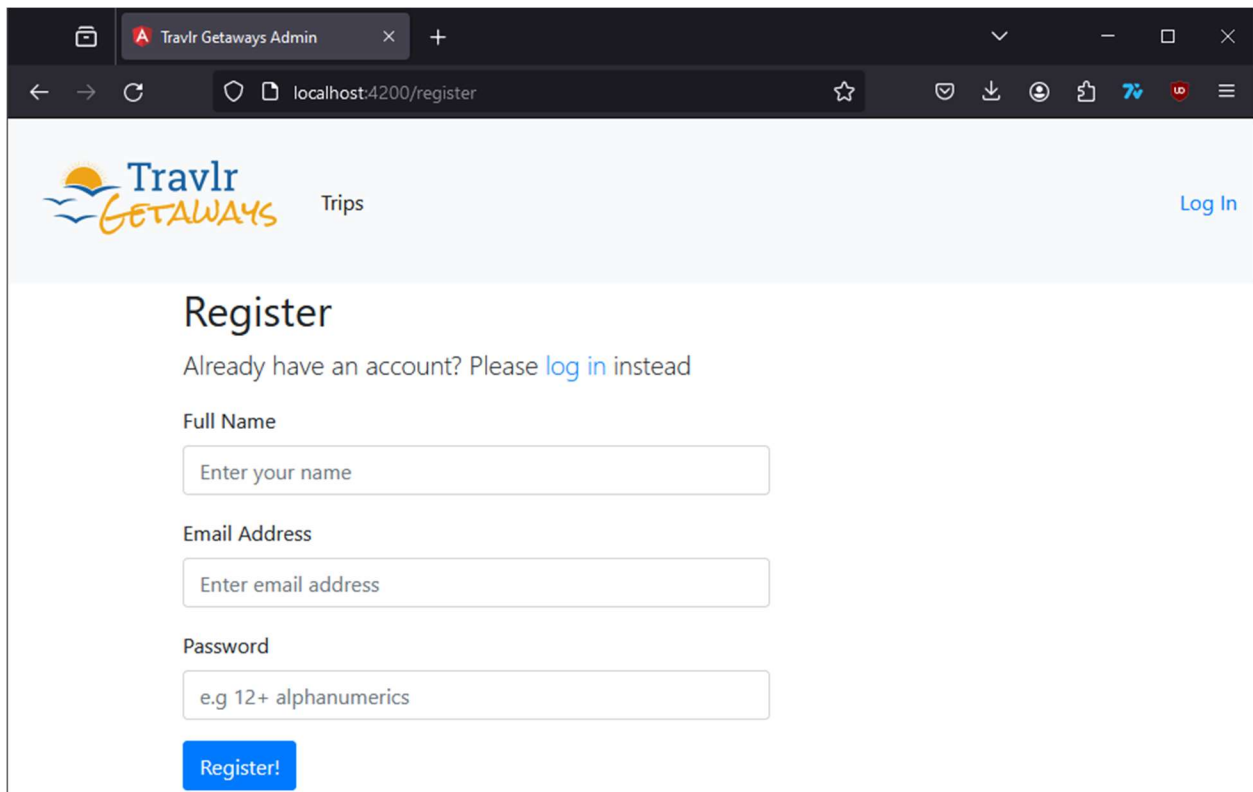
Method	Purpose	URL	Notes
GET	Retrieve list of trips	/api/trips	Returns all active trips
GET	Retrieve single trip	/api/trips/:tripCode	Returns single trip instance, identified by the tripCode passed on the request URL
GET	Retrieve list of meals	/api/meals	Returns all active meals
GET	Retrieve single meal	/api/meals/:mealCode	Returns single meal instance, identified by the mealCode passed on the request URL
GET	Retrieve list of rooms	/api/rooms	Returns all active rooms
GET	Retrieve single rooms	/api/rooms/:roomsCode	Returns single room instance, identified by the roomCode passed on the request URL
POST	Add a single trip	/api/trips	New single trip instance as seen in the database, identified by the tripCode
PUT	Update a single trip	/api/trips/:tripCode	Updated single trip instance as seen in the database, identified by the tripCode
DELETE	Delete a single trip	/api/trips/:tripCode	Returns Null on successful deletion

## The User Interface



The screenshot shows a web browser window with the title "TravlR Getaways Admin". The address bar displays "localhost:4200/login". The page header features the "TravlR GETAWAYS" logo, the word "Trips", and a "Log In" link. The main content area is titled "Login" and includes a link "Don't have an account? Please [create an account](#)". Below this are two input fields: "Email Address" with a placeholder "Enter email address" and "Password" with a placeholder "e.g 12+ alphanumerics". A blue "Sign In!" button is positioned at the bottom of the form.

Figure 1: Admin SPA account login screen



The screenshot shows a web browser window with the title "TravlR Getaways Admin". The address bar displays "localhost:4200/register". The page header features the "TravlR GETAWAYS" logo, the word "Trips", and a "Log In" link. The main content area is titled "Register" and includes a link "Already have an account? Please [log in](#) instead". Below this are three input fields: "Full Name" with a placeholder "Enter your name", "Email Address" with a placeholder "Enter email address", and "Password" with a placeholder "e.g 12+ alphanumerics". A blue "Register!" button is positioned at the bottom of the form.

Figure 2: Admin SPA account registration screen

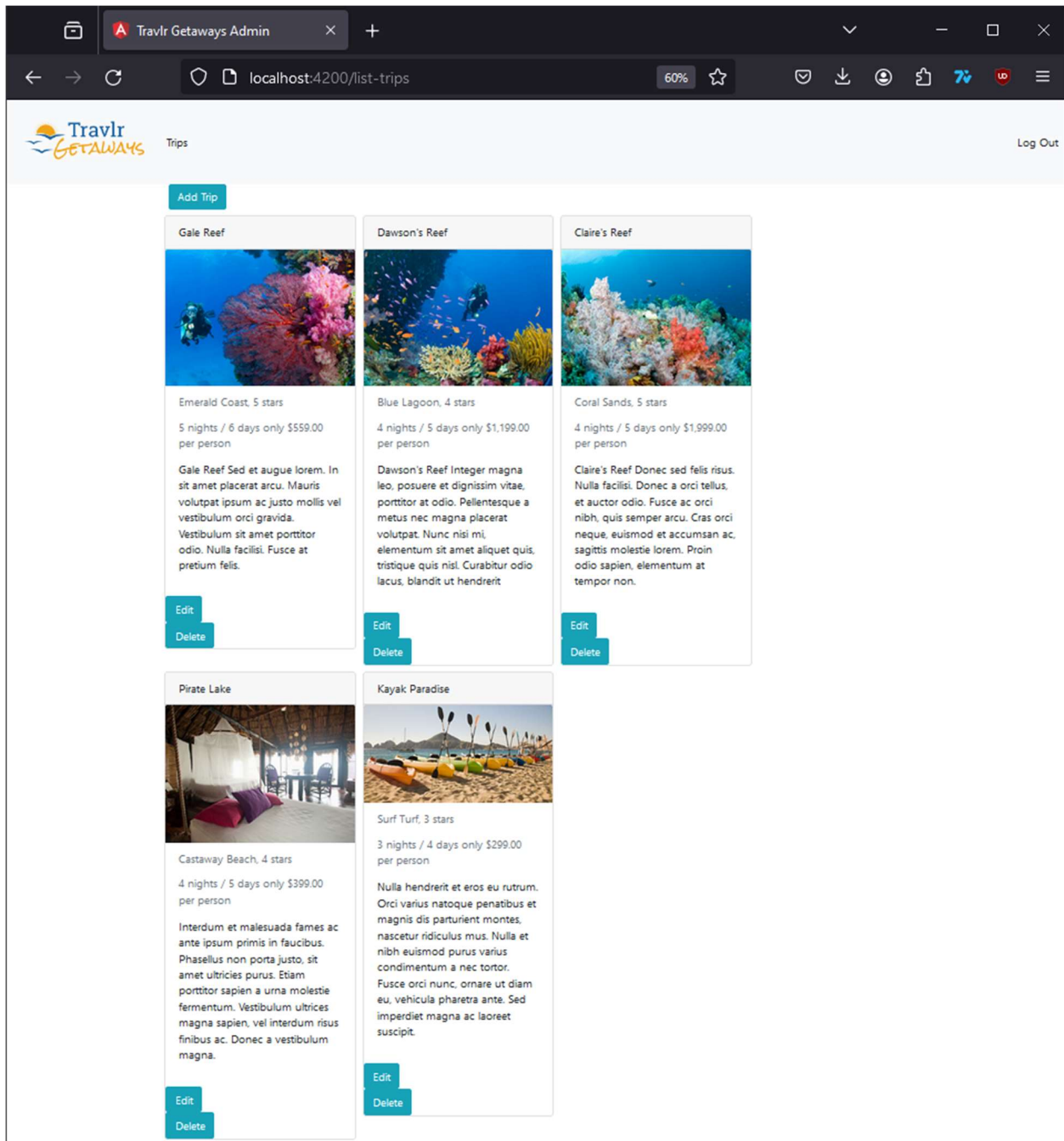


Figure 3: Trip listing admin SPA with a newly added trip.

Travlr Getaways Admin × +

localhost:4200/edit-trip 90% ☆

Travlr GETAWAYS Trips Log Out

## Edit Trip

Code:  
CLAR210621

Name:  
Claire's Reef

Length:  
5 nights / 6 days

Start Date:  
2024-10-24T08:00:00.000

Resort:  
Coral Sands, 4 stars

Per Person (\$):  
749.00

Image:  
reef3.jpg

Description:  
<p> Claire's Reef - Updat

Save

Figure 4: Edit screen to modify an existing trip (Claire's Reef) demonstrating pre-filled information.

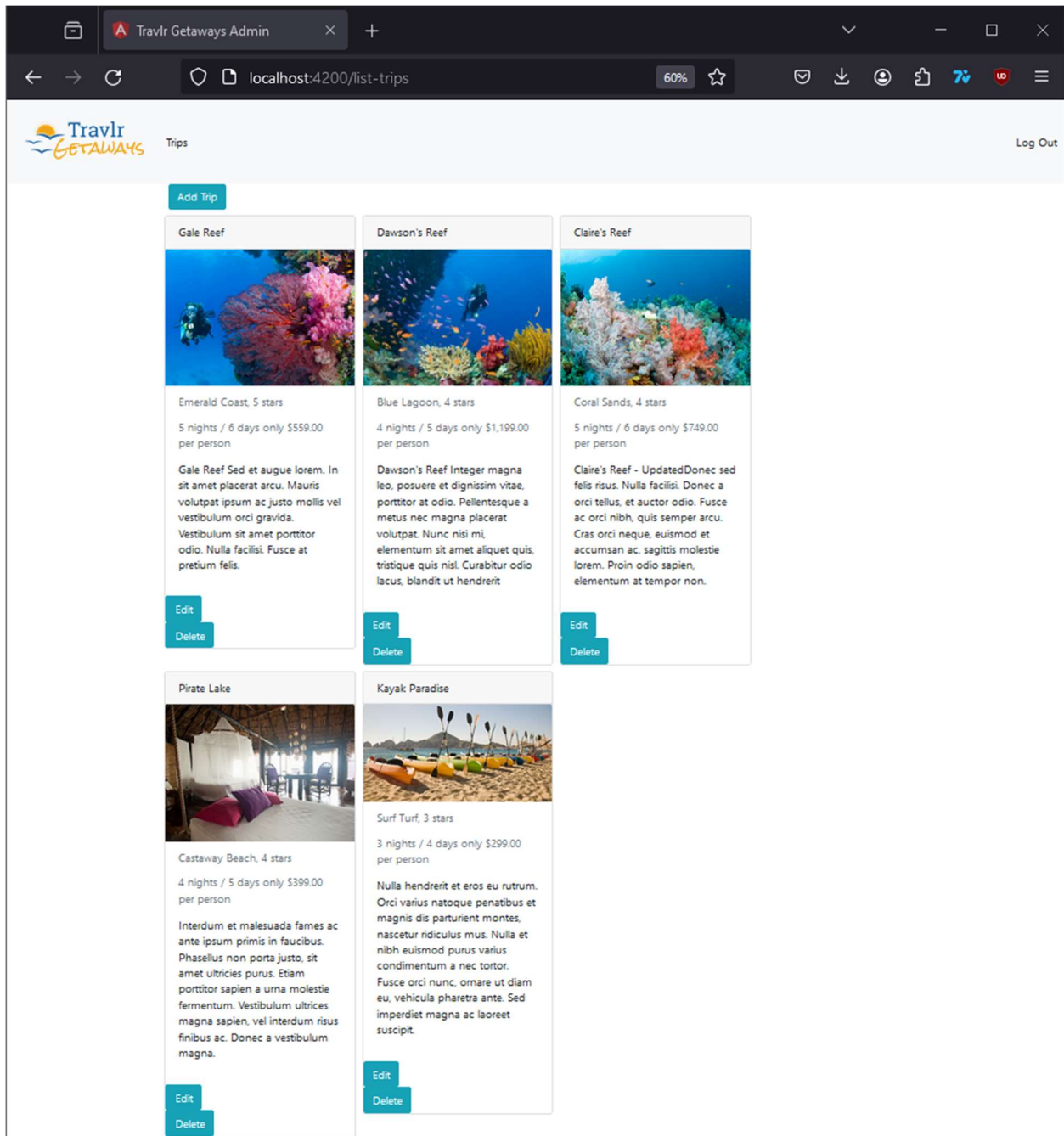


Figure 5: Trip listing admin SPA with newly modified trip details.

Angular is a front-end framework to create single-page applications (SPAs), with data processing occurring on the client side. Alternatively, Express is a back-end framework to create a Representational State Transfer Application Programming Interface (RESTful API), with data processing occurring on the server side and transferring data back to the client side. In practical terms, Angular project structure differs from the Express Hypertext Markup Language (HTML) customer-facing page by consolidating the entire application logic into the browser rather than on a server. Angular's project structure is comprised of models for structuring data, components for basic functionality, routers to enable navigation, and services to abstract critical application

data and functions. Express' project structure is comprised of models for structuring data, routes to enable data and user navigation, views to display data, and controllers to update views based on model data. Much of the data for an Angular SPA is retrieved upon initialization, enabling client-side processing, while Express will see continuous communication and data transfer between the client-side and server-side. Subsequently, Angular can dynamically update a single HTML page with data for the user. However, it is critical to note that both project structures leverage APIs to communicate with databases and utilize Create, Read, Update, and Delete (CRUD) functionality.

Some SPA advantages include:

- Reduced server load from minimized client-server communications.
- Faster page loading time and responsiveness as processing occurs on the client side.
- Modular project structures enhance testing and troubleshooting efforts.

Some SPA disadvantages include:

- Increased initial load resulting from retrieving the entire application logic from the server side.
- There are increased security risks associated with malicious code injections.
- Difficulty optimizing for Search Engine Optimization (SEO) resulting from a single, dynamically updated HTML page.
- Clients must refresh to receive content updates.

Additional SPA functionality includes:

- It enables the entire application logic to be executed on the client side, increasing responsiveness, and reducing server loads.
- Enables thorough debugging with browser developer tools.
- It provides the ability to work offline or with a poor internet connection, as much of the data is stored in the browser cache.

The process of testing if the SPA is appropriately communicating with the API for common CRUD operations is conducted by performing various actions on the SPA and monitoring the App API terminal. This is because the API controller contains status codes that relay vital information about the outcome of various operations. The GET API function can be tested by simply visiting the Travlr Getaways Admin page. This page performs a GET call to the API to retrieve a list of all trips in the MongoDB database. If the terminal relays a status code of 200 on a GET request, the operation is executed successfully. This can be corroborated by the trips populating the trip listing Admin page. Alternatively, if the terminal relays a status code of 404 on a GET request, the operation fails either due to a lack of trips in the database or an error. Subsequently, the trip listing Admin page will not be populated with trips, and further debugging is required.

The PUT API function involves updating a single trip, which modifies data in the MongoDB database. This API function can be tested by clicking the edit button at the bottom of a trip on the

trip listing Admin page. The form will be automatically populated with the trip data. Following the modification of some data fields, the save button can be clicked, which performs a PUT call to the API to update the specific trip with the newly modified data in the database. If the terminal relays a status code of 200 on the PUT request, the operation is executed successfully. This can be verified by ensuring the changed parameters are appropriately reflected on the trip listing Admin page. However, if the terminal relays a status code of 404 on a PUT request, the operation has failed, which is likely attributable to the trip not being found in the database. A status code of 500 can also be thrown by a PUT request, which indicates there was a server error. In any case, troubleshooting is required to identify why the trip cannot be found in the database.