CS 465 Project Software Design Document

Version 1.2

Table of Contents

Document Revision History

Version	Date	Author	Comments
1.0	<01/28/2024>	<Kyle Dale>	Completed and revised the Executive Summary, Design Constraints, and System Architecture View (Component Diagram). Added a references section for personal use throughout the course.
1.1	<02/10 - 02/11/2024>	<Kyle Dale>	Updated to version 1.1 in revision history, completed the various diagrams, completed diagram descriptions, completed API endpoints, updated references.
1.2	<2/24 - 2/25/2024>	<Kyle Dale>	Refactored table of contents, Design Constraints, Sequence Diagram, Class Diagram. Completed UI section and references 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

In response to Travlr Getaways' strategic initiative to enhance and extend their exceptional service to an online platform, we have been engaged to provide our expertise and support in this transformational period. Travlr Getaways requires a travel booking site so their customers can book travel packages. The site must allow users to create an account, search for packages via multiple filters, book vacation packages, and view itineraries prior to trips. Additionally Travlr Getaways would like an admin-only site where administrators are able to maintain the site and its functionality.

With these requirements firmly in mind, we will utilize the MEAN stack and implement MVC (Model-View-Controller) architecture for development. In order to understand the benefits of MVC and the MEAN stack, we must confirm a foundational understanding of what they are and how they function.

The MVC framework is an architectural pattern that separates application processes into three main logical components, the Model, View, and Controller. When conceptualizing the difference between the three, the Model can be thought of as the data, while the view is the display, and the controller is the application logic (Holmes et al. 2019). A basic MVC pattern may typically demonstrate the following response flow: a request comes into the application, gets routed to the controller, the controller may send this to the model, the model responds to the controller, the controller merges the data with the view, and finally the controller sends the response to the requestor (Holmes et al. 2019). The MVC pattern aims to isolate the program logic and presentation layers from one another, increasing code readability and maintainability, which are pivotal aspects in software development. Additionally the MVC pattern fits nicely with rapid prototype development and test driven development (TDD), both of which will benefit Travlr Getaways.

In contrast with the MVC pattern (application organization), the MEAN stack encompasses the software tools necessary for full-stack development and meeting Travlr Getaways' development goals. The 'MEAN' in the MEAN stack stands for MongoDB, Express, Angular, and NodeJS, each of which play an important role in the development of the site.

MongoDB is an open source NoSQL database management program, meaning it does not utilize SQL to manage relational databases (Gillis et al. 2023). MongoDB was designed for high-performance and easy scalability. Additionally, MongoDB stores data in flexible JSON-like documents, meaning it works seamlessly with NodeJS. Due to the open-source, high performance nature of MongoDB, as well as its pairing with NodeJS, it will enable Travlr Getaways to maintain a powerful, lightweight (in terms of cost) database for their application, that has the benefit of public backing and scrutiny.

Defining the 'E' in 'MEAN', Express is a lightweight framework that provides an easy-to-use interface to connect NodeJS and MongoDB (Eddie, 2022). As will become more clear when we reach the 'N' in 'MEAN', NodeJS is not a server, it is a platform. This allows for more creative implementations of servers, but compounds the difficulty in getting a basic site up and running. Express lessens the challenges inherent to NodeJS by setting up a webserver to listen to incoming requests and return relevant responses (Holmes et al. 2019). Essentially, Express can be thought of as the connector between the data in the database (MongoDB) and the HTML page. It allows the data present in the site's database to be actively displayed on the site. This will allow Travlr Getaways to host a more robust, feature rich, and relevant site for their customers.

Next is the ‘A’ or Angular, a robust, open-sourced front-end framework that is maintained by Google (Google, 2023). Angular is essentially a framework that aids the developer in creating the user interface of a web application. Additionally, the component-based architecture of Angular supports our adoption of the MVC pattern discussed earlier, while the powerful routing abilities present in Angular allow for the creation of single page applications (SPAs). SPAs differ from traditional web pages in that rather than routing a user to a different page when clicking a link, SPAs load one page and dynamically update content as the user interacts with the page (Google 2023). Although SPAs have many benefits, including user experience and eventual performance boosts, they are not ideal for search engine optimization, and can have long initial load times. As a result, for the case of Travlr Getaways, an SPA would be perfect for the administrator-side of the application. As the usage of an SPA would allow for a more dynamic and feature-rich product, that the administrator could effect in real-time. As we can see, Angular is important to the MEAN stack for a multitude of reasons, and will be invaluable in the development goals of Travlr Getaways.

Finally comes the ‘N’ or NodeJS. Node is a software platform that allows one to create their own web server and build web applications on top of it, it is not a server itself, nor is it a language, but it contains a built-in HTTP server library so developers do not need to run separate web server programs (Holmes et al. 2019). Node utilizes an event-driven, non-blocking, I/O model, making it lightweight and efficient (Eddie, 2022). Essentially, Node is the platform by which the web application and server will be built upon, and as touched upon earlier, utilizes the same language as Express. This will benefit the development team as development will be simplified and have increased efficiency due to a uniform language. The flexibility and power of Node combined with Express allow for the relatively rapid creation of dynamic and powerful web applications, and are excellent choices for tackling Travlr Getaways needs.

Now, with a foundational understanding of the MVC pattern and MEAN stack, we can begin to glean the advantages of utilizing them for Travlr Getaways’ purposes. Ultimately the MEAN stack contains the potential to meet every one of Travlr Getaways’ development requirements in a robust and efficient manner, while the open-source scaleable nature of the MEAN stack will further result in a reduction in overall time and money expenditure for the creation and maintenance of the project.

Design Constraints

When considering the design constraints for the Travlr Getaways application, there are a few inherent limitations within the MEAN stack that must be considered. Beginning with MongoDB, as a NoSQL database it offers powerful flexibility and scalability but does have certain disadvantages to be aware of.

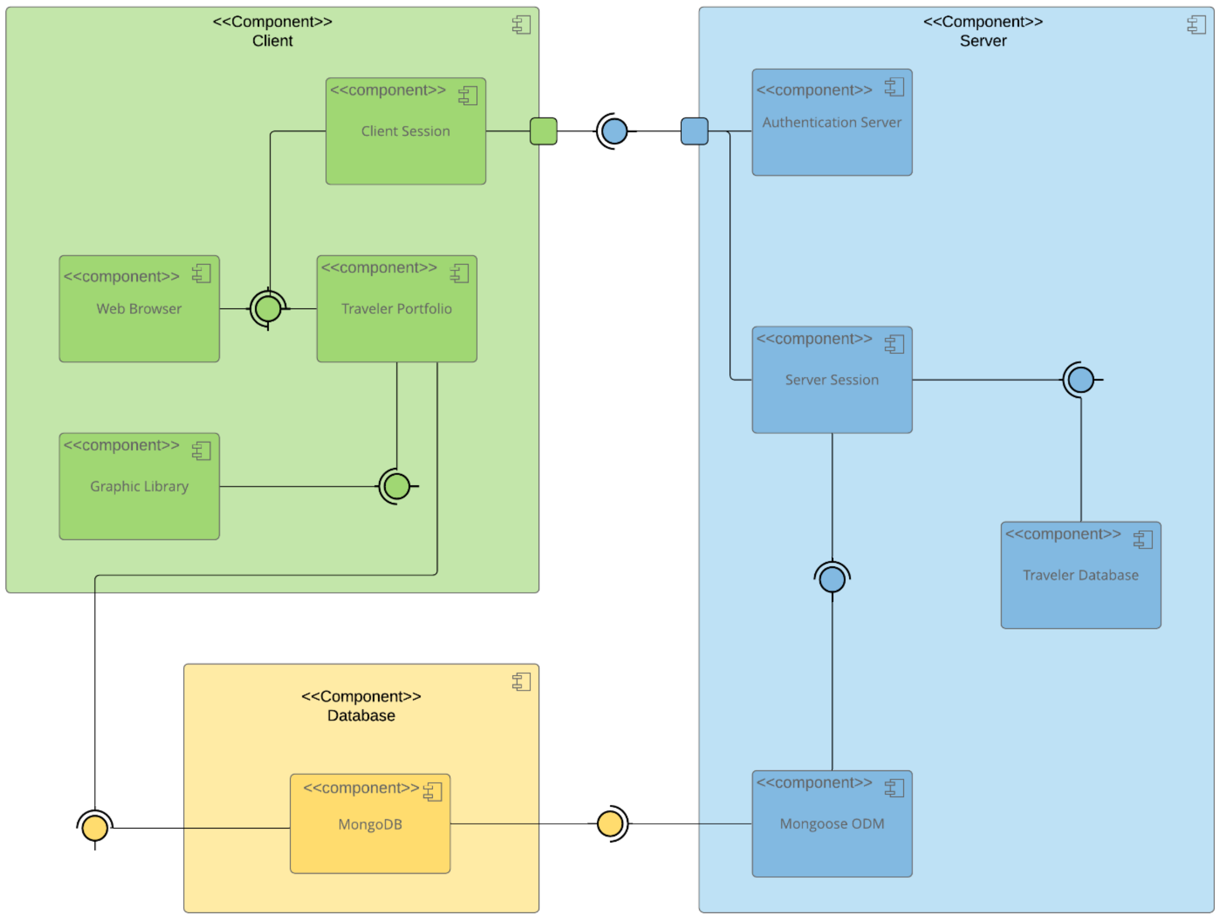
Firstly, while MongoDB provides atomicity, consistency, isolation, and durability (ACID) at the document level, it does not provide full ACID compliance across multiple documents or collections (GeeksforGeeks, 2023). This limitation in MongoDB arises from the nature of its architecture, where data is shared and distributed across multiple nodes for scalability and performance, and as a result, MongoDB does not support multi-document transactions that span multiple shards or collections. Additionally, unlike traditional relational databases, MongoDB does not support joins in the same manner. Developers can add join functionality by manually adding the code, but this can be tedious, time consuming, and leave the code prone to error. Lastly MongoDB would not be ideal for larger applications. This does not strictly apply to Travlr Getaways’ at the moment, but were the application to grow, the problems with redundancy in memory usage and document nesting and size limits may become relevant. As MongoDB stores key names with each value pair, some redundancy is introduced and may increase memory usage beyond what is necessary (GeeksforGeeks, 2023). Lastly, MongoDB limits the overall document size to 16 MB and the maximum possible nesting level to 100 levels (GeeksforGeeks, 2023). Although MongoDB has some drawbacks, in the case of Travlr Getaways web application, it is a powerful and scalable solution.

Another important constraint to be aware of in development of the Travlr Getaways' web application are the security risks due to the flexible nature of the MEAN stack. As an example, the dynamic nature of JS in Node.js can increase the risk of injection attack if proper security measures, such as input validation and sanitization are not considered. In a similar vein, due to the schemaless nature of MongoDB, data exposure is a possibility if access controls are not adequately configured. Although the MEAN stack is powerful, considering and mitigating these potential vulnerabilities will be an important factor in achieving the desired functionality for the Travlr Getaways web application, while providing robust security.

Lastly, performance optimization can be tricky within the MEAN stack, due to the distributed nature of components. As an example, ensuring efficient communication between the client-side Angular framework and the server-side Node.js application requires careful attention to latency and transfer optimization (Angular, 2024). Additionally, optimizing MongoDB queries for performance requires a deep understanding of the subject matter and increases the overall program complexity and thus development time.

By acknowledging and addressing the constraints present in the MEAN stack, the development of the Travlr Getaways application will better align with project goals, and ensure successful implementation within the defined requirements.

Component Diagram



A text version of the component diagram is available: CS 465 Full Stack Component Diagram Text Version.

System Architecture View

When reviewing the component diagram of the Travlr Getaways' web application (seen above) we can identify much about the system architecture and the components contained within. The three overarching components of the application that will interact to facilitate its function are the Database, Client, and Server component boxes. Within each of these are processes that interact, perform, and track various aspects of the program's function.

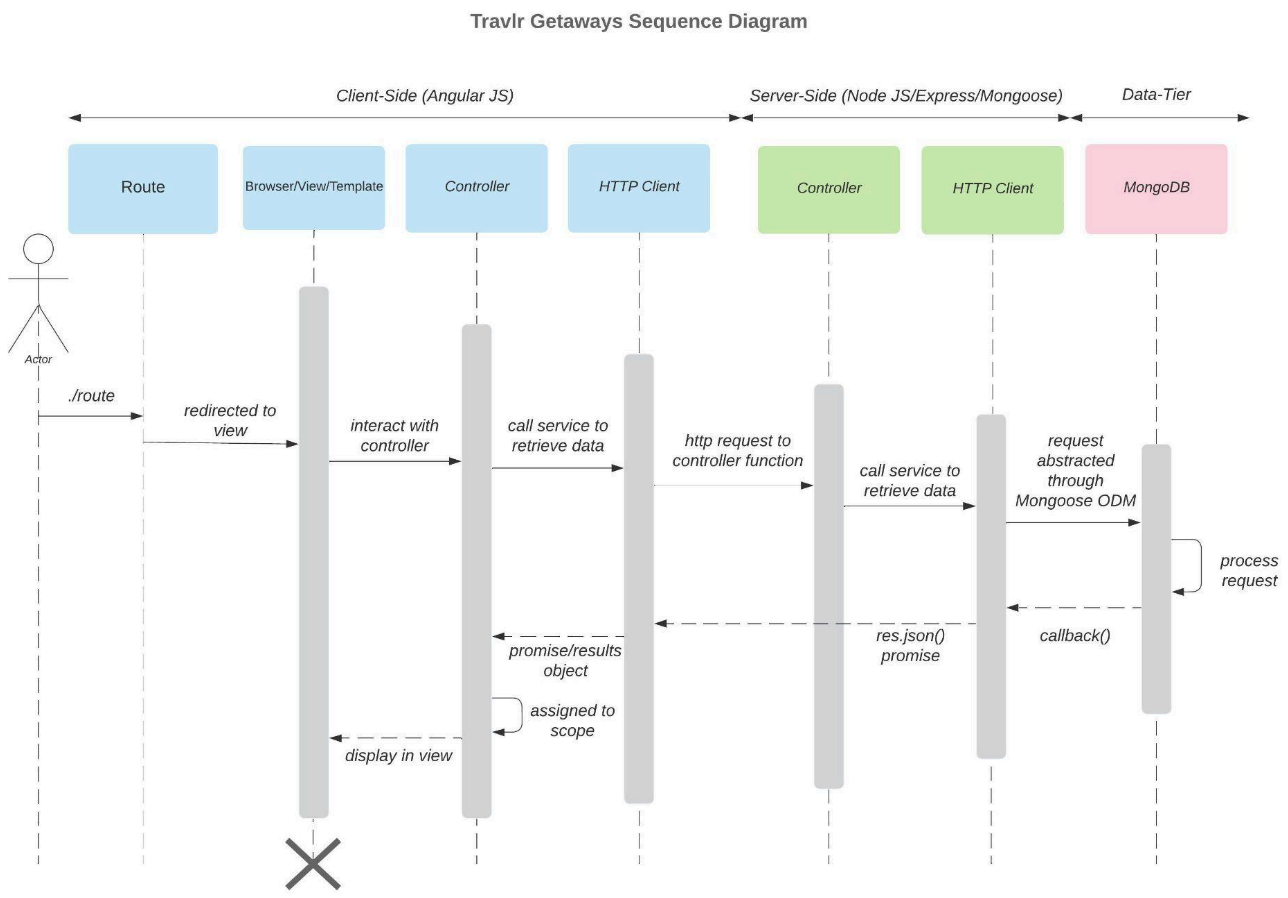
To begin, the Client component box (green) can be seen to contain the following components or processes: the client session, traveler portfolio, web browser, and graphic library. The web browser represents the client-side interface through which end-users will interact with the system, while the client session will manage and maintain the session information for the end-user interaction. The traveler portfolio represents the user's portfolio, personalized data pertaining to their travel history, preferred destinations, booked packages, itineraries and more. Finally, the graphic library will handle the graphical elements and resources required for rendering within the web browser. Ultimately, this overarching Client component is responsible for managing, tracking, and facilitating end-user interaction with the web platform.

Next, the Server component box (blue) contains the following components or processes: the authentication server, server session, traveler database, and MongooseODM. The authentication server component is somewhat self-explanatory in purpose, and is responsible for handling the user authentication and authorization process. The server session (similar to the client session), is responsible for managing and maintaining the session information on the server side, facilitating communication with other server components. The Traveler Database is responsible for storing traveler-related data, including user profiles. Finally, the Mongoose ODM component acts as a bridge between the Server component box and the Database component box (MongoDB). Ultimately, this overarching Server component is responsible for managing, tracking, and facilitating back-end processes and interactions with the web platform.

Finally, the Database component box (yellow) only contains the single, MongoDB component. By now we are quite familiar with MongoDB and can likely deduce its purpose within the diagram and program flow. MongoDB or the database is responsible for storing and retrieving information relating to the users and their profiles.

Ultimately, the diagram is providing a visual representation of the flow and processes that will take place to allow the Travlr Getaways web platform to function. Thanks to our foundation understanding of the MEAN stack, many of the components, their relationship, and contribution to the application’s function can now be comprehended visually, to even further enhance our understanding and continuously ensure the product is relevant to the requirements.

Sequence Diagram



When reviewing the sequence diagram of the Travlr Getaways' web application (seen above), we can identify much about the system architecture and the flow of logic that takes place within the application. Similar to the previously reviewed component diagram, we can see that the logical flow of the program is broken down into three distinct sections, the client-side, server-side, and data-tier sections. Requests made to the site will flow through each of the respective layers until the request is processed and a response is returned.

Before we dissect the logical flow of the diagram, it is important to understand what the various symbols of the diagram indicate. Beginning on the top with the object boxes, as sequence diagrams are used to illustrate the simple or complex interactions between objects (Athuraliya, 2022). On the left we can see the actor. The actor represents the user interacting with the system, and in this instance lays on a lifeline. In basic terms, the actor sitting on the lifeline indicates that the actor is involved in the sequence of events being depicted, and can be seen as beginning the process. The solid black arrows seen throughout the diagram represent synchronous messages, where the sender must wait for the receiver to process the message and return a response before carrying on with another message. Next, we can see gray boxes overlapping lifelines throughout the diagram; these are known as activation bars, and indicate that an object is active (instantiated) during an interaction between two objects. The length of the rectangle indicates the duration the object stays active (Athuraliya, 2022). Finally, we can see a reflexive message on the right, where MongoDB processes the request and begins the response chain. Once the data has been returned to the client-side, the activation bar is terminated to illustrate process completion.

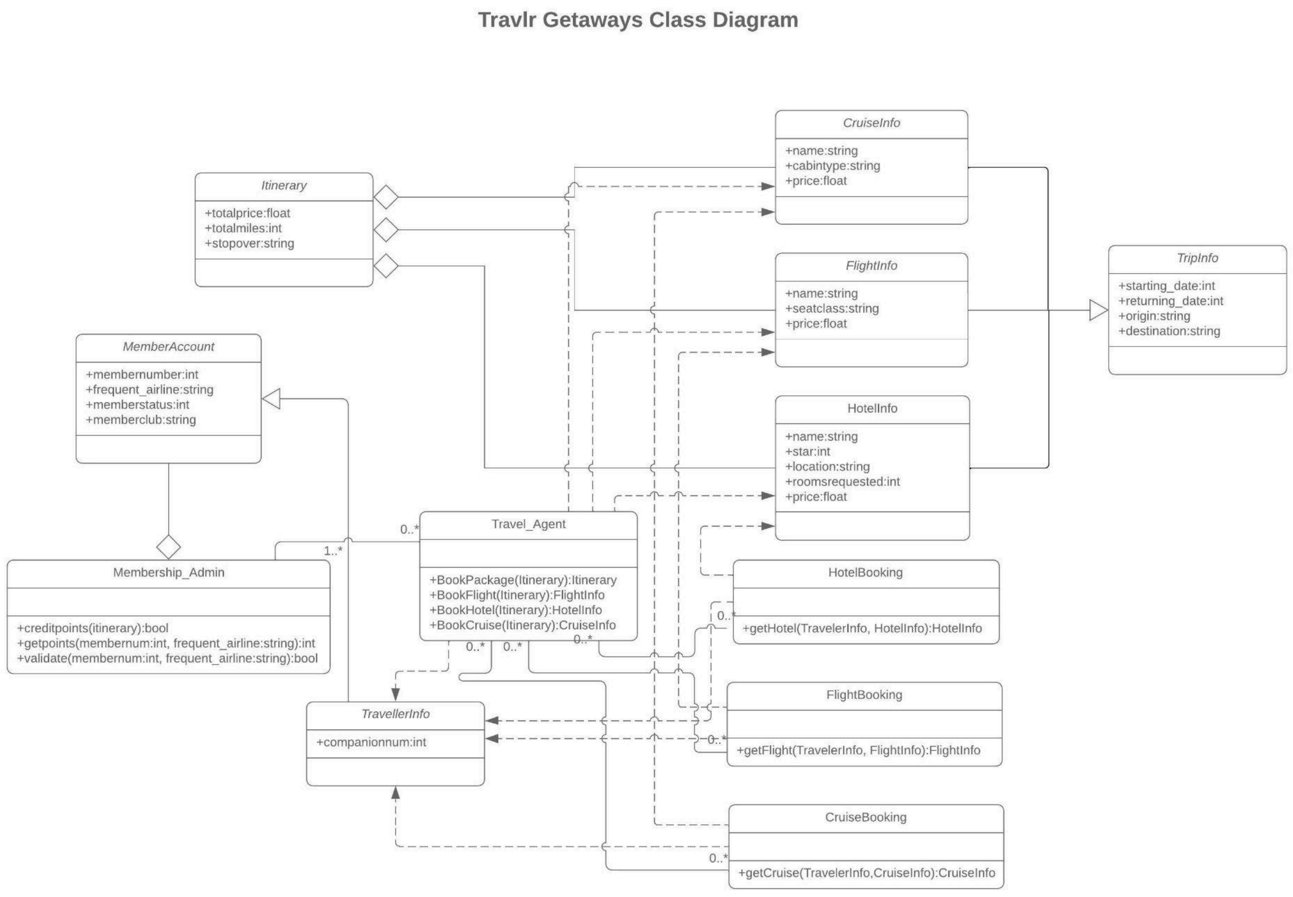
Now that we understand sequence diagram notation, we can view an example. To illustrate the flow of logic in the program, we can imagine a user logging into the system. The logical flow of the program will begin on the left of the diagram with the user or actor. When the user enters the URL and attempts to log in to the web application, JavaScript will be run and will hit the router of the client-side server, also known as the routing module. The router will then redirect the request to the browser/view/template object of the program and a process, denoted by the action bar, will begin to process the user request. The browser/view/template object will then interact with the controller so that the controller may request the proper data from the server-side of the application. Once the controller calls the proper service to retrieve the data, the HTTP client object will issue an http request to the server-side controller function. Now inside of the server-side portion of the application, the controller will interact with the HTTP client object within the server-side to request the proper data from the DB. The server-side HTTP client will put in a request that is abstracted through MongooseODM to the final tier of the application, the data tier. In the data tier, we can see the reflexive message mentioned earlier where MongoDB is processing the requested data and preparing to begin the return process.

As we can see, the return of the request is similar to the request, but invokes fewer objects. As an example, once the HTTP client on the server-side receives a return from the data-tier, it can immediately return the request to the client-side HTTP client, skipping an object. Finally on the client-side once again, the request is returned to the Controller where it is assigned proper scope before finally being returned to the proper browser/view/template, and the user request is fulfilled and processes terminated. This process will work similarly whether the user is requesting authentication or attempting to dynamically load alternate content. Basically, the client-side of the application will route the request to the client-side controller, where the control will then engage the API to request the proper information from the database. Once the data is retrieved it will be returned to update the user's view.

When concerning the admin functionality, the processes required would be handled similarly in terms of logical flow. The request would still be made from the admin client-side, and routed to the server-side and eventually the database. However, although the logical flow will be similar, the processes, tools, and user interface will be much different. Although this will not cause differences in the sequence diagram’s flow, as we will soon see in the class diagram, there will be more processes and functionality involved with the admin-specific page. However, the overarching logical architecture will remain similar.

In conclusion, the sequence diagram of the Travlr Getaways' web application provides valuable insights into the system architecture and the flow of logic governing interactions between different layers. Through the careful examination of symbols and notation, we've gained a clear understanding of how requests traverse through the client-side, server-side, and data-tier sections, ultimately resulting in the fulfillment of user requests.

Class Diagram



When reviewing the class diagram of the Travlr Getaways' web application (seen above), we can identify much about the system architecture and the processes that take place within the application.

As this diagram and the application's function primarily center around the Travel_Agent class, we will begin by exploring it and in relation to the other classes. As we can see, the Travel_agent class has four operations defined within, BookPackage, BookFlight, BookHotel, and BookCruise, with their purpose being fairly self-explanatory. We can further see that the Travel_Agent class has many relationships with the other class objects within the application. Beginning with dependency (denoted by a dotted line and black arrow), we can see that Travel_Agent has dependency with CruiseInfo, FlightInfo, HotelInfo, and TravellerInfo. This means that Travel_Agent relies on information present in the four class objects, so altering them could alter Travel_Agent. Next we can see that Travel_Agent has several associations. These include an association with Membership_Admin, HotelBooking, FlightBooking, and CruiseBooking, and each association includes a cardinality of zero or more, meaning that zero or more instances of the previously named class objects can be related to a single instance of the Travel_Agent class. As an example, a single Travel_Agent instance may have many HotelBooking instances associated with it alone.

Moving onto the HotelBooking, FlightBooking, and CruiseBooking classes, we can see that, other than the previously mentioned associations, each has two dependencies and a get operation associated. All three classes have a dependency pointing toward TravellerInfo, and each of the three 'Booking' classes has a dependency with the corresponding 'Info' class; for example, CruiseBooking is dependent on CruiseInfo, and FlightBooking is dependent on FlightInfo.

When concerning the 'Info' classes, we can see that, in addition to the previously mentioned dependencies, these classes all possess inheritance of the TripInfo class. Inheritance represents an is-a relationship between classes, where the subclass or child class inherits attributes and behaviors from the parent class. In this case, each of the 'Info' classes is a child class of TripInfo . Additionally, each 'Info' class contains various attributes related to the class object's function, such as name, cabin, and price for CruiseInfo. These attributes are what are being inherited from the TripInfo class, and represent the only relationship the TripInfo class has.

When analyzing TravellerInfo, we see it has the attribute of 'companionnum' as well as the previously mentioned dependency with Travel_Agent and the three 'Booking' classes. Additionally, TravellerInfo inherits from the MemberAccount class.

MemberAccount has multiple attributes and only one other relationship besides the inheritance by TravellerInfo, and that is aggregating Membership_Admin. This means that the MemberAccount object encapsulates functionality provided by Membership_Admin, but Membership_Admin also exists independently from MemberAccount. This specific relationship will allow Membership_Admin to manage multiple membership statuses, for example.

Finally, we have the Itinerary class object. Itinerary has totalprice, totalmiles, and stopover as the attributes and is aggregated by the 'info' classes for its relationships. Similar to previously discussed, this means that the 'Info' class objects encapsulate some of the functionality within Itinerary , but Itinerary still exists independently.

Ultimately, we can see that although initially confusing, the class diagram for the Travlr Getaways' web application provides invaluable insight into the system and class object architecture and processes. Through the careful examination of the diagram, we've gained a clear understanding of how the various class objects interact to perform the functionality required in the Travlr Getaways' web application.

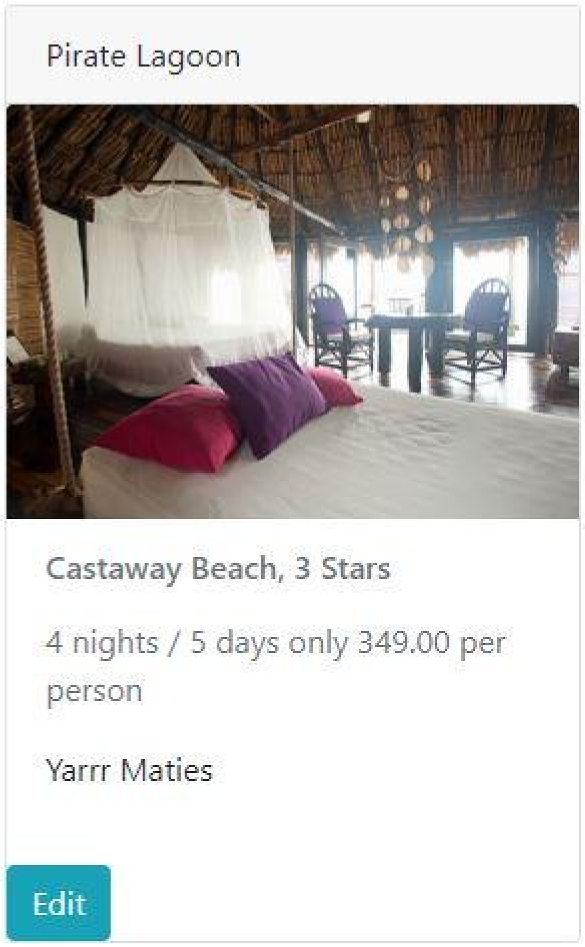
API Endpoints

Method	**Purpose**	**URL**	**Notes**
GET	<Retrieve list of available trips>	</api/trips>	<Returns all active trips>
GET	<Retrieve single trip based on trip code>	</api/trips/[trip code]>	<Returns single trip instance, identified by the trip code passed on the request URL>
GET	<Retrieve list of available rooms>	</api/rooms>*	<Returns all active rooms>
GET	<Retrieve single room based on room ID>	</api/rooms/[room ID]>*	<Returns single room instance, identified by the room ID passed on the request URL>
GET	<Retrieve list of available meals>	</api/meals>*	<Returns all active meals>
GET	<Retrieve single meal based on meal ID>	</api/meals/[meal ID]>*	<Returns single room instance, identified by the room ID passed on the request URL>

*(Please note URL with an * have not been implemented at the time of writing.)*

The User Interface

Added Trip:



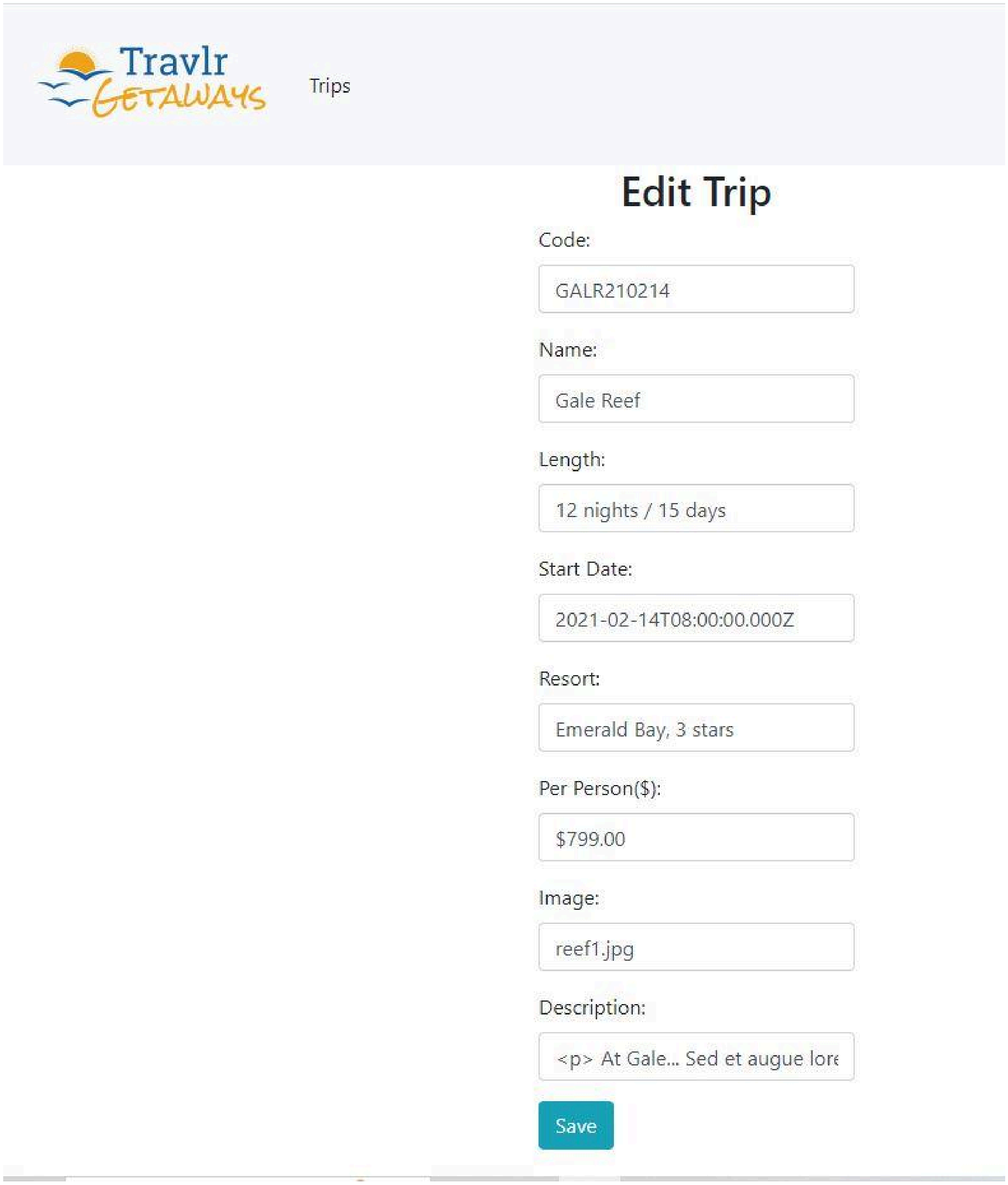
Pirate Lagoon

Castaway Beach, 3 Stars

4 nights / 5 days only 349.00 per person

Yarrr Maties

Edit Trip:


Trips

Edit Trip

Code:
GALR210214

Name:
Gale Reef

Length:
12 nights / 15 days

Start Date:
2021-02-14T08:00:00.000Z

Resort:
Emerald Bay, 3 stars

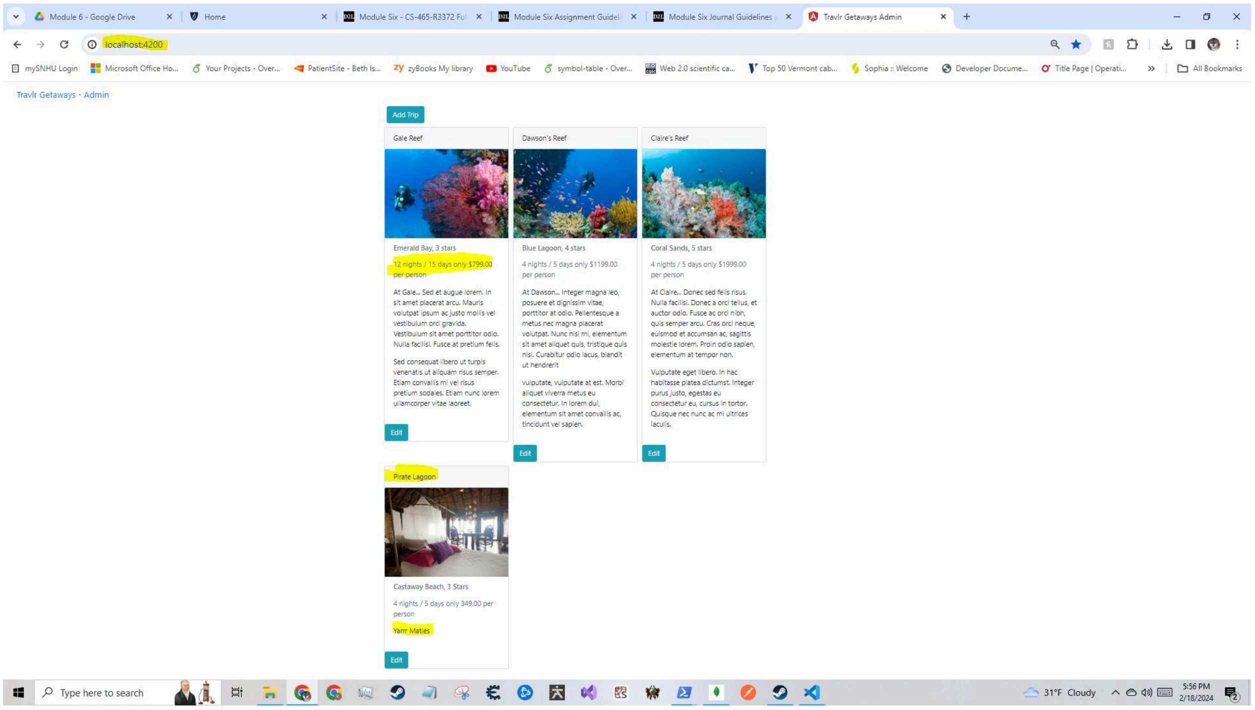
Per Person($):
$799.00

Image:
reef1.jpg

Description:
<p> At Gale... Sed et augue lore

Save

Updated Screen:



Gale Reef

Emerald Bay, 3 stars

12 nights / 15 days only $799.00 per person

At Gale... Sed et augue lorem. In sit amet placerat arcu. Mauris volutpat ipsum ac justo mollis vel vestibulum orci gravida. Vestibulum sit amet porttitor odio. Nulla facilisi. Fusce at pretium felis.

Sed consequat libero ut turpis venenatis ut aliquam risus semper. Etiam convallis mi vel risus pretium sodales. Etiam nunc lorem ullamcorper vitae laoreet.

Dawson's Reef

Blue Lagoon, 4 stars

4 nights / 5 days only $1199.00 per person

At Dawson... Integer magna leo, posuere et dignissim vitae, porttitor at odio. Pellentesque a metus nec magna placerat volutpat. Nunc nisi mi, elementum sit amet aliquet quis, tristique quis nisl. Curabitur odio lacus, blandit ut hendrerit

vulputate, vulputate at est. Morbi aliquet viverra metus eu consectetur. In lorem dui, elementum sit amet convallis ac, tincidunt vel sapien.

Claire's Reef

Coral Sands, 5 stars

4 nights / 5 days only $1999.00 per person

At Claire... Donec sed felis risus. Nulla facilisi. Donec a orci tellus, et auctor odio. Fusce ac orci nibh, quis semper arcu. Cras orci neque, euismod et accumsan ac, sagittis molestie lorem. Proin odio sapien, elementum at tempor non.

Vulputate eget libero. In hac habitasse platea dictumst. Integer purus justo, egestas eu consectetur eu, cursus in tortor. Quisque nec nunc ac mi ultrices iaculis.

Pirate Lagoon

Castaway Beach, 3 Stars

4 nights / 5 days only 349.00 per person

Yarrr Maties

How is the Angular project structure different from that of the Express HTML customer-facing page?

Although both Angular and Express can be and are utilized to create the Travlr Getaways' web application, there are distinct differences between the two platforms. The architecture, data flow, and rendering/routing methods between Angular and Express HTML function and operate distinctly.

Regarding file architecture, Angular typically follows a modular structure where components, services, modules, and other artifacts are organized into separate directories based on their function, such as the 'app', 'assets', or 'environments' folders present within the source folder for the SPA (app_admiin). Express HTML, meanwhile, generally has a simpler structure and can be organized into popular organizational methods such as MVC, which was utilized for Travlr Getaways' web application. The MVC architecture allows HTML, CSS, and JS files to be placed in a directory that is easily accessible by Express (as well as increasing readability).

The data flow of Angular and Express is also quite distinct from one another. In Angular, the data flow is managed through uni-directional data binding using observables or the 'ngOnChanges' lifecycle hook for component communication (Angular, 2024). In Angular, services play a critical role in managing shared data and logic across different components, as can be seen in the trip-data.service.ts file within the app/services folder(s). In contrast, Express typically involves a request-response model where data is requested from the server/DB and rendered client-side. This philosophy can be observed in the MVC architecture of the user-application created for Travlr Getaways.

Finally, for routing and rendering, Angular renders views dynamically on the client-side using its templating engine and component-based architecture. This means that changes to the data will automatically reflect in the view, unlike with Express HTML. Angular also has a built-in routing module so that navigation between views is handled client-side without full page reloads (Angular, 2024). Express HTML, however, renders views on the server-side and sends the pre-rendered HTML to the client. Express can also handle routing on the server-side using middleware functions (Holmes et al, 2019). Routes are defined based on the HTTP method and URL pattern, and the server sends the appropriate responses based on the requested route. This can be seen in both the app_api and app_server portions of the Travlr Getaways' web application.

What are some advantages and disadvantages of the SPA functionality? What additional functionality is provided by a SPA compared to a simple web application interaction?

SPAs have many unique advantages and disadvantages when compared to MPAs that a developer or client should be aware of before deciding which to employ. Firstly, when building an SPA, it is necessary to have at least a foundational understanding of JavaScript (JS), while this is not always the case with MPA. If a developer is unfamiliar with JavaScript, they will either need to learn to utilize an SPA, or choose to use an MPA. Another factor to consider is speed, as what SPAs trade in initial load time they gain in a faster user experience overall. Since SPA loads once and dynamically updates content as the user interacts with the application, this results in faster performance after the initial load. In MPA, each new link must load an entirely new page, which creates a faster first load, but slower subsequent operation speed (Holmes et al, 2019). Another benefit similar to the previous, is that with SPAs, the server primarily serves data through API calls, rather than rendering HTML pages for each request. This can reduce server load, especially in applications with heavy client-side interaction. Finally, SPAs can be easily designed to format properly across different devices and screen sizes. This will more easily allow developers to ensure a consistent experience across various devices when compared to MPAs.

Although there are many benefits to SPAs, there are drawbacks as well. As SPAs rely heavily on JS to render content dynamically, search engine optimization (SEO) can be affected (BasuMallick, 2022). Search crawlers may not properly execute the JS and can affect the discoverability of SPA content. Another drawback of SPAs, as touched upon previously, are the longer initial load times when compared to MPA. Depending on the purpose of the application, this can have a marked effect on user enjoyment and retention. To compound this issue, SPAs may not be compatible with some older web-browsers, and could cause accessibility issues. Finally, with SPAs, much of the application's logic and data processing occurs on the client-side, which can make clients particularly vulnerable to client-side attacks like Cross-Site Scripting.

There are a few additional features offered by SPAs not present in MPAs. As touched upon previously, SPAs dynamically load content from the server without full page reloads. This is accomplished via client-side routing, which allows users to move between views without triggering a page refresh. Additionally, as much of the application processes occur client-side, SPAs can leverage certain techniques and client-side caching to provide some offline support. Once internet connection returns, the SPA will sync changes made offline (Doggett, 2023).

What is the process of testing to make sure the SPA is working with the API to GET and PUT data in the database? What are some errors you ran into or what are some errors you could expect to run into?

Although there are a plethora of methods to ensure the SPA is working with the API to GET and PUT data, for the Travlr Getaways' web application, I primarily relied upon error handling, input validation and logging and monitoring. I have not yet stress tested the platform and attempted to break it, but thanks to security by design, I was able to include safeguards against malicious user input in the design itself. Additionally, the inclusion of log messages within the various processes of the SPA allows me to ensure that the SPA is functioning properly. Thanks to these log messages, I was able to catch and solve a few errors affecting site performance.

The primary two errors I ran into were my version of Node.js being too high for the other dependencies present in the application, and attempting to pass an invalid symbol to the ‘perPerson’ field of the trips. I was able to solve the Node.js error by uninstalling Node, and installing node-version-manager or nvm. Once nvm was installed I could reinstall the various versions I required and switch between them at will.

The other error took me a bit longer to solve, as it occurred twice, the second instance being less obvious. Initially, my trip data included a ‘$’ within the string containing the perPerson amount for the various trips. This was causing an error as ‘$’ was an invalid symbol for the pipe. I was able to solve this by removing the dollar sign from the data itself. This fixed the issue until I ran into the same issue again when attempting to format and output the data to the SPA from the API. I eventually realized the problem lay in a piece of code taken from the module resources which included the addition of a symbol with the perPerson output, not present in the data itself. Once I removed this I was able to run the SPA and properly execute the desired functionality.

References

Angular. (2024). *Component Lifecycle*. Angular. https://angular.io/guide/lifecycle-hooks.

Athuraliya, A. (2022, December 12). *Sequence diagram tutorial – complete guide with examples*. Creately. https://creately.com/guides/sequence-diagram-tutorial/.

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

Doggett, B. (2023, December 12). *Single Page Applications: A deep dive into their function and value*. Xtremepush. https://www.xtremepush.com/blog/single-page-applications-value/#:~:text=The%20last%20advantage%20of%20SPAs,unstable%20or%20they%20are%20offline.

Eddie, J. (2022, January 7). *Why mean stack is best for web development?*. Medium. https://enlear.academy/why-mean-stack-is-best-for-web-development-55b7e4a00d72.

GeeksforGeeks. (2023, April 21). *Acid properties in DBMS*. GeeksforGeeks. https://www.geeksforgeeks.org/acid-properties-in-dbms/

Google. (2023). *What is Angular?*. Angular. https://angular.io/guide/what-is-angular.

Google. (2024b). *Server-side rendering*. Angular. https://angular.io/guide/ssr

GeeksforGeeks. (2023, August 29). *MVC Framework introduction*. GeeksforGeeks. https://www.geeksforgeeks.org/mvc-framework-introduction/.

Gillis, A. S., & Botelho, B. (2023, March 7). *What is mongodb? features and how it works – techtarget definition*. Data Management. https://www.techtarget.com/searchdatamanagement/definition/MongoDB.

Holmes, S., & Harber, C. (2019). *Getting mean with Mongo, express, angular, and node*. Manning.

IBM. (2024). *Routing*. Express routing. https://expressjs.com/en/guide/routing.html.

MongoDB. (2024). *Optimize Query Performance*. MongoDB.

https://www.mongodb.com/docs/manual/tutorial/optimize-query-performance-with-indexes-and-projections/.

Nagar, T. (2022, March 1). *Know advantages and disadvantages of Mean Stack & PHP Development*. Dev Technosys. https://devtechnosys.com/insights/mean-stack-development-vs-php-development-advantages-and-disadvantages/.

VisualParadigm. (2024). *UML Class Diagram Tutorial*. VisualParadigm. https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial/.