Travlr Getaways Online User Connection

CS 465 Project Software Design Document

Version 2.0

Table of Contents

Document Revision History

Version	Date	Author	Comments
1.0	09/11/2022	L. Artl	Routing for all main pages; Interactive header menu; Database connection added
1.1	10/2/2022	L. Artl	Routing Diagram, Class Diagram, API Endpoints inserted and updated
1.2	10/8/2022	L. Artl	The User Interface updated
2.0	10/15/2022	L. Artl	Document completed, submitted

Executive Summary

For a smooth and user-friendly customer experience, an easy-to-navigate intuitive interface will be developed using the latest web technologies. The overall structure of the application will incorporate the MEAN stack development process. This process uses a MongoDB database for storing relevant website data (in this case, available or selected trips and all relevant properties). MongoDB is ideal for an application of this sort because it is very fast and very easy to implement using the other tools of the MEAN stack process. The NoSQL structure of MongoDB allows administrators to add properties to database entries without compromising the structure of the database as a whole. Additionally MongoDB allows for indexing which can significantly speed up the querying process for larger datasets.

For the front end, Angular will be used to develop a website that users can navigate easily once logged in. The front-end presentation will utilize Angular's page widgets that allow for a smoother interface within the page; some of these are dropdown menus, animations, and stylized buttons for submitting forms and updating information.

To connect the front to the back (Angular web pages to the database) the application will use node.js and the express framework. These frameworks will use Javascript and will make up the controller aspect of the application.

To make managing the website and its data easy for administrators, a backend site that utilizes the single-page application feature of Angular will be built alongside the frontend. This SPA will allow admin to easily add, remove, or update website content and information by providing a simple interface to interact with the database. Admin won't have to directly access the database via a third party application (like Compass). The backend will also logically handle all requests from the admin SPA to insert new data into the customer facing site and remove old data when requested to do so.

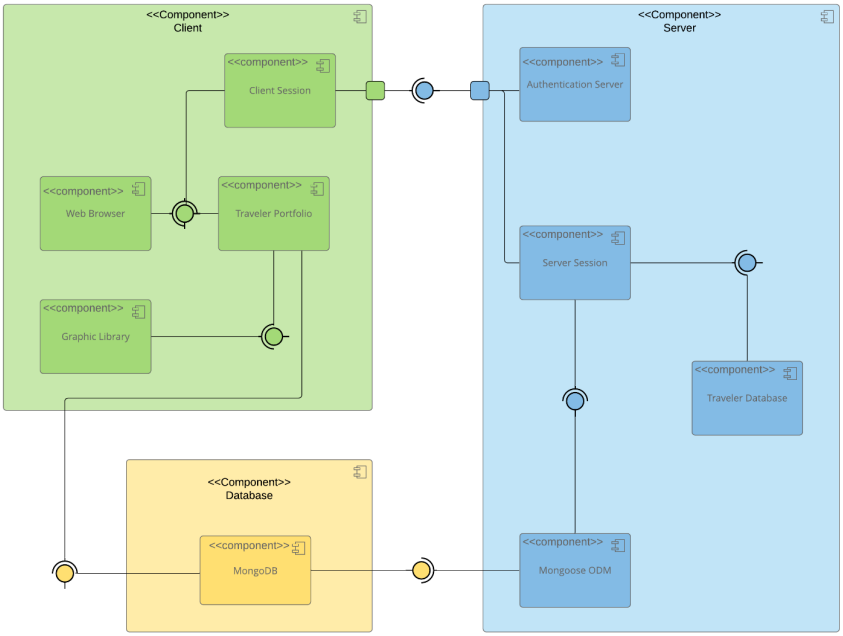
Design Constraints

When developing the website for "Travlr Getaways" and following the MEAN stack development process, there are a few things to consider. MongoDB (the 'M' in MEAN) is not designed for very large datasets, and the application can see some slow down if the database becomes too large. "Too large" typically means millions of documents, however, and therefore Travlr is unlikely to see this as an issue in the future.

Using node.js, express framework, and Angular can make a very intuitive and easy-to-use web interface, but the caveat with these is that the web application can contain large amounts of code and files. These can become cumbersome should the site continue to expand at a steady rate. This is not to say the code will be unmanageable, but this should be taken into consideration when assigning teams and product owners to the project.

Finally, once the MEAN stack is implemented, it can be very difficult to roll back the application to an "older" style of development. The integration of Angular and node.js makes this so, and it should be understood by the product owners that once the process has been completed the code will need to be maintained by teams that can continue working with these frameworks.

System Architecture View: Component Diagram



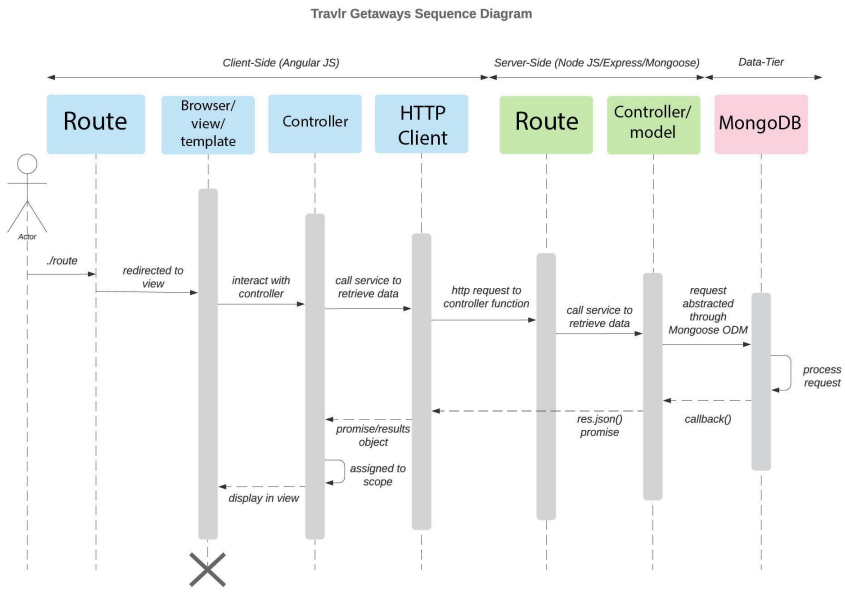
The client-side of the MEAN stack is run by Angular.js. The application will be displayed in a user's web browser, allowing them to view and log in for a more personalized experience. The Angular framework allows a developer to extend HTML tags with metadata to help them create a more dynamic, interactive web experience that is much more powerful than static HTML with javascript.

On the server-side of the stack, the user's information is validated and a server session is started (once authenticated). The server component handles the communication between the database and the front-end client-side, providing the latter with information to display in the view templates. The database (in this case, MongoDB) is accessed using the Mongoose middleware in the server component.

The database component consists of a MongoDB database that houses all information to be displayed in the web browser. This is a NoSQL database, consisting of JSON documents with no need for rigid relational schemas (as would be the case in a SQL database). MongoDB is fast and easy to scale.

With the Angular framework, developers can build a responsive and interactive front-end web application, complete with all the "bells and whistles" that users have come to expect from websites. For developers, Angular provides important components like validation, localization, and back-end service communication (What is the mean stack? 2022). The back-end service for this web application is then run with Express, running on the node.js server. The Express framework is easy to work with and provides powerful models for routing URLs and handling HTTP requests from the Angular front-end. Storing data for the application is done in the database, which is accessed through the node.js drivers and Mongoose middleware. The MongoDB stores the JSON documents that are passed to it from the Express server; these documents are created by the Angular front-end. Likewise, documents retrieved from the database are in JSON format, and passed upwards through Express to the Angular front-end, where they are displayed in the view templates.

Sequence Diagram



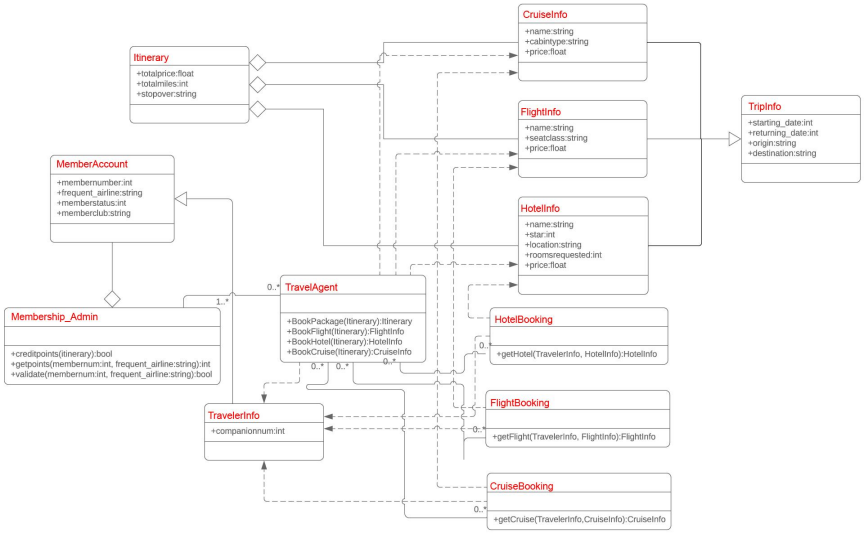
Referring to the diagram to the left:

A user types a route as a page address into the browser bar. The browser reads the route and forwards the request to the appropriate controller. The controllers are listed in the app.js file of the application, so the browser knows which routes are available to use, and where they can be found. The route directs the browser to the appropriate controller, which then gets the requested data from the models on the server side. The controller creates an HTML page that displays the requested data (for a generic web page, in this case) and returns it to the browser so the user can view it.

In the event that a browser makes a request for data that is housed on the database, a second controller for the API is employed to fetch and return this data. For instance, if the page in question lists data from the database in its view, the progression of displaying the html page follows as above. However, during the rendering of the page a call to a second controller is made, one that utilizes a middleware device to facilitate the fetching of the requested data. In the case of the above diagram, the middleware “mongoose” is used to get data from the MongoDB. This data, if it exists, is returned to the API controller (the second controller here) as either one or many JSON strings. This response from the API controller is forwarded to the HTTP client, which passes it back up the chain (along with the HTML website data) to the first controller, which then renders it to the browser.

Class Diagram

Referring to the diagram to the right:



Cruise, Flight, and Hotel-Info classes should all have the name, price, and specific attributes for the given type of booking as variables that are accessible from outside the class. These classes are derived from the superclass “TripInfo”, as logically an itinerary would include information about the Cruise, Flight, and/or Hotel a traveler is using for their trip. TripInfo class would also ideally include a starting date, return date, and other important general information about the trip as a whole. Since all of the “____Info” classes are a part of a trip's information, “Itinerary” is an aggregate of these classes.

All of the “_____Booking” classes (lower right-hand side) contain methods necessary to “book” their respective item (Cruse, Flight, and Hotel). These classes can be implemented in different ways by the TravelerInfo class and the “____Info” classes above.

The TravelAgent class in the center is the class that will access all others, so that it can book items and access the information necessary to view the trip information.

The Membership_Admin class (far left) is an aggregate of all MemberAccount classes. The MemberAccount class is the superclass to the TravelerInfo class, where the number of traveler companions can be accessed.

API Endpoints

Method	Purpose	URL	Notes
GET	Retrieve one trip by code	/api/trip/:tripcode	Returns a single trip object from the database using the “tripcode” as reference for which trip to retrieve
GET	Retrieve all trips	/api/trips	Returns all trips from the database
POST	Add a trip to the database	/api/trips	Adds a trip to the database
PUT	Update a single trip in the database	/api/trip/:tripcode	Updates a single trip object using inputs from an html form in Angular application
DELETE	Delete a single trip in the database	/api/trip/:tripcode	Takes a trip code and deletes that trip
POST	Register a new user	/api/register	Takes a name, email, and password and registers a user by placing them in the database after hashing the password
POST	Login a user	/api/login	Takes an email and password to login a user, returns a jwt

The User Interface
Main Log-in Page:

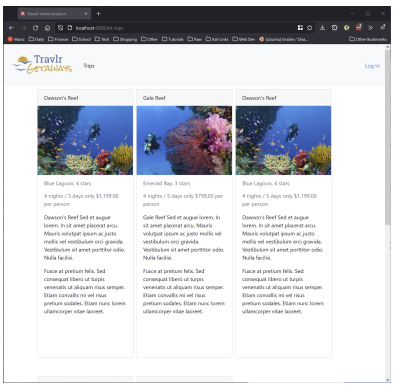


Fig 1: User is logged out

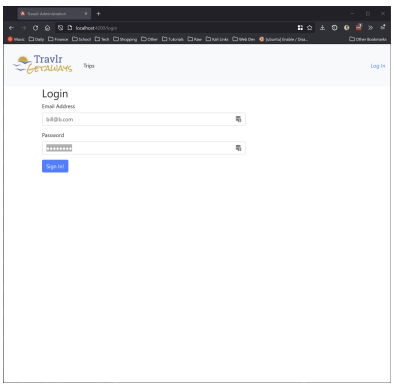


Fig 2: User logging in

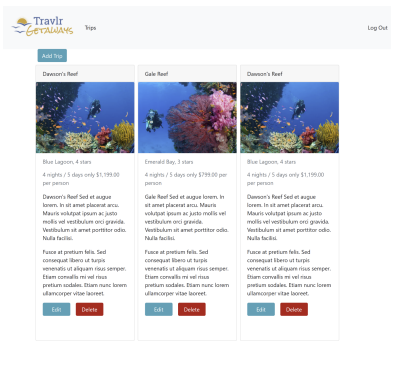


Fig 3: User logged in

Adding a Trip:

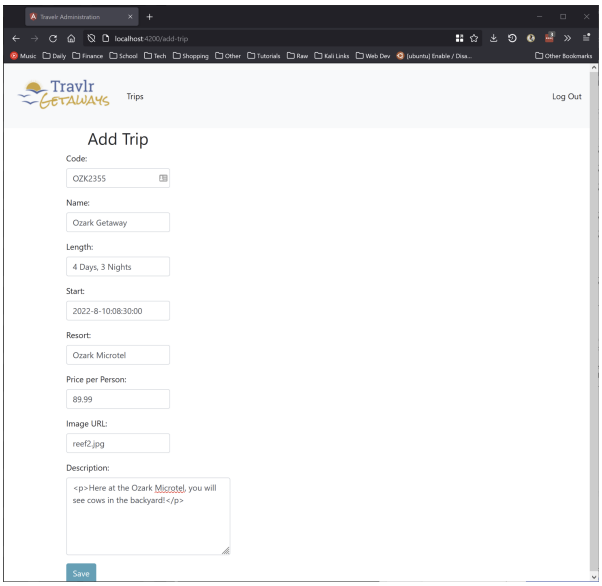


Fig 4: Adding a Trip

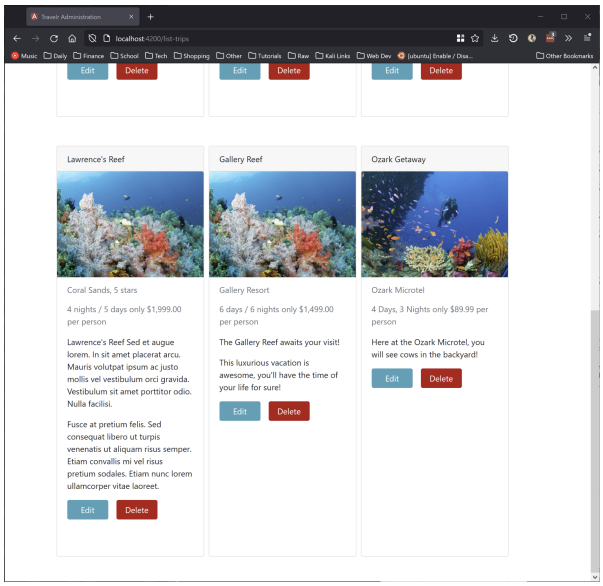


Fig 5: Trip Added

Editing a Trip:

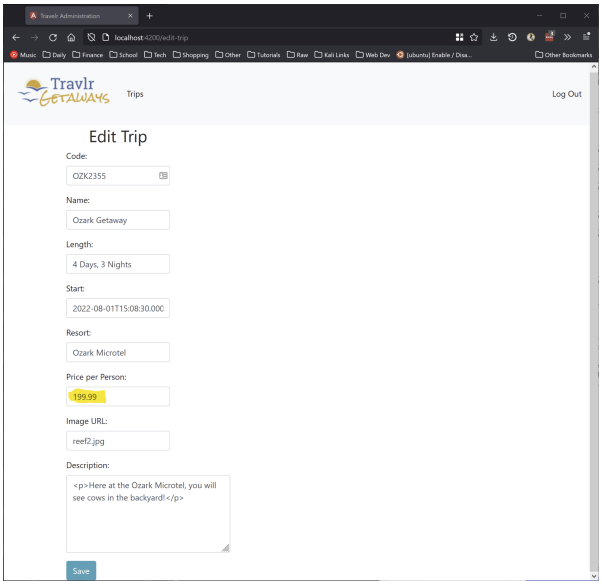


Fig 6: Editing a Trip

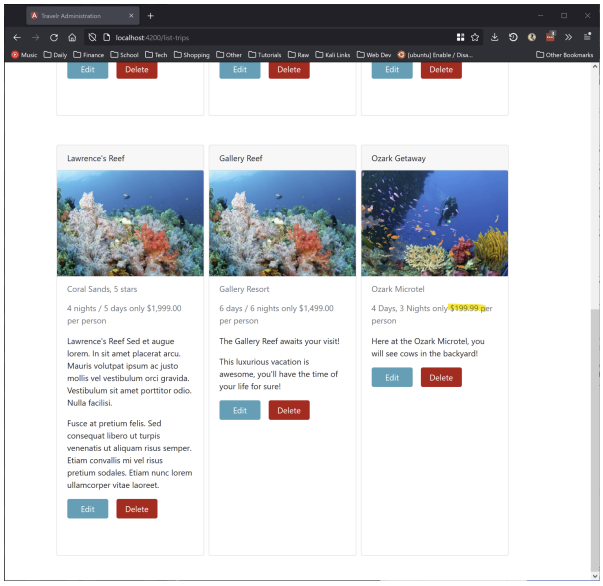


Fig 7: Trip Edited Successfully

Summarization of Angular Project Structure and the SPA

The backend application utilizes Angularjs, node.js, and MongoDB to access and display Trip information onto a single-page application for logged in administrative users. The complexity of the application is immense, so in the interest of concicity we will summarize the project here.

The Angular application allows for a single web page to serve all pertinent information to a user. Angular accomplishes this by making use of components, modular application parts that can be dropped into any part of the application html to be displayed on the page. Angular routing connects all components to one another through a service. As components call the service, they are routed to the appropriate call (PUT, POST, GET, etc), which is then sent to the API for processing. Unlike an Express application, which requires each page to have its own route and html page, the Angular SPA can manage multiple components through a single service. This allows all pages in the SPA to access necessary information quickly.

Angular also supports many other components through third party's like "bootstrap", including dropdowns, nav bars, and dynamic buttons. Because the components can be added to other components, Angular supports a modular structure, allowing developers to create interactive web-pages quickly and without rewriting or copying code. This also makes refactoring code much easier.

In order to test an application like this one it is important to ensure that routing to the API works first. To do this, Postman is used to quickly check all HTTP requests to the API, including logging in and retrieving data from the database. Once the routes are established, the application can be tested in a standard web browser with developer tools opened up. It is important to check for various errors as the application is running, and to test all potential areas of issue (mainly connections to the backend). Because Angular is based in javascript, using "console.log()" and ".json({"message":" "}) are extremely helpful in narrowing down the source of a bug or error. Debugging can also be done through Visual Studio with the installation of a few third party extensions.

References:

What is the mean stack? introduction & examples. MongoDB. (2022). Retrieved September 24, 2022, from https://www.mongodb.com/mean-stack