Travlr Getaways Web Application

CS 465 Project Software Design Document

Version 1.0

Table of Contents

Document Revision History

Version	Date	Author	Comments
0.1	01/22/22	Craig O'Loughlin	Added Executive Summary, Design Constraints, and Overview
0.5	02/05/22	Craig O'Loughlin	System Architecture View / API Endpoints
1.0	02/19/22	Craig O'Loughlin	Completed API Endpoints and User Interface description

Executive Summary

The Travlr Getaways web application is a website where users can browse and book travel with the client agency. Users may also create and access a personal account through which they can provide booking details and thereafter view itineraries. There is also a single-page application component where Travlr can update the website with new information as necessary.

As a web application, this project is built to be fully accessible to a user through the internet, either customer or administration. The primary components of this project are:

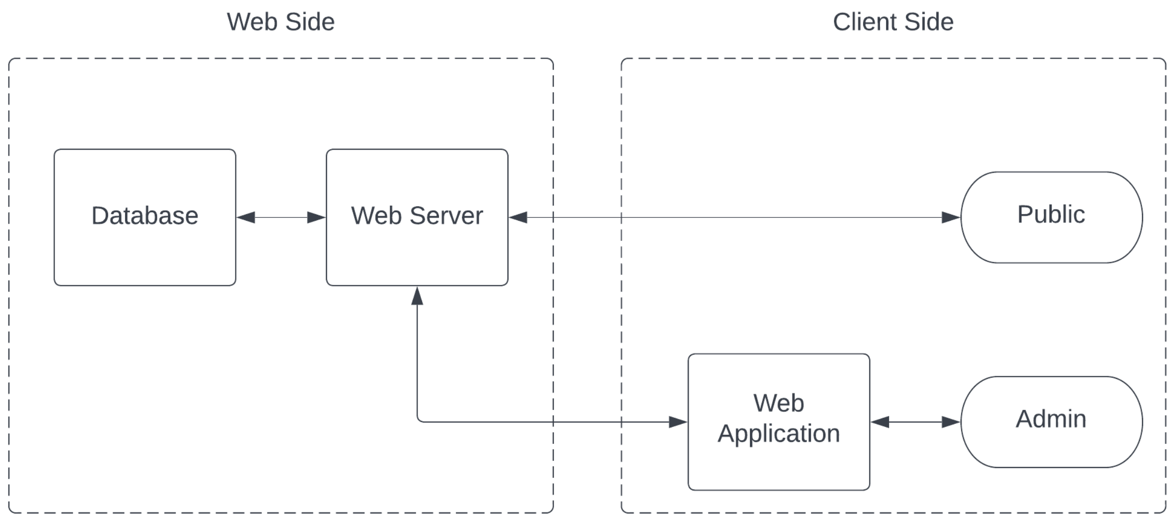
- A database, where travel and customer information are physically stored
- A web server, which can service users who access this application through the internet
- A web application framework, to create a web-accessible admin application

We have chosen a set of reliable, modern, and well-supported frameworks that work well together to support the development of this project. We are using the MEAN tech stack:

- MongoDB *(database)*
- Node.js with Express *(web server framework)*
- Angular *(web application framework)*

This stack has been successfully employed by many companies, is well-established in the field, and provides a consistent development environment from front to back [1].

The web application system has two front-facing access points: a **public website** and an **administration application**, which are supported by a single backend framework for efficiency. A brief overview of the top-level architecture of this project is provided:



The **public website** features a set of pages where users can login, browse travel, and book travel. They can also view active and future itineraries for their travel. The **admin channel** is a single-page application which can be accessed with an administration login and allows modifications to the public website and provides management tools.

Design Constraints

Identifying and documenting design constraints allows us to properly scope development and the project outcome. By doing so we assure that we will neither miss key features nor provide unnecessary functionality.

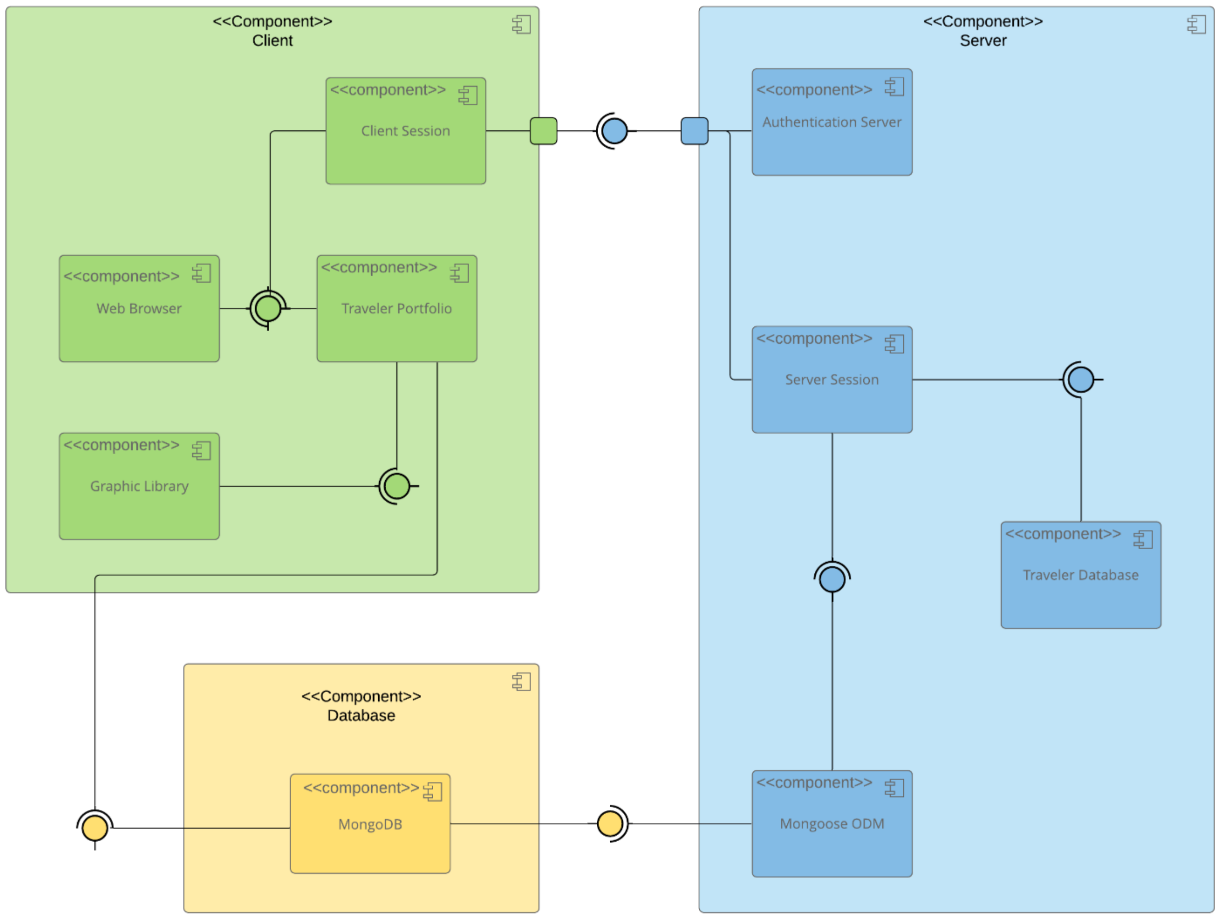
The following constraints frame the project development:

- Final price of development must fall within the agreed upon budget, including maintenance and serving costs.
- System must be publicly accessible through a computer or other web-connected device with no specific requirements other than use of the latest supported web browsers and operating systems.
- Administration single-page application accessible only with proper authentication.
- Public website must provide secure login access when provided with proper credentials.
- Server / Database must conform to industry-standard safety and privacy requirements as it is handling potentially sensitive user information.
- System is accessible with uptime 99.9% or greater ("standard availability" [2]).
- Users must have an account in order to book travel and view itinerary, otherwise they must be allowed to create one.

System Architecture View

Component Diagram

The below diagram presents the Travlr Getaways web system architecture in further detail:



Functional components are divided into Client, Server, or Database and represent the key processes that occur in the system. The **client** processes are addressed on the local user computer session, **server** processes occur on the Travlr web server, and **database** processes (of which there is one) run on the database, which is a separate entity from the server for increased security and performance.

Each process is detailed on the following page.

Client

Web Browser: Manages data presentation and UI components for the user. This is a standard browser such as Edge or Firefox that renders server data into a readable format and sends/receives HTTP requests.

Client Session: The user browser and server work together to create an authenticated access session for the client. Marked with a unique session ID, all interactions are processed through this channel and logged by the server.

Graphic Library: Client cache for public server data such as images and logos. May also refer to general graphic libraries that may be used by a JavaScript-powered frontend such as WebGL.

Traveler Portfolio: Managed through the client session, the traveler portfolio should contain received information relevant to the user including itinerary or account details.

Server

Authentication Server: Handles authentication (login) requests from the client and may respond with a session access or disallow as needed. All requests should be logged here.

Server Session: Since the server may handle many requests, each unique server session for an authenticated client must be stored. The server can then properly route requests to information in the database as needed.

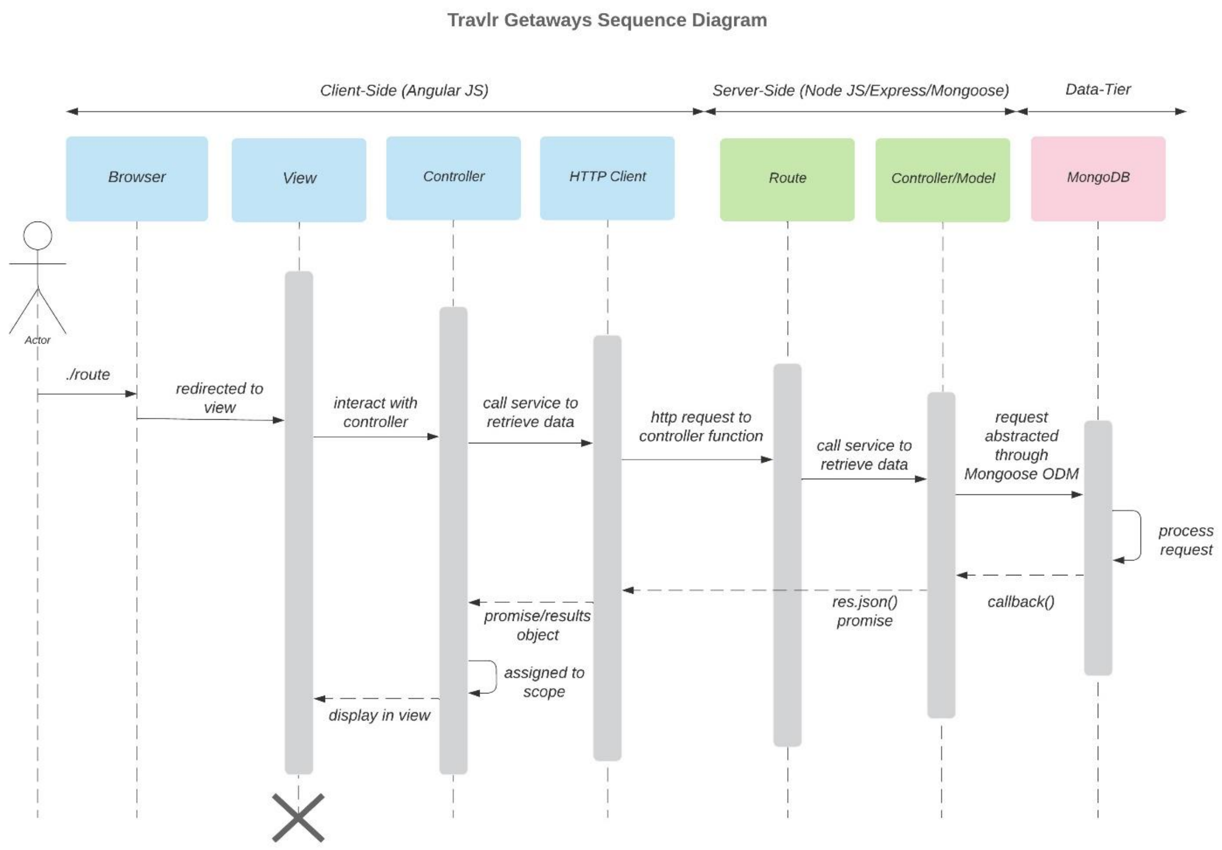
Traveler Database: A server-cached version of relevant database information available for the specific user, so the user may view, add, or remove information. The server can store this information on request or at the end of a session.

Mongoose ODM: Middleware for convenient, safe, and ordered access to information on the actual database, built specifically for Node.js development.

Database

MongoDB: The database is implemented using MongoDB, which provides a flexible data storage scheme, storing data essentially in JSON format, and a uniform access API in Javascript that mirrors the design pattern of the rest of the system.

Sequence Diagram

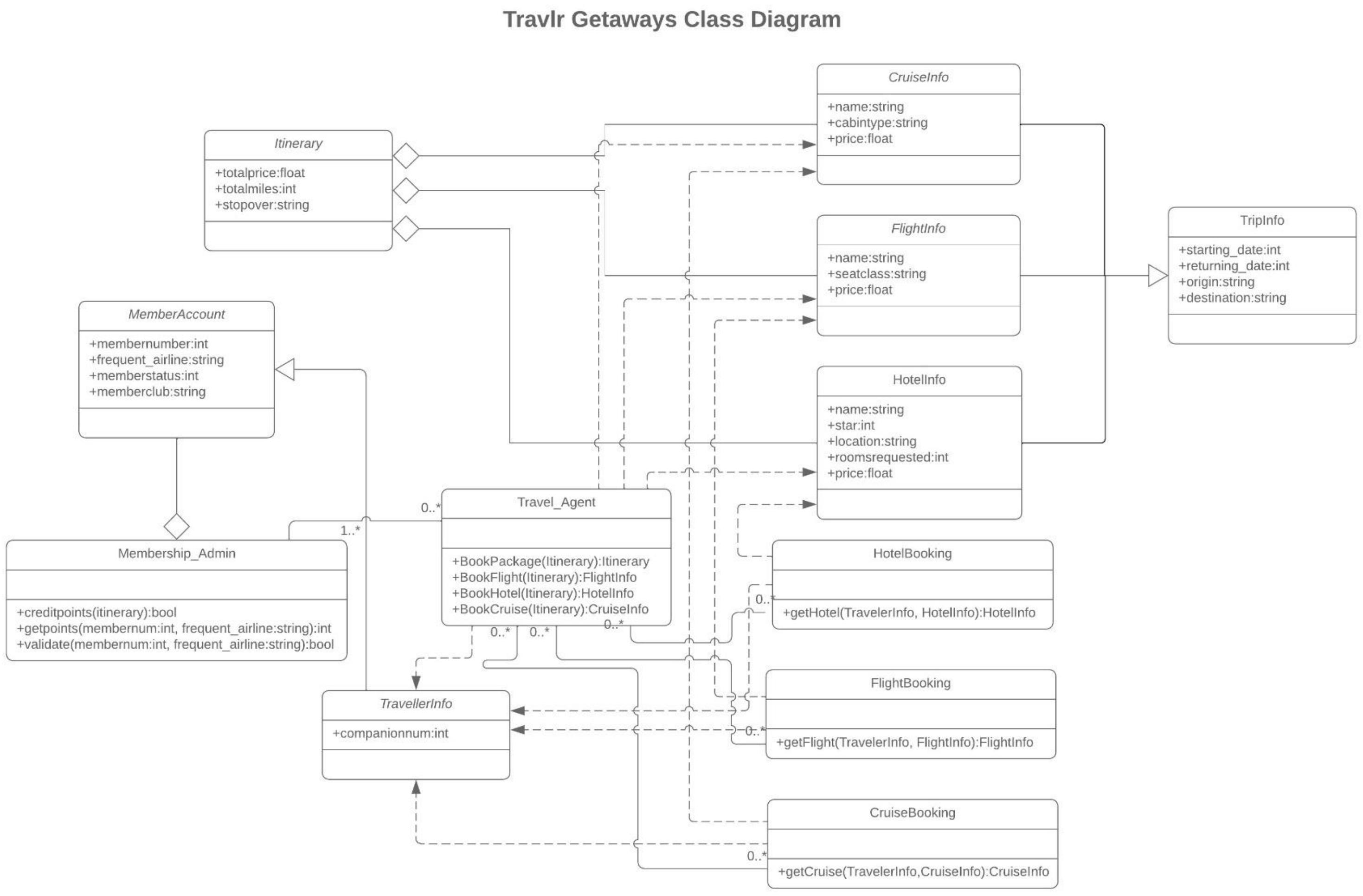


The sequence diagram illustrates how commands and their corresponding responses travel through the system for any session. The session always starts with the user and a web browser, through which the client side SPA view/controller is accessed. It is through this controller the user may sign-in and make requests to view or modify appropriate data. The controller contains no data itself to verify or provide such information (this would be a major security risk and would make managing business data near impossible), so must pass all data requests to the server instead. The HTTP client is used to forward data requests in a standard format for the server to process.

The server-side application contains a router, which sends commands to the appropriate service. Each service individually makes up the Controller/Model, which contains the business logic to carry out the requests from the client. This may include validating login information or making requests for travel data.

The endpoint and final layer of the application is the data layer, a database implemented in MongoDB and abstracted through Mongoose. This is where application data lives. Requests are processed by the database engine and returned to the server, which will forward the information back to the client side.

Class Diagram



The class diagram illustrates how each individual service is organized and interacts with each other to create a complete booking system. Each service is detailed on the following page.

Trip Info / Itineraries

There is a single TripInfo base class that is extended to support specialized information for either a cruise, flight, or hotel. Each of these classes contain all the information pertaining to an individual trip segment. Further, there is an Itinerary class which can aggregate one or multiple cruises, flights, or hotels in a single accessible package.

Travel Agent / Bookings

The system class design revolves around the Travel Agent, a central service that performs booking services given information on a traveler and details on what to book. Each detailed booking is held in either a CruiseBooking, FlightBooking, or HotelBooking class, which implements the associated travel information and contains the necessary logic to translate such information into booking orders.

Member Accounts / Administrative

Each traveler, as represented by an individual TravellerInfo class, must have a MemberAccount. The MemberAccount class contains the details of each membership and status. These accounts can be verified through the Membership Admin service, which can validate information and update reward points among others.

API Endpoints

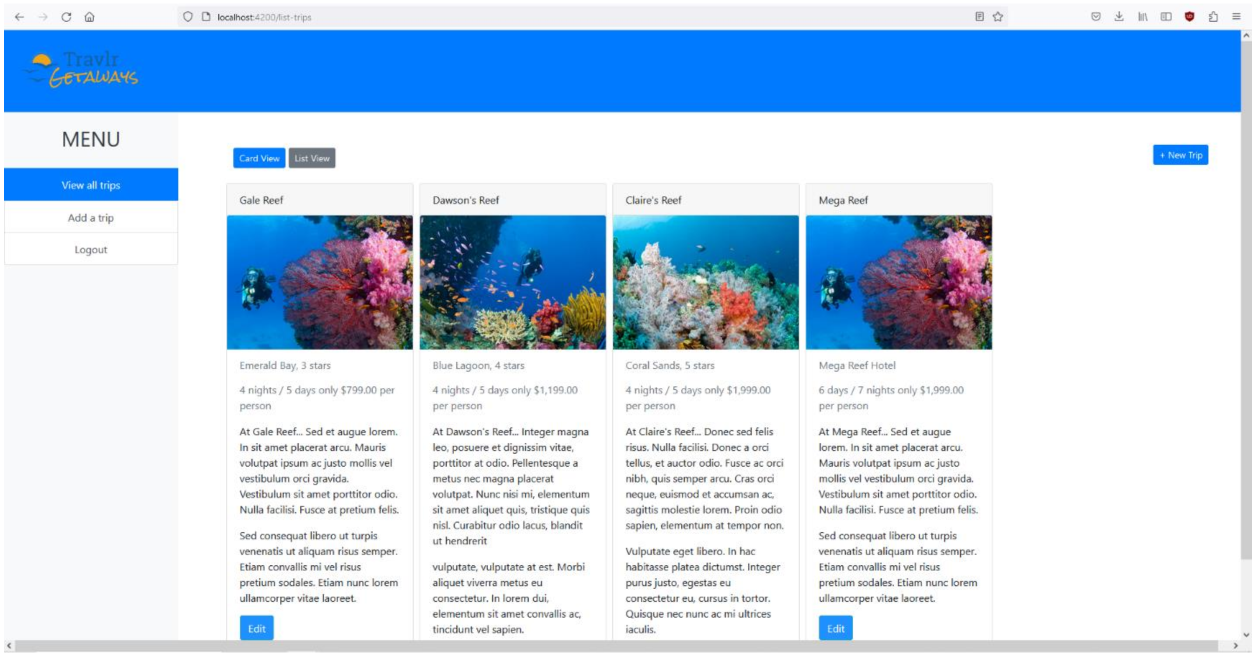
The server application implements a uniform access API accepting HTML requests. This allows consistent and predictable access to resources from any number of clients and their associated architecture. The API is summarized on the following pages and further documented at the **/api-docs** URL.

Method	**Purpose**	**URL**	**Notes**
GET	Retrieve all trips currently active in the database	/api/trips	Returns a list of trip documents ordered in JSON format, and an HTTP status OK (200). If the database could not be reached or something else went wrong, returns no document and HTML status NOT FOUND (404).
GET	Retrieve a single trip by identifier code. The :code segment is replaced with the identifier code.	/api/trips/code/:code	Returns a trip document in JSON format, and an HTTP status OK (200). If nothing was found, the document is empty. If the database could not be reached or something else went wrong, returns no document and HTTP status NOT FOUND (404).
POST	Add a single new trip to the database	/api/trips	***REQUIRES AUTHORIZATION VIA BEARER TOKEN*** Send new trip data as JSON in request body. Returns JSON data of the trip added and HTTP status 201 if successful. Otherwise, returns status 400 and error message.

PUT	Update a single trip currently on the database. The :code segment is replaced with the identifier code.	/api/trips/code/:code	***REQUIRES AUTHORIZATION VIA BEARER TOKEN*** First request a trip via GET, then send it back with updated info in JSON format in request body using this call. Returns HTTP code 204 on success, code 404 if no trip was found for that code or invalid content, or code 500 if a server error occurred.
POST	Register a new account to the user database.	/api/register	Send name, email, and password information as JSON in request body. Returns a new timed bearer token (JWT format) and HTTP code 200 on success. Returns code 401 for invalid information, and 404 for connection errors.
POST	Login to an existing account on the database.	/api/login	Send email and password information as JSON in request body. Returns a new timed bearer token (JWT format) and HTTP code 200 on success. Returns code 401 for invalid credentials, and 404 for connection errors.

The User Interface

Admin Portal SPA (Angular Client) – Trip View – Card View



Gale Reef

Emerald Bay, 3 stars

4 nights / 5 days only $799.00 per person

At Gale Reef... 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.

Edit

Dawson's Reef

Blue Lagoon, 4 stars

4 nights / 5 days only $1,199.00 per person

At Dawson's Reef... 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 $1,999.00 per person

At Claire's Reef... 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.

Mega Reef

Mega Reef Hotel

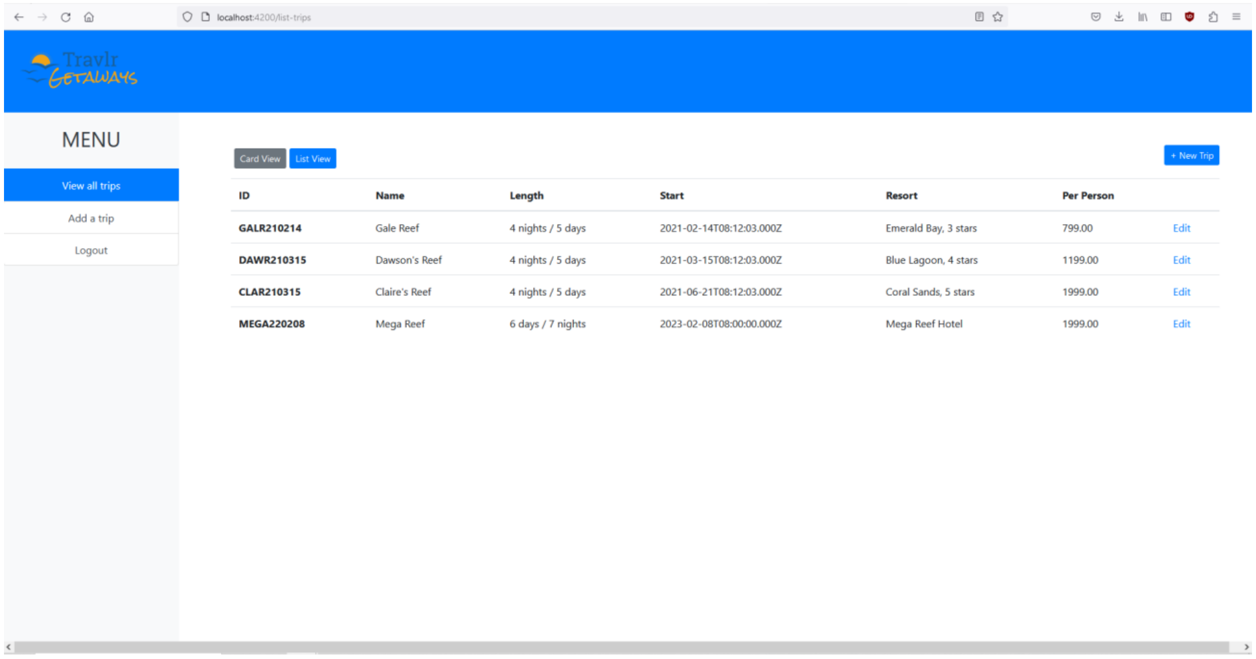
6 days / 7 nights only $1,999.00 per person

At Mega Reef... 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.

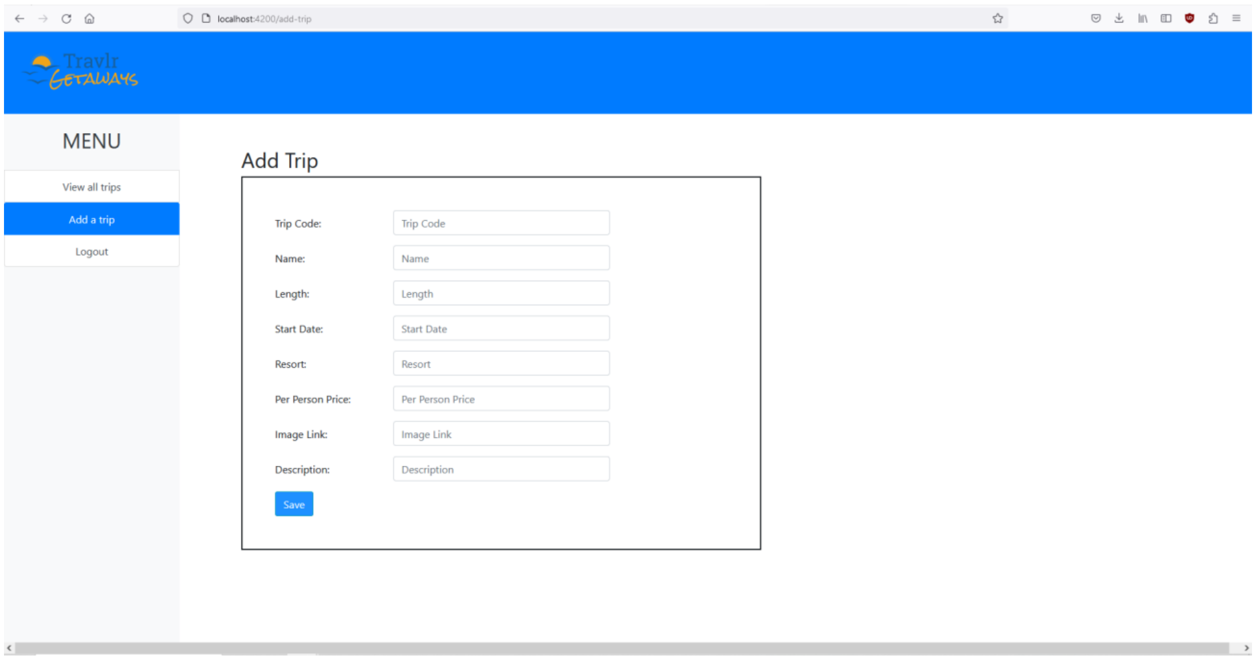
Edit

Admin Portal SPA (Angular Client) – Trip View – List View



ID	Name	Length	Start	Resort	Per Person	
GALR210214	Gale Reef	4 nights / 5 days	2021-02-14T08:12:03.000Z	Emerald Bay, 3 stars	799.00	Edit
DAWR210315	Dawson's Reef	4 nights / 5 days	2021-03-15T08:12:03.000Z	Blue Lagoon, 4 stars	1199.00	Edit
CLAR210315	Claire's Reef	4 nights / 5 days	2021-06-21T08:12:03.000Z	Coral Sands, 5 stars	1999.00	Edit
MEGA220208	Mega Reef	6 days / 7 nights	2023-02-08T08:00:00.000Z	Mega Reef Hotel	1999.00	Edit

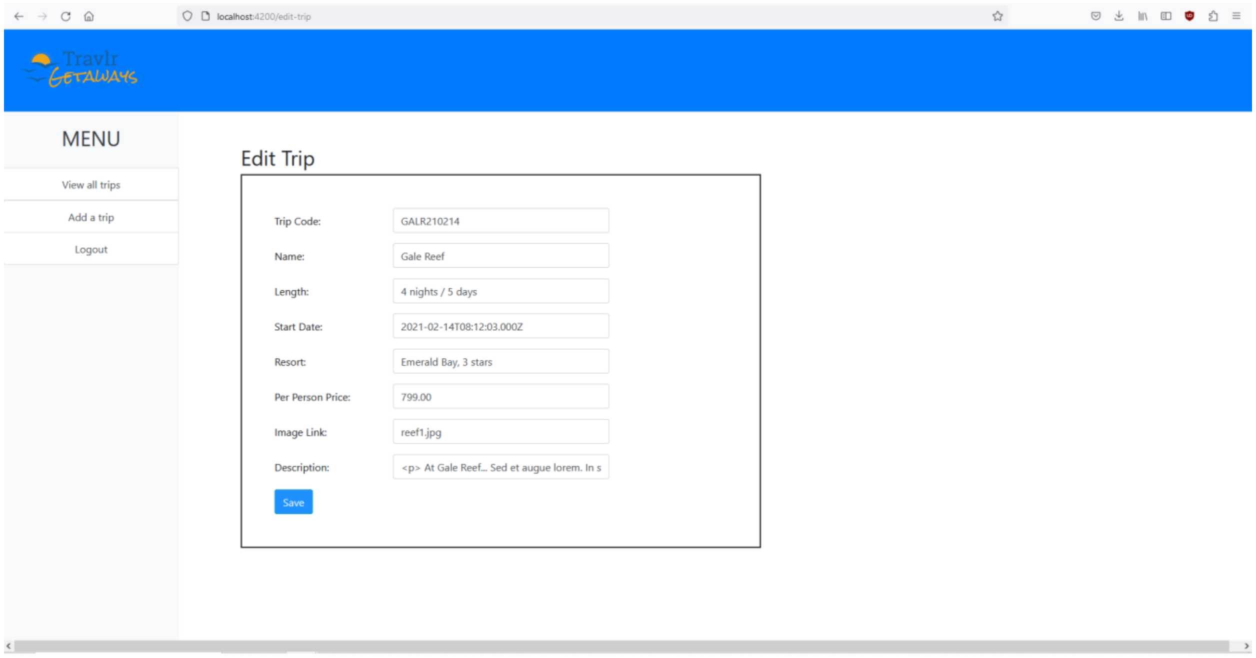
Admin Portal SPA (Angular Client) – Add Trip



Trip Code:	Trip Code
Name:	Name
Length:	Length
Start Date:	Start Date
Resort:	Resort
Per Person Price:	Per Person Price
Image Link:	Image Link
Description:	Description

Admin Portal SPA (Angular Client) – Edit Trip

Trip data is prepopulated based on selected trip to edit



Trip Code:	GALR210214
Name:	Gale Reef
Length:	4 nights / 5 days
Start Date:	2021-02-14T08:12:03.000Z
Resort:	Emerald Bay, 3 stars
Per Person Price:	799.00
Image Link:	reef1.jpg
Description:	<p> At Gale Reef... Sed et augue lorem. In s

Overview

The user interface for the public facing travel/booking service is a 'traditional' multi-page website built on Express. The Express server provides each page on request as the user navigates, meaning that no page takes very long to load altogether, and the user can navigate with steady, consistent timing between pages. It allows us to use a combination of HTML, CSS, and JavaScript to deliver an attractive UI and a templating engine to provide dynamic content without having to change the view structure.

The user interface for the admin portal is a single-page application (SPA) built on Angular. Angular makes it easy to package an entire multi-page application in a single delivery as soon as the application is started. For a small upfront cost in loading time, the rest of the application can be navigated as seamlessly as any desktop or smartphone application. This lends itself well to applications where the user is engaging for a longer period of time, such as in this case.

Angular provides a suite of tools that allow us to easily build a responsive, modern UI with consistent view elements. Angular allows us to change parts and pieces of an application UI on command using JavaScript functionality. Further, we use the Bootstrap CSS library to provide additional UI elements and tools such as table views for data.

The single page application in Angular uses the same Express / Node.js backend for authenticating users and for data transactions. This means that the SPA uses the API that was built on the Express server. Before building the SPA to use these API endpoints, they were thoroughly tested with reliable third-party HTTP software such as Postman. This gives us a chance to isolate bugs before they become a problem for the SPA. After the SPA HTTP client was built for this API, it itself is then tested for correctness both in cases of normal operation (such as adding and editing trips) and in cases of abnormal operation (such as cases where there is no database connection or using invalid credentials).

References

[1] Souvik. (2022, Feb 2). *Famous companies using mean tech stack for development.* RS Web Solutions. https://www.rswebsols.com/tutorials/programming/mean-tech-stack-development

[2] Leslie, A. (2022, Aug 23). *What is an uptime guarantee? Can hosts really stay online forever?.* MUO. https://www.makeuseof.com/what-is-uptime-guarantee/