

Travlr Getaways

# **CS 465 Project Software Design Document**

Version 1.2

## Table of Contents

[**CS 465 Project Software Design Document** 1](#_Toc36198462)

[Table of Contents 2](#_Toc36198463)

[Document Revision History 2](#_Toc36198464)

[Instructions 2](#_Toc36198465)

[Executive Summary 3](#_Toc36198466)

[Design Constraints 3](#_Toc36198467)

[System Architecture View 3](#_Toc36198468)

[Component Diagram 3](#_Toc36198469)

[Sequence Diagram 4](#_Toc36198470)

[Class Diagram 4](#_Toc36198471)

[API Endpoints 4](#_Toc36198472)

[The User Interface 4](#_Toc36198473)

## [Document Revision History](#_heading=h.lnxbz9)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 01/25/24 | Jeremy Baldner | Template received. Updates to the Executive Summary, Design Constraints, and System Architecture View: component diagram. |
| 1.1 | 02/11/24 | Jeremy Baldner | Sequence and Class diagram complete. API endpoints complete. |
| 1.2 | 2/23/24 | Jeremy Baldner | Updated content to better reflect the application. Screenshots of final project included. |

## [Executive Summary](#_heading=h.35nkun2)

To create an optimal application for Travlr Getaways, two different architectures will be required. The customer-facing side of the application will use URL routing and model-view-controller (MVC) patterns to create a multi-page application. This allows for faster first page loads and improved SEO analytics. The administrator side of the application will utilize a single-page application (SPA) architecture. This will be feature rich, private, and provide a highly interactive environment to meet the requirements of site maintenance.

Both architectures will utilize the MongoDB, Express, Angular, and Node.js (MEAN) stack. MongoDB will be the database that will hold the data for the website and customers. Express is a backend framework that will create the application programming interface (API). The API allows for communication between the website and the database. Angular is a framework used to create dynamic webpages for the SPA. Node.js is used to control the server that the website will run on.

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

* The design of Travlr Getaways will require two separate architectures which means more time will be required to develop this application.
* Since Angular is part of the MEAN stack, its usage will need to be limited to promote proper Google Analytics and SEO ranking of the customer-facing website (Holmes & Harber, 2019).
* SPAs are feature rich, but the initial load time will be slower than multi-page applications since it must load the content for the entire application.
* The MEAN stack does not automatically maintain user sessions and JSON Web Tokens will need to be used for accurate authentication (Holmes & Harber, 2019).
* Usage of Express and Angular requires Node.js. Node.js has frequent updates with their long-term support typically lasting 30 months (Node.js, 2024). Frequent updates to the Node.js framework may result in more technical debt as the application ages.

## [System Architecture View](#_heading=h.44sinio)

### Component Diagram



A text version of the component diagram is available: [CS 465 Full Stack Component Diagram Text Version](https://learn.snhu.edu/d2l/lor/viewer/view.d2l?ou=6606&loIdentId=24342).

The database component contains MongoDB from our MEAN stack. This database provides an interface to the traveler portfolio and Mongoose object-document mapper (ODM).

The client component contains the client session, web browser, traveler portfolio, and graphic library. The traveler portfolio requires interface from MongoDB and the graphic library to receive data. The web browser and the client session require interface from traveler portfolio to receive data. The client component requires interface with the entire server component to display appropriate user webpages.

The server component contains the authentication server, server session, traveler database, and mongoose ODM. Mongoose ODM requires interface with MongoDB and provides interface to server session. The server session requires interface from Mongoose ODM and traveler database. The server component provides interface to the client component.

### Sequence Diagram

A diagram with different colored squares

Description automatically generated

On the client-side, the user will send a path request to the app’s router. The route will be sent to the template/view to create the view. The view interacts with the controller to determine the HTTP request to send to the RESTful API. The API is located in Express on the server side.

Sever-Side routes the request to the controller. The controller submits the request to the MongoDB database. The request is processed, and data is returned. The data is sent to the client-side view/template. The amount of data is determined by the scope of the account. The view is rendered in the application.

## Class Diagram

A diagram of a company

Description automatically generated

**Travel Agent** class has four public methods: book a package, book a flight, book a hotel, and book a cruise. It has an association with the classes: Membership Admin, Hotel Booking, Flight Booking, Cruise Booking. It is implemented from the classes: Cruise Info, Flight Info, Hotel Info, and Travel Agent.

**Membership Admin** class has three public methods: credit points, get points, validate. It has an association with Itinerary.

**Member Account** class has four public variables: member number, frequent airline, member status, and member club. It is aggregated with the Membership Admin class. Member Admin is required to have data for the Member Account class.

**Traveler Info** class has one public variable named companion number. It inherits from the Member Account class.

**Itinerary** class has three public variables: total price, total miles, stop over. It is aggregated with the classes: Cruise Info, Flight Info, Hotel Info. An aggregated class is required to have data for the Traveler Info class.

**Cruise Info** class has three public variables: name, cabin type, price. It inherits from the Trip Info class.

**Flight Info** class has three public variables: name, seat class, price. It inherits from the Trip Info class.

**Hotel Info** class has five public variables: name, star, location, rooms requested, price. It inherits from the Trip info class.

**Hotel Booking** class has one public method called get hotel. It is implemented from the Hotel Info and Travel Agent classes. It has an association with the Itinerary class.

**Flight Booking** class has one public method called get flight. It is implemented from the Flight Info and Travel Agent classes. It has an association with the Itinerary class.

**Cruise Booking** class has one public method called get cruise. It is implemented from the Cruise Info and Travel Agent classes. It has an association with the Itinerary class.

**Trip Info** is a parent class that has four public variables: starting date, returning date, origin, and destination.

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

| **Method** | **Purpose** | **URL** | **Notes** |
| --- | --- | --- | --- |
| **GET** | Retrieve list of trips | ./api/trips | Returns all the trips found in the database. If no trips are found, it returns an error. |
| **GET** | Retrieve one trip | ./api/trips/code | Returns a trip that matches the code sent to the database. If no trip is a match, it will return an error message. |
| **POST** | Login | ./login | Verifies email and password. |
| **POST** | Register | ./register | Creates an account in the users database. |
| **POST** | Add trip | ./api/trips/code | Creates a trip in the trips database. |
| **PUT** | Updates a trip | ./api/trips/code | Updates a trip in the trips database. |
| **DELETE** | Delete a trip | ./api/trips/code | Deletes a trip in the trips database. |

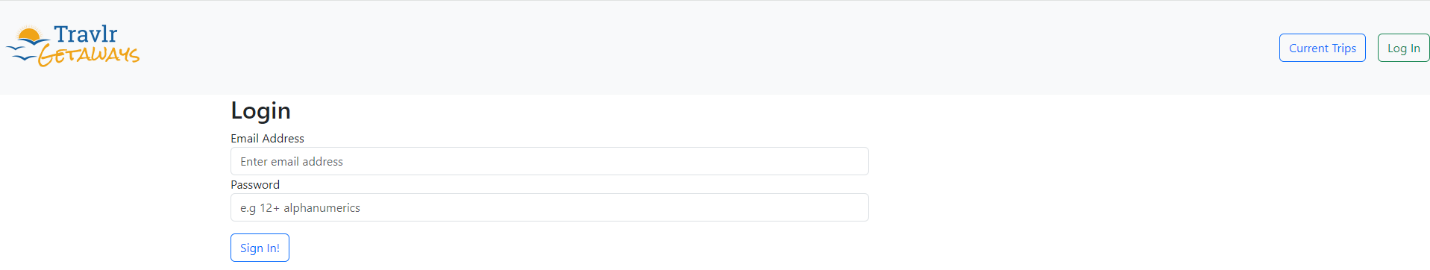
## The User Interface

Initial Page:

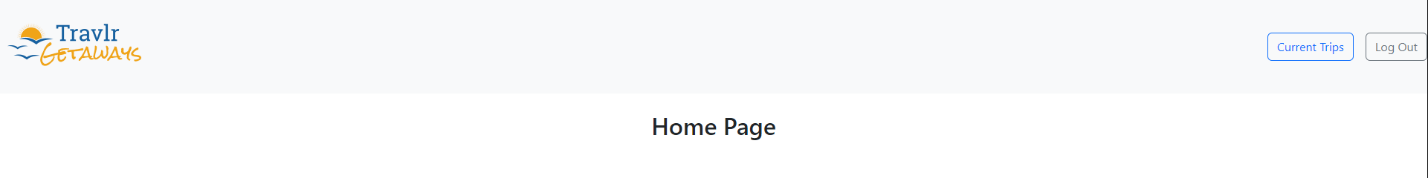
A screen shot of a computer

Description automatically generated

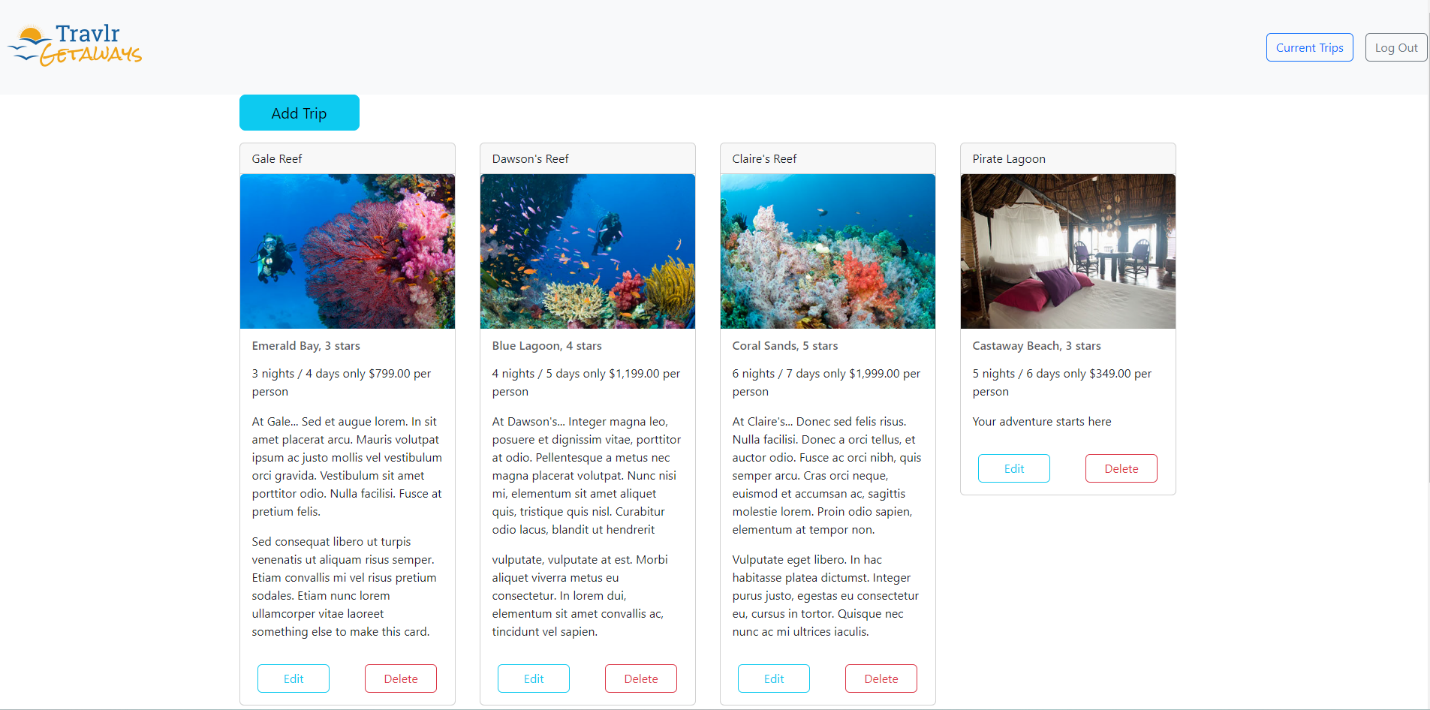
Login Page:



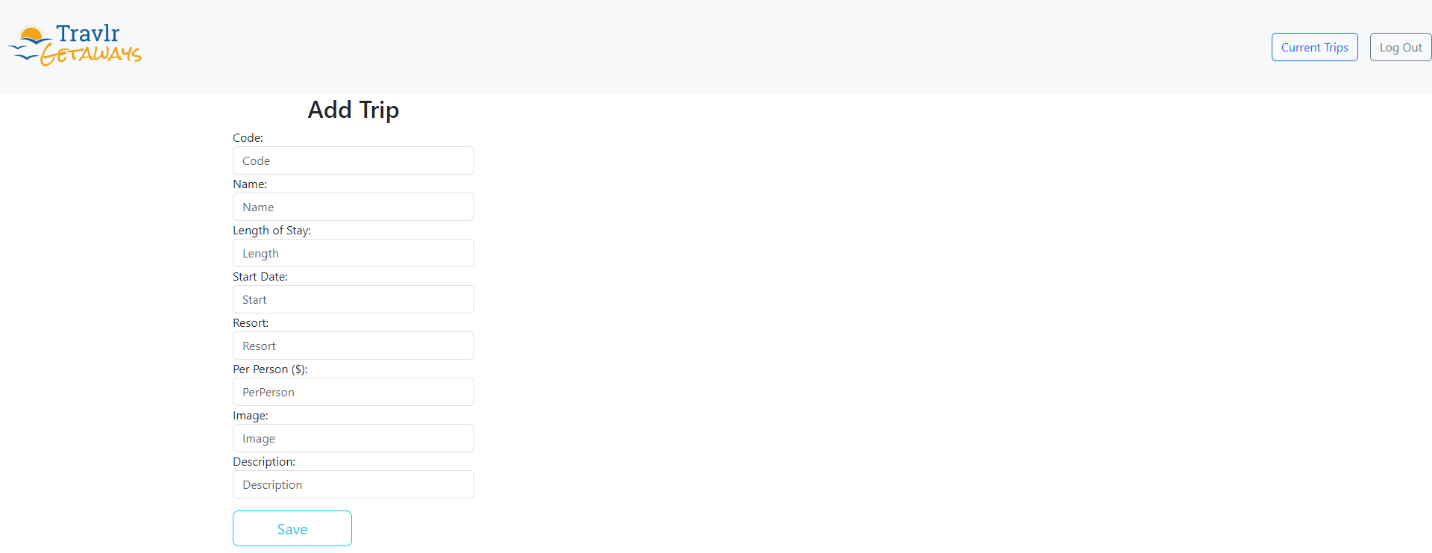
Home Page (After Login):



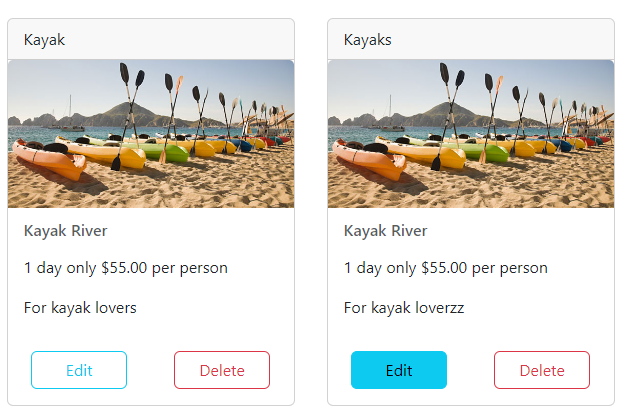
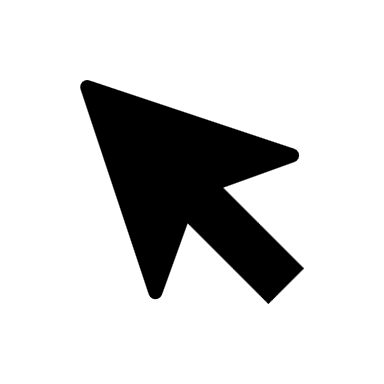
Current Trips Page:



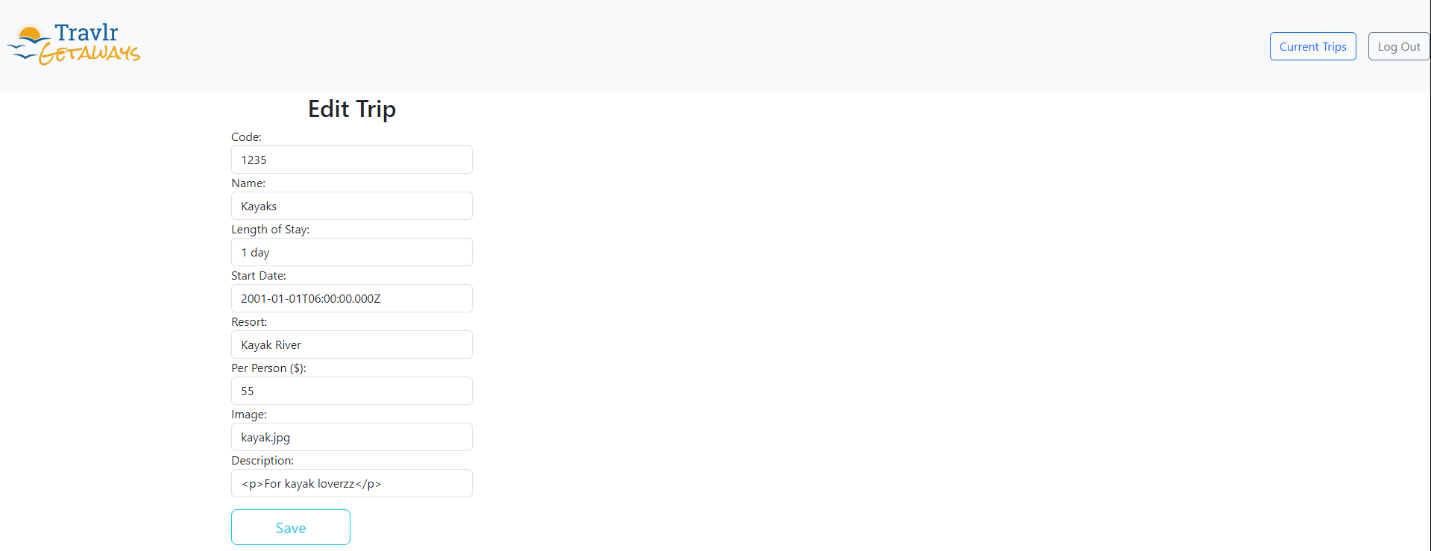
Add Trip Page:



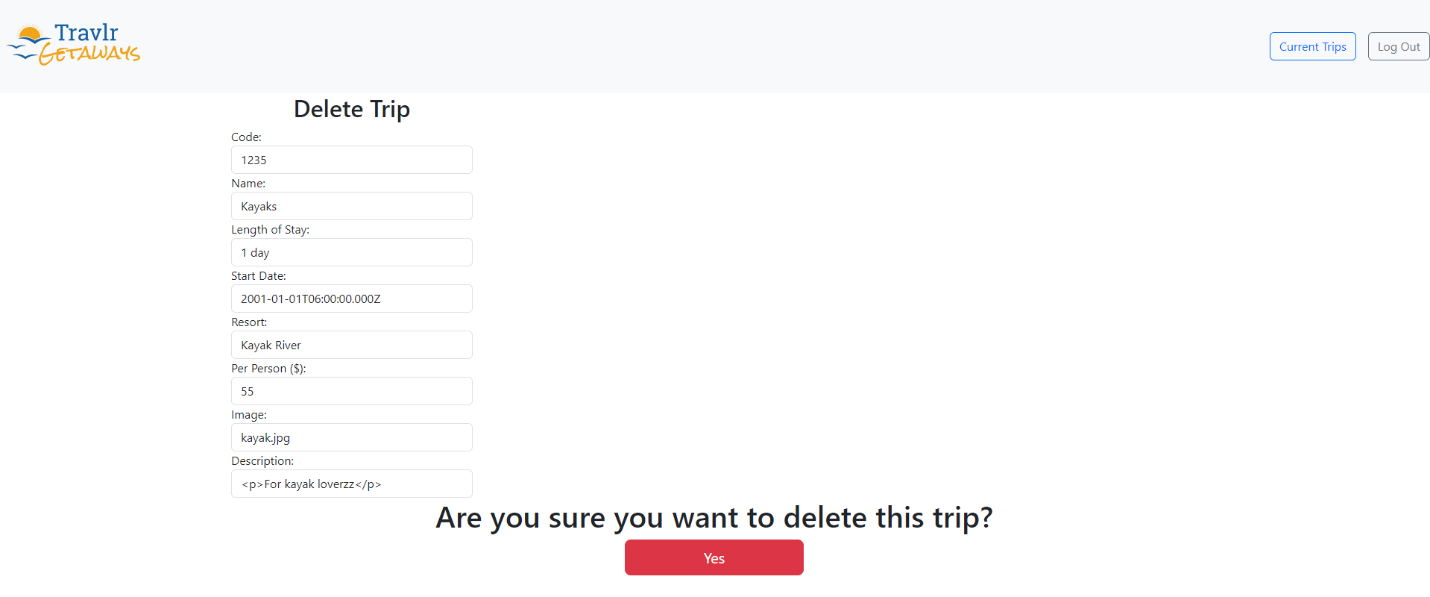
Edit and Delete Options:



Edit Trip Page:



Delete Trip Page:



The Angular project structure involves html, css, and TypeScript for each component. Each component utilizes a data service and router files to receive data and render it on the appropriate view. We utilized a front-end toolkit, Bootstrap, and did not need to write with javascript or css. Angular was much more complex to connect and debug.

The Express project structure involved css, handlebars and JavaScript. Each component was setup in the MVC pattern which made it easier to manage. The css had to be written. Express was less complex to connect and debug.

SPAs provide dynamic functionality to webpages for an interactive and fast environment. Once the application is loaded, the response times will be much quicker than an MPA. Also, since SPAs are sent to the client, they reduce the load on the server.

At this current time, SPAs are typically ranked worse in SEO rankings due to the difficulty to crawl. Since the SPA needs to load all the data onto the client at first load, the initial load time may be slower. Also, since the entire application is loaded, there are increased security risks due to possible reverse engineering (BasuMallick, 2022).

When testing the SPA, we used integration testing with Postman and user acceptance testing to verify that the website performed as expected. POST was tested by submitting valid and invalid requests to the API. This verifies that the logic inside the controller worked properly. Initially, there were issues with token verification and the code needed to be refactored. Ultimately, all POST tests returned the expected code. GET was tested by verifying the data matched the application output on the front end. PUT was tested by adding and editing data from the SPA. The Chrome developer tool and CLI were utilized to verify the correct http response codes were returned.

I ran into errors with the PUT request sending an error status after attempting to add a trip. This was due to typos in the edit-trip.component.html formControlNames. I would expect errors from the connections between the TypeScript components, data service, and API router. The complex connections between the files can increase the likelihood of errors. Making sure to output consult messages was helpful to determine the location of errors that occurred.

# References

BasuMallick, C. (2022). *What Is a Single-Page Application? Architecture, Benefits, and Challenges*. Retrieved from Spiceworks: https://www.spiceworks.com/tech/devops/articles/what-is-single-page-application/#:~:text=SPAs%20are%20more%20prone%20to%20getting%20hacked%20through%20cross%2Dsite,finding%20vulnerabilities%20through%20reverse%20engineering.

Holmes, S., & Harber, C. (2019). *Getting MEAN with Mongo, Express, Angular, and Node, Second Edition.* Manning Publications.

Node.js. (2024). *Previous Releases*. Retrieved from Node.js: https://nodejs.org/en/about/previous-releases