

Travlr Web Application

# **CS 465 Project Software Design Document**

Version 3.0

## 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 | 07/15/2023 | Jonathan Boeglin | Added content to Executive Summary, Design Constraints, and System Architecture View sections. |
| 2.0 | 7/30/2023 | Jonathan Boeglin | Added content to the Sequence Diagram, Class Diagram, and API Endpoints sections. |
| 3.0 | 8/13/2023 | Jonathan Boeglin | Added User Interface section and updated API Endpoints. |

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

This project is composed of a website for customers, a single-page application (SPA) for administration, and a server-based backend for data storage and computation, using MongoDB, Express, Angular, and Node.js (MEAN) for development. MongoDB will be used for data storage. Express is used as a toolset and middleman for communication between the front and backend. Angular is the toolset to build the frontend framework for the users and admins, as well as security. Node is used for backend framework development and will be used for the frontend until Angular is implemented.

The customer-facing side of this application will be a website composed of multiple pages including a homepage, login option, and access to travel options and reservations. The admin SPA will have access to administrative tools for the application. An SPA is better for this as it performs better than traditional layouts and is easier to use.

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

The three main constraints for this application are time, knowledge, and scope. The amount of time to develop this application is limited by course time, and any development after the deadline, while personally fulfilling, will not help towards the main goal of the project. Knowledge is a limiting factor, as the creation of the application is influenced by what knowledge the creator has already, and what help can be found for development within temporal and moral reasons. The scope of what is wanted in the application means that any ideas that do not conform to what the client wants, even if helpful, cannot be implemented without client permission. To summarize, this project needs to be done by the end of class, with the development knowledge given/previously retained, and without adding anything unnecessary.

## [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 application will be split into three separate parts-a frontend accessible by users and a separate site for administrative access, a backend with server functionality and user data storage, and another database used for data storage. The frontend is split into the web browser that the user is using to access the application, the individual client session that is giving access to the server, the portfolio displaying relevant information to the user, and the graphics library that hosts images relevant to the pages accessed by the user. The backend is composed of the authentication server that checks user credentials before establishing a link to the next component, a server session that is the current instance of the running server, a database containing users and relevant data, and a Mongoose ODM that allows modeling of the data stored in the MongoDB database. Finally, the MongoDB database is considered separate from the other two components and stores all relevant data that users would need to access.

### Sequence Diagram

A diagram of a diagram

Description automatically generated

A user first accesses the site through proper routing, where the main site page is displayed. Then, from that view, they can interact with the site. Once doing so, this interaction is sent to a controller that requests access to the data needed to display what the user is looking for. This happens over an HTTP connection between the user’s client and the server’s client. Then, that call for service is forwarded to another controller of the server’s database, which requests relevant information. The request is processed, and then the return trip starts. A callback is initiated with the requested data, and json package is forwarded back over from client to client. The user’s client controller receives the information, and forwards it back to the user’s browser/view, where the user can now see the data requested.

## Class Diagram

A diagram of a travel company

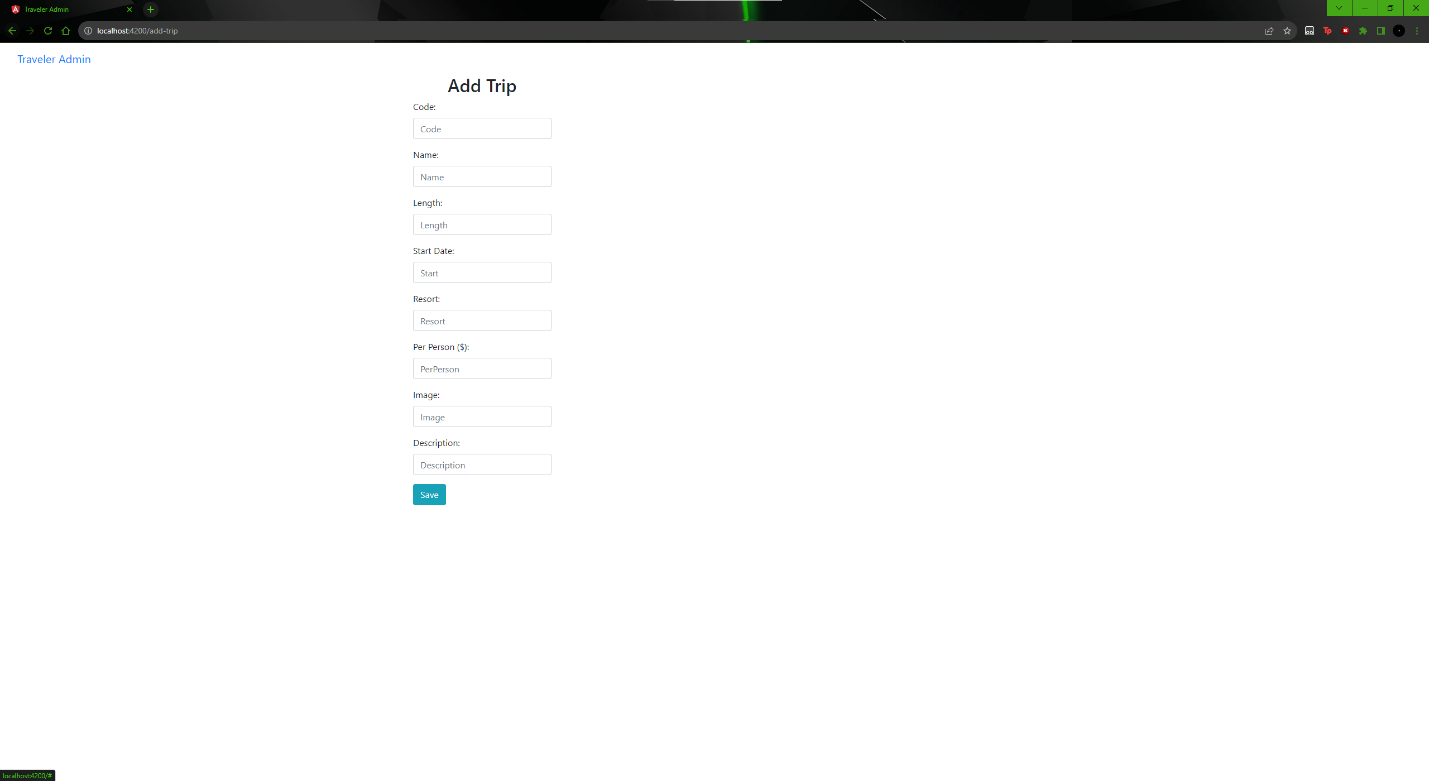
Description automatically generated

To start, the Flight, Cruise, and Hotel Info classes interact with their Booking counterparts, giving set variables to fill in the functions with. They also interact with the Itinerary and TripInfo classes, and are effected by the Travel\_Agent class as well. The Travel\_Agent class connects to the Info and Booking classes, as well as the TravelerInfo and MembershipAdmin classes. This allows a Travel Agent to book cruises, flights, and hotel stays for travelers while having access to their membership information and how many guests they wish to include on the trip. Travelers have access only to their account, where they can see details on their membership.

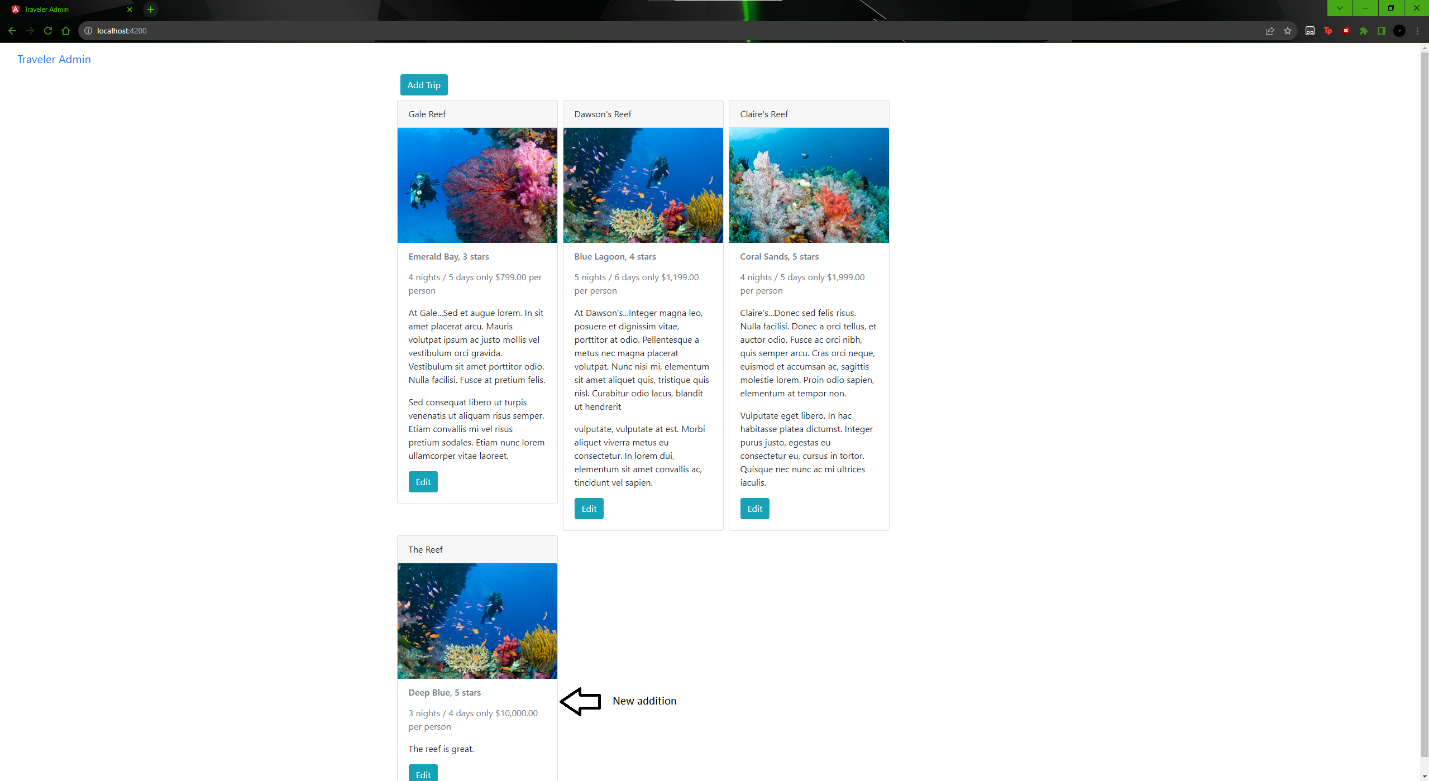
## [API](#_heading=h.2jxsxqh) Endpoints

| **Method** | **Purpose** | **URL** | **Notes** |
| --- | --- | --- | --- |
| **GET** | Retrieve list of trips | /api/trips | Returns all active trips in database |
| **GET** | Retrieve single trip | /api/trips/:tripCode | Returns single trip instance, identified by the trip ID passed on the request URL |
| **POST** | Login user | /api/login | Grants access to admin options |
| **POST** | Register new user | /api/register | Grants access to admin options for new account |

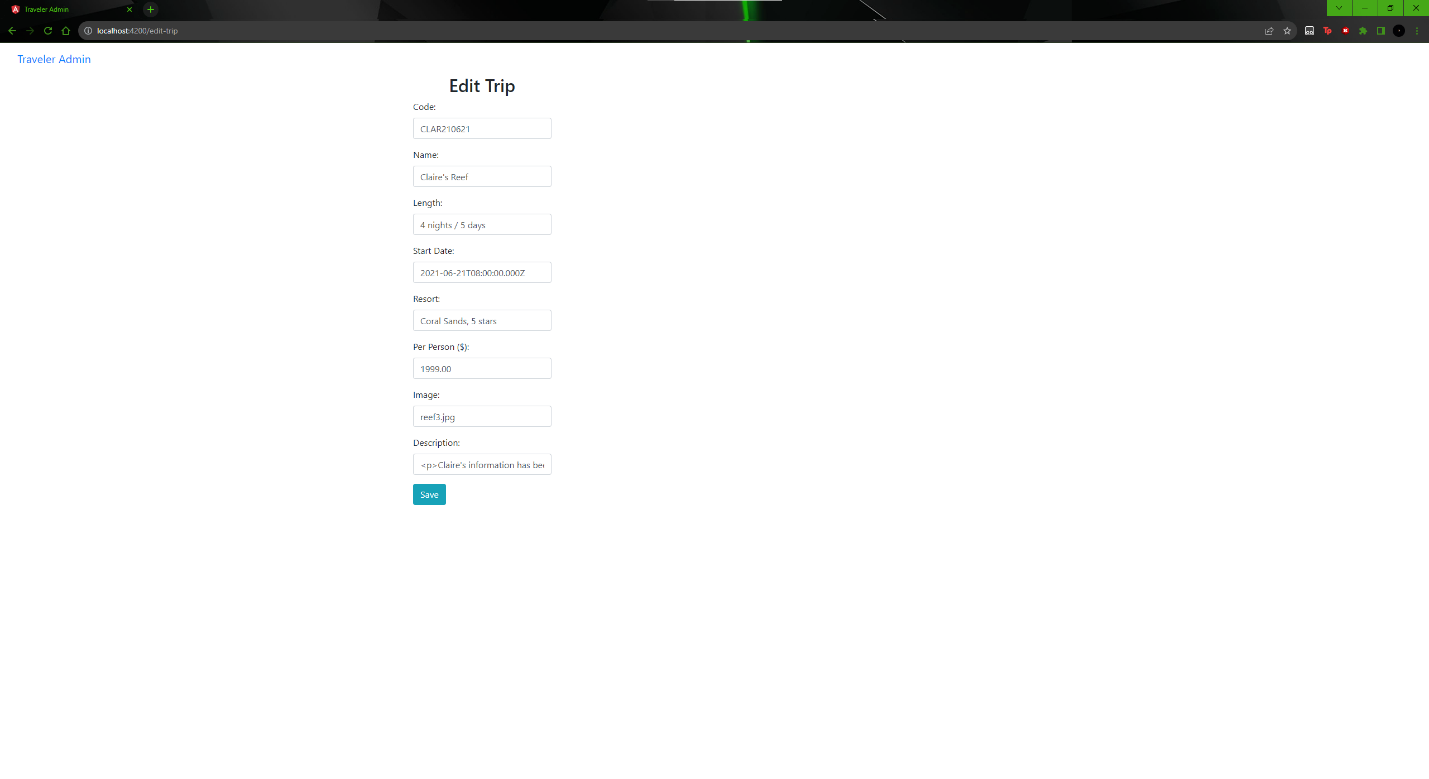
## The User Interface



Add Trip screen.



Added unique trip.



Edit Trip screen.

A screenshot of a computer

Description automatically generated

Updated Trip screen

The Angular project is a single page application, with options to edit and add trips to the server for display on the customer-facing page, whereas the customer page displays these trips in a different layout, as well as including navigation to other pages of the site as well. The Angular project is much more simplistic visually as well, as the admins are not the customer base for the site and do not need to be enticed to use it.

Some advantages of an SPA include quick loading time, ease of use for both users and builders, and small bandwidth usage. Disadvantages include heavy use of browser resources and security issues like ease of data breaches and script injections. Additional functionality that they provide over simple web apps includes data updates without page refreshing, quicker responsiveness due to not having to load separate pages during navigation, and, as for development, debugging can be monitored easily through actual browsers.

The process used for testing the SPA for GET and PUT was to add a new trip to the database and then edit an old trip with the respective buttons, and then check the database through Studio 3T to make sure the data was properly added to the repository. The main error I ran into was a human one, where I forgot to include the backslash after trips. Because of this, when I pressed the edit button, the prefilled data was not collected because the wrong address was being accessed. Once fixed, the data showed up correctly.