

Travlr Getaways

# **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 | 03/15/2023 | Valerie Smith | Executive Summary, Design Constraints, Component Diagram Description |
| 2.0 | 04/01/2023 | Valerie Smith | Design Documents and Descriptions |
| 3.0 | 04/12/2023 | Valerie Smith | Screenshots and Summary |

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

The goal of this project is to create a customer facing website for the client, Travlr Getaways. This project will be a client-server web application that features an administrator single-page application along with the customer facing website in order for Travlr Getaways to fulfill their need of providing a travel based website to their customers.

The client-server web application will be created with the MEAN technology architecture, which combines MongoDB, Express, Angular, and Node.js. MongoDB will be utilized as the database, providing high performance and scalability for web applications. ExpressJS is a framework that enables interactivity via an interface in order to allow NodeJS to connect with the Mongo database by the use of RESTful routes.

AngularJS is used to create the client-facing parts of the website as well as the single page Administrator site. AngularJS allows extended use of HTML combined with the JavaScript programming language. The client facing website will allow the users to view and book their travel plans as well as view contact, about, meals, news, and rooms pages. The Administrator single page application will allow the administrator to update and remove website content by interacting with the database.

The Node.js platform uses an event-driven model that can handle concurrent requests efficiently. Node.js is also a JavaScript based platform with a large library of various JavaScript modules. As the application will be using JavaScript for the entire project, front and back end, the application will have a high performance and security aspect. As a feature, NodeJS provides a single-threaded model with which it will provide scalability for applications that receive a lot of traffic or have multiple users accessing data simultaneously.

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

As this project will be created utilizing the MEAN technology stack, there is a constraint of the choice of programming language such as Node.js, AngularJS, and ExpressJS are coded with JavaScript. JavaScript is a programming language that does not have established coding guidelines. Developers that are working and creating in this project must have a functional knowledge of the JavaScript programming language, as well as anyone who is onboarded once the project is completed in order to perform support maintenance on the application.

Node.js and AngularJS libraries and imported modules can take up a lot of space and will need to be updated on occasion due to the availability and upgrades on certain libraries that they contain and pull into the application.

The MEAN stack development is not recommended for large enterprise web applications due to the MongoDB not being designed for extremely large datasets which can cause slowness in the application, as well as the potential loss of records that are written to MongoDB.

It is difficult to isolate the server from the business logic by utilizing the MEAN stack, often causing backwards compatibility challenges.

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

### Component Diagram



.

The architecture shown will work together to deliver the client server application. The user will view the client via a web browser, starting a client session. The user will be authenticated by the Authentication Server, and if validated, will be able to navigate the Travlr portfolio, which is created with AngularJS. The Graphic Library will contain various images that are used in the website and that are updatable by the administrator.

Any request made by the user will be further processed by the client as it will determine where to send that request. If a user makes a server type request, such as submitting information in a form or requesting data in a webpage, the request will move to NodeJS and be forwarded. The Server Session will use ExpressJS to connect with the MongoDB and fetch or store the requested data and return a response through ExpressJS. NodeJS, or the Server Session, will return the result of that request to the Client Session, which will then display the results to the user on the appropriate page of the Travlr Portfolio, where the user will be able to view the results in the web browser.

The MongooseODM is an object data modeling library for MongoDB and NodeJS. This will be used to manage the relationships between data in the database as it provides validation of the schema and is used to translate between the objects that are in the code and their representation in the database.

### Sequence Diagram

Diagram

Description automatically generated with medium confidence

The Sequence Diagram shown describes the interactions between the various components in the application. A user, or an Admin, will log into the application. Based upon which screen the user selects, they will be routed to the screen they selected using the Router, which contains the paths to all of the individual screens that are available in the application. Each route contains a view/template that will be shown in the browser for each specific screen. The controller will provide the interactivity for each screen, such as showing the text content and images that make up each specific screen. If a screen contains the functionality to call a service, such as retrieving trip details, the HTTP Client component will handle these requests and also handle the return responses.

The HTTP Client will send a request to the server based upon the route that it is calling, which determines which controller to use to get the requested data from the Mongoose DB via the Mongoose ODM. The DB will process the request and return the data back to the controller via the appropriate route, which is then received by the controller and formatted for viewing in the browser.

## Class Diagram

**Diagram, schematic

Description automatically generated**

The Itinerary class is an aggregate of other classes, such as CruiseInfo, FlightInfo, and HotelInfo, as well as including its own properties of totalprice, totalmiles, and stopover.

The Membership\_Admin class is an aggregate class that contains the details of the MemberAccount class, as well as its own functions of creditpoints, getpoints, and validate. This class has an association with the TravelAgent class of at least one, but possible many TravelAgent classes.

The MemberAccount class contains the properties membernumber, frequent\_airline, memberstatus, and memberclub. This is used by the Membership\_Admin class as well as a generalization of the TravelerInfo class.

The TravelerInfo class contains the property companionnum and is a dependency of multiple classes, such as CruiseBooking, FlightBOoking, HotelBooking, and TravelAgent. It also has a generalization to the MemberAccount class.

The TravelAgent class has a dependency relationship with the CruiseInfo, FlightInfo, HotelInfo, and TravelerInfo classes. It also has a directed association with the CruiseBooking, FlightBooking, and HotelBooking class. The TravelAgent class contains the methods BookPackage, BookFlight, BookHotel, and BookCruise, utilizing the classes that it has a direct association with.

CruiseInfo class is a dependency of the CruiseBooking and TravelAgent classes. It contains the properties name, cabintype, and price. It is an aggregate of the Itinerary class. It has an association with the HotelInfo class.

FlightInfo class is a dependency of the FlightBooking and TravelAgent classes. It contains the properties name, seatclass, and price. It is an aggregate of the Itinerary class. It has a generalization with the TripInfo class, as well as an association with the CruiseInfo class.

HotelInfo class is a dependency of the TravelAgend and HotelBooking classes. It contains the properties name, star, location, roomsrequested, and price. It is an aggregate of the Itinerary class.

HotelBooking contains the method getHotel, and has the HotelInfo class as a dependency. It is a dependency of the TravelerInfo class. It has a many to many association with the TravelAgent class.

FlightBooking contains the method getFlight and has a dependency of the FlightInfo class. It has the dependency of the TravelerInfo class and has a many to many association with the TravelAgent class.

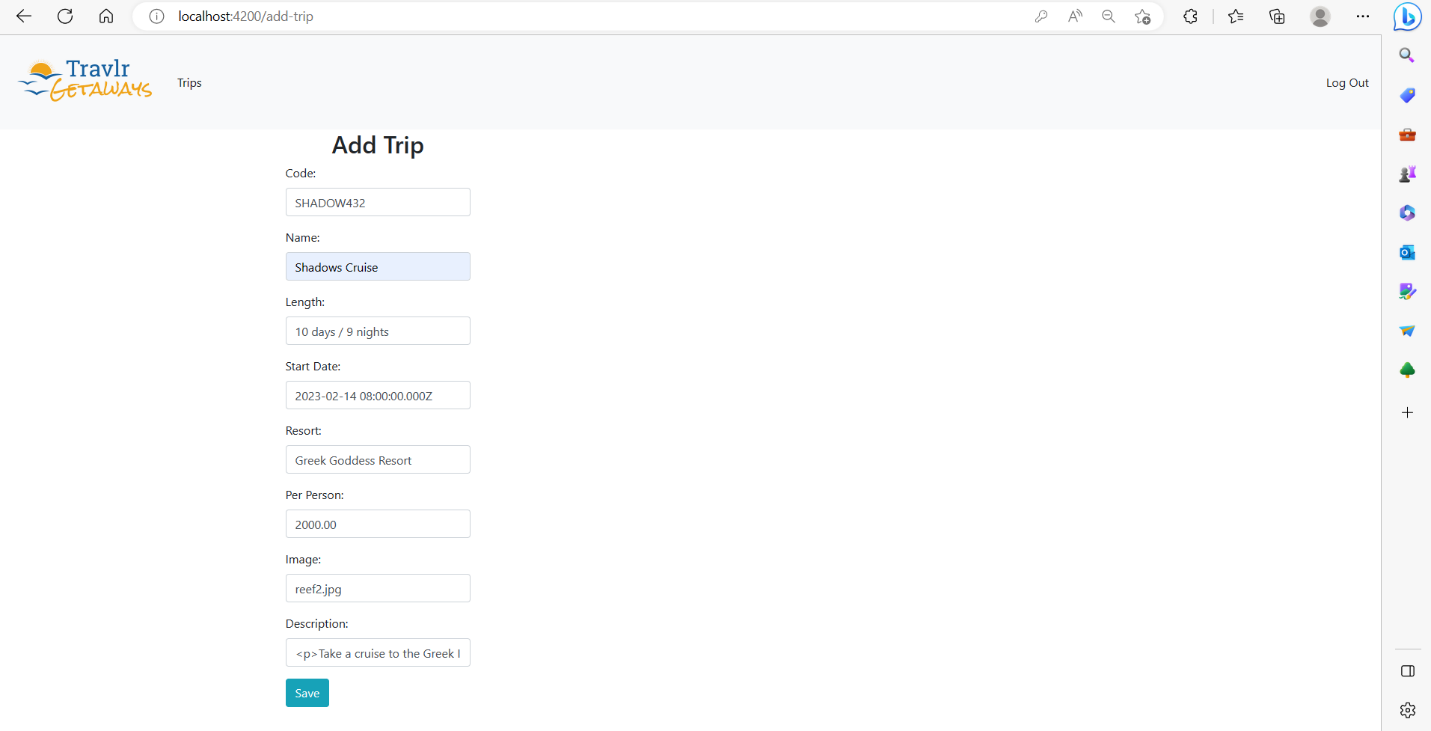
CruiseBooking contains the method get Cruise, and has a dependency of the CruiseInfo class. It has the TravelerInfo class as a dependency, and a many to many association with the TravelAgent class.

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

| **Method** | **Purpose** | **URL** | **Notes** |
| --- | --- | --- | --- |
| **GET** | Return one trip by code | /api/trips/:tripcode | Returns a single trip object from the database using the trip code as the reference |
| **GET** | Return all trips | /api/trips | Return a list of all trips stored in the database |
| **POST** | Add a trip | /api/trips | Add a trip to the database |
| **PUT** | Update a trip | /api/trip/:tripcode | Update a trip that already exists in the database |
| **POST** | User register | /api/register | Register user into the database |
| **POST** | User login | /api/login | Login the user |
| **DELETE** | Delete a trip | /api/trip/:tripcode | Delete a trip from the database using the tripcode |

## The User Interface

**Add Trip Screen**



**Trip List Screen (with new trip visible)**

Graphical user interface, application

Description automatically generated

**Edit Trip Screen**

Graphical user interface, application, Teams

Description automatically generated

**Edited Trip in Trip List Screen**

Graphical user interface, application

Description automatically generated

The Angular project structure consists of JavaScript components in a Module-View-Controller implementation. The Express project structure also features a MVC implementation style, but is tightly coupled with the server code, whereas the Angular project is a stand-alone Single Page Web Application that is calling the same API.

Due to the modular nature of the Angular components, the Single Page Application, or SPA, offers a richer and more fluid functionality as compared to the Express web application. The various elements and modules offer reusability as well, whereas the Express application is designed specifically for the data that it is supporting. As the Angular application is based on individual components, it is easier to test and to update when necessary.

The Express application requires each webpage to have its own route and HTML page, whereas the Angular app can manage multiple components on the same page. The Angular SPA will display the contents of a component based upon the selection from the user with the router, and can display one or more components at the same time.

There are many ways to test the SPA to ensure that it is working with the API for the various calls to the API, such as GET and PUT calls. The most intuitive way is to use the developer tools in the browser and monitor the Network tab to ensure that the call is successful as this will allow an inspection of how the application is acting when making these calls. For the GET calls, the API call will be visible as a GET call and will return a message. If it is successful, it will return the requested data object in JSON format. The data can be viewed in the developer tools in order to ensure that it is being returned in the expected format.

The PUT call can also be viewed in the developer tools in the Network tab. This call will contain a request body that asks the server to update an object that is already in the database. This request object will be visible in the developer tools along with the API call that is used. Once the call is made, the response from the API will be visible in the developer tools Network tab. The application can be inspected visually to ensure that the data is updated successfully.

To ensure that the data in the database is getting updated, the Studio 3T GUI was used to monitor the data in the Mongo database. While making the PUT calls from the SPA, the data can be viewed from Studio 3T to ensure that it is correctly updated and stored. For GET calls that return data from the database, the data returned from the API can be inspected to ensure that it matches the data in the database.