

Ronald W. Sudol III

# **CS 465 Project Software Design Document**

Version 1.0

## 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 | <mm/dd/yy> | Ronald W. Sudol III | Software Design Document |

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

An appropriate architecture for the web application based on our client’s software requirements is a M.E.A.N. Full stack application. M.E.A.N. employs MongoDB, Express.js, Angular, and Node.js. This architecture allows for communication between a database and webpage by utilizing a model-view-controller (MVC) pattern. This allows for a webpage that is functional, aesthetically pleasing, and efficient. Further it allows for the simple creation of a customer-facing webpage that is user-friendly and fast-loading as well as an administrator’s portal that is stable and powerful. Users will be able to sign into the customer portal and quickly view and interact with data. Admins will be able to log into a portal that allows for managing databases and user accounts.

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

There are a number of design constraints for developing the web-based Travlr Getaways application. The most glaring is that the front-end application and the admin portal need to both be able to directly access the database. Users need to be able to create an account, search travel packages based on different parameters, and book travel accommodations. Admins will need to be able to add new travel packages, edit existing packages, and manage user accounts. For these reasons the application must be designed in a way that is modular and scalable. The MVC pattern applied to a MEAN full stack architecture should meet the demands of these constraints.

## [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 overall system architecture of the web application consists of three major components. These are the Client, Server, and Database components. The Client component consists of the user’s traveler portfolio, the client’s side of the session initiated with the web application via the user’s web browser as well as a cache of locally stored images that populate the web page. The Client component initiates a session with the webserver by first gaining access through the authentication server. Once authenticated, the server then acknowledges and allows the session to access data stored in the traveler database that is stored in the server component as well as via a Mongoose Database manager. The Mongoose database manager is used to access data stored in the Database component consisting of MongoDB. Additionally, the Traveler Portfolio piece of the Client component can access certain data in the Database component directly.

### Sequence Diagram

A diagram of a diagram

Description automatically generated

This sequence diagram shows the passage of a request into the application by an end user and the subsequent response from the database. First, the actor sends a request to navigate to a specific route. This request is sent from the browser to a controller. The controller realizes the need for the request to be passed on to the server side of the application and sends the request to the HTTP client.

The HTTP client then passes a request the server side controller. As the request is received server side it is checked for assigned a route and the controller is called. The controller then communicates with the database to pull information necessary to process the request.

The database responds via a callback and the controller then passes resources back the HTTP client that made the client-side request. Once the HTTP client passes these resources to the client side controller, the controller assigns scope and the resources are displayed in the browser.

## Class Diagram

A diagram of a computer

Description automatically generated

There are a number of classes that compose our application.First off there is the MemberAccount class. This class contains variables about individual user accounts. Membership\_Admin class validates frequent flier points and passes the information to the Travel\_Agent class.

Travel\_Agent class holds functions for booking trips including packages, flights, hotels, and cruises. Travel\_Agent updates information in TravelerInfo.

HotelBooking, FlightBooking and CruiseBooking facilitate the booking functions of Travel\_Agent by providing functions to pull information from HotelInfo, FlightInfo, and CruiseInfo respectively.

Each of CruiseInfo, FlightInfo, and HotelInfo is an instance of TripInfo and therefore inherits variable information about trips from the TripInfo class.

Each of CruiseInfo, FlightInfo, and HotelInfo has an instance of the Itinerary class that includes information like the total price and distance of the trip/accommodation.

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

| **Method** | **Purpose** | **URL** | **Notes** |
| --- | --- | --- | --- |
| **GET** | Retrieve list of trips | /api/trips | Returns all trips. |
| **GET** | Retrieve single trip | /api/trips/:tripCode | Returns single trip identified by the tripcode. |
| **DELETE** | Delete a single trip within the database | /api/trips/:tripCode | Deletes single trip identified by the tripcode. |
| **POST** | Add a single trip to the database | /api/trips | Create a new trip in the database with passed parameters. |
| **PUT** | Update a single instance of an existing trip in database. | /api/trips/:tripCode | Updates single trip identified by the tripcode. |

## The User Interface

The Angular project structure utilized in this assignment is different from that of the Express HTML customer-facing page in a number of ways. Angular is more polished and modular. Specifically, Angular breaks an application down into components, services, and modules. Routing was also handled in a separate module. Express is far more static.

This modularity creates a number of advantages to implementing an Angular SPA page. Because a SPA page only loads once and dynamically updates without reloading the page, users experience a more fluid and seamless interaction with the application. This also makes for faster navigation of the page. Furthermore, the modular nature of Angular allows for code to more easily be reused throughout the application. There are some disadvantages. SPA pages experience a heavier initial load time. In addition, SPA pages are not as easily parsed by web-crawlers and search engines which can make a company’s web application harder to be found by new customers.

The process for testing to make sure the SPA is working with the API to GET and PUT data to/from the database is pretty straight forward. We tested by adding and updating entries in the database via the application. We then implemented database management tools like Postman or Mongo Studio 3T to view the database entries and check if they were in fact updated or created. Further, built in testing by way of status codes code be viewed in the command line terminal. GET and PUT requests that returned 200 range codes showed successful execution of the functions whereas 400 or 500 range codes showed errors that occurred either on the server-side or in the execution of the requests.