

Travlr Getaways Website

# **CS 465 Project Software Design Document**

Version 2.0

## 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 | 09/16/24 | Thomas Z McMahon | Added executive summary, design constraints, and explained component diagram. |
| 1.1 | 10/08/24 | Thomas Z McMahon | Added Sequence Diagram, Class Diagram, and filled out API Endpoints table |
| 2.0 | 10/26/24 | Thomas Z McMahon | Updated API Endpoints table and User interface information |

## Instructions

Fill in all bracketed information on page one (the cover page), in the Document Revision History table, and below each header. Under each header, remove the bracketed prompt and write your own paragraph response covering the indicated information.

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

A customer facing website that needs to hold data, allow customers to make changes and bookings, allow administrators to make changes to various packages and data, and update those pieces of information in real time requires at least three parts: a customer facing dynamic and interactable webpage, an administrator single-page application, and a database to house all of the information they both interact with. MangoDB is a good database that is efficient especially as your customer base and bookings data increases, expressJS and angularJS are frameworks that help build the web and customer facing web pages, and NodeJS is used as the server code to handle requests and changes to the database.

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

1. No security built into the development: This doesn’t mean the website won’t be secure but as it is not part of the requested project the product provided will not have any built-in security thus making it not production ready.
2. Small database server: During development I am working with a small database server, and this puts limitation on customer and product data storage capabilities. As the company expands either more robust servers will be needed or cloud-based storage.
3. MangoDB limitations: MangoDB has a smaller maximum document size and needs to split up larger documents and it also lacks the ability to make mass changes right out of the box.
4. Search engine optimization and display: AngularJS and dynamic website parts that are not preloaded by the search engine will not be displayed for the user as compared to static web data.

## [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).

Here are the three major components of the web application. First, we have the database that stores all the data. It communicates to the client by providing the traveler portfolio data which then uses that data to fill out the client webpage. On the other end the database connects to the server component via mongoose ODM which then connects to the server session and fills out the server side database making it faster for the server to interact with data.

The server component gets data from the database first for the authentication data for a client to login, then it is used for interactions between the database and the client in a controlled server session.

The final part is the client component. This is what the client interacts with here the client component pulls the page data from the database to fill out the traveler portfolio and website graphics and stored data. The client also connects to the server first to login and then to interact with the data in the database through the server functions.

### Sequence Diagram

A diagram of a company

Description automatically generated

When the user arrives on the webpage the browser will connect to the application and look for the appropriate route based on the address. The route will direct to the appropriate view for that page. The view will talk to the controller to receive data. The controller will send a request through the HTTP client which will then look for the appropriate route in the APIs. The route will direct the request to the appropriate controller and model to request and format data from the database at the back. Once the database responds back the controller uses the model to format the data and then sends it back to the client that requested it. The client then gives those results to the controller in charge of it which then fills that data into the view for the user to see it.

## Class Diagram

A diagram of a travel application

Description automatically generated  
Trip info is a base class for Cruise, Flight, and HotelInfo to inherit from. The Cruise, Flight, and HotelInfo classes hold data on any individual cruise, flight or hotel for a trip booking. That data when booked will be shared with the Itinerary which will aggregate the data to give the user an overview of the trip. The data will also be shared with the Travel\_Agent which will take that data and use that to book cruises, flights, and/or hotels via the relevant booking class/es. The Travel\_Agent will share that data with The TravellerInfo to update rewards and points. TravellerInfo gets data from the MemberAccount to use when interating with the Travel\_Agent. The MemberAccount is a base class for the Membership\_Admin to inherit from. The Membership\_Admin then sharesthat data with the Travel\_Agent where appropriate for bookings.

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

| **Method** | **Purpose** | **URL** | **Notes** |
| --- | --- | --- | --- |
| **GET** | Retrieve all items in trips collection | /api/trips | Returns all items in trips collection |
| **GET** | Retrieve a specific item in trips collection | /api/trips/:tripCode | Returns a single item from trips collection found using the tripCode |
| **POST** | Adds a new trip item to trips collection | /api/trips | Adds a single item to the trips collection |
| **POST** | Updates an existing trip item in trips collection | /api/trips/:tripcode | Updates a selected item in the trips collection |
| **POST** | Adds a user to the users collection | /api/register | Adds a new user to the users collection for authentication purposes |
| **POST** | Checks if user is in users collection and if password is correct | /api/login | Sends email and password entered by user and checks the users database to see if it finds a match. Returns a session token if successful. |

## The User Interface

User Interface Examples

A screenshot of a computer

Description automatically generatedA screenshot of a trip

Description automatically generatedA screenshot of a computer

Description automatically generated

The biggest differences between the Angular page and the express HTML page were the reactiveness and the single page load. The Angular page updates as changes are made and gives instant feedback for both the development and usage whereas the HTML page is static until reloaded for the developer and changes only between refreshes for the user. Additionally, the Angular page might take longer for the initial load, but it doesn’t ever send the user to another page. Everything is dynamically changed in the same page only updating the parts that need to be while in use. The HTML page on the other hand must send you to another page for a different piece of information and has less interactivity.

SPAs are more interactive and keep the user from moving back and forth between web pages. They give the user the ability to see changes in real time and see issues as they pop up both in usage and during development. They also tend to have longer initial load times and have more moving parts where a user could mess something up or where security could become an issue.

When testing the SPA’s connection to the API and if it is functioning correctly you need to check a few things. Make sure that you see appropriate changes within the SPA itself when calls are made to the API such as seeing the appropriate data filled in when a GET call is made or seeing data updated when a PUT or POST call is made. Then you need to check if when the SPA is restarted it still reflects the right data and changes. Finally, you need to make sure that the database itself shows the changes. Theoretically if a restart still shows the changes, you are probably good but to be safe look into the database to ensure that the data is updated there.