

<Name of Software Application>

# **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 | 11/25/2023 | Caleb Partain | System Architecture View: Class Diagram and API endpoints table |

## 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)

The customer has declared that the web application should include what seems to be 3 main pages. A login page will be required, a booking page, and an admin site. The accounts and listings will need to be stored in a database somewhere. After careful consideration, I have decided that the best architecture would utilize the MEAN stack. The mean stack stands for MongoDB, Express.js, Angular, and Node.js. MEAN would allow for easy future scaling and a simple development process. MongoDB will be the database that will hold all of the user information and booking information. Express will be involved in all of the server side logic such as user authentication. Angular will be the frontend framework that will create the single page application that the users will interact with. Node.js will serve as the runtime environment.

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

The customer has asked that we create a single page web application for their customers to book travel reservations. It is required that the users have logins associated to their identity and an admin specific page. There are only a few constraints that were actually given by the customer, but there are a lot of other implied constraints that must be taken into consideration. The first of which would be legal obligations that pertain to user information. Any user data that is tracked from the user must be openly stated to be tracked and must only be for communicated reasons. There was no budget expressed, but the development process will have to be complete within 7 weeks to adhere to project deadlines.

## [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).

There are three main components in which the diagram can be broken down into. The first of which is the client component. The client component is the front end which holds the client session. Web browser, traveler portfolio and the graphic library. The client session is the connection between the server and the client. The web browser is simply the interface in which the user interacts with. All of the graphics are handled by the graphic library and the traveler portfolio is the users profile that is pulled from the Database. The database is comprised of a single MongoDB and is connected to the Traveler portfolio and the servers Mongoose ODM. The Mongoose ODM is simply a way for the server to interact with the database. Inside of the server, there is a smaller database for the user data. Server session communicated with the client through the authentication server to provide requests.

### Sequence Diagram

A diagram of a travel way

Description automatically generated

The depicted cycle illustrates the interaction among three essential elements: the server, client, and database. The sequence begins with user authentication or registration to acquire access privileges. Subsequently, the server initiates loading and establishes a connection with the travlr database. Utilizing the information, the mongoose Object-Document Mapper (ODM) submits requests for the retrieval of models from the database. The database, in turn, presents a library for the user to peruse and express their opinions.

## Class Diagram

A diagram of a computer

Description automatically generated

The depicted class diagram illustrates the functionality of a web application. To ensure proper operation, users are required to input the "http" of the website. When conducting searches within a database, the website engages in interaction with the database. Subsequently, the database responds by presenting relevant data to the user, while concurrently securing the backend connection of the website.

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

<Exposing RESTful endpoints is a design approach to enable an application to participate in a larger ecosystem. Document each endpoint in the table below, including the HTTP method, purpose, URL, and notes.>

| **Method** | **Purpose** | **URL** | **Notes** |
| --- | --- | --- | --- |
| **GET** | <Retrieve list of things> | </api/things> | <Returns all active things> |
| **GET** | <Retrieve single thing> | </api/things/:thingId> | <Returns single thing instance, identified by the thing ID passed on the request URL> |
| **POST** | Create new list of things | /api/things | Creates a new list of things |
| **POST** | Create a single item | /api/things/:thingId | Creates a single thing with the thing id |
| **PUT** | Update a list in the database | </api/things> | Updates and entire list of things |
| **PUT** | Update a single item in the database | </api/things/:thingId> | Updates a single thing with the thingId |
| **DELETE** | Delete all things | </api/things> | Deletes all things in list |
| **DELETE** | Delete a single thing | </api/things/:thingId> | Deletes only thing with thingId |

## The User Interface

<Insert screenshots from the development of the SPA development to show the following: (1) a unique trip, added by you, (2) the Edit screen, and (3) the Update screen.>

A screenshot of a website

Description automatically generated

A screenshot of a computer

Description automatically generated

<Summarize the Angular project structure and how it compares to the Express project structure. Be sure to describe the rich functionality provided by the SPA compared to a simple web application interaction. Describe the process of testing to make sure the SPA is working with the API to GET and PUT data in the database.>

There are a few key differences you will notice when it comes to t the Angular project structure and the HTML customer facing page. When first looking at the two, you may notice the large difference in scale. The Angular project has a CSS, HTML and TS file for each component. If there are a lot of components on the page, this can become quite cumbersome to work with. That being said, there are also some key advantages to the structure. One of the biggest being the ability to reuse the components. With each component having it’s separate module to be imported, we can simply reuse the code by importing the component in later versions. The testing process is actually quite tedious. I would recommend starting with postman to ensure that your API is working properly. Once you are sure that the codes are correct, Ensuring all of the buttons work properly is very important. During this stage, it is a good idea to try and find edge cases.