

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 | 09-16-23 | John Brungard | Updated the following sections: Executive Summary, Design Constraints, and System Architecture View. |
| 2.0 | 09-26-23 | John Brungard | Updated the following sections: Sequence Diagram, Class Diagram, and API Endpoints |
| 3.0 | 10-12-23 | John Brungard | Updated the following sections: User Interface |

## 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)

Working together with the client Travlr Getaways, a travel booking website is being developed using the MEAN Stack Architecture which will be written in JavaScript and TypeScript and formatted in JSON. The MEAN Stack consists of MongoDB (the database), Express (the web framework), Angular (the front-end framework), and Node.js (the web server).

The application from the customer’s side will utilize a Single-Page Application (SPA), which will allow for communication between the server and database with quick page loading. This will assist in endeavors performed by the customer such as booking travel packages, creating accounts, reviewing itineraries and filtering searches for travel packages by location and price-point.

An administrative-only site is to be developed which will support the maintaining of the customer base, viewing available trip packages, and pricing items and packages. This will also support SPA, which means everything runs and is managed inside the browser.

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

One possible constraint is that the server will be single-threaded. While this certainly offers many benefits, we must ensure the development of the code is non-blocking, meaning we must run blocking operations asynchronously to prevent disruptions of the flow concerning the main process. There is also no storing of session states as it uses a series of HTTP requests. This means that visitors will not be remembered with requests. Using Express, the complexities that may come with using node.js will be lessened and put more focus on the building of the web application.

Another constraint, which again while it does offer benefits, is our web application being a Single Page Application. Because the web application is written in JavaScript, it makes it harder for search engines to crawl and index the application as most look at the HTML content of a page. This makes the indexing of the web application uncertain and may not be as good as the crawling as delivered by the server. One way to overcome this limitation is to create separate HTML pages to mirror the content of the SPA although this can be hard to maintain, and it offers limitations of potential search engine optimization.

The final constraints to be covered are time and budget. This is a general constraint on most projects and a balance must be struck to ensure requirements from the client are being met. This also can affect the hardware used based on the money and period allotted to complete the project. If we go with hardware in which one REST API serves as the medium between the database and application due to costs, this may be a potential problem as an application grows as the application and database would fight over the same resources. With more money and time, we can have multiple instances of the mentioned architecture which would help as traffic increases on the application.

## [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 diagram above shows the 3 main components of the client-side, server-side, and the database. MongoDB stores documents as BSON and Mongoose is complementary to this for data modeling, which is defined in a schema. Mongoose also helps with connections, queries, validations, aggregations and much more. Within the server-side is an authentication server, which helps determine which users have what privileges concerning the application. When users log in successfully, a session is created from both the server and the client to provide communication.

Within the server-side holds the traveler database, which holds information about locations, itineraries, etc. The web browser on the client-side provides the view of the application while the model is brought by the server and database. The controller, which would be defined in the code, determines what information is passed along and helps present the traveler portfolio in conjunction with the Graphic Library. This is demonstrated as users navigate through the different sections of the web browser such as the rooms and travel sections. An API is used in place of an integration approach as it provides the interface between services without having the added complexities and fragmentation an integrated approach would have as the application grew. All of which is to be contained in a single Express project.

### Sequence Diagram

<Illustrate the flow of logic in a web application by completing a sequence diagram. Insert an image of the sequence diagram here.>

A diagram of a diagram

Description automatically generated

The Sequence Diagram can be broken into 3 main components: the client, the server, and the database. Beginning with the actor/user, they are routed to the view. The controller and model then interact to establish the HTTP Client. Service is then called to retrieve the browser using MVC. When the user tries to access certain parts of the website, such as the travel section, the server side would call a service to retrieve data from the database using the controller/model, which will then deliver HTTP to the user and update their view.

The controller is used to pass information along. The model contains data such as the server and database and the view are the browser or template displayed to the user. In summary, the user is presented with a view in which when a user interacts with it causes the controller to pass their request along to the server which represents the model. This server then coordinates with the database to send information back through the controller and update the user’s view.

## Class Diagram

<Illustrate the JavaScript classes of the web application by completing a class diagram for the web application. Insert an image of the class diagram here.>

A diagram of a travel geoways class diagram

Description automatically generated

The class diagram displays a member’s account that is made up of a unique member number/identity, the airline they most travel with, their status as a member (this can be upgraded), and the club in which they reside. There is a generalization/inheritance in which the Travel\_Agent (who also has a unique number, but as a companion) is based on the MemberAccount. There is also an aggregation relationship between the MemberAccount and Membership\_Admin as the MemberAccount is part of the Membership\_Admin. The admins are responsible for credit points and validation. There is a dependent relationship between the Travel\_Agent and the different types of bookings as the Travel\_Agent is responsible for these. Likewise, there is also a dependent relationship between the information of each type of vacation and their booking counterparts as you need the information to book. The overall TripInfo inherits the information from the cruise, flight, and hotel information to organize the starting and end dates, origin, and the destination(s). The itinerary holds the information and booking of the trip in one package. It has a 0 to many relationship with the booking categories and is dependent on the information from the hotel, flight, and cruise info, to be complete. The Membership\_Admin has a 1 to many relationship then with the itinerary as one admin can handle many itineraries.

## [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(tripsController.tripsList)** | Retrieves a list of all trips. | </api/trips> | Used with the Mongoose.find() method with no filter criteria to return all instances. |
| **GET(tripsController.tripsFindCode)** | Retrieves a single trip based on the trip code. | </api/trips/:tripCode> | Used with the Mongoose.find() method with a filter set to the tripCode as the argument, which is passed on the URL to receive a single instance or specific trip. |

## 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.>

<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.>

A screenshot of a cellphone

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a cellphone

Description automatically generated

Angular organizes all the material into modules to include an app root module. This is the entrance to the application and the one that is bootstrapped. Applications are also created by composing HTML templates with Angularized markup component classes to manage the templates by adding logic in the services folder and boxing services in the beforementioned modules. The components control the view, and a class is used in conjunction through an API of methods. Angular will then go through the creation and destruction of components as the user navigates the application. The HTML template tells Angular during this how to render the component. This is part of Angular’s strength of databinding where the template and Angular are connected at both ends. Meta Data is used to process a class through Angular. This is usually done in a typescript file and done with a decorator. Furthermore, Angular templates are dynamic and are guided by directives. Finally, services from Angular use dependency injection to provide new components with services they need.

An SPA dynamically rewrites the current page rather than loading new pages from a server. This means no page reloads and less time wasted refreshing pages. An example of an SPA is Facebook. SPA’s also consume less bandwidth since they only load once. Unfortunately, search engines use page numbers as a metric, so SPAs suffer from this. SPA’s are also resource-hungry as it requires a lot from the web browser meaning an updated and modern browser is nearly a must. Finally, SPA’s are more vulnerable to cross-site scripting attacks and to expose sensitive data if an unauthorized client-side script is injected into the web app.

We use API Testing to ensure the SPA is working with the API in which we make requests to the API endpoints and compare their responses with the actual results. For our project, we performed this by checking the backend API by checking MongoDB for data that was added and the data that was retrieved. We also analyzed the powershell for certain codes to determine success such as 200 and 201 for correct actions and 400 and 500 for unintended results. On the front end, we checked this by analyzing the view after being updated by the model as we navigated through the application. For example, adding the trip ‘Mega Reef’ through the front end would display a card view with the specified contents. We could also see the path in the url. This is because, in essence, the web browser will take requests from the SPA, secure them with CORS, and forward the request to the secured API.