

Travlr Website

# **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 | 1/21/2025 | Sarah Marshall | Edited executive summary, design constraints and system architecture view: component diagram. |
| 2.0 | 2/7/2025 | Sarah Marshall | Edited sequence diagram, class diagram and API endpoints. |
| 3.0 | 2/14/2025 | Sarah Marshall | Reviewed entire document and edited the user interface. |

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

The client Travlr Getaways, wants a travel booking website where their customers can book travel packages, make an account, search for travel packages by location and price, book reservations with Travlr Getaways travel agency and have constant access to a stable website that is always available. The appropriate web application based on my client’s software requirements must include the MEAN stack. This means using these four technologies: MongoDB, Express, Angular and Node.js. MongoDB is the database which will be used to store all data. Express is the web framework which listens “to incoming requests and return relevant response[s]” (Harber & Holmes, 2019). Angular is the front-end framework “for creating the interface for [a] website or application” (Harber & Holmes, 2019). Lastly, Node.js is the web server (Harber & Holmes, 2019). Overall, MongoDB is the database, Node.js and Express is the application server, and Angular is the front end (Harber & Holmes, 2019). The customer-facing side of the application allows the clients to book travel packages, make an account, search for travel packages by location and price, book reservations with Travlr Getaways travel agency and have constant access to the stable website. The administer SPA allows authorized users to maintain a customer base, available trip packages, and pricing for each item and package.

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

For any application, design constraints may follow. In this web-based Travlr Getaways application, it may be technical, financial, legal and regulatory, time and user acceptance testing (UXPin, 2025). Technical constraints refer to device and operating system limitations, such as screen size and processing power (UXPin, 2025). For this application we are unsure what clients will be accessing this website on, therefore this can lead to issues as devices and OS’s can be different. Financial constraints refer to a limitation on the budget that can impact other processes during designing the website which may be testing and access to resources/tools (UXPin, 2025). For this application I am unsure what tools I may run into needing in the future and they may end up being a financial constraint. Legal and regulatory constraints may include privacy laws and accessibility laws (UXPin, 2025). While designing this website application, it is important to follow users’ privacy laws and accessibility laws as well as protect all data exchanged. Time constraints refer to us not having enough time to complete the website application for the client Travlr Getaways. User acceptance testing can be a design constraint as well because if the client tests the website application product, and they are not fully satisfied with the product then it will need to be modified.

## [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 major components, Client, Server and Database, and all work together to form a website application. Inside each of these major components there are minor/subcomponents. In the Client component there is Client Session, Web Browser, Traveler Portfolio, and Graphic Library. In the Database component there is MongoDB. In the Server component there is Authentication Server, Server Session, Traveler Database and Mongoose ODM. Inside the Client component the Client session is connected to the Web Browser and Traveler Portfolio. The Traveler Portfolio is connected to the Graphic Library. Inside the Database component MongoDB is not connected to anything else as there is no other subcomponents in this major component. In the Server component the Authentication Server is not connected to other subcomponents. The Server Session is connected to the Traveler Database and the Mongoose ODM. The Client component connects to the Database component through the Traveler Portfolio into MongoDB of the Database component. The Client component connects to the Server component through Client Session into Authentication Server of the Server component. The Database component connects to the Server component through MongoDB into Mongoose ODM of the Server component. These three major components utilize the MEAN stack. MongoDB is used in the Database component. Express and Node.js is used in the Server component. Angular is used in the Client component.

### 

### Sequence Diagram

A screenshot of a diagram

AI-generated content may be incorrect.

In the sequence diagram above there is the Client-Side (Angular JS), the Server-Side (Node JS/Express/Mongoose) and the Data-Tier. In the Client-Side (Angular JS), in order, there is Route, Browser/View/Template, Controller, and HTTP Client. In the Server-Side (Node JS/Express/Mongoose), in order, there is Route and Controller/Model. In the Data-Tier there is MongoDB. All of these layers and tiers work together in this full stack application. This whole process is started with an actor, such as a user, when they enter a route into a browser. This is then redirected to a view of the browser. This view interacts with the controller which calls to retrieve data from the HTTP client. The HTTP client works with the server-side through the route by API calls such as GET, PUT and POST. The route then calls to retrieve data from the controller/model which uses Mongoose ODM to get into the database MongoDB. Once in the database the request of information/data is processed, and returns the results by displaying the view to the actor.

In the sequence diagram users(customers) would be able to see the option for a Sign in or Trips in the Browser/View/Template portion. When the user(customer) sends a request to do a Sign in or Trip interactions this would be sent through each stage, once it reaches MongoDB that process would be requested and returned back to the user by rendering the new view. In MongoDB, if the sign in attempt was verified to be admin credentials their interaction would go through similar phases by rendering the new view for this specific log in, enabling them to modify, view and request different data’s than a customer’s credentials.

## Class Diagram

**A diagram of a class diagram

AI-generated content may be incorrect.**

In the class diagram above there are 12 classes that all work together. The classes are Itinerary, MemberAccount, Membership\_Account, TravelInfo, Travel\_Agent, CruiseInfo, FlightInfo, HotelInfo, HotelBooking, FlightBooking, CruiseBooking, and TripInfo. Itinerary consists of totalprice, totalmiles and stopover, and is connected to CruiseInfo, FlightInfo and HotelInfo. CruiseInfo consists of name, cabintype and price. FlightInfo consists of name, seatclass and price. HotelInfo consists of name, star, location, roomsrequested and price. These three classes are connected together though TripInfo which consists of starting\_date, returning\_date, orgin and destination. These three classes are also connected together though Travel\_Agent which consists of BookPackage, BookFlight, BookHotel and BookCruise. The three classes are also connected to their own booking class, CruiseBooking, FlightBooking and HotelBooking. CruiseInfo is connected to CruiseBooking which consists of getCruise. FlightInfo is connected to FlightBooking which consists of getFlight. HotelInfo is connected to HotelBooking which consists of getHotel. These three classes are connected to the Travel\_Agent class, which was defined earlier, and TravellerInfo which consists of companionnum. The TravellerInfo class is connected to the MemberAccount class which consists of membernumber, frequent\_airline, memberstaus and memberclub. The MemberAccount is connected to Membership\_Account which consists of creditpoints, getpoints and validate. The Membership\_Account is then connected to the Travel\_Agent class, which was defined earlier. Overall, each of these classes work together so users can book a trip through a travel agent.

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

| **Method** | **Purpose** | **URL** | **Notes** |
| --- | --- | --- | --- |
| **GET** | Retrieve list of all trips | /api/trips | Returns all trips in database |
| **GET** | Retrieve single trip | /api/trips/:tripCode | Returns single trip from database using tripCode |
| **GET** | Retrieve home page | /api/home | Returns all home content from database |
| **GET** | Retrieve travel page | /api/travel | Returns all travel content from database |
| **GET** | Retrieve room page | /api/rooms | Returns all room content from database |
| **GET** | Retrieve meal page | /api/meals | Returns all meal content from database |
| **GET** | Retrieve news page | /api/news | Returns all news content from database |
| **GET** | Retrieve about page | /api/about | Returns all about content from database |
| **GET** | Retrieve contact page | /api/contact | Returns all contact content from database |
| **POST** | User login | /api/login | Allows a user to log in after verifying credentials in database |
| **POST** | Create new trip | /api/trips | Allows admin to add a new trip to the database |
| **POST** | Register a new user | /api/register | Allows a new user to register and added to the database |
| **POST** | Login a user | /api/login | Allows a user to login to the website by verifying through the database and have access to adding a trip and editing it, which is not accessible without logging in |

## The User Interface

A screenshot of a computer

AI-generated content may be incorrect. A unique trip, added by me.

A screenshot of a computer

AI-generated content may be incorrect.The Edit screen.

A screenshot of a computer

AI-generated content may be incorrect.The Update screen.

A screenshot of a computer

AI-generated content may be incorrect.The SPA before a user is logged in.

A screenshot of a computer

AI-generated content may be incorrect. The SPA after a user is logged in.

A screenshot of a computer

AI-generated content may be incorrect.The travlr customer-facing page.

The Angular project structure is different from the Express HTML customer-facing page. In the Angular project structure there is different folders that are specific to the task, such as trip-listing and trip-card with only component files in them, such as trip-listing-component.html or trip-card-component.ts. The Express HTML customer-facing page structure has controllers, routes, and views that have js or hbs files in them. Some advantages of the SPA functionality is how all data is on one page, it is rendered very quickly and provides the user with all information on one page, making it more user-friendly. Some disadvantages of the SPA functionality is that there can be limited info on one page, meaning users may not be able to gain all of the information they are wishing to gain on one page. It can also look too congested if not done right. An additional functionality provided by a SPA is the API interaction, which can be more secure and perform better. The process of testing to make sure the SPA is working with the API to GET and PUT the data into the database can be done with Postman. In Postman we were able to test if data could be taken from a website using GET with the url and the data we want to get which was name, perPerson, description and more. This was similarly done with PUT. If PostMan was able to verify both were valid, then we knew that the SPA was working correctly with the API to GET and PUT data into the database, it also worked for POST. An error I ran into was while doing POST, was that I had an extra space in one of my key values that did not match what it was expecting. For example, I was trying to test and send POST, but my start date key value was entered “2022- 01- 1908:00:00Z” and it kept giving me a socket-hang up error. I was confused because there was no issues with my code, and my localhost to my SPA was loading but not updating information on the new trip I was trying to add. After an hour I realized the extra spaces in the start key value, after removing the extra spaces and sending the POST with the start key value as “2022-01-1908:00:00Z” it effectively posted my new trip on my SPA and Postman verified it was posted as well. Another common error to run into is a 404 error, meaning the API endpoint was not found.

References:

Harber, C., & Holmes, S. (2019). *Getting MEAN with Mongo, Express, Angular, and Node, Second Edition*. O’Reilly Online Learning. https://learning.oreilly.com/library/view/getting-mean-with/9781617294754/?sso\_link=yes&sso\_link\_from=SNHU

UXPin. (2025). *7 constraints in design and how to overcome them*. Studio. https://www.uxpin.com/studio/blog/constraints-in-design/#:~:text=Design%20constraints%20are%20limitations%20or,these%20limitations%20before%20every%20project.