

Travlr Getaways

# **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 4](#_Toc36198469)

[Sequence Diagram 5](#_Toc36198470)

[Class Diagram 6](#_Toc36198471)

[API Endpoints 7](#_Toc36198472)

[The User Interface](#_Toc36198473) 7

## [Document Revision History](#_heading=h.lnxbz9)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 11/11/24 | Tiffany McDonnell | Completion of Executive Summary, Design Constraints, and System Architecture View: Component Diagram. |
| 1.1 | 11/26/24 | Tiffany McDonnell | Completion of Sequence Diagram, Class Diagram, and API Endpoints |
|  |  |  |  |

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

Travrl Getaways wants us to create a full stack web application to fulfill their needs. To create this full stack application we will be using MEAN stack for the developing process. This will include utilizing multiple software programs and computer languages. We will be utilizing handlebars to control duplicate code from being used based on html files.

MongoDB will be used to store and access data. Express will be used to communicate data between the front end and database. Node. js will be used as a framework for processing multiple tasks at the same time. AnglerJS will be utilized for most of the front-end development. It is essential to the int interaction communications of the client side of the web application. Angler also allows us to create Dynamic Single-Page Applications(SPAs). SPAs allow the clients to use the application easily without having to continually load a whole page for and minor update. This will in turn make things faster and give the client a better experience.

**Requirements:**

* Must have a customer facing website
* Must have a database
* Must have an administrative single-page application
* Everything must work together appropriately
* Login in interface for user

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

* Everything must work together seamlessly
* Must be able to handle thousands of connections in a short time frame.
* Website security for personal information
* Webpage must include scalability
* Code must be able to maintain manageability for updates

We must use these constraints to help maintain a proper working website. These constraints also guide us in completing the client’s requirements. When all constraints are met, then the website should run the way the client expects for the finished product. This will allow us to develop and build upon in a smooth motion and make the project finish in an expected manner.

## [System Architecture View](#_heading=h.44sinio)

The diagram below shows the architecture of the application being developed. It shows a type of web application architecture. On the client end, there will be four different components. Starting with the client’s session, the web browser and traveler portfolio connect to the session. Both these components need each other for the client’s session on the web application to be successful. The main part would be the traveler portfolio since that is what the client would like to access, but that can’t be viewed without some sort of browser to load the portfolio. The last component on the client’s end is the graphic library. The travel portfolio can access the graphic library to display any images the web application is designed to show the client.

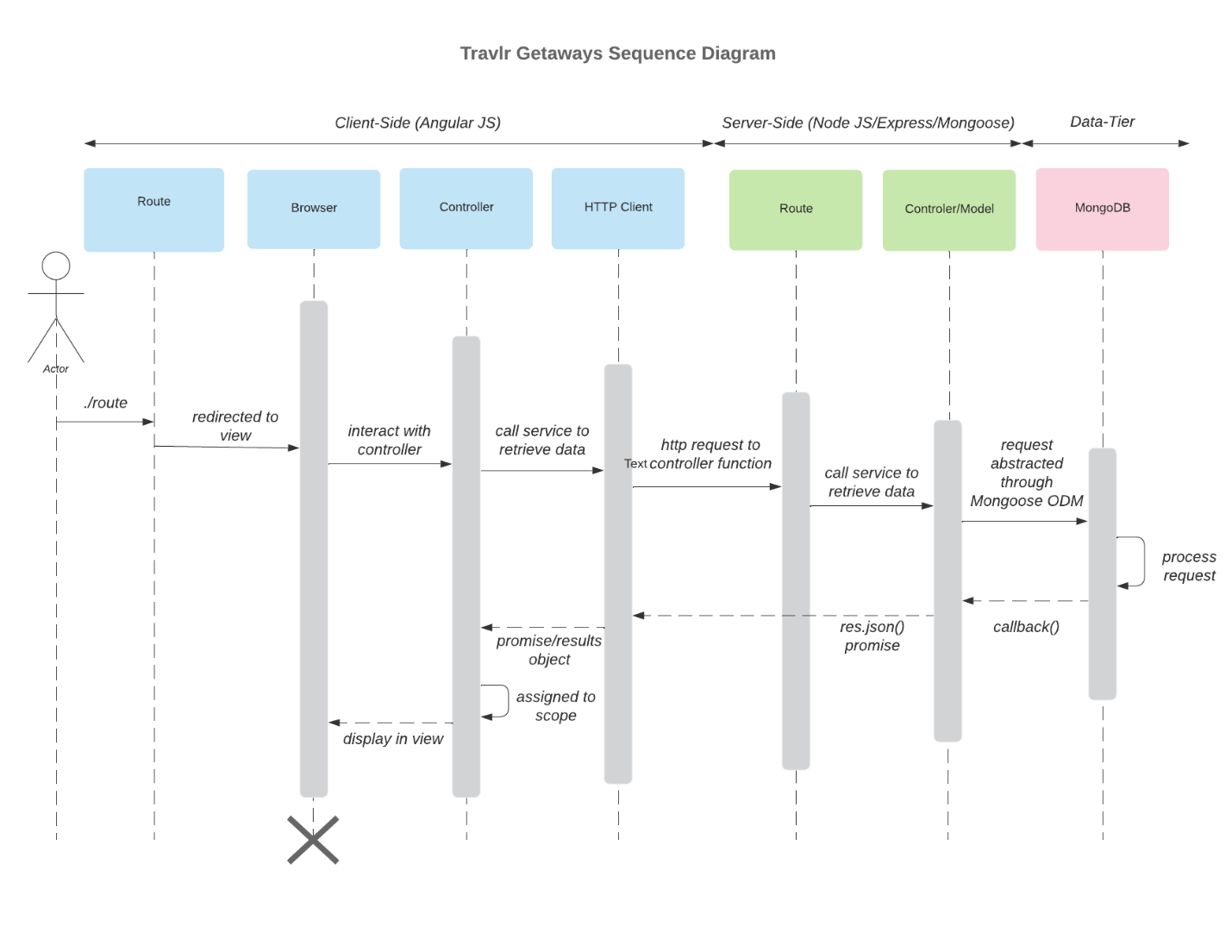
On the sever side there are also four components. The first component is the authentication server. This component also affects the client side since the client will utilize this component within the clients session. It is also the connecting point from the client session to the server session component. The server session has connections to both the traveler database and the Mongoose ODM. Following the diagram there are also symbols used to show the relationships between the different components. The little squares are used to show the ports between major components. The circles in the relationship lines between components tell us possible two different things. If a structure of an outer circle is connected to a line to one component, that component is a required interface. If the other line(s) does not have the structure of an outer circle, then it is a provided interface.

### Component Diagram



### 

### Sequence Diagram



Using the actor in the Sequence diagram as a user, who starts by signing into the internet and choosing a desired route. The route then will take us to our view, also known as the browser for the user to look at and book trips. The user then is able to interact with the browser via the controller. As they make selections, the controller then calls on the HTTP Client to retrieve data. From there, actions are made on the backend side of the full-stack application, and the http request gets routed to the server’s controller. The server’s controller is then able to abstract the data from Mongoose ODM. Mongo DB will process the request and start to send it back via the controller’s callback() function. The Http Client is then able to receive the processed data via the res.json() function. The front end’s controller then receives the resulting data object and assigns it to a scope. Once assigned the view is able to display the data to the user with the browser.

## 

## Class Diagram

A diagram of a travel application

Description automatically generated

The class diagram shows a total of twelve classes and their relationship to one another. Looking at the diagram we can begin with the TripInfo class. Inside TripInfo there are three attributes: totalprice:float, totalmiles:int, stopover:string. Attached to the class, there are three non-filled diamonds that connect to three aggregated classes: CruiseInfo, HotelInfo, FlightInfo. CruiseInfo has three attributes: name:string, cabintype:string, and price:float. FlightInfo contains three attributes as well: name:string, seatclass:string, and price:float. The HotelInfo class has five attributes: name:string, star:int, location:string, roomrequested:int, and price:float. The Itinerary class inherites the data from any of or all three of the previous classes and adds four more attributes: startingdate:int, returningdate:int, origin:string, and destination:string to create a package. HotelBooking is a class with a single method that retrieves HotelInfo:getHotel. FlightBooking retrieves FlightInfo just as CruiseBooking retrieves CruiseInfo. The TravelAgent class has four methods BookPackage(), BookHotel(), BookFlight(), BookCruise(). All of these methods are able to utilize their relationships to process the data needed to book the user’s trip. The Travel agent is also able to use its relationship to the TravelerInfo class, which contains an attribute of companionnum:int to save the all retrieved data that is necessary for booking a trip for the user. The TravelerInfo also inherits a MemberAccount. A MemberAccount contains a membernumber:int, frequent\_airline:string, memberstatus:int, and a memberclub:string. The MemberAccount class has an aggregated relationship with Membership\_Admin. Membership\_Admin has methods creditpoints(), getpoints(), and validate().

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

| **Method** | **Purpose** | **URL** | **Notes** |
| --- | --- | --- | --- |
| **GET** | Retrieve a list of trips | /api/trips | Returns all active trips |
| **GET** | Retrieve a single trip | /api/trips/:tripcode | Returns single trip, the trip’s ID, uses individual trips URL |
| **POST** | Adds a trip | /api/trips | Adds a new trip object to the list of trips |
| **DELETE** | Deletes a trip | /api/trip/:tripcode | Deletes a specified trip based on tripcode |
| **PUT** | Updates a trip | /api/trip/:tripcode | Updates specified trip based on tripcode |

## 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 computer

Description automatically generatedA screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a website

Description automatically generated

With Angular and Express, one is primarily meant for front end while the other is designed more for the server side of things. Express uses the Node framework to help build and use functions on the server side of the web application while Angular utilizes JavaScript  to build a scalable and fast functioning front end for a web application. An advantage of using a SPA application is with it being a single page that only updates what needs to be updated it helps keep the application fast. This allows for better experience for the users.

Testing the SPA is extremely important. We need to make sure all endpoints that we created are working as they should. In this case tested the API to GET and to PUT within the database.

**References**

Acharya, D. P. (2024, November 4). *What are Single Page Applications? examples, frameworks, and more*. Geekflare. https://geekflare.com/single-page-applications/

What is component diagram? (n.d.). https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-component-diagram/