

<Name of Software Application>

# **CS 465 Project Software Design Document**

Version 1.0

## Table of Contents

[**CS 465 Project Software Design Document** 1](#_heading=h.gjdgxs)

[Table of Contents 2](#_heading=h.30j0zll)

[Document Revision History 2](#_heading=h.1fob9te)

[Instructions 2](#_heading=h.1fob9te)

[Executive Summary 3](#_heading=h.3znysh7)

[Design Constraints 3](#_heading=h.2et92p0)

[System Architecture View 3](#_heading=h.tyjcwt)

[Component Diagram 3](#_heading=h.3dy6vkm)

[Sequence Diagram 4](#_heading=h.4d34og8)

[Class Diagram 4](#_heading=h.2s8eyo1)

[API Endpoints 4](#_heading=h.17dp8vu)

[The User Interface 4](#_heading=h.4h5b3qv007h8)

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

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | <09/25/22> | Derek Paterson | Executive Summary, Design Constraints, Sys Arch |
| 1.1 | <10/11/22> | Derek Paterson | Sequence Diagram, Class Diagram, Endpoints |
| 1.2 | <10/23/2022> | Derek Paterson | User Interface, Updated API Endpoints |

## 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)

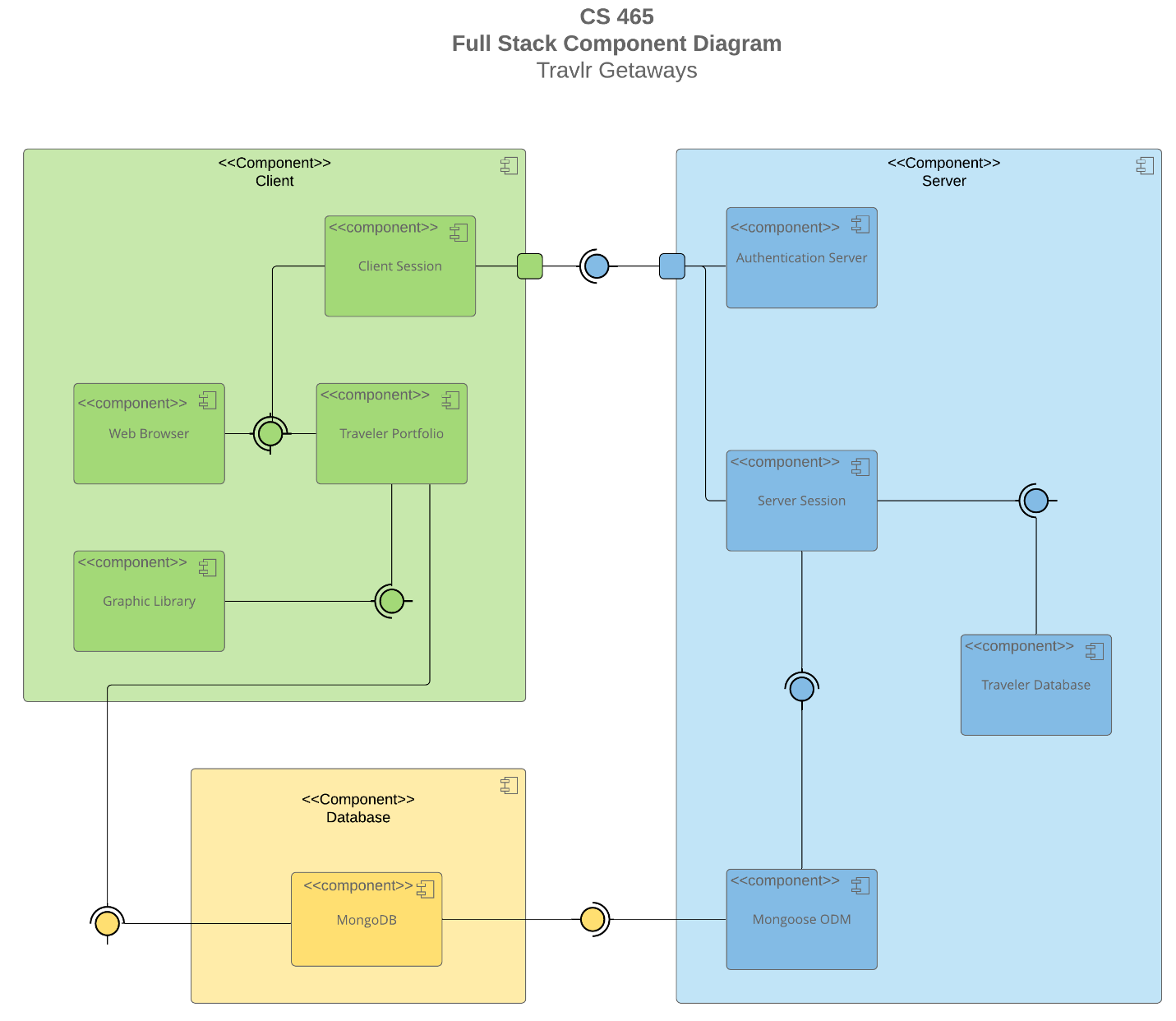
This project is a web-app for the Travlr Getaways company. The web app is built on MEAN, which is Mongo, Express, Angular, and Node. Mongo is the database system, Express and Angular are frameworks for web applications, with angular in particular being a scalable component based framework. Node is an asynchronous javascript platform that makes loading web content fast, easy, and scalable. The customer-facing side of the application has a ton of detailed pages so they can browse and find the best deals, and the admin single page application is fast and runs right with the server.

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

The design constraints we have are working within MEAN, needing a timeline for the customer, and adhering to a budget. The primary drawback with MEAN is that node is not good for big applications, as long as there is not significant feature bloat we should be fine for a small website. The advantages of a MEAN stack, however, mean that we can develop fast reliable sites, and we can do it quickly, so that should provide a satisfactory timeline for the customer and the speed of development will cut down on expenses.

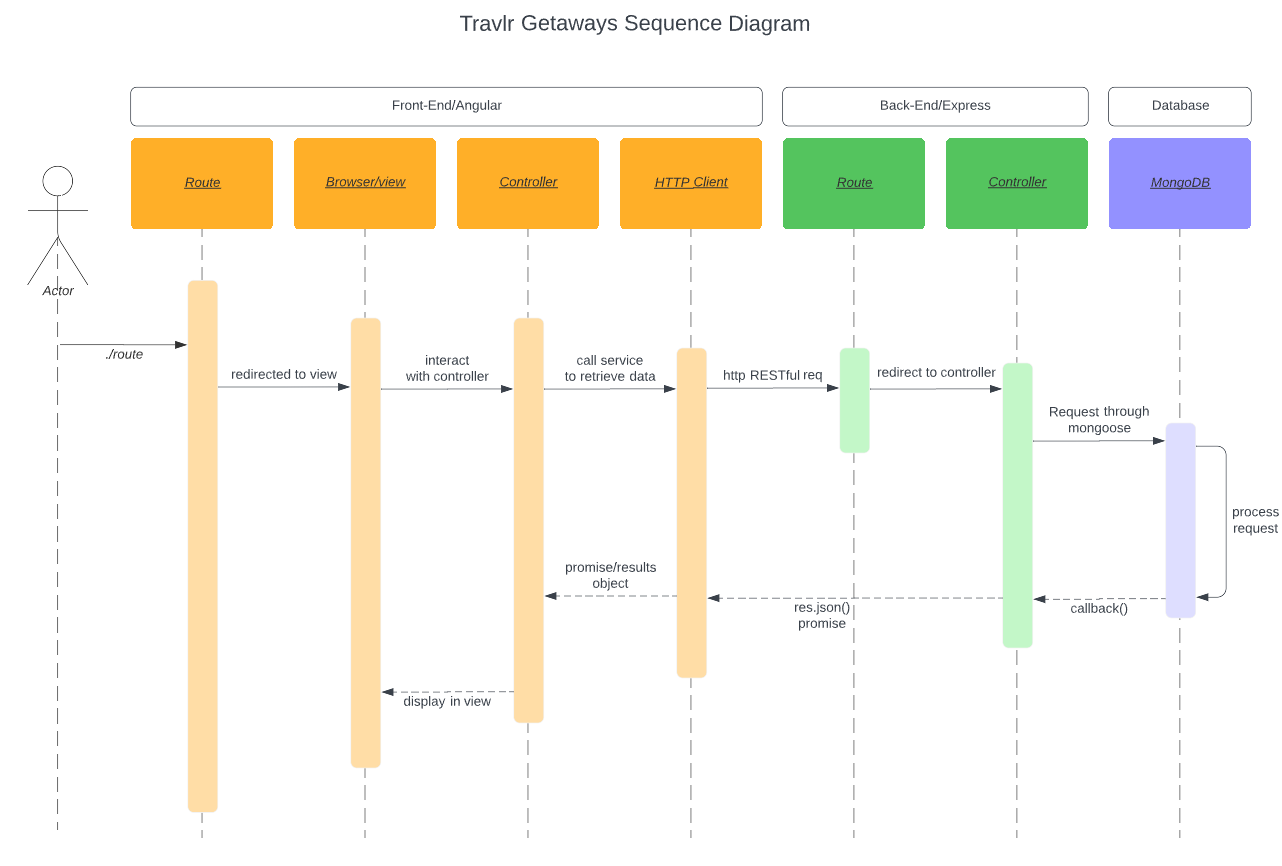
## [System Architecture View](#_heading=h.44sinio)

### Component Diagram



The system architecture for this project has three components, the database, the client, and the server.  
  
The database is the link between the client and the server, after authentication is handled with the server, the database will serve up the content to the client. The client side is a browser that loads the graphics and the html, css, and javascript. Finally the server authenticates the client session and processes requests from the client to tell the database what to serve to the client.

### Sequence Diagram



The user makes a request via the routing, which is then sent to the view (browser), which talks to the controller to get an updated view, and then the controller asks the HTTP client to send a RESTful request to the back end routing. The Backend route tasks the request and talks to the controller which sends a request via mongoose to the database. The database processes the request and issues a callback with the relevant information, which goes through the controller as a JSON response. the http client returns the resulting JSON object to the controller which serves it to the browser and updates the display.

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

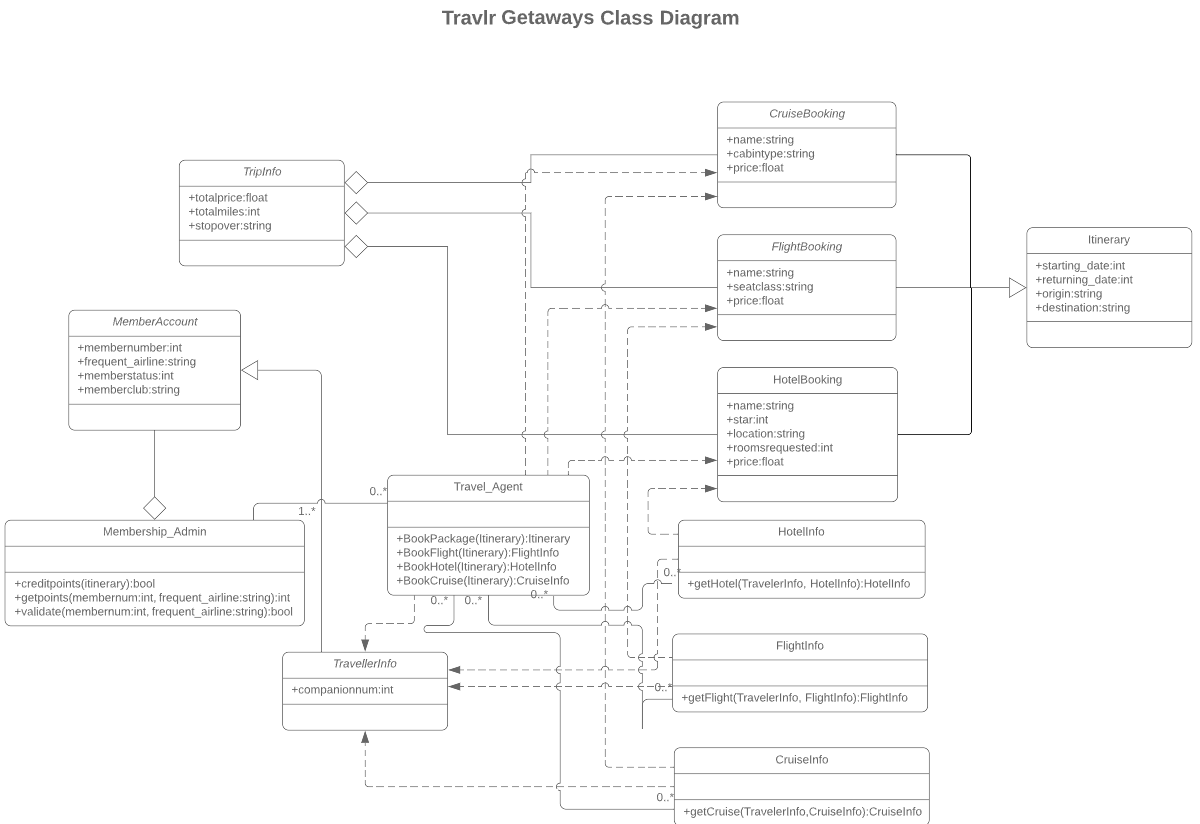
## 

## 

## 

## 

## Class Diagram



Since Travlr Getaways is based around booking entire vacations for people, functioning as a travel agent, it only makes sense for the core of the system to be the Travel\_Agent class. The Travel\_Agent class takes information from the MemberAccount and the TravelerInfo classes to get info from the flight/hotel/cruise classes and then goes to the booking classes to schedule the tip. These booking classes have detailed information on prices and accommodation. The end output of this is an Itinerary class that gives all the details of the trip, meanwhile the TripInfo class also inherits this information basically as a summary/cursory glance. The Membership\_Admin class allows management of the MemberAccounts and issuing credit and rewards points to member accounts.

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

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

| **Method** | **Purpose** | **URL** | **Notes** |
| --- | --- | --- | --- |
| **GET** | Retrieve a list of trips | app\_api/trips | Returns all trips in database |
| **GET** | Retrieves a single trip instance | app\_api/trips/:tripCode | Returns a single instance of a trip, based on the entered trip code. |
| **POST** | Login user | app\_api/login | logs in the user |
| **POST** | Register user | app\_api/register | registers a user to the database, hashes the password |
| **POST** | Add single trip | app\_api/trips | adds a trip with correct data structure to the database |
| **PUT** | Edit single trip | app\_api/trips/:tripCode | modifies an existing trip to update the parameters |
| **DELETE** | Delete single trip | app\_api/trips/:tripCode | deletes a trip |

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## The User Interface SPA backend with default trips and a custom trip Edit screen showing the details auto filled in the form

Updated trip card after submitting the description edit.

The Angular project structure is much more modular than the an express page, with angular each component you generate with the ‘ng’ command has its own html and css as well as various javascript functionalities, so we can leverage this buy having a lot of complex functions nested into the SPA angular backend without needing to load it all at once.   
  
We use SPAs because they are very fast and flexible, and easy to use. That being said, it can be difficult to keep upgrading them with new content and features and managing SEO is not great. Plus, with an SPA there is no back button so this can be a hassle unless you design around saving a lot of state locally for the user. One significant addition is SPAs cache information just one time, they don’t need to constantly download from the server, although this can be problematic if there is constantly changing content and older information is cached.   
  
In order to test the SPA , the easiest method is to use Postman. You make a call to the API (GET/PUT/POST) using the application and the route on your local express server (which needs to be running) (ie. localhost:3000/api/login) and test the API calls with that. Then you can boot up the angular server and test your functions and check the error codes in your console and the database to see if all your api endpoints are working correctly when applied to the SPA.