

Travelr 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 4](#_Toc36198468)

[Component Diagram 4](#_Toc36198469)

[Sequence Diagram 5](#_Toc36198470)

[Class Diagram 6](#_Toc36198471)

[API Endpoints 7](#_Toc36198472)

[The User Interface 8](#_Toc36198473)

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

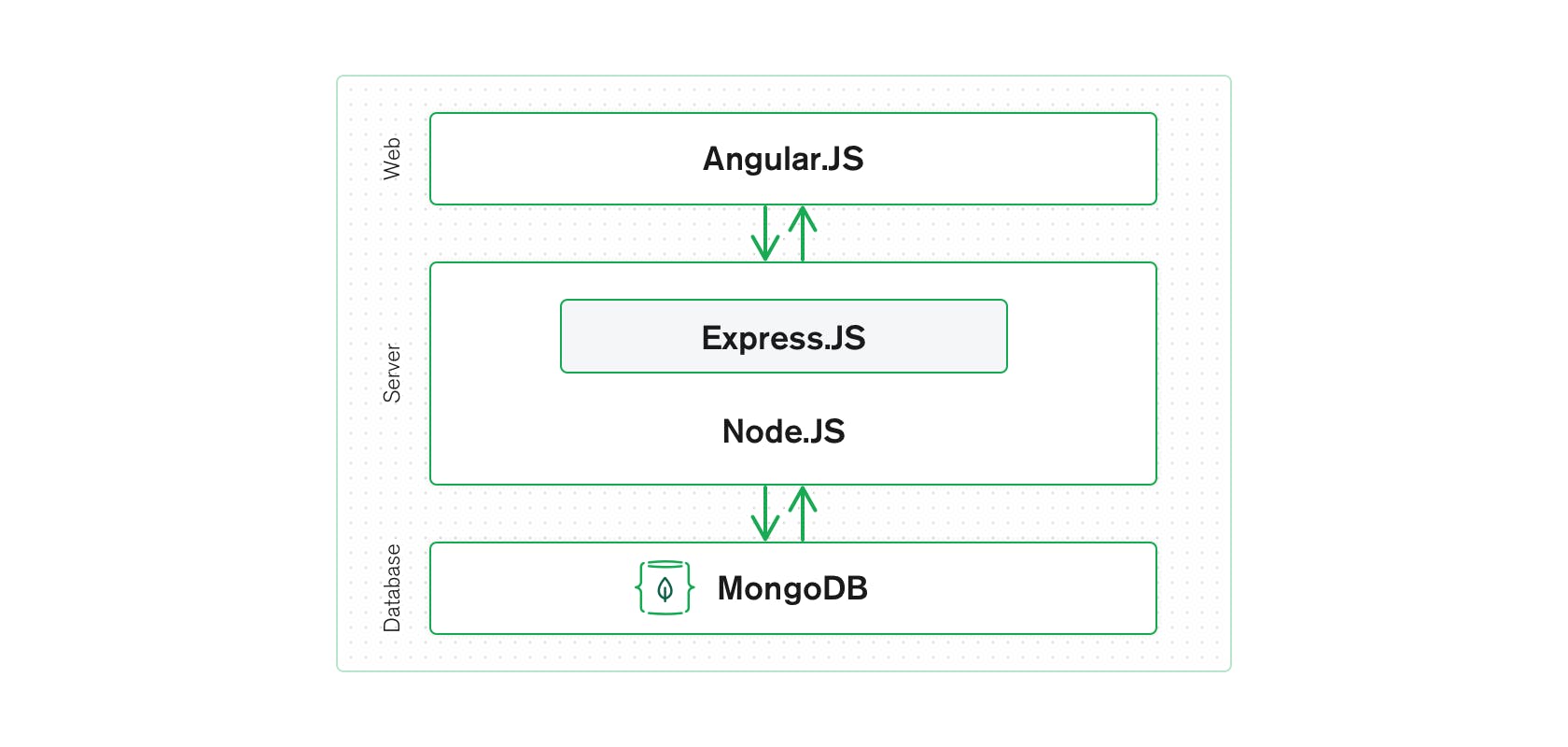
| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 07/11/23 | Derek Bamford | 1. Set up Model View Controller (MVC) pattern. 2. Took the static HTML site and moved it to a dynamic JSON file using Handlebars (HBS) |
| 2.0 | 7/25/23 | Derek Bamford | 1. Sequence Diagram added. 2. Class Diagram added. 3. API Endpoints |
| 3.0 | 8/8/23 | Derek Bamford | 1. The 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)

Travlr Getaways has asked that we flesh out their web-based application to bring it more in line with their competitor’s websites. This project will require a variety of front and back-end software to complete it is with this in mind that we have decided to go with MEAN as our stack of choice. MongoDB will be used as our database to store user information, travel bookings, and administrator privileges. Express, a Node.js framework, will allow us to build out both the web and mobile based application in parallel as well as construct our APIs. Angular, a JavaScript framework, will be used for our front-end development and includes things like validation, localization, and communication with the back end. Finally, Node.js will be employed as our server-side proxy because it can handle large amounts of connections simultaneously.



A single-page application was required by the company, this will allow for the application to dynamically rewrite the current web page with new data from the web server, instead of the default method of a web browser loading entirely new pages.

The customer facing side should allow users to see various travel packages offered, see room rates, and meals offered at the resort. A contact page must have a form with fields that can be filled out by the user so they can ask for additional information.

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

· Security Issues

Input Validation: All user input should be treated as potentially hazardous, and the proper steps should be taken to prevent bad actors from infiltrating the system such measures could be requiring charter limits and not having user input be run directly as code.

APIs: They should be RESTful and not contain a state are their core function.

Client/Server: Login attempts should be monitored, and a secure connection should only be made using SSH authentication. The use of CSF should also be implemented.

Code quality: Code reviews should be performed as well as dependency checks to prevent known vulnerabilities.

· Web server to host the web application.

The web server must be able to handle dynamic web browsing because of our use of software such as an application server and database. Apache HTTP Server could be a good choice as it is a free open-source web browser for Windows, Mac OS X, Unix, Linux, Solaris, and other operating systems. Stress tests should be run to prevent unexpected outages that may occur due to a potential increase in the user base.

· Storge of all associated images and data.

How much information will be stored server-side and how much can be cached on the client side will need to be determined. There are many options for cloud storage including AWS and Azure, price and timeline will be a factor in deciding which to go with.

· How quickly can pages be rendered across various platforms.

Application should run without issue across multiple hardware and software configurations, this includes various browsers on both desktop and mobile platforms.

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

### Component Diagram

Bold and italics are used to designate hierarchy of components.



The three main components of the system architecture are the ***Client***, the ***Server***, and the ***Database*** and within each are multiple subcomponents. The ***Client*** component houses the *Client* *Session* which interacts with the *Web Browser* and the *Traveler Portfolio* as shown by the assembly icon, this indicates that the three component interfaces are intrinsically linked to one another. The *Traveler Portfolio* links to the *Graphic Library* component with an assembly icon as well. The ***Client*** component makes a distinct interaction with the ***Server*** component, as shown by the port and assembly icons. Within the ***Server*** component we have the *Authentication Server* and *Server Session* components, each of which only shares an association to the ***Client*** component and are separate from each other. The *Server Session,* however, does interact with the *Traveler Database* and Mongoose ODM, a Node.js-based Object Data Modeling library for MongoDB, which is why it is shown to interact with *MondoDB* in the ***Database*** component.

Sequence Diagram

A diagram of a trade end

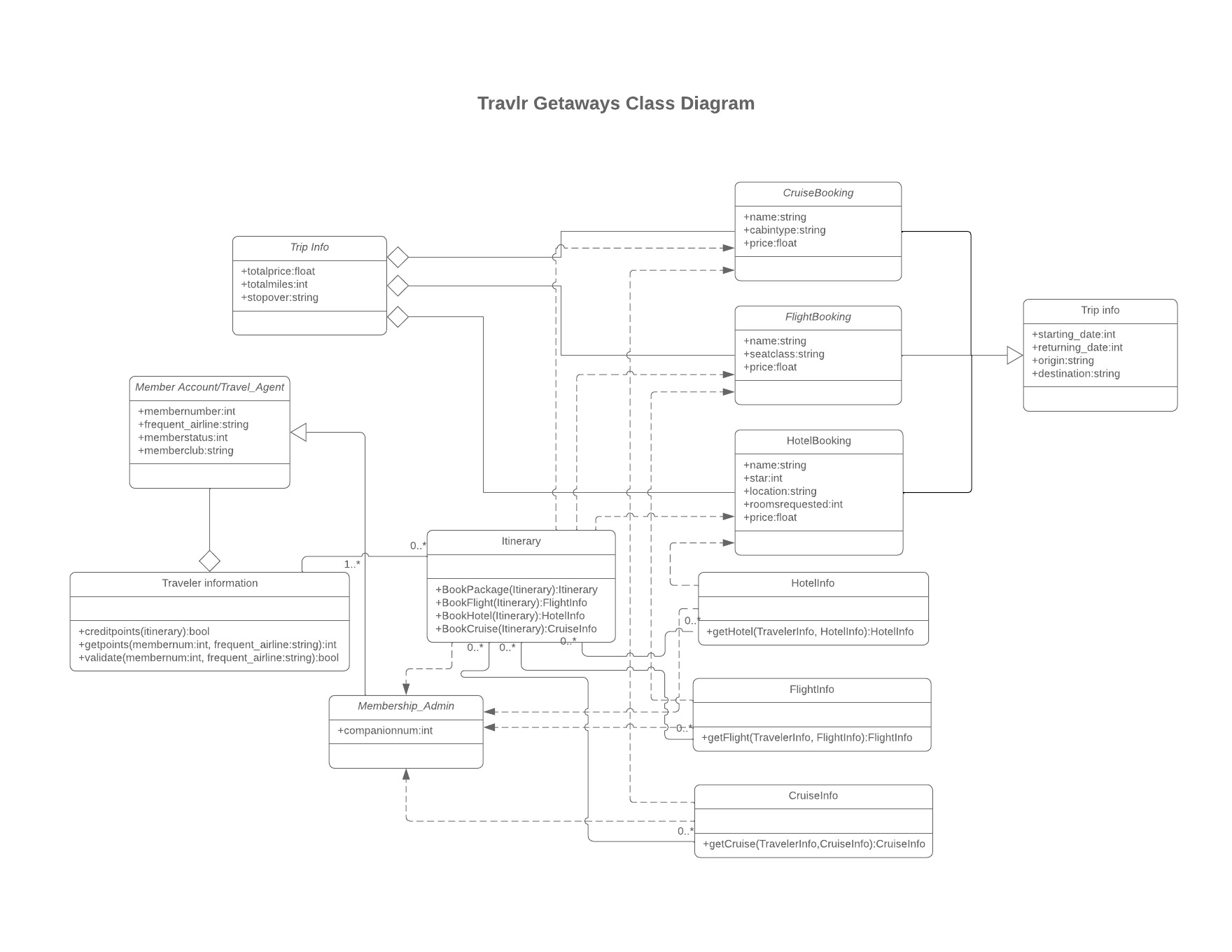
Description automatically generated

When a user interacts with a website, they are only interacting with the frontend’s graphical user interface (GUI), while not only aesthetically pleasing it also separates users from interacting with the codebase itself. The backend is where the magic happens, this is where the code interacts with several systems to allow for the frontend to dynamically change and display a multitude of information.  I will briefly discuss those backend interactions in this section.

Users who visit Travlr Getaways web page will always be brought to the site’s homepage, from here they have navigational options such as Travel, Food, About, etc. The backend handles these changes through routes, which allows for multiple views and navigation between them. Routes allow for the users to switch between these views without losing the application state and its properties. The browser then uses controllers, where we have defined values and functions that make up the behavior of the application, to get the data onto our browser view.  More than likely this data will be stored on server database in which case more interactions are needed to complete the task of updating the view.

The HTTP client performs HTTP requests to consume, read engage, with an API. RESTful APIs are standard and allow for interactions only upon request, which helps prevent data from bad actors. Once the client-side has made a request to the server-side another controller handles the call and passes it to the server-side model which exists between the MongoDB database storage and the logic of the application.  Good models are based on “schema”, which describes the properties of the model. The information is then pulled from the database and the call sequence is performed in reverse, where it finally renders the new data to the viewing window.

## Class Diagram



*Member Account/Travel\_Agent* Class: Holds the public attributes (+) of member ID number, most frequent airline used member status and memberclub. Italicized class name indicates and abstract class.

Traveler Information Class: Is associated with the *Member Account/Travel\_Agent Class* through aggregation (represents a “part of” relationship) The public attributes are creditpoints, getpoints, and validate. Multiplicity is set to 1 to many.

*Membership\_Adim*: An abstract class that inherits the attributes of the *Member Account/Travel\_Agent* Class and has Realization relationship with many other classes.

Itinerary Class: Contains public attributes for BookPackage, BookFlight, BookHotel, and BookCruise. This class is associated with the info classes and has a realization relationship with the booking classes.

Booking classes: All are abstract and contain the public attributes price and name. Additionally, the *CruiseBooking* has cabintype, *FlightBooking* has seatclass, and *HotelBooking* has star, location, and roomsrequested attributes.

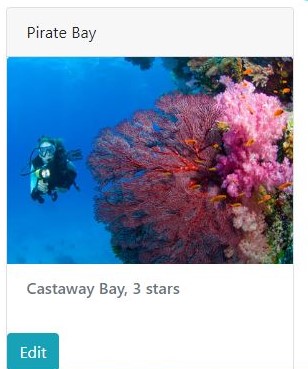
Trip Info Class: I elected to have two Trip info classes, one that contains the information regarding price, miles, and stop overs and has an aggregation relationship with the BookingClasses and the other contains information regarding the start and return date, the origin of the trip, and the destination. This class has an inheritance relationship with the BookingClass.

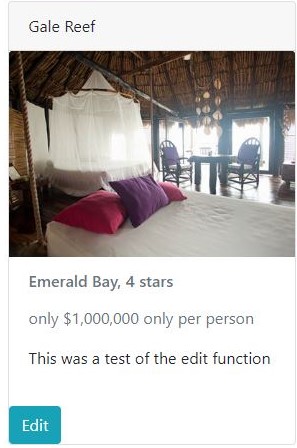
Info Classes: The contain public methods to retrieve the data related to Hotel, Flight, and Cruise information. The have a general association with the Itinerary class and have their multiplicity set to “

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

| **Method** | **Purpose** | **URL** | **Notes** |
| --- | --- | --- | --- |
| **GET** | Retrieve a representation of a resource.  SAFE - IDEMPOTENT  Multiple identical requests return the same result. | /api/trips | Returns MongoDB database data for all trips through the api and returns it to the browser |
| **GET** | Retrieve a representation of a resource.  SAFE - IDEMPOTENT  Multiple identical requests return the same result. | /api/trips/DAWR210315 | Returns MongoDB database data for a single trip through the api and returns it to the browser |

## The User Interface

 A screenshot of a phone

Description automatically generated 

Express is used for building customer facing HTML web pages, it is a NodeJS framework for server-side web applications using RESTful APIs. Originally, we built a web application using Express for the server-side rendering of the application, the user would navigate the application to which the input would be read, and the pages updated on the server before displaying to the user. Eventually we moved to the Angular framework to convert the application to a single-page application (SPA) where HTML is cached client-side and only uses the RESTful API for data updates.

It is always good to have error handling in your code so that you can deduce when something goes wrong, I also found that keeping an eye on PowerShell while Express was running allowed me to check if my GET requests were working, which brings me to the next point. Methods such as GET, POST, PUT, and DELETE are used to interact with the database. It allows a user to manipulate the data from the application without having to go directly into the database, but for this to occur we need to talk about endpoints. Endpoints are a specific location within an API that accepts requests and sends back responses. This was tested using Postman which simplifies each step of the API lifecycle so you can create better APIs. Finally, we added security to the application which allowed the methods to be protected from someone who may not have access to these methods, this was also tested through Postman.