

<Travlr Getaways Web Application Project CS 465 – Noah Khomer>

# **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 | <07/09/2025> | <Noah Khomer> | <Starting Milestone 1 > |
| 2.0 | <08/20/2025> | <Noah Khomer> | <Ending Final Project> |

## Instructions

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

The company Travlr Getaways require that a fully integrated full stack web application functionality to provide travel booking capabilities for their customers along with administrative tools for their internal staff such as help desk, management etc. The system is going to be developed using the MEAN stack of MongoDB, Express.js along with Angular and Nodse.js. The following will be used to offer a scalable and efficient architecture approach for building the web application dynamically. The customer-facing website will also be made functional by establishing the website to allow users to browse and book trips, view their details, and lastly interact with the interface with a design that is both functional and appealing for the customer. The administrative component meanwhile will be a single page application that will enable the staff members to securely manage trip listings, view all the booking data, and lastly perform the updates to the content regularly. In this manner, architecture assures that a responsive experience for both user and the administration is defined and supported by the unified tech stack and secure API integration.

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

There are a few design constraints that need to be kept in mind when it comes to building the Travlr Getaways web application. First, we are limited with potential since only the MEAN stack can be used. It is a bit tricky when working with a limited tech stack because the application must support modern web browser integration, and it must provide a good layout that is responsive for not just one screen but multiple screen sizes. The second element that I find in the design constraint is the security element, and the reason for this is because administrators are going to be getting their roles, and there must be some sort of access control for authenticity. And security. Lastly, the hosting must be compatible with Node.js and MongoDB, which is also limiting a lot of deployment options to the platform when it comes to server-side applications. These constraints are defining the boundaries of the system, how much it can influence science around performance, security, and lastly, scalability.

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

### Component Diagram



### System Architecture View

### For the application component view. From the component diagram, the web application is composed of multiple components that are going to be working together with the standard MEAN architecture, the front end. It's going to be built with Angular so it can include the customer interface and the admin SPA. Both must connect with the back-end services, which are going to be built using Node.js and Express.js. They can handle business logic, routing, and their API endpoint management. The data is going to be stored and retrieved from MongoDB, which is a NoSQL database that can support different flexible documents. Based on the amount of storage we have. And lastly, authentication services require that security access for the administrative features exist. From the diagram it can be concluded that the components are loosely interacting with each other with RESTful API, so it can allow for scalability and maintainability as the system continues to grow. In the future.

### Sequence Diagram

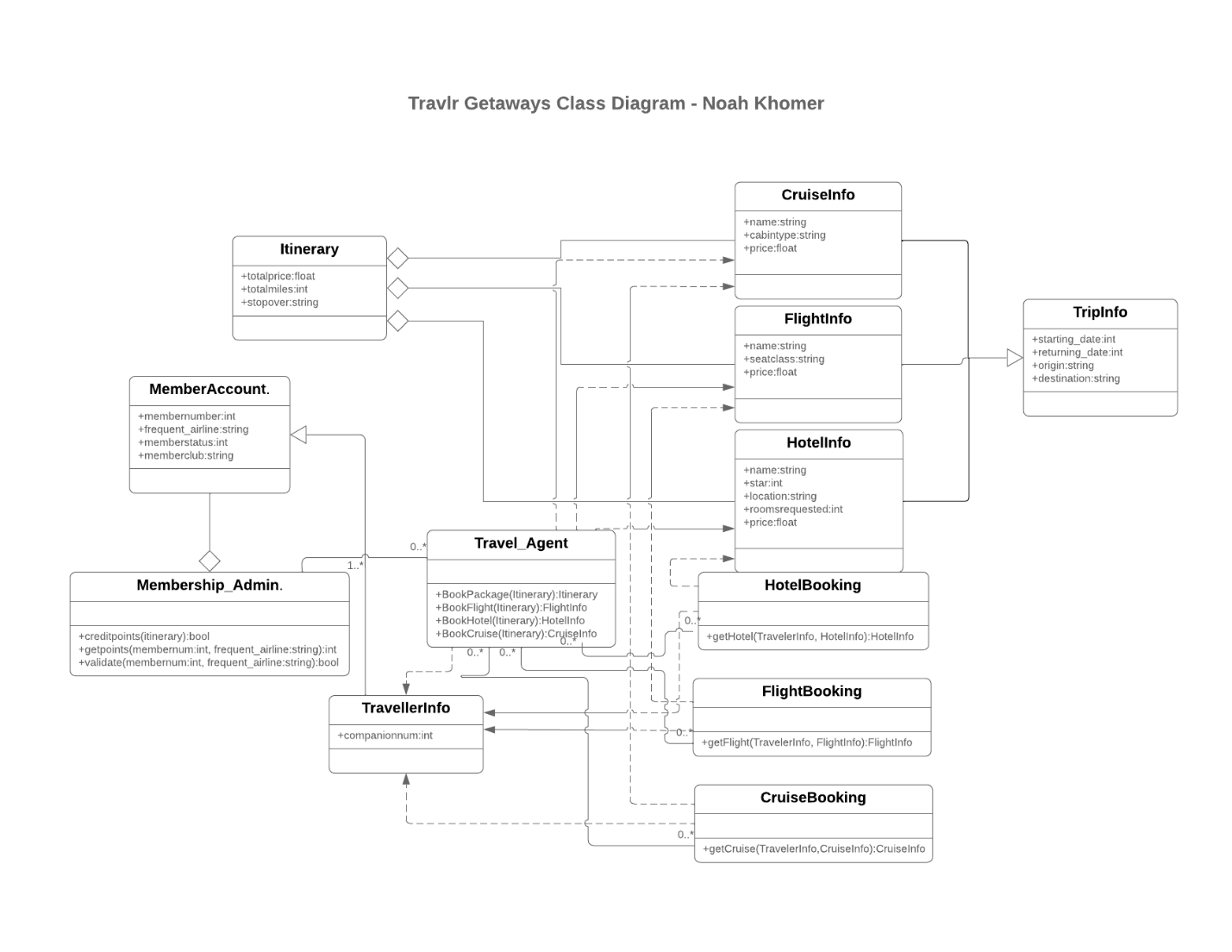
A diagram of a diagram

AI-generated content may be incorrect.

Through the diagram, we see the finishing observance upon the requested end via the Travlr Getaways stack. A user selects any route in the browser, the Angular view loads. It then calls its controller to assemble some data for it. The controller calls an Angular service that makes an HTTP request of some Express route on the Node server. The route passes the call to the server-side controller. That controller instructs Mongoose to read or write some data in MongoDB. MongoDB returns some results to Mongoose. The server responds with a JSON object to the browser. The Angular service resolves its promise, the controller populates the view model, and finally, the data is reflected on the page.

Sign-in goes through the same phases but with an extra verification check. The server authenticates the user and sends the session token or message. Trips Simple read: The client calls the trips endpoint, the server queries the MongoDB, and the list or a single trip is returned as JSON. Admin operations count one more rule: The server verifies that the request is from an authorized user before the writing operation is created, updated, or deleted and then performs the write operation and returns the result.

## Class Diagram



Focusing on the main travel ideas and their interrelation, there is TripInfo holding the actual trip details: origin, destination, dates, etc. Itinerary is a group of all the pieces of the trip and tracks totals like price and miles. FlightInfo, HotelInfo, and CruiseInfo are all eligible options for every possible segment. TravellerInfo stores traveler details that are reused across bookings. MemberAccount stores membership number, status, and program data. Membership\_Admin allows checking or crediting member points.

The Booking classes associate a traveler with a chosen option. FlightBooking keeps track of a traveler and a flight. HotelBooking keeps track of a traveler and a hotel. CruiseBooking keeps track of a traveler and a cruise. Travel\_Agent is a service class that generates bookings from an itinerary. This layout keeps the focus of each class narrow, promotes reuse, and allows for fairly easy extension to another transport type with little change to the rest of the model.

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

**Documentation of API points**

| **Method** | **Purpose** | **URL** | **Notes** |
| --- | --- | --- | --- |
| **GET** | <Retrieve list of things> | </api/things> | <Returns all active things> |
| **GET** | <Retrieve single thing> | </api/things/:thingId> | <Returns single thing instance, identified by the thing ID passed on the request URL> |
| |  | | --- | | **POST** |  |  | | --- | |  | | |  | | --- | | Sign in |  |  | | --- | |  | | |  | | --- | | /api/auth/signin |  |  | | --- | |  | | |  | | --- | | Validates credentials  and returns token  or session info |  |  | | --- | |  | |
| |  | | --- | | **GET** |  |  | | --- | |  | | |  | | --- | | Get all trips |  |  | | --- | |  | | |  | | --- | | /api/trips |  |  | | --- | |  | | |  | | --- | | Returns an array of  trip documents |  |  | | --- | |  | |
| |  | | --- | | **GET** |  |  | | --- | |  | | |  | | --- | | Get one trip |  |  | | --- | |  | | |  | | --- | | /api/trips/:tripId |  |  | | --- | |  | | |  | | --- | | Looks up by id or code and  returns details |  |  | | --- | |  | |
| |  | | --- | | **GET** |  |  | | --- | |  | | |  | | --- | | Get current user |  |  | | --- | |  | | |  | | --- | | /api/auth/me |  |  | | --- | |  | | |  | | --- | | Reads user info from token |  |  | | --- | |  | |
| |  | | --- | | **POST** |  |  | | --- | |  | | |  | | --- | | Create a trip |  |  | | --- | |  | | |  | | --- | | /api/admin/trips |  |  | | --- | |  | | |  | | --- | | Admin only.  Creates a new trip |  |  | | --- | |  | |
| |  | | --- | | **PUT** |  |  | | --- | |  | | |  | | --- | | Update a trip |  |  | | --- | |  | | |  | | --- | | /api/admin/trips/:tripId |  |  | | --- | |  | | |  | | --- | | Admin only.  Updates fields on an  existing trip |  |  | | --- | |  | |
| |  | | --- | | **DELETE** |  |  | | --- | |  | | |  | | --- | | Delete a trip |  |  | | --- | |  | | |  | | --- | | /api/admin/trips/:tripId |  |  | | --- | |  | | |  | | --- | | Admin only. Removes a trip |  |  | | --- | |  | |
| |  | | --- | | **GET** |  |  | | --- | |  | | |  | | --- | | List member bookings |  |  | | --- | |  | | |  | | --- | | /api/bookings/:memberId |  |  | | --- | |  | | |  | | --- | | Returns bookings for a  member |  |  | | --- | |  | |
| |  | | --- | | **POST** |  |  | | --- | |  | | |  | | --- | | Create a booking |  |  | | --- | |  | | |  | | --- | | /api/bookings |  |  | | --- | |  | | Creates a booking from selected options |

## The User Interface

The user interface concerning this project is particularly divided into client / travlr-admin and Angular application, offering Create, Read, Update, Delete operations on resources named "trips." Most probably, for reading activities, looking at trip-listing.component.ts and trip-card.component.ts having a list of trips is displayed and explicitly referred to as an individual card that includes information. A person can choose, and by clicking on one trip, trip-detail.component.ts to view all the details of a trip. The third-feature set this project brings is in making (Create) and Controlling (Update) activities toward a trip that trip-form.component.ts would primarily offer an interactive form. Create one or update an item with initial data available and allow the user to change and save the details. Similarly developed, you would include a mechanism to delete this trip, perhaps through a button or icon in one of the trip-detail or trip-listing components by ensuring you got all the functionality normally integrated. With all these seamless components, the management of trip data directly from the user interface becomes such a very fluid and intuitive experience.

A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

The implementation of this system project will take place in two primary sections, namely the folder of `client/travlr-admin`, which will have within itself an Angular single page application (SPA), and the folder of `cs465-project`, which will contain an Express.js-backed server. While the frontend based on Angular separates responsibilities by components in order to implement the user interface, and services in order to interact with the data in the database and the server, as well as environments and security features by separate modules, the backend as based on Express is additionally broken down into `app api` - which handles the RESTful API (provides controllers, models, routes for authentication and data manipulation) and another section less visible the `appserver’ - that will mainly render the static files and also some server side elements.

In comparison with common web applications, the Angular single page application has more functionalities and has a more pleasant interaction with the user. Every time the user adds, requests, updates, or removes an element of the interface, only the relevant parts of the page are updated, making the transition smoother and much more efficient. This approach also eliminates the tight binding that the traditional approach has in place and allows both front-end and backend systems to move at different times, and to be deployed in multiple release cycles. Here, on the one hand, the developers work only on Angular app creation, managing mostly UI; and on the other hand, the necessary backend processes and data supply are provided by Express API.

Verifying the proper functionality of a SPA API database combination is not limited to just one method. A test view that spans multiple strategies such as correct Angular components and services, back routing pages with model and controller incline to figure out the working logic. In the absence of these types of tests, the development environment virtually simulates an actual preproduction their PUT and GET requests and goes into another cycle, to download it again into the current environment fresher than before. Backtesting data both facilities to a 100% reliable idea, including application routing and call service shifting.