Travlr Gateways Web-based Application

CS 465 Project Software Design Document
Version 1.0

Table of Contents

Document Revision History

Version	Date	Author	Comments
1.0	11/10/23	Stephen Owusu-Agyekum	Documentation of the executive summary, design constraints, and the system architecture view to the client web-based app

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

This project aims to create a full-stack web application for our customer, Travlr Getaways. Travlr Getaways has outlined specific requirements for a full-stack web application, including a customer-facing website, a database, and an admin single-page application (SPA). The MEAN stack, which consists of MongoDB, Express.js, Angular, and Node.js, was chosen as the development framework for this project due to its flexibility and scalability. At the front end, the customer-facing website will be built using Angular, a robust front-end framework. The angular modular architecture allows for the creation of dynamic and responsive user interfaces. It will have swift page loading, and the client can communicate with the database and the server through the interface.

At the back end of the customer-facing, Node.js and Express.js will be utilized for the backend to handle server-side logic and API interactions. RESTful API endpoints will be created to facilitate communication between the front end and the database, ensuring smooth data flow.
The framework is developed in JavaScript, and the MongoDB database will be used on the client-server side, while the JavaScript web server will be JavaScript Node. MongoDB, a NoSQL database, was chosen for its flexibility in managing unstructured data, making it ideal for storing travel-related information such as locations, accommodations, and user preferences.

The admin SPA will also be built with Angular on the front end, offering a uniform and seamless experience across the customer and admin interfaces. The admin SPA will use the same backend as the customer-facing website to ease development and ensure consistency. Using data with multiple dimensions, we may transport data between the MongoDB server and the client to ensure it performs best. The MEAN stack's front end is Angular JavaScript, which will be used to build the application. The Administrator SPA is designed for internal use, providing administrators with powerful tools to manage and oversee the application. The admin SPA will interact with the same MongoDB database used by the customer-facing website, ensuring a unified data source. The requirement for the MEAN stack design for Travlr Getaways' online application seeks to create a strong, scalable, and secure solution that meets the required customer and admin functionality.

Design Constraints

While we start putting things together to develop the web-based application for Travlr Gateways, we first need to focus more on the client's requirements. Travlr Getaways, our client, needs to know what their clients desire from their web application. We should also prioritize a seamless and intuitive user interface, responsive design for various devices, and efficient navigation to enhance overall user satisfaction. The app should be user-friendly. This may necessitate numerous modifications to the application's user interface, as new functionality may make it less user-friendly.

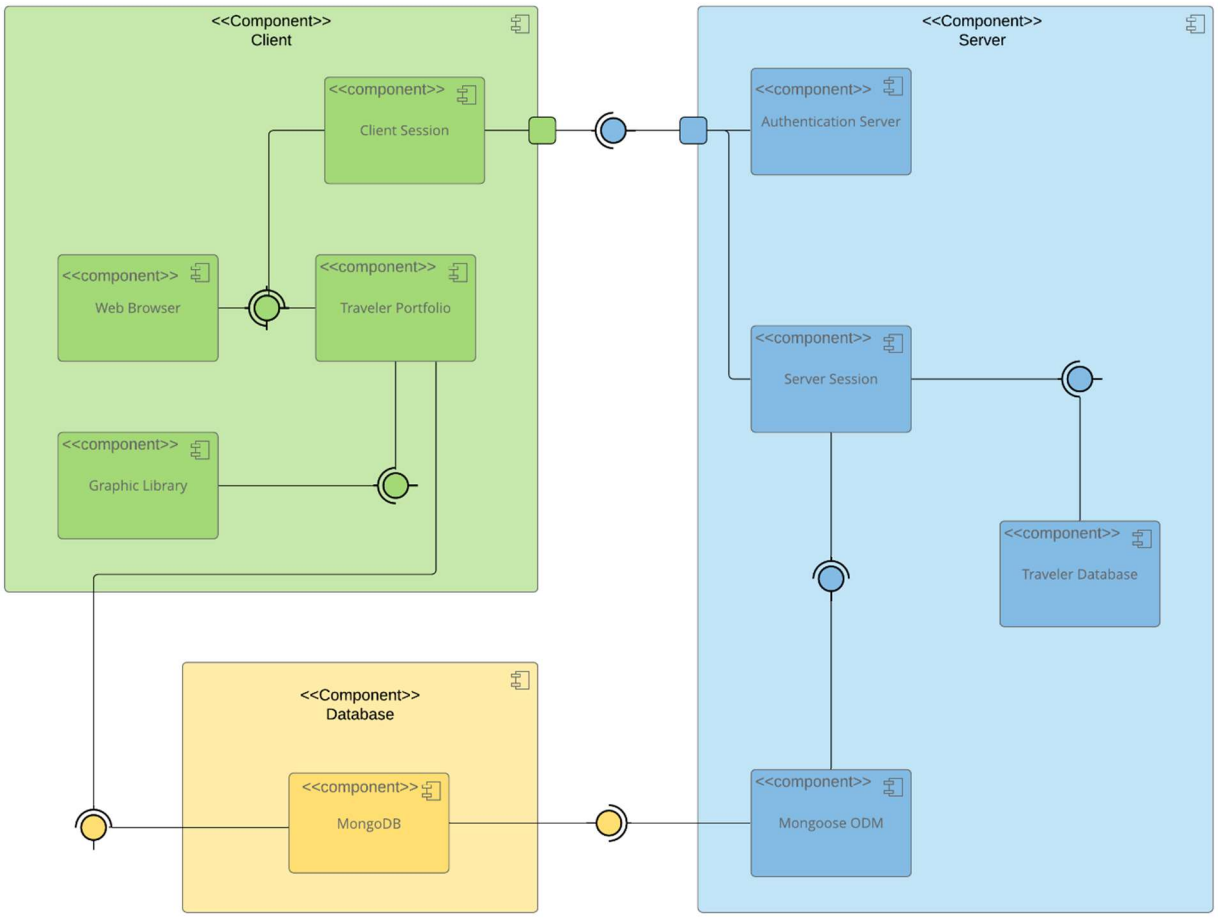
Financial constraint. The budget constraints necessitate careful resource allocation and may influence the extent of features and scalability achievable within the project. We must align development efforts with budget constraints, making strategic decisions on features, technologies, and development processes. Constraints in this area may limit the extent of customization and feature richness.

Time constraint. We should also efficiently manage development timelines by prioritizing essential features and functionalities. Time constraints may impact the depth of testing and the thoroughness of feature implementation. The compatibility constraints with the current framework, such as third-party APIs or older systems, may pose hurdles to effortless integration and entail a lot of testing. A greater amount of time may necessitate more significant expenditures to complete the web-based application for Travlr Gateways, which depends on the budget.

Security constraints. Data security and privacy rules will influence the design decisions, potentially influencing the deployment of robust authorization and authentication procedures. Implement strong security measures to safeguard user data and guarantee that we comply with data protection laws. Security limitations directly impact user trust and the application's overall integrity.

System Architecture View

Component Diagram



A text version of the component diagram is available: [CS 465 Full Stack Component Diagram Text Version].

The component diagram above illustrates the web-based application's key components:

- The components of the client side on the left
- The components of the server side on the right
- The database

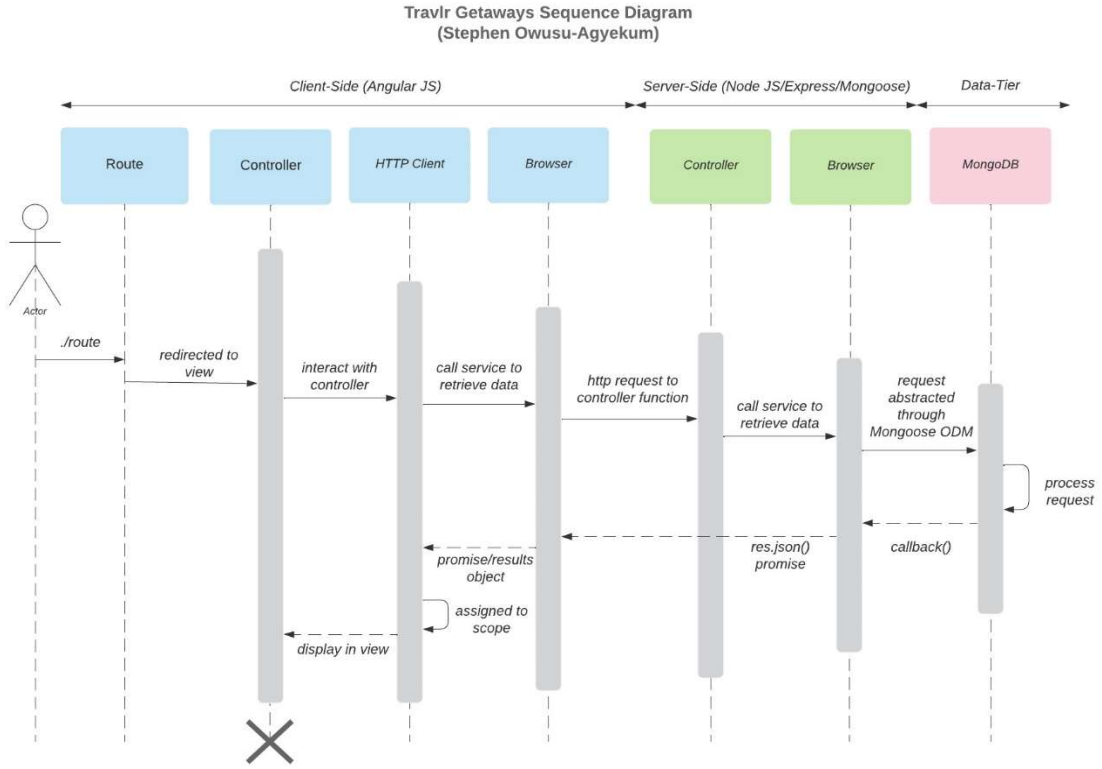
The client-side component comprises a web browser, traveler portfolio, and graphic library. The server-side component consists of an authentication server, server session, traveler database, and Mongoose ODM. The app will have a user interface allowing users to interact with the data stored in the database. The ports connect the client and server sides. Both server and client sides are connected to the MongoDB database to store information which is responsible for managing and interacting with the traveler database.

The web server will be responsible for handling incoming HTTP requests from clients. It may include components like Apache, Nginx, or another web server software. The web browser in the diagram represents the client-side runtime environment that renders HTML, executes JavaScript, and manages the user interface.

The authentication server will manage user authentication and authorization processes. It will provide secure access control mechanisms for the web application by interfaces with the Web Browser for user login and session management. The graphic library will provide a set of functions and tools for rendering graphics on the client-side, and it would be used by the Web browser to display graphical elements on the user interface.

The Traveler portfolio will manage, and display information related to a traveler's portfolio, which could include travel history, preferences, and other relevant details. It will interact with the Graphic Library to visualize travel-related data as well as retrieving and storing data from or to the Database. Also, the Client Session represents the client-side storage for maintaining user sessions, stores session-related data such as user credentials or authentication tokens. The Traveler Database represents the MongoDB database where traveler-related data is stored including user accounts, traveler portfolios, and other application data.

Sequence Diagram



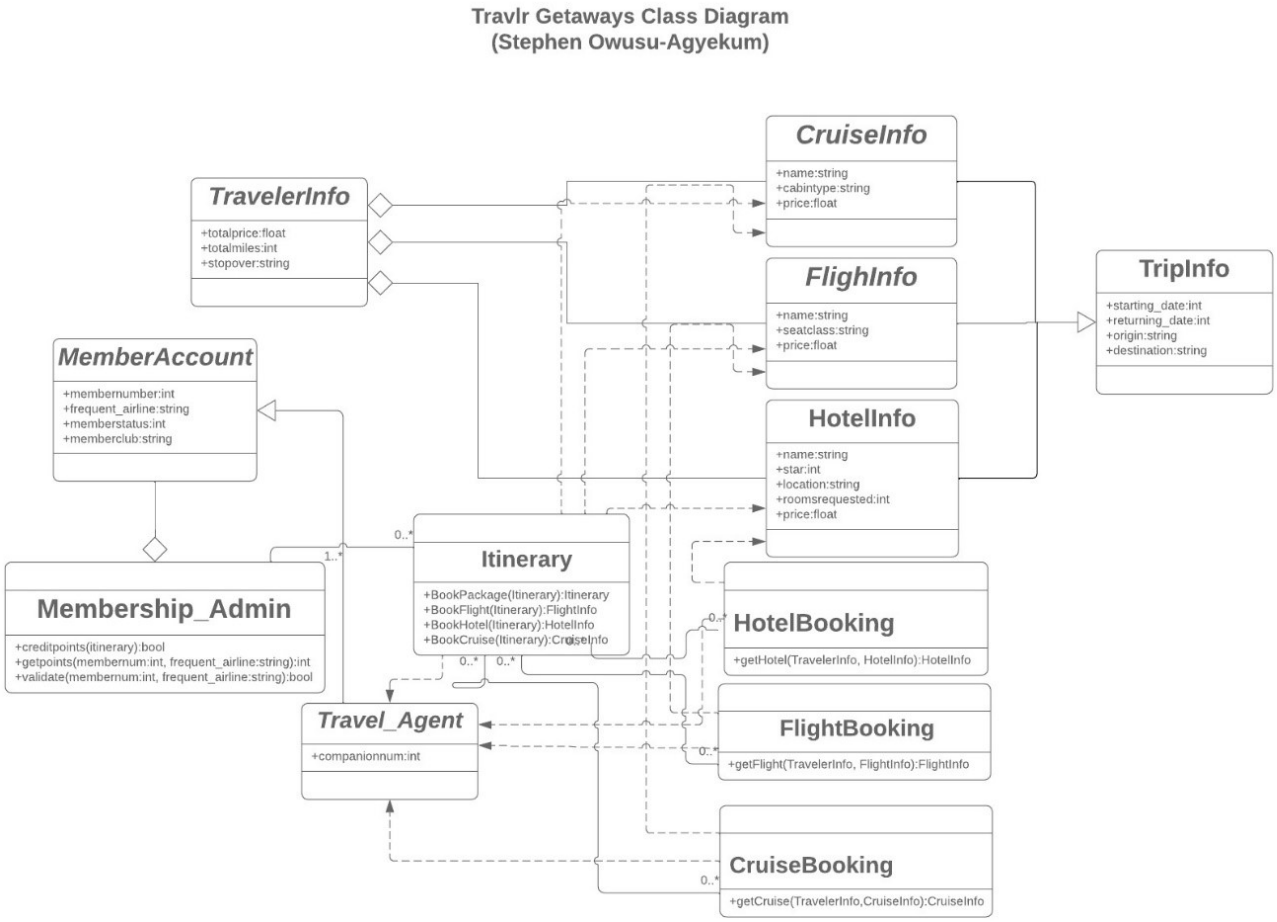
The sequence diagram above has three main components involved in the web application: database (Data-Tier), Client, and Server. The sequence diagram showcases the interactions and logic flow among these components during various processes like sign-in (authentication), Trips, and Admin interactions. Clients initiate the sign-in request by sending user credentials such as username and password through the browser.

The server side would then use MongoDB to send a call to the website. The server would then receive the client's sign-in request and validate the credentials against the database. If the credentials are valid, the Server sends back a successful response and an authentication token. If the provided credentials are invalid, the Server will send an error response. When the client is successfully authenticated based on the credentials provided, they are stored as authentication tokens in local storage or as cookies for subsequent authenticated requests.

During the Trip process, the Client requests trip data from the Server, including the authentication token in the request header. The server then receives the request and validates the token to ensure authentication. The Server will retrieve trip data from the database if it is valid. If invalid, the Server will send back an authentication error. After successful validation, the client will receive and display the trip data to the user.

For the Admin interaction, the Client initiates actions like creating, updating, or deleting trips, users, or other administrative tasks. The Server then receives admin requests, performs the necessary validations, and updates the database accordingly if it is valid. The client, or the Admin Interface, receives success or error responses and updates the interface accordingly.

Class Diagram



The class diagram illustrates the many relationships among the classes. Some of the classes include CruiseInfo, TravellerInfo, FlightInfo, and HotelInfo. These classes represent specific details or information related to different aspects of a trip. Every user will initially have an ordinary account. Travel agent, for instance, would be a typical account role. The itinerary will be used to manage and organize a trip's various components, possibly including CruiseInfo, TravellerInfo, FlightInfo, and HotelInfo. The travel agent can also use the itinerary to view the various items and communicate with one another.

The TravellerInfo stores details about individual travelers, like personal information, preferences, and special requirements. FlightInfo holds flight information, including departure or arrival times, airlines, flight cabins, and seat availability. CruiseInfo contains data about cruises, such as

schedule dates, destinations, and amenities. HotelInfo also manages data regarding various hotels, such as room types, facilities, locations, and pricing.

Now, let's come to the TripInfo class. TripInfo aggregates and manages instances of CruiseInfo, TravellerInfo, FlightInfo, and HotelInfo and also provides a cohesive representation of a complete trip. The Membership_Admin class represents administrative privileges and functionalities for managing memberships and all possible administrative tasks. Travel_Agent class contains some functionalities for travel agents, such as booking management and customer interactions. The MemberAccount also handles general account functionalities like authentication, user profiles, and settings.

Let's describe the last three classes: HotelBooking, CruiseBooking, and FlightBooking. HotelBooking will manage all hotel reservations, room choices, availability, and booking-related activities. All reservations for cruise vacations, cabin preferences, itinerary modifications, and associated booking services are managed by CruiseBooking. On the other hand, FlightBooking handles ticketing, seat assignments, and other associated airline booking procedures. The information on the website will be visible to the user after a connection with the data and the website. This happens as long as the user is interacting with the page.

In summary, the class diagram shows how these JavaScript classes handle many online application capabilities, including bookings, user roles, trip information, and membership-related tasks. They also serve as the application's fundamental framework, facilitating the management and arrangement of trip data, interaction with users, and service reservations.

API Endpoints

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	<Create new list of things>	</api/things>	< We will create a new list of things.>
POST	<Create a single thing>	</api/things/:thingid>	< We will create a single instance of the object, denoted by the item ID passed through on the request URL>
PATCH	<Update full list of things>	</api/things>	<We will update and modifies full list of things>
PATCH	<Update and modify single thing>	</api/things/:thingid>	<One instance of the object will be updated and modified using the thing ID provided on the request URL>
PUT	<Update and replace full list of things>	</api/things>	<We will update and replaces full list of things>
PUT	<Update single thing>	</api/things/:thingid>	<With reference to the thing ID supplied on the request URL, we shall update and replace a single instance of that thing>
DELETE	<Delete full list of things>	</api/things>	<Deletes full list of things>
DELETE	<Delete single list of thing>	</api/things/:thingid>	<Delete single thing instance, identified by the thing ID passed on the request URL>

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.>

<Summarize the Angular project structure and how it compares to the Express project structure. Be sure to describe the rich functionality provided by the SPA compared to a simple web application interaction.

Describe the process of testing to make sure the SPA is working with the API to GET and PUT data in the database.>