

ryanshatch\_app

# **CS 465 Project Software Design Document**

Version 1.0.1

## 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 | 11/11/24 | Ryan Hatch | Added the /travel page |
| 1.0.1 | 11/22/24 | Ryan Hatch | Edited the Design Document |

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

The Travlr Getaways app is built using the MEAN stack—MongoDB, Express.js, Angular, and Node.js. It powers a customer website for browsing and booking trips and an admin single-page application (SPA) for managing bookings. This setup makes for smooth performance, secure operations, and an easy-to-use experience.

**Design Constraints:**

* **Scalability:** The app must be able to handle thousands of users at the same time without any performance issues.
* **Security:** Sensitive data, such as user credentials and bookings, must be protected with encryption.
* **Compatibility:** The system needs to work smoothly on all major browsers and devices.
* **Timeliness:** Development must stay on schedule and work within the available resources.

*These constraints guide design choices, ensuring system efficiency and functionality while addressing potential performance bottlenecks and security vulnerabilities.*

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

### Component Diagram:

### Please see the hyperlinked Word Document "CS 465 Full Stack Component Diagram Text Version" for alternative text. Client

### Web Browser: Hosts the client-side interface.

### Client Session: Maintains session data for logged in users.

### Traveler Portfolio: Shows trip details and bookings.

### Graphic Library: Enables rich visual elements.

### Server

### Authentication Server: Manages user login and token validation.

### Server Session: Handles session data on the server side.

### Traveler Database: Stores user and trip details.

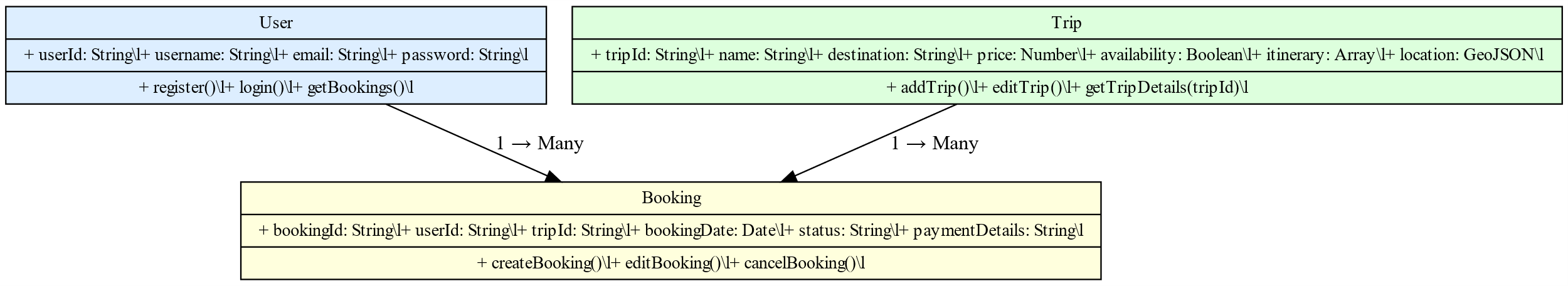
### Mongoose ODM: Uses schemas to simplify interactions with MongoDB.

### Database

### MongoDB: Serves as the main data storage system.

*Each component communicates through RESTful APIs. The Client retrieves data from the Server, and the Server uses Mongoose for database interactions.*

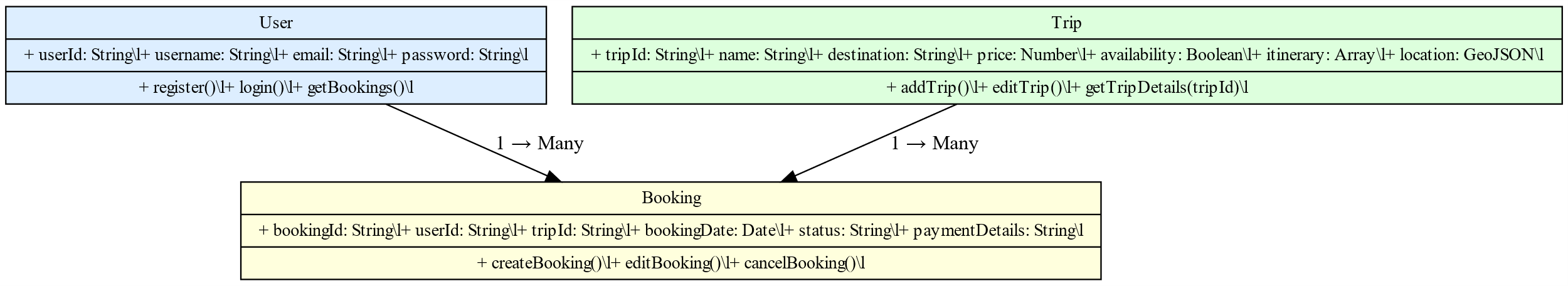
### Sequence Diagram:

****

**Logic Flow Description:**

1. **Sign-In Process:**
   * The client sends user credentials to the server through the Authentication Server.
   * The Authentication Server checks the credentials in the MongoDB database and provides an authentication token upon validation.
2. **Viewing Trips:**
   * The client sends a GET request to the server to fetch trip details.
   * The server uses Mongoose to retrieve trip data from MongoDB and returns it to the client.
3. **Admin Operations:**
   * Admins use the SPA to perform actions like adding or editing trips via POST/PUT requests.
   * The server validates these requests and updates the MongoDB database as needed.

## Class Diagram:



The **User** class represents app users with attributes like username, email, and password. The **Trip** class contains details of travel choices, such as the destination, description, price, and availability.   
The **Booking** class connects User and Trip data, with attributes such as bookingDate, status, and paymentDetails.

These classes are all interconnected, whereas a user can have multiple bookings, and each booking is tied to a specific trip. Class methods will all handle CRUD operations like creating users, retrieving trip details, and managing bookings, creating a smooth backend functionality.

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

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Purpose** | **URL** | **Notes** |
| **GET** | Retrieve all trips | /api/trips | Returns a list of all available trips. |
| **POST** | Add a new trip | /api/trips | Adds a new trip to the database; requires admin authentication. |
| **GET** | Retrieve a specific trip | /api/trips/:tripId | Returns details of a trip identified by tripId. |
| **PUT** | Update an existing trip | /api/trips/:tripId | Updates details of a specific trip; requires admin authentication. |
| **DELETE** | Delete a specific trip | /api/trips/:tripId | Removes a trip identified by tripId; requires admin authentication. |
| **POST** | Create a new booking | /api/bookings | Adds a new booking to the database. |
| **GET** | Retrieve all bookings by user | /api/bookings | Returns bookings for the authenticated user. |
| **GET** | Retrieve a specific booking | /api/bookings/:id | Returns details of a specific booking for the authenticated user. |
| **PUT** | Update an existing booking | /api/bookings/:id | Updates details of a specific booking; requires user or admin authentication. |
| **DELETE** | Delete a booking | /api/bookings/:id | Cancels a booking identified by id; requires user or admin authentication. |
| **POST** | Register a new user | /api/users | Adds a new user to the database; validates unique email and username. |
| **POST** | Authenticate user | /api/auth/login | Validates user credentials and issues an authentication token. |
| **POST** | Logout user | /api/auth/logout | Ends the user's session by invalidating the authentication token. |
| **GET** | Error handling example | /api/errors | Returns a structured error response for invalid or unauthorized requests. |

***Example of `/api/bookings` request and response payloads:***{

"id": "12345",

"name": "Beach Getaway",

"price": 500,

"availability": true

}

## The User Interface:

**A screenshot of a computer

Description automatically generated**

The Angular-based SPA offers richer and more enhanced functionality in comparison to today’s traditional web apps, using reusable components and dynamic data binding for better features like real-time updates and client-side validations. It also communicates with the backend through RESTful APIs, which helps to give the users the ability to view and book trips while also allowing admins the ability to manage the trips and bookings. Using tools like Postman and browser debugging, the testing consists of verifying **GET**, **POST**, and **PUT** requests to help keep and maintain proper data integrity and smooth functionality for the web app.