TICKETIFY

Table of Contents

1. Introduction

The **Event Ticket Booking App** is a comprehensive, full-stack web application designed to provide a seamless platform for booking tickets to various events. Built using the **MERN stack** (MongoDB, Express, React, Node.js), the app serves both event organizers and users by providing them with distinct functionalities.

Key Features:

User Experience: Users can easily browse a variety of events, including concerts, theater performances, workshops, festivals, and more. They can filter events by category, view event details, and book tickets based on availability. The platform also supports secure user registration and login via JWT (JSON Web Token) authentication, ensuring data security and privacy.

Event Publishing: Event organizers can publish, manage, and promote their events. They can provide comprehensive details about their events, including name, description, location, date, time, ticket prices, and more. Organizers can also edit or delete their events at any time.

Subscription and Notifications: Users can subscribe to receive event updates or newsletters to stay informed about upcoming events. This ensures that users never miss out on events they are interested in.

Scalability: With the use of MongoDB as the database, the app is highly scalable and flexible, allowing the addition of new event categories, user features, and other improvements as the platform grows.

Secure Authentication: The app employs JWT-based authentication to ensure secure login for both users and event organizers, providing an added layer of security to sensitive data.

The app leverages the power of the **MERN stack** to create a dynamic and efficient platform. The frontend, built with **React.js**, provides an interactive and responsive user interface, while the backend, powered by **Node.js** and **Express.js**, handles all API requests, user authentication, event management, and ticket processing. MongoDB, a NoSQL database, is used to store event data, user information, and ticket bookings in a highly flexible and scalable manner.

Overall, the **Event Ticket Booking App** aims to simplify the process of event ticketing, making it accessible to a wide range of users while offering powerful tools for event organizers to manage their events effectively.

2. Functional Requirements

2.1 System Requirements

Backend:

- **Node.js**
- **Express.js**
- **MongoDB**
- **JWT Authentication** for secure user login

Frontend:

- **React.js**
- **Redux** (for state management)
- **React Router** (for navigation)
- **TailWind-CSS** (for styling)

3. Design

3.1 System Architecture

The app follows a client-server architecture with the following components:

Frontend (React): The frontend is responsible for rendering the user interface, displaying event information, and interacting with the backend APIs. It handles user registration, login, browsing events, booking tickets, and managing the shopping cart.

Backend (Node.js + Express): The backend handles API requests, manages user authentication, stores event data, and processes ticket bookings. It also integrates with payment gateways to handle payments.

Database (MongoDB): MongoDB is used to store user information, event data, and booking details. It is a NoSQL database, which provides flexibility in handling different types of data, such as event descriptions, prices, and booking statuses.

3.2 Database Schema

USER SCHEMA

```
const UserSchema = new mongoose.Schema(
  {
    fullname: {
      type: String,
      required: true,
    },
```

```
    email: {
      type: String,
      required: true,
      unique: true,
    },
    password: {
      type: String,
      required: true,
    },
  },
  { timestamps: true }
);
```

EVENT SCHEMA

```
const EventSchema = new mongoose.Schema(
  {
    eventName: {
      type: String,
      required: true,
    },
    description: {
      type: String,
      required: true,
      trim: true,
    },
    imageURL: {
      type: String,
      required: true,
    },
    location: {
      type: String,
      default: "Online",
      trim: true,
    },
    price: {
      type: Number,
      required: true,
    },
    date: {
      type: String,
      required: true,
    },
    time: {
      type: String,
      required: true,
    },
    duration: {
      type: String,
      required: true,
    },
    eventType: {
      type: String,
      required: true,
    },
    eventCat: {
      type: String,
      required: true,
    },
```

```
  },
  { timestamps: true }
);
```

SUBSCRIBER SCHEMA

```
const SubSchema = new mongoose.Schema(
  {
    email: {
      type: String,
      required: true,
      unique: true,
    },
  },
  { timestamps: true }
);
```

3.3 API Endpoints

3.3.1 User Authentication Routes

POST /SignUp
Description: Registers a new user.
Body:

```
{
 "username": "user_name",
 "email": "user_email@example.com",
 "password": "user_password"
}
```

Response:

- 200 OK: User registered successfully.
- 400 Bad Request: Missing or invalid data.
- 500 Internal Server Error: Error in processing the request.

POST /LogIn
Description: Logs in a user.
Body:

```
{
  "email": "user_email@example.com",
  "password": "user_password"
}
```

Response:

- 200 OK: Successfully logged in, returns a token.
- 401 Unauthorized: Invalid credentials.

- 500 Internal Server Error: Error in processing the request.

3.3.2 Event Routes

POST /Event
Description: Creates a new event.
Body:

```
{
  "eventName": "Concert Name",
  "description": "Event Description",
  "imageURL": "http://image.url",
  "location": "Event Location",
  "price": 100,
  "date": "2024-12-25",
  "time": "19:00",
  "duration": "2 hours",
  "eventType": "Music",
  "eventCat": "Concert"
}
```

Response:

- 201 Created: Event created successfully.
- 400 Bad Request: Missing or invalid data.
- 500 Internal Server Error: Error in processing the request.

GET /Events
Description: Fetches all events.
Response:

- 200 OK: Returns a list of all events.
- 404 Not Found: No events found.
- 500 Internal Server Error: Error in processing the request.

3.3.3 Subscription Route

POST /letter
Description: Allows users to subscribe to event updates or newsletters.
Request Body:

```
{
"email": "user_email@example.com"
}
```

Response:

- 200 OK: Subscription successful.
- 400 Bad Request: Invalid email or missing data.
- 500 Internal Server Error: Error in processing the request.

4. Work flow

4.1 User Registration & Login

A user registers with an email and password, which is securely stored in the database after hashing. After registration, the user can log in using the same credentials.

4.2 Browsing Events

Users can browse events and filter them by category. Each event has a detail page where users can see the description, date, time, location, and ticket prices.

4.3 Event Publishing

Event organizers can log in and publish events with all necessary details (event name, description, date, time, etc.). Organizers can edit or delete their events.

5. Conclusion

The Event Ticket Booking App provides a seamless platform for users to browse and book tickets for various events. With the MERN stack, the app is highly scalable, and the use of JWT for authentication ensures secure user login. Event organizers have an easy-to-use interface to publish their events, making the app suitable for both users and event organizers.