22. Event Ticketing Platform

Creating an Event Ticketing platform using the MEAN stack (MongoDB, Express.js, Angular, Node.js) is a comprehensive project. Below is a high-level overview of how to build such a platform, along with some code snippets to guide you:

Project Setup and Structure

Set up a new project folder and structure for your Event Ticketing platform. Install the required Node.js packages and create a basic Angular application.

```
# Create a new Angular application

ng new event-ticketing-app
```

- Backend (Node.js & Express.js)

Create the backend of your Event Ticketing platform using Node.js and Express.js.

Installation of Packages

Install the necessary packages for Express.js, Mongoose (for MongoDB), and other dependencies.

```
npm install express mongoose cors
```

Setting up Express.js

Create your Express.js server, set up middleware, and handle routes.

- **javascript**

```javascript
// server.js

const express = require('express');
const mongoose = require('mongoose');
const cors = require('cors');

const app = express();

// Middleware
app.use(express.json());
app.use(cors());

// Database connection
mongoose.connect('mongodb://localhost/event-ticketing-app', {
  useNewUrlParser: true,
  useUnifiedTopology: true,
  useCreateIndex: true,
});

// Define Mongoose models for Event, Ticket, and User data
const Event = mongoose.model('Event', {
  name: String,
  date: Date,
  description: String,
  // Add more event-related fields as needed
});

const Ticket = mongoose.model('Ticket', {
  eventId: mongoose.Schema.Types.ObjectId,
  price: Number,
  // Add more ticket-related fields as needed
});

const User = mongoose.model('User', {
  username: String,
  password: String, // Use hashing for security
  // Add more user-related fields as needed
});
```

```
// Routes for managing events, tickets, and user authentication
app.post('/api/register', async (req, res) => {
  // Register a new user
  // Store hashed password in the database
});

app.post('/api/login', async (req, res) => {
  // Authenticate user and generate a JWT token
});

app.post('/api/events', async (req, res) => {
  // Create a new event
  // Save the event to the database
});

app.get('/api/events', async (req, res) => {
  // Retrieve a list of events
});

// Create similar routes for managing tickets, event details, and ticket purchases
```

- Frontend (Angular)

Create the frontend of your Event Ticketing platform using Angular. Design the user interface for searching, purchasing tickets, and user accounts.

Design and UI

Design the user interface for your Event Ticketing platform using Angular components, templates, and styles.

Event and Ticket Listings

Create components for users to browse and search for events and tickets.

Ticket Purchase

Design components for users to view event details and purchase tickets.

User Authentication

Implement user registration and login functionality.

- typescript

```
// event-listing.component.ts

import { Component } from '@angular/core';
import { EventService } from './event.service';


@Component({
  selector: 'app-event-listing',
  templateUrl: './event-listing.component.html',
})
export class EventListingComponent {
  events: any[];


  constructor(private eventService: EventService) {}


  ngOnInit() {
    this.eventService.getEvents().subscribe((data) => {
      this.events = data;
    });
  }
}
```

MongoDB

Create a MongoDB database to store event details, ticket data, user profiles, and purchase history.

Putting It All Together

Integrate the frontend and backend by making API requests from Angular components to Node.js routes. Ensure that you handle event creation, ticket management, user authentication, and ticket purchasing properly.

Building an Event Ticketing platform is a significant project with many opportunities for expansion, including payment gateway integration, event scheduling, event promotion, event organizers' dashboards, and more. It's a versatile platform that can be customized to your specific requirements.