18. Task Management App

Creating a basic Music Player using the MEAN stack (MongoDB, Express.js, Angular, Node.js) involves several components, including audio streaming and playlist management. Here's a high-level overview of how to build a simple music player, along with some code snippets to guide you:

Project Setup and Structure

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

```
# Create a new Angular application

ng new music-player-app
```

- Backend (Node.js & Express.js)

Create the backend of your Music Player 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 multer
```

Setting up Express.js

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

- **javascript**

```
// server.js

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

const app = express();

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

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

// Define Mongoose models for Playlist and Track data
const Playlist = mongoose.model('Playlist', {
  name: String,
  tracks: [{ type: mongoose.Schema.Types.ObjectId, ref: 'Track' }],
});

const Track = mongoose.model('Track', {
  title: String,
  artist: String,
  audioFile: String,
  duration: Number,
  // Add more fields as needed
});

// Multer storage for uploading audio files
const storage = multer.diskStorage({
  destination: (req, file, cb) => {
    cb(null, 'uploads/');
  },
  filename: (req, file, cb) => {
    cb(null, Date.now() + '-' + file.originalname);
  },
});

const upload = multer({ storage });

// Routes for managing playlists, tracks, and audio streaming
app.post('/api/playlists', async (req, res) => {
  // Create a new playlist
  // Save the playlist to the database
});

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

app.get('/api/playlists/:id', async (req, res) => {
  // Retrieve a specific playlist and its tracks
```

```
});

app.post('/api/tracks', upload.single('audioFile'), async (req, res) => {
  // Upload and save a new audio track to the database
});

app.get('/api/tracks/:id', (req, res) => {
  // Stream audio file to the client
  // You'll need to send audio chunks as a response
});
```

- ## Frontend (Angular)

Create the frontend of your Music Player using Angular. Design the user interface for managing playlists, tracks, and audio controls.

Design and UI

Design the user interface for your Music Player using Angular components, templates, and styles.

Playlist Management

Create components and forms for users to create playlists, add tracks, and manage playlists.

Audio Controls

Design components for playing audio tracks, including controls like play, pause, skip, and volume.

Audio Streaming

Implement audio streaming from the backend to play audio tracks in the player.

- **typescript**

```
// audio-player.component.ts
import { Component, Input } from '@angular/core';

@Component({
  selector: 'app-audio-player',
  templateUrl: './audio-player.component.html',
})
export class AudioPlayerComponent {
  @Input() audioUrl: string;
  audio: HTMLAudioElement = new Audio();

  play() {
    this.audio.src = this.audioUrl;
    this.audio.play();
  }

  pause() {
    this.audio.pause();
  }

  stop() {
    this.audio.pause();
    this.audio.currentTime = 0;
  }
}
```

MongoDB

Create a MongoDB database to store playlists and track data.

Audio Streaming

To stream audio from the backend, you'll need to implement a route that sends audio chunks to the client. This will involve using the Express.js res.write and res.end methods to stream audio data.

Putting It All Together

Integrate the frontend and backend by making API requests from Angular components to Node.js routes. Implement playlist management, track uploading, and audio streaming properly.

Building a Music Player is an exciting project, and you can expand it with additional features like user accounts, album artwork, song lyrics, and advanced audio controls for a more complete music player experience.