Mood-based song recommender

Mini Project: Mood-Based Song Recommendation System
1) Introduction

This project is about making a web app that can look at your face through the webcam, understand your mood, and then suggest songs that fit your feelings.

👉 Example: If you look happy, it will suggest cheerful songs. If you look sad, it will suggest slow or emotional songs.

We use a library called face-api.js (runs in browser) to detect moods, and a MongoDB database to store songs with labels like happy, sad, neutral, angry.

2) Objective

Detect user's mood in real time.

Connect the detected mood to a list of songs in the database.

Show matching songs to the user on the website.

Make it easy to add new moods and songs in the future.

👉 In short: Face → Mood → Songs.

3) Technologies Used
Frontend (what user sees)

React.js → Makes the website interface.

Vite → Runs the project fast during development.

face-api.js → Detects emotions from your face.

TensorFlow.js → Supports face-api.js in the browser.

Axios → Sends/receives data from backend.

CSS3 → Designs and styles the web page.

Backend (hidden server)

Node.js + Express.js → Backend server that handles requests.

MongoDB + Mongoose → Database to save songs with their moods.

Multer → For uploading songs (if admin adds songs).

dotenv → To keep secret keys safe.

CORS → Lets frontend and backend talk safely.

4) System Workflow (How it works step by step)

User clicks "Detect Mood".

Webcam opens → face-api.js checks the user's face.

The library finds the main mood (happy/sad/neutral/angry/surprised).

That mood is sent to the backend API.

Backend looks into MongoDB and finds songs with that mood tag.

Songs are sent back to the frontend.

User sees the Recommended Songs list on the screen.

👉 Think of it like a chain:
Face → Mood → Backend → Database → Songs → User.

5) Applications (Where it can be used)

Music apps → Create automatic mood playlists.

Entertainment systems → Suggest movies/music depending on emotions.

Therapy/wellness → Play calm music if someone is stressed.

Advertising → Show ads or content based on facial mood.

6) Advantages (Why it’s useful)

Works directly in the browser → No need for heavy backend servers.

Real-time → Instant results (few seconds).

Flexible → Easy to add more moods and songs.

Lightweight → Uses small models (TinyFaceDetector).

7) Future Enhancements (How it can be made better)

Detect mood automatically without clicking the button.

Improve accuracy with better ML models.

Connect with Spotify/YouTube to suggest real playlists.

Let users sign in and save personal playlists.

Learn from user feedback (like/dislike songs).

8) System Architecture (Diagram in words)
User → Webcam → face-api.js (Detect Mood)
→ Backend (Express.js) → MongoDB (Songs with moods)
→ Send songs back → Show on frontend

9) Database Design (MongoDB example)

Each song is stored with details:

```
{
  "title": "On Top of the World",
  "artist": "Imagine Dragons",
  "mood": "happy",
  "url": "https://link-to-song.mp3",
  "coverImage": "https://link-to-image.jpg"
}
```

👉 So if mood = "happy", the backend fetches all songs where "mood": "happy".

10) APIs (Backend Routes)

POST /api/recommendations

Input: { "mood": "happy" }

Output: list of songs for happy mood.

POST /api/songs (optional)

For admin to upload new songs with mood.

11) Testing (Checking if it works)

Test if the mood detection works in good light and poor light.

Test if songs match the mood correctly.

Test if no face is detected → app shows message (“No face found”).

Test speed (should respond within 1–2 seconds).

12) Privacy & Safety

Webcam is only used in browser, not uploaded anywhere.

Only the mood label (like "happy") is sent to backend, not the face image.

Sensitive keys (DB URL) are stored in .env, not shared in code.

13) Conclusion

This mini-project shows how AI + Web Development can be combined to make real-world apps. It is lightweight, works in the browser, and gives a personalized experience. With improvements like Spotify integration, it can become a full product.

Awesome mini-project! Here's a clean, detailed write-up you can drop into your report or README. I kept the language simple, added concrete steps, sample code, and test checklists so it's easy to build and evaluate.

Mini Project: Mood-Based Song Recommendation System

1) Introduction

This web app watches your face via the webcam, estimates your current mood in the browser using **face-api.js**, and recommends songs that match that mood from a **MongoDB** database.

Example: look happy → get upbeat songs. Look sad → get mellow songs.
Everything runs client-side for emotion detection; only the **mood label** goes to the server.

2) Objective

- Detect the user's mood in (near) real time from the webcam.
- Map the detected mood to mood-tagged songs in the database.
- Display matching songs immediately in the UI.
- Make it simple to add new moods and new songs later.

In short: Face → Mood → Songs.

3) Technologies Used

Frontend

- **React + Vite** — fast dev server and modern React tooling.
- **face-api.js** (on top of **TensorFlow.js**) — face detection + emotion classification in the browser.
- **Axios** — call backend APIs.
- **CSS3** — styling (or Tailwind if you prefer).

Backend

- **Node.js + Express** — REST API.
- **MongoDB + Mongoose** — song storage with mood labels.
- **Multer** (optional) — admin upload for songs / cover images.
- **dotenv** — load environment variables.
- **CORS** — allow frontend ↔ backend securely.

4) System Workflow (Step by Step)

1. User clicks **Detect Mood**.
2. App asks for **camera permission** and starts the webcam stream.
3. **face-api.js** loads lightweight models and infers the dominant emotion (e.g., happy/sad/neutral/angry/surprised).
4. Frontend sends `{ mood: "happy" }` to the backend.
5. Backend queries MongoDB for songs where `mood === "happy"`.
6. Backend returns a list of song objects.
7. Frontend renders the **Recommended Songs** list.

Chain: Webcam → face-api.js → Mood → API → MongoDB → Songs → UI.

5) Applications

- **Music apps**: auto mood playlists.
- **In-car/TV entertainment**: adapt content to mood.
- **Wellness/therapy**: calm playlists when stressed.
- **Advertising**: mood-aware content (with user consent).

6) Advantages

- **Browser-first**: processing stays on the device; low server cost.
- **Real-time**: feedback in seconds.
- **Flexible**: new moods/songs are easy to add.

- **Lightweight**: works with TinyFaceDetector models.

7) Future Enhancements

- Auto-detect mood continuously (no button).
- Better accuracy with improved/finetuned models.
- Integrations: **Spotify/YouTube** for live playlists.
- Sign-in: save favorites, history.
- Feedback loop: thumbs up/down to learn user taste.
- Multi-mood blends (e.g., 60% happy, 40% surprised).

8) Architecture (Diagram in Words)

```bash
User → Webcam
    → face-api.js (in-browser emotion)
    → POST /api/recommendations { mood }
    → Express.js → MongoDB (songs)
    → Response: [ {title, artist, mood, url, coverImage}, ... ]
    → React renders Recommended Songs
```

9) Database Design (MongoDB)

Collection: `songs`

```json
{
  "_id": "ObjectId",
  "title": "On Top of the World",
  "artist": "Imagine Dragons",
  "mood": "happy",
  "url": "https://link-to-song.mp3",
  "coverImage": "https://link-to-image.jpg",
  "createdAt": "2025-08-20T00:00:00.000Z"
}
```

Mongoose schema (example):

```js
```

```
// models/Song.js
import mongoose from "mongoose";

const songSchema = new mongoose.Schema({
  title: { type: String, required: true, trim: true },
  artist: { type: String, required: true, trim: true },
  mood: { type: String, required: true, enum: ["happy", "sad", "neutral", "angry", "surprised"]
},
  url: { type: String, required: true },
  coverImage: { type: String }
}, { timestamps: true });

export default mongoose.model("Song", songSchema);
```

10) APIs (Backend Routes)

`POST /api/recommendations`

- **Request body:** `{ "mood": "happy" }`
- **Response:** `[{ title, artist, mood, url, coverImage }, ...]`

```
js

// routes/recommendations.js
import express from "express";
import Song from "../models/Song.js";
const router = express.Router();

router.post("/", async (req, res) => {
  try {
    const { mood } = req.body;
    if (!mood) return res.status(400).json({ error: "mood is required" });
    const songs = await Song.find({ mood }).limit(25).lean();
    res.json(songs);
  } catch (e) {
    res.status(500).json({ error: "Server error" });
  }
});

export default router;
```

`POST /api/songs` (optional admin)

- **Request body:** `{ title, artist, mood, url, coverImage }`
- Validates `mood` is one of the allowed values; saves a new song.

```
js

// routes/songs.js
import express from "express";
import Song from "../models/Song.js";
const router = express.Router();

router.post("/", async (req, res) => {
  try {
    const song = await Song.create(req.body);
    res.status(201).json(song);
  } catch (e) {
    res.status(400).json({ error: e.message });
  }
});

export default router;
```

11) Frontend Implementation (React)

Install models and load once:

```
bash

npm i face-api.js axios
```

Load models & detect mood:

```
jsx

// components/MoodDetector.jsx
import { useEffect, useRef, useState } from "react";
import * as faceapi from "face-api.js";
import axios from "axios";

export default function MoodDetector({ onSongs }) {
  const videoRef = useRef(null);
  const [ready, setReady] = useState(false);
  const [mood, setMood] = useState(null);
```

```
useEffect(() => {
 (async () => {
   const MODEL_URL = "/models"; // publi_
```