SRS

Intelligent Music Mood Generator

Team 2

Team Members: Julia Kim, Gio Ong, Jenny Nguyen, Zijie Mu

October 2, 2025

Problem Statement

Users want playlists that match their current mood using the music they already like. The problem is that current music apps either give the user generic recommendations that miss the mark, or end up spending forever manually building the perfect playlist themselves. What we need is something that can understand when the user types "feeling lazy Sunday morning vibes" and turn that into an actual playlist, one that flows well and gets created right in their Spotify account.

Objective / Goal

We're building a system that lets the user describe their mood and automatically generates a playlist from their own library. It'll match the energy, tempo, and overall feel they're going for, make sure the songs flow nicely together, and get smarter over time based on what they skip or like all from their Spotify account.

One Use Case (Primary)

User: Authenticated Spotify user

Use Cases:

1. User types "cozy rainy focus, mid-tempo, no vocals, low brightness."
2. The app figures out what the user types in and finds music that matches the tempo range "mid-tempo" is, how bright or dark the songs should sound, etc.
3. The system grabs all the user's saved songs and their audio features from Spotify, then filters down to songs that could work
4. It finds similar tracks that vibe together and arranges them so the playlist flows smoothly without feeling repetitive
5. A brand new playlist gets created right in the user's Spotify account with the best tracks
6. As the user listens, the app will track what they skip to get a general sense of what the user likes

Scope of the Project

- Mood detection through text input and manual mood selection

- Integration with music streaming APIs (Spotify)
- Playlist generation based on mood classification
- User profile management and preference storage
- Basic machine learning model for mood-to-music mapping
- Feedback mechanism for continuous improvement
- Cross-platform web application
- Mood transition feature ("Energetic" to "Calm" to “Pump”)
- Music metadata analysis (tempo, energy)
- User history and saved playlists

Tech Stack

Frontend:

- **Framework:** React.js with TypeScript
- **UI Library:** Material-UI or Tailwind CSS
- **State Management:** Redux or Context API
- **Data Visualization:** Chart.js or Recharts (for mood analytics)

Backend:

- **Framework:** Node.js with Express.js
- **API:** RESTful API architecture
- **Authentication:** JWT (JSON Web Tokens) or OAuth 2.0

Database:

- **Primary Database:** MongoDB (for user profiles, preferences, and playlist data)
- **Caching:** Redis (for session management and frequently accessed data)

Machine Learning:

- **Language:** Python
- **Libraries:** scikit-learn, TensorFlow/PyTorch (for mood classification)
- **Model:** Classification model trained on mood-labeled music datasets

External APIs:

- **Music Services:** Spotify Web API

Development Tools:

- **Version Control:** Git with GitHub/GitLab
- **Project Management:** Jira with Confluence
- **CI/CD:** GitHub Actions or Jenkins

- **Testing:** Jest (frontend), Pytest (backend)
- **Containerization:** Docker

Hosting/Deployment:

- **Cloud Platform:** AWS, Google Cloud Platform, or Heroku
- **Web Server:** Nginx or Apache

External Interface Requirements

User Interfaces

- **Landing Page:** Introduction to MoodTune with mood selection preview
- **Dashboard:** Main interface showing current mood, generated playlist, and playback controls
- **Mood Selection Interface:** Visual mood picker with 8-12 mood options
- **Playlist View:** List/grid view of recommended songs with album art
- **Profile Page:** User preferences, listening history, and saved playlists
- **Settings Page:** Account settings, privacy controls, and API connections

Hardware Interfaces

- No specific hardware interfaces required
- Standard web browser on desktop/mobile devices
- Audio output through device speakers/headphones

Software Interfaces

- **Spotify Web API:** For music search, metadata retrieval, and playback control
  - Authentication: OAuth 2.0
  - Data Format: JSON
- **Database Interface:** MongoDB connection for CRUD operations
- **Machine Learning Model Interface:** REST API endpoint for mood prediction

Functional Requirements

- User login with Spotify (authentication and token)
- Import user library and fetch track audio features
- Parse mood text into constraints (energy, tempo, etc.)
- Build playlist with similarity search
- Create playlist in user’s Spotify and return playlist link
- Collect feedback (likes, skips) per session
- Store playlist history and allow quick regeneration

Non-Functional Requirements

- Playlists should be generated within a 5 second window
- Simple, clean, and intuitive user interface

Estimated Timeline

1. Planning and Design (2 weeks)

- **Week 1-2:**
  - Set up development environment and tools (Jira, Confluence, Git)
  - Finalize requirements and SRS documentation
  - Create system architecture and design documents
  - Design database schema
  - Create wireframes and UI mockups

2. Development - Core Features (2 weeks)

- **Week 3-4:**
  - Set up frontend framework and basic UI components
  - Implement user authentication and authorization
  - Set up backend API structure
  - Database integration and basic CRUD operations
  - Develop mood selection interface
  - Implement external music API integration (Spotify)
  - Create basic playlist generation algorithm
  - Develop music playback interface

3. Development (2 weeks)

- **Week 5-6:**
  - Implement machine learning model for mood classification
  - Develop feedback collection system
  - Create user profile and preference management
  - Implement playlist saving and history features
- **Week 7:**
  - Develop mood transition feature
  - Implement analytics and visualization
  - Add recommendation refinement based on feedback

4. Testing and Quality Assurance (2 weeks)

- **Week 8:**
  - Integration testing
  - API testing
  - Security testing
- **Week 9:**
  - User acceptance testing
  - Performance and load testing
  - Bug fixes and optimization

5. Deployment and Documentation (1 week)

- **Week 10:**
  - Deploy to production environment
  - Final documentation (user manual, API docs)
  - Create demo video and presentation
  - Project final submission