

Computer Science: Mood-Based Playlist Recommender Documentation

Mauricio Cubillos, Roseline Damiscar, Paola Montejo, Zara Moreno, Angie Salazar
FIU Knight Foundation School of Computing and Information Sciences
CIS4951 U01
Dr. Masoud Sadjadi

1. Executive Summary

Moodify is a full-stack web application designed to help university students quickly discover music that matches their mood. The solution leverages Natural Language Processing (NLP) techniques to recommend relevant Spotify playlists based on either direct mood selection or a user-written description. The system includes a ReactJS frontend and a Node.js/Express backend with MongoDB for data persistence. Its core feature is a pattern-matching algorithm that classifies input into one of five moods: Chill, Focus, Energetic, Happy, Sad.

Key engineering accomplishments include seamless Spotify API integration with automated token refreshing and successful validation of the NLP logic using more than 50 realistic test descriptions, resulting in a smooth, responsive user experience.

2. Problem & Requirements (O1)

Context & Stakeholders:

University students often struggle to create playlists that match their desired mental state, whether for studying, relaxing, or motivating themselves. Moodify automates this process to save time and improve the listening experience.

Functional Requirements:

- The system must allow users to select from predefined moods (chill, focus, energetic, happy, sad).
- It must accept user-entered, free-text mood descriptions.
- It must assign a mood using a pattern-matching NLP algorithm.
- It must perform authenticated Spotify API calls to retrieve relevant playlists and tracks.
- It must display up to 5 recommended playlists and up to 10 tracks per playlist.
- It must support GET and POST requests to store playlist data in MongoDB.

Non-Functional Requirements:

- Backend must automatically refresh expired Spotify tokens.
- Implement strong error handling, especially for backend API calls.
- Provide clear visual feedback and responsive UI behavior.

Constraints:

- Third-Party API Limits: The system depends on Spotify's API rate limits, data availability, and authentication model (Client Credentials Flow).
- No User Authentication: Without user login, personalization is limited to public data only.
- Rule-Based NLP: The mood detection pipeline uses pattern matching rather than ML models, imposing limitations on accuracy and nuance.

- **Deployment Simplicity:** Architecture assumes a single backend instance with in-memory token caching, restricting scalability.
- **Dataset Access:** The app relies solely on Spotify's public playlist corpus; no local or proprietary dataset is used.

Success Criteria:

- **Accuracy:** Pattern-matching mood classification correctly identifies the intended mood for at least 80% of the tested user descriptions.
- **Reliability:** Spotify token refreshes occur seamlessly, with no more than 1% request failures due to authentication issues.
- **Performance:** Playlist and track data should load within 1–3 seconds under typical network conditions.
- **User Experience:** Users should be able to obtain mood-based playlist recommendations in two or fewer interactions.
- **Data Integrity:** All successfully fetched playlists can be stored and retrieved from MongoDB without failure.

Risks:

- **API Dependency:** If Spotify modifies its API or rate limits, functionality could break.
- **NLP Misclassification:** The simple pattern-matching algorithm may misinterpret nuanced or slang-heavy descriptions.
- **Token Expiration:** In-memory token caching may fail under high concurrency or server restarts.
- **Data Persistence:** MongoDB connection issues could prevent playlist saving or retrieval.
- **User Expectation:** Without personalization or advanced NLP, recommendations may not meet user expectations for precision or relevance.

3. System Overview & Architecture (O2)

Architecture:

1. **Frontend:** ReactJS interface handling mood input and user interaction.
2. **Backend:** Express REST API managing business logic, NLP processing, token handling, and routing.
3. **Services/Data:** Spotify API and MongoDB.

Component	Type	Technology	Description	Interactions
Moodify Client	Web Application (Frontend)	ReactJS	Displays the UI, handles user input (button clicks/text), and performs Pattern Matching NLP.	Moodify API (REST over HTTP)
Moodify API	Web Server (Backend)	Node.js + ExpressJS	Hosts the REST API, manages Spotify Authentication, handles core logic, and proxies requests to external services.	MongoDB (Read/Write) Spotify API (Authenticated Calls)
MongoDB	Database (Persistence)	Mongoose	Stores saved playlist objects and related metadata.	Moodify API (Reads/Writes)
Spotify API	External System	Axios, OAuth 2.0	Provides public search results for playlists and tracks.	Moodify API (Authenticated HTTP Requests)

Key Technologies:

<u>Layer</u>	<u>Component</u>	<u>Technology</u>
Frontend	UI & logic	ReactJS, HTML/CSS, JavaScript
Frontend	Routing	React Router DOM
Backend	API server	Express.js (Node.js)

Backend HTTP client Axios

Backend Database MongoDB with Mongoose

Tooling Environment dotenv, nodemon, cors

API Endpoints:

- GET /api/spotify/token — Retrieves a fresh Spotify access token
- GET /api/spotify/playlists?mood= — Returns 5 playlists matching the provided mood
- GET /api/spotify/playlists/:id/tracks — Returns up to 10 tracks from a selected playlist

Data and Storage Overview:

- The playlist model schema includes fields for mood, playlist name, Spotify playlist ID, and an array of tracks (each with name, artist, and URL).
- Authentication data, including the Spotify access token and expiration time, is stored in memory on the backend to provide lightweight caching and reduce unnecessary authentication requests.

4. Discipline Specific Depth (O6)

Algorithms & Data Structures:

- The mood detection uses a deterministic Pattern Matching approach on the frontend (MoodSelectionPage.js).
- It uses a moodDictionary containing synonyms and mood-related phrases for five moods (Chill, Focus, Energetic, Happy, Sad). The algorithm iterates over the user's input, counts keyword matches for each mood, and selects the mood with the highest score.
- This method is simple and performant ($O(M * W)$, where M is moods and W is total synonym words), but the developers noted its lack of nuance compared to proper sentiment analysis, making it a viable but limited solution for complex language inputs.

Architecture & Patterns:

- The backend implements a Spotify Client Credentials Flow.
- The utility getSpotifyToken utilizes an in-memory cache (spotifyToken) to store the access token and its expiration time. This prevents redundant API calls to Spotify for the token, a key performance optimization. The token is retrieved only if it's null or expired.
- The backend fetches 20 playlists from Spotify and then randomly shuffles and selects 5 to return to the frontend. This ensures a degree of variety for each request, enhancing the user discovery experience.

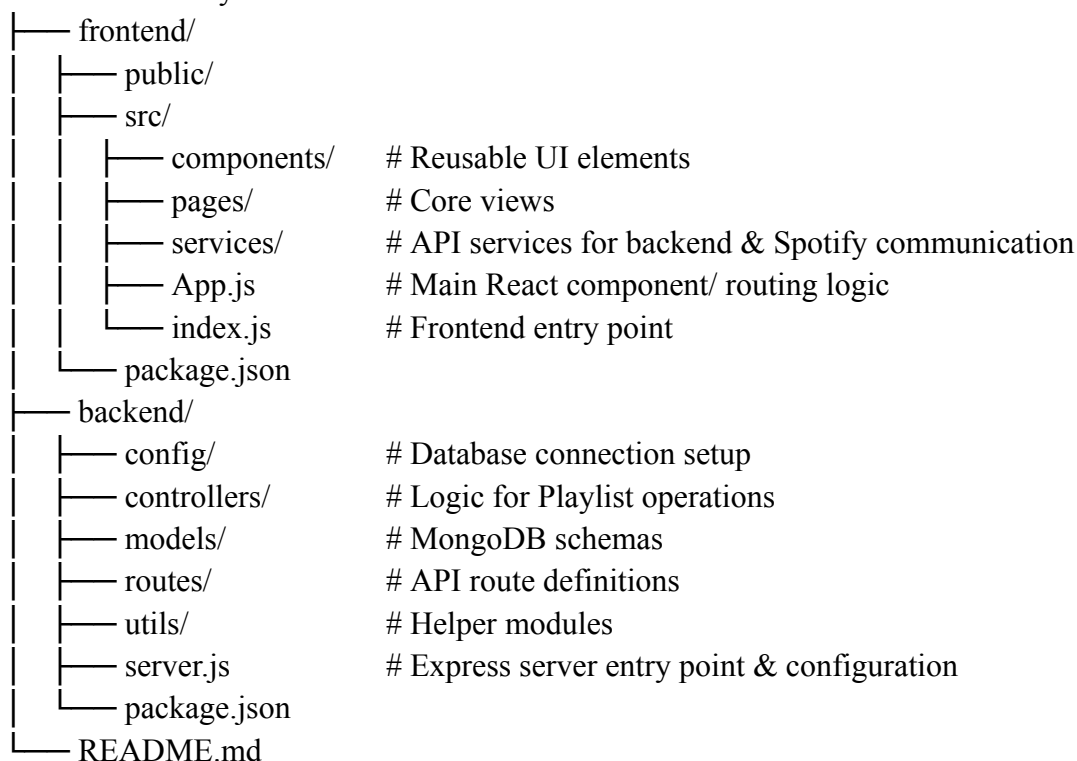
Engineering Evidence:

- More than 50 realistic mood descriptions were used to evaluate the accuracy of the pattern-matching algorithm.
- Console-based logging in the backend confirmed successful token retrieval, playlist searches, and MongoDB interactions.
- Error handling was validated through controlled failure tests, such as inducing incorrect tokens and simulating network interruptions.
- The system demonstrated stable performance under typical load and produced consistent results that were repeatable across runs.

5. Implementation Notes (O2)

Repository Structure:

Mood-Based-Playlist-Recommender/



Coding Conventions, Notable Patterns, and Tradeoffs:

- The entire project is written using ES6+ syntax, employing JavaScript modules (“type”: “module” in package.json) and async/await for asynchronous control flow (e.g. playlistController.js, spotifyAuth.js).
- Frontend API calls are abstracted into service modules (api.js, spotify.js), isolating the data access layer for a cleaner architecture.

- The use of React Hooks (useState, useNavigate) in page components (MoodSelectionPage.js, PlaylistPage.js) reflects modern React functional programming patterns.

6. Testing & Evaluation (O3)

Strategy:

- Focused heavily on the NLP/Pattern Matching logic to ensure specific keywords correctly triggered the associated mood categories. Frontend component rendering was verified using the standard React testing library setup (App.test.js, setupTests.js).
- Validated the communication between the React frontend and the Express backend, ensuring that mood inputs correctly traveled to the API and returned formatted JSON responses.
- Manual testing covered the full user journeys (Journey 1 & 2), verifying that a user click or text entry resulted in a playable Spotify link displayed on the interface.

Coverage:

- The pattern matching algorithm was rigorously tested against a dataset of over 50 realistic testing descriptions to ensure it could handle diverse user inputs beyond simple keywords.
- Validation covered both explicit inputs (button clicks) and implicit inputs (text descriptions) across all five target moods (Chill, Focus, Energetic, Happy, Sad).

Performance & Reliability Testing:

- Testing specifically targeted Spotify API interactions, identifying them as the primary point of failure. Error handling wrappers were implemented on the backend to manage token expiration and external API outages gracefully (returning 503 status codes instead of crashing).
- The backend's response time was optimized by implementing an in-memory caching mechanism for the Spotify authorization token. This ensures that the application does not waste time re-authenticating with Spotify for every single user request, significantly reducing latency.

KPIs/Metrics:

- The primary metric was the success rate of the algorithm in correctly categorizing the 50+ test descriptions into the expected mood buckets.
- Successful retrieval of a valid authorization token and subsequent playlist data (monitored via console logs like "Spotify token retrieved successfully").

Defect Summary:

- Initial testing with “Sentiment Polarity Score” revealed a defect where the model lacked nuance (understanding only positive/negative) and failed to scale to new vocabulary.
 - The system was refactored to use the Pattern Matching algorithm, which proved more flexible and scalable for the project’s scope.
- Early tests showed the application failing after one hour due to Spotify's token lifecycle.
 - An auto-refresh logic was implemented in spotifyAuth.js to check tokenExpirationTime before every request.

7. Security, Privacy, Accessibility & Compliance (O5)

Security:

- Credentials stored in a backend-only .env file; never committed to version control.
- Frontend communicates only with the Express backend, preventing exposure of API keys or database URLs.
- Spotify token handled server-side using Client Credentials Flow with automatic refresh logic.

Privacy:

- No user accounts or PII collected; all inputs processed transiently.
- Database stores only generic playlist info (mood, name, spotifyId).
- Future personalization will require OAuth 2.0 and a formal privacy policy.

Ethical, Legal, and Societal Impacts:

- Supports student well-being by reducing effort in finding mood-aligned music.
- Complies with Spotify API requirements (authentication, rate limits).
- Mood detection uses a simple, transparent rule-based algorithm.

Accessibility:

- High-contrast mood buttons offer an accessible alternative to text input.
- Clear visual feedback helps users understand system status and errors.

8. Deployment & Operations

Backend Deployment:

- Node.js/Express application
- Navigate to the backend directory using `cd ~/Mood-Based-Playlist-Recommender/backend`
- Start the server with `node server.js`
- The server runs on a configurable port (default 5001) and connects to MongoDB via Mongoose for data persistence.

Frontend Deployment:

- React Single Page Application
- Navigate to the frontend directory using `cd ~/Desktop/Mood-Based-Playlist-Recommender/frontend`
- Start the development server with `npm start`
- By default, the frontend runs on port 3000; for production, it can be built into static assets using `npm run build` and served via a web server or CDN

Configuration, Operations, and Monitoring:

- All sensitive credentials, including `MONGODB_URI` and Spotify API keys (`SPOTIFY_CLIENT_ID`, `SPOTIFY_CLIENT_SECRET`), are stored in backend-only `.env` files and injected into the server environment at runtime.
- The system uses console logs to monitor server startup, database connections, and Spotify authentication events. Logs provide feedback on operational health and error states.

Supporting Documentation:

- Full installation, setup, and maintenance instructions are available in Appendix A: Installation Guide and Appendix C: Admin/Operations Guide.

9. Limitations & Future Work

Known Limitations:

- Without user login, all recommendations are based solely on the selected mood and are identical for all users; no personalization or exclusion of previously heard tracks.
- The keyword pattern-matching NLP struggles with complex sentences, negation (e.g. “I am not happy”), and nuanced emotional descriptions outside the hardcoded dictionary.
- The app is a discovery tool only; users must navigate to Spotify to play music.
- Playlist discovery is ephemeral; session history is not stored, so users cannot revisit previous searches without starting over.

Shortcuts & Technical Debt:

- Using Client Credentials Flow limits the app to public Spotify data and prevents features requiring user consent (private libraries, playlist modification).
- Spotify access tokens are stored in a global variable; this design is not stateless and limits horizontal scaling.
- Mood dictionary is hardcoded on the frontend; expanding vocabulary requires a code deployment instead of a dynamic update.

Prioritized Roadmap:

- Enable Spotify user login to unlock personalization features.

- Replace pattern-matching with a lightweight text classification model (e.g. BERT) for more accurate mood detection.
- Use the user's "Liked Songs" and listening history to filter and rank playlist suggestions.
- Allow users to save or edit playlists directly within the app.

10. References

Axios. <https://axios-http.com/>

ExpressJS. <https://expressjs.com/>

MongoDB. <https://www.mongodb.com/>

Mongoose. <https://mongoosejs.com/>

Node.js. <https://nodejs.org/>

ReactJS. <https://reactjs.org/>

Spotify Web API. <https://developer.spotify.com/documentation/web-api/>

Appendices

A. Installation Guide

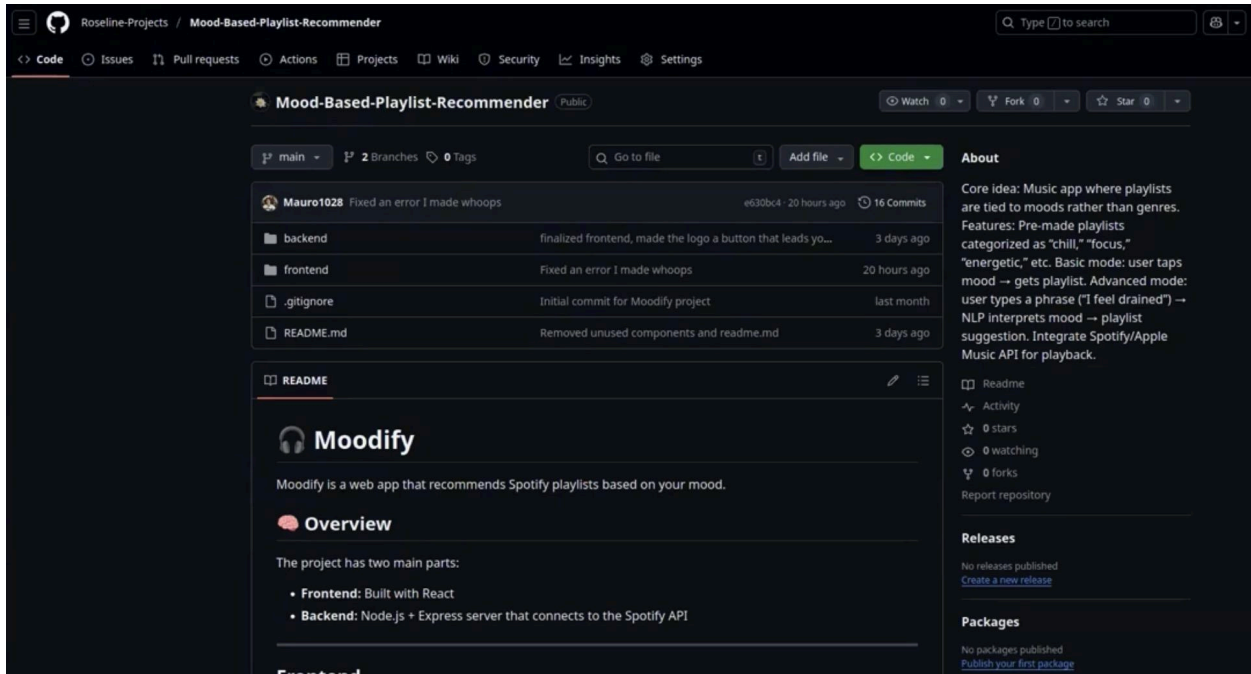
Introduction:

Moodify is a web-based application that helps university students quickly discover Spotify playlists that match their current mood. The system uses pattern-matching NLP to classify either a selected mood or a written mood description into one of five categories: Chill, Focus, Energetic, Happy, and Sad.

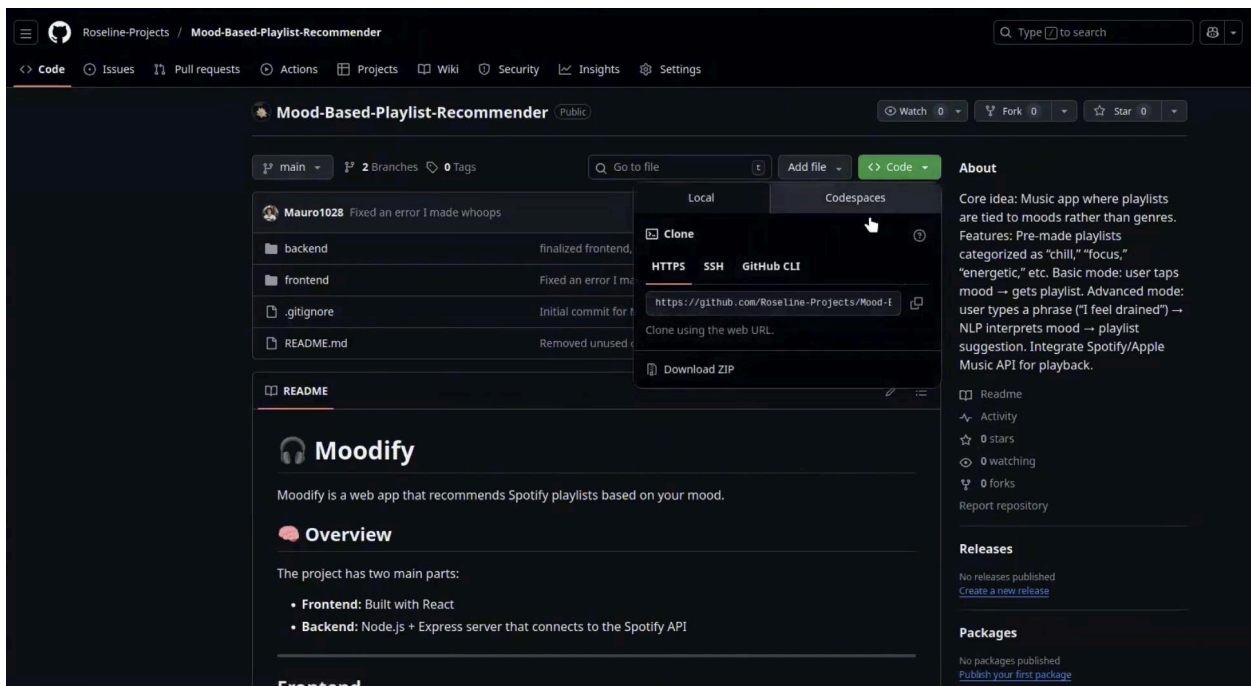
Based on this classification, Moodify retrieves up to 5 recommended playlists and 10 preview tracks per playlist from the Spotify Web API. This manual explains how to install Moodify on a new device.

Follow the steps with screenshots below:

1. Go into our GitHub page as shown below

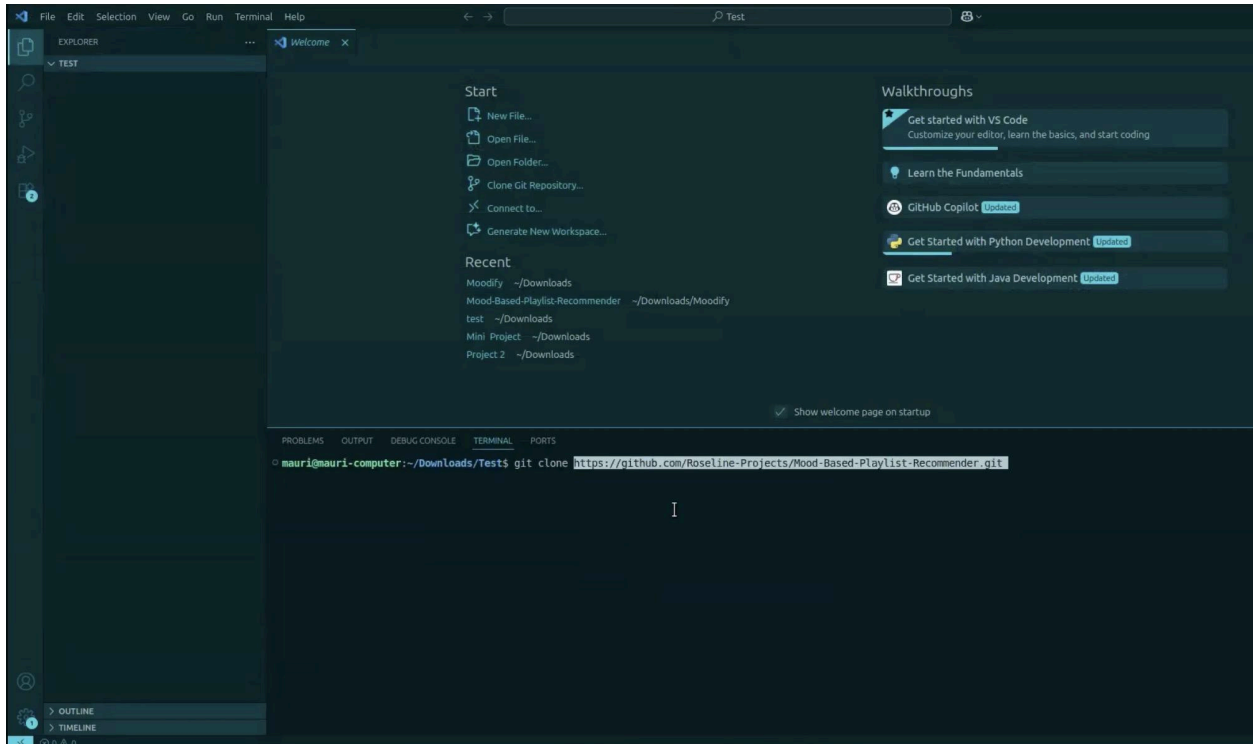


2. Copy GitHub url

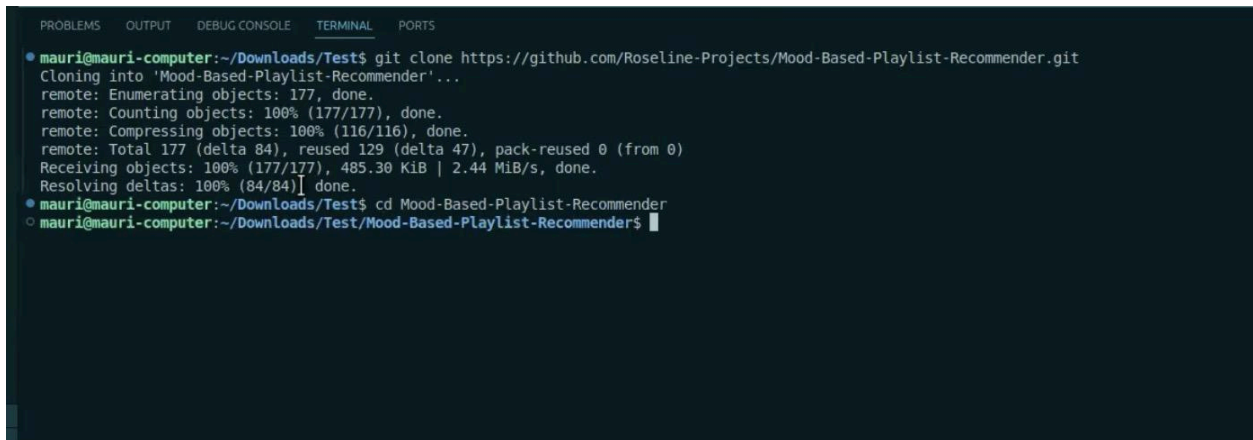


- Then go into your IDE of choice and set it in the location you would like the repository to be stored.

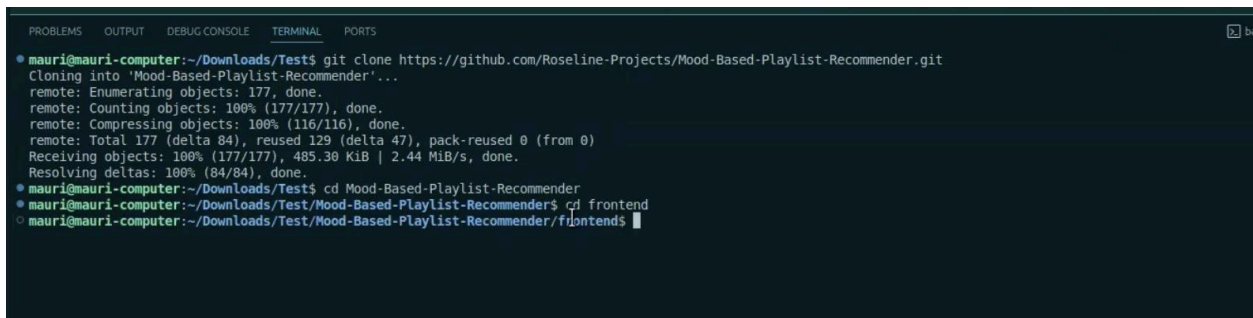
Below we use VSCode and typed the following command: `git clone github-url`



4. Once cloned you will then type `cd Mood-Based-Playlist-Recommender` as shown below



5. Then get the directory into frontend using `cd frontend` to install dependencies



6. Install dependencies as shown below: npm install

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
mauri@mauri-computer:~/Downloads/Test$ git clone https://github.com/Roseline-Projects/Mood-Based-Playlist-Recommender.git
Cloning into 'Mood-Based-Playlist-Recommender'...
remote: Enumerating objects: 177, done.
remote: Counting objects: 100% (177/177), done.
remote: Compressing objects: 100% (116/116), done.
remote: Total 177 (delta 84), reused 129 (delta 47), pack-reused 0 (from 0)
Receiving objects: 100% (177/177), 485.30 KiB | 2.44 MiB/s, done.
Resolving deltas: 100% (84/84), done.
mauri@mauri-computer:~/Downloads/Test$ cd Mood-Based-Playlist-Recommender
mauri@mauri-computer:~/Downloads/Test/Mood-Based-Playlist-Recommender$ cd frontend
mauri@mauri-computer:~/Downloads/Test/Mood-Based-Playlist-Recommender/frontend$ npm install
npm WARN EBADENGINE Unsupported engine {
  package: 'react-router@7.9.5',
  required: { node: '>=20.0.0' },
  current: { node: 'v18.19.1', npm: '9.2.0' }
}
npm WARN EBADENGINE
npm WARN EBADENGINE Unsupported engine {
  package: 'react-router-dom@7.9.5',
  required: { node: '>=20.0.0' },
  current: { node: 'v18.19.1', npm: '9.2.0' }
}
npm WARN EBADENGINE
(#####.#####) : idealTree: timing idealTree Completed in 978ms
```

7. Once frontend installation is done, do the same in the backend using cd backend with the proper output shown below

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
run `npm fund` for details
11 vulnerabilities (4 moderate, 7 high)
To address issues that do not require attention, run:
  npm audit fix
To address all issues (including breaking changes), run:
  npm audit fix --force
Run `npm audit` for details.
mauri@mauri-computer:~/Downloads/Test/Mood-Based-Playlist-Recommender/frontend$ cd ../
mauri@mauri-computer:~/Downloads/Test/Mood-Based-Playlist-Recommender$ cd backend
mauri@mauri-computer:~/Downloads/Test/Mood-Based-Playlist-Recommender/backend$ npm install
added 127 packages, and audited 128 packages in 777ms
24 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities
mauri@mauri-computer:~/Downloads/Test/Mood-Based-Playlist-Recommender/backend$
```

8. Add a .env file in the backend folder in this format

```
$ .env.example x
Mood-Based-Playlist-Recommenderr > backend > $ .env.example
1 # .env.example
2 PORT=5001
3 MONGODB_URI=your_mongodb_connection_string_here
4 SPOTIFY_CLIENT_ID=your_spotify_client_id
5 SPOTIFY_CLIENT_SECRET=your_spotify_client_secret
6
```

9. After setting up the .env in the backend folder we will type npm start in the backend folder for the backend to start

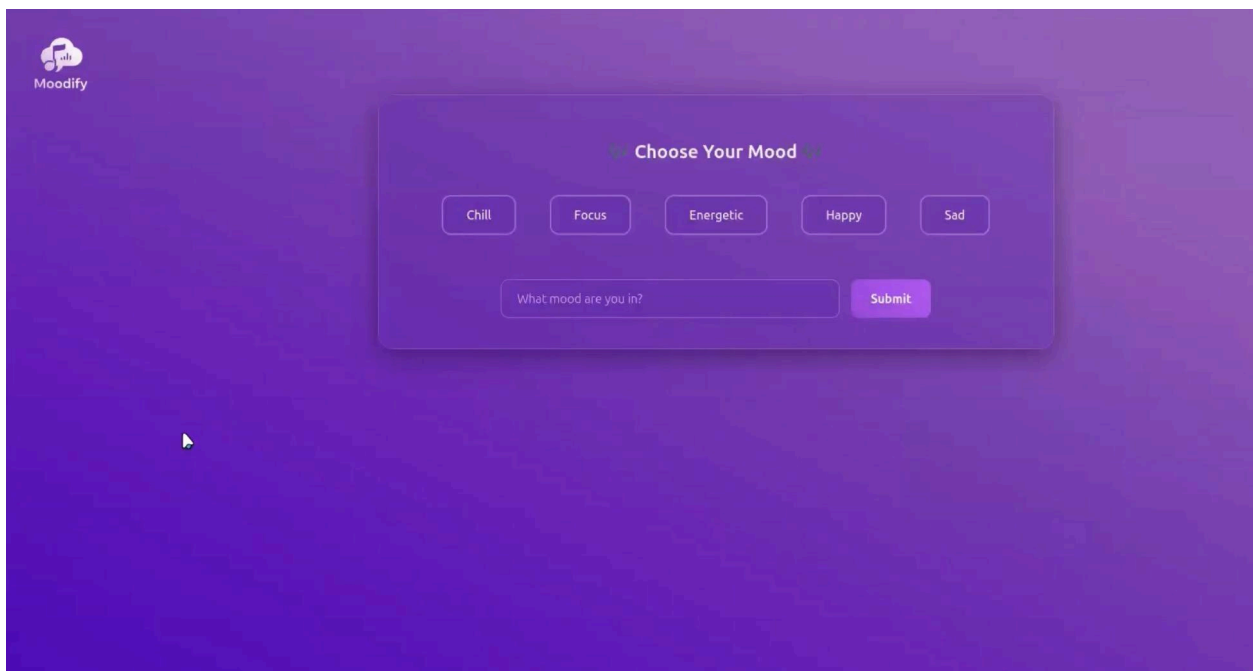
```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
added 127 packages, and audited 128 packages in 777ms
24 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities
mauri@mauri-computer:~/Downloads/Test/Mood-Based-Playlist-Recommenderr/backend$
* History restored
mauri@mauri-computer:~/Downloads/Test/Mood-Based-Playlist-Recommenderr/backend$ cd ../
mauri@mauri-computer:~/Downloads/Test/Mood-Based-Playlist-Recommenderr$ cd backend
mauri@mauri-computer:~/Downloads/Test/Mood-Based-Playlist-Recommenderr/backend$ npm start
> backend@1.0.0 start
> node server.js
[dotenv@17.2.3] injecting env (5) from .env -- tip: add access controls to secrets: https://dotenvx.com/ops
[dotenv@17.2.3] injecting env (0) from .env -- tip: prevent building .env in docker: https://dotenvx.com/prebuild
Server running on port 5001
MongoDB Connected: ac-qbkzbtj-shard-00-00.wkgkz8w.mongodb.net
I
```

It should run and have the output shown above

10. Then we create another terminal and set the directory into frontend and repeat the command npm start which should run the app

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
mauri@mauri-computer:~/Downloads/Test$ cd frontend
bash: cd: frontend: No such file or directory
mauri@mauri-computer:~/Downloads/Test$ cd Mood-Based-Playlist-Recommender
mauri@mauri-computer:~/Downloads/Test/Mood-Based-Playlist-Recommender$ cd frontend
mauri@mauri-computer:~/Downloads/Test/Mood-Based-Playlist-Recommender/frontend$ npm start
```

11. Once running, it should automatically open this page as shown below



B. User Manual

Introduction:

Moodify is a web-based application that helps university students quickly discover Spotify playlists that match their current mood. The system uses pattern-matching NLP to classify either a selected mood or a written mood description into one of five categories: Chill, Focus, Energetic, Happy, and Sad.

Based on this classification, Moodify retrieves up to 5 recommended playlists and 10 preview tracks per playlist from the Spotify Web API. This manual explains how to use Moodify and its main features.

System Requirements:

- Supported Devices
 - Any device with an Internet-connected web browser
- Browser Compatibility
 - Chrome
 - Firefox
 - Safari
 - Microsoft Edge
- Dependencies
 - No user login required
 - Spotify account is not required

Follow the steps with screenshots below:

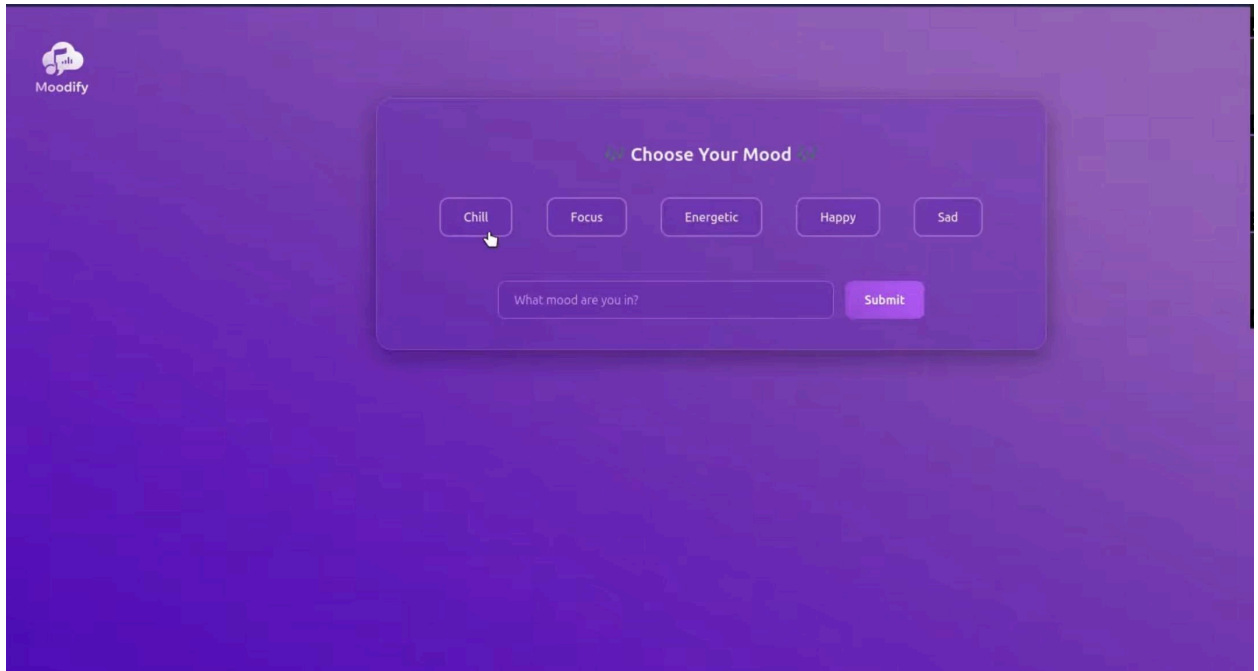
Launching Moodify

1. Open your web browser
2. Navigate to Moodify URL

Home Screen

The home page displays:

- 5 mood buttons:
 - Chill
 - Focus
 - Energetic
 - Happy
 - Sad
- A text box for typing your own mood description
- A submit button to generate playlist
- A navigation bar
- Logo on the top left of the screen



Main Features

Select a Predefined Mood

1. Click one of the mood buttons:
 - Chill
 - Focus
 - Energetic
 - Happy
 - Sad
2. The button will highlight to confirm your choice.
3. Click one of the mood buttons.

Moodify will fetch Spotify playlists related to your chosen mood.



Above example is of the Choose Your Mood display

Text Input Mode

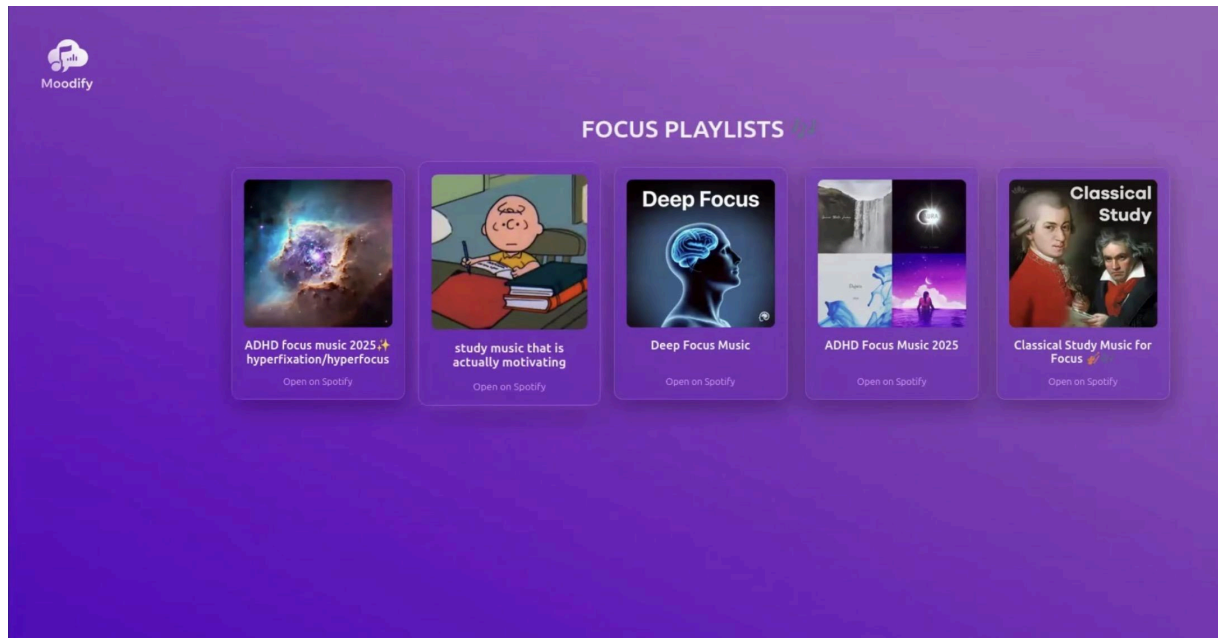
If your feelings don't fit a preset mood:

1. Locate the text input box labeled "What mood are you in?"
2. Type anything, such as:
 - "I'm stressed."
 - "I feel heartbroken."
 - "I feel super amazing."
3. Click Submit.

Below is an example of how to use the text box. User enters "I am locked in" into the text box:



Then Moodify generates playlists for the mood "Focus," which was obtained from the text box.

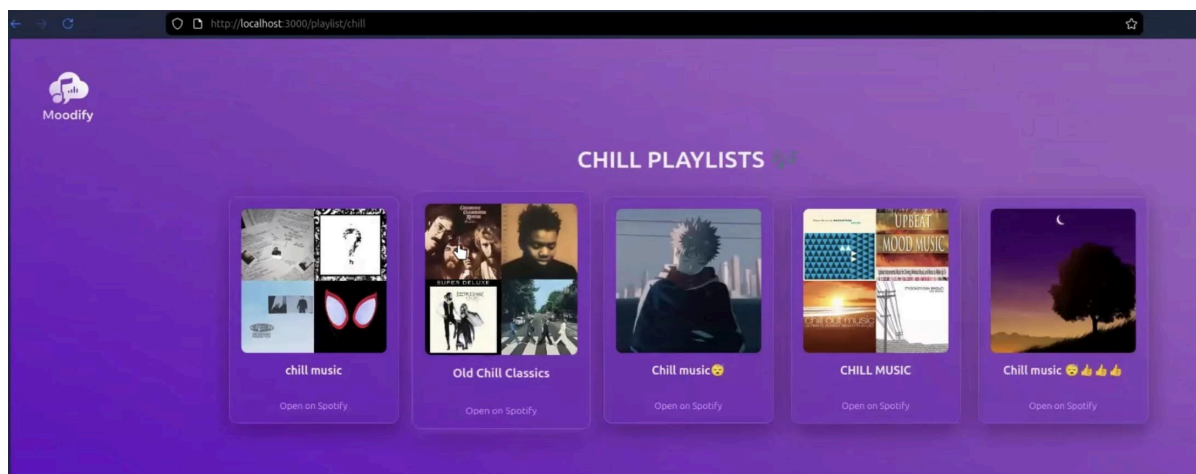


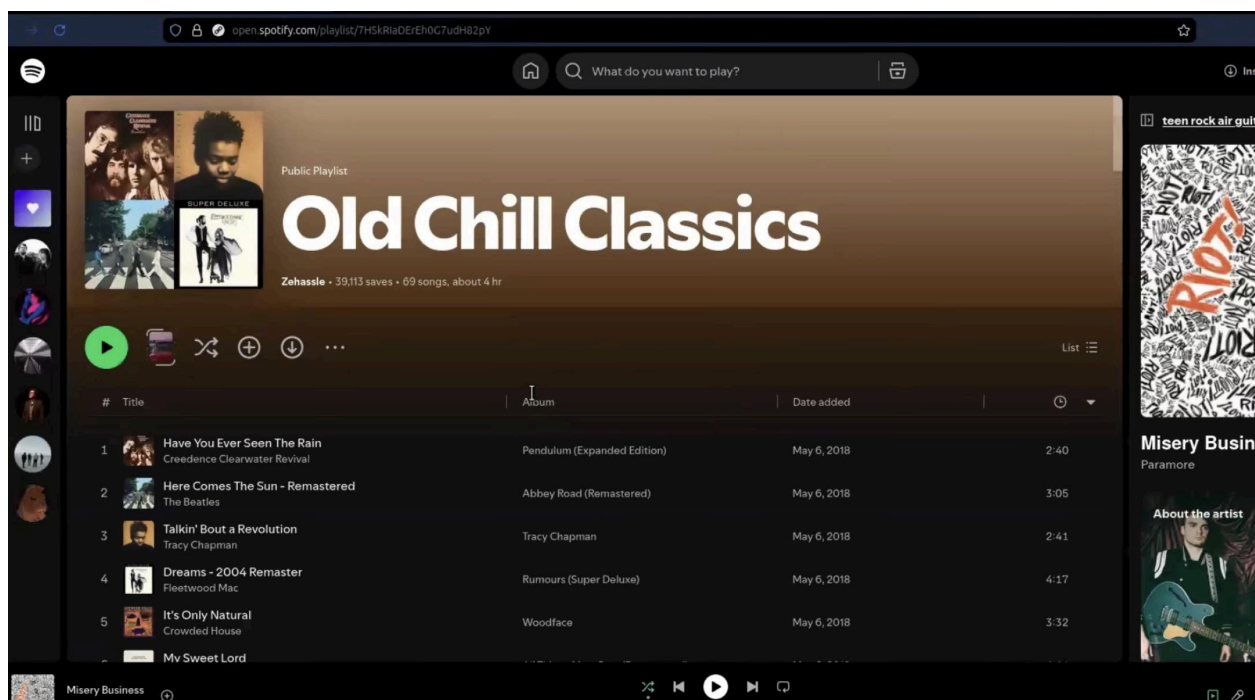
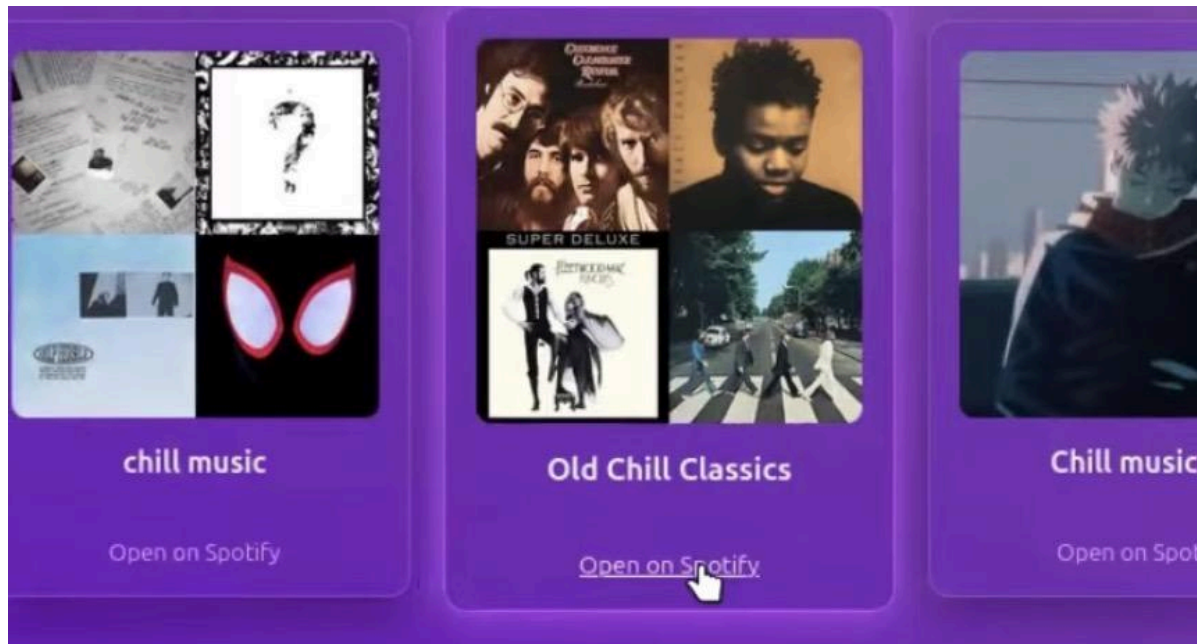
The NLP engine scans your text for mood-related keywords and assigns the mood with the highest match score. If it can't match with one of our assigned moods, it will default to the mood being chill.

Viewing Recommended Playlists

After you submit a mood:

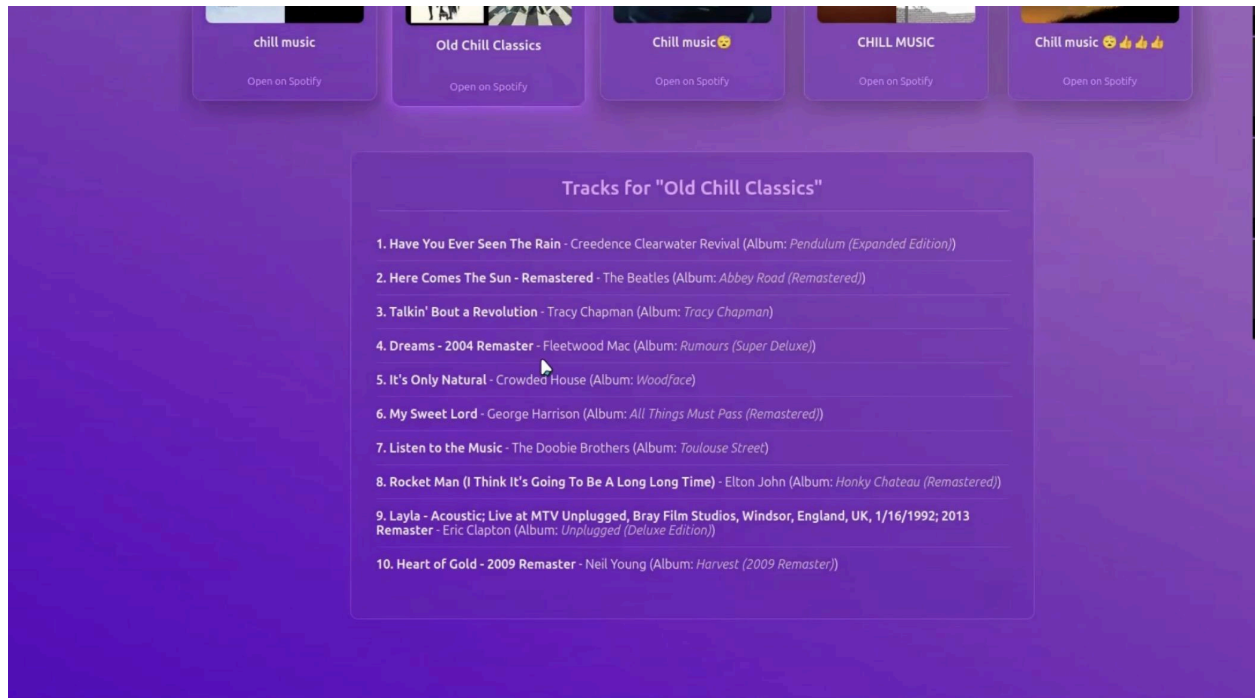
1. The app displays up to 5 playlists, each showing:
 - Playlist name
 - Cover image
 - Playlist name
 - “Open on Spotify” button





The new tab when “Open in Spotify” is clicked above

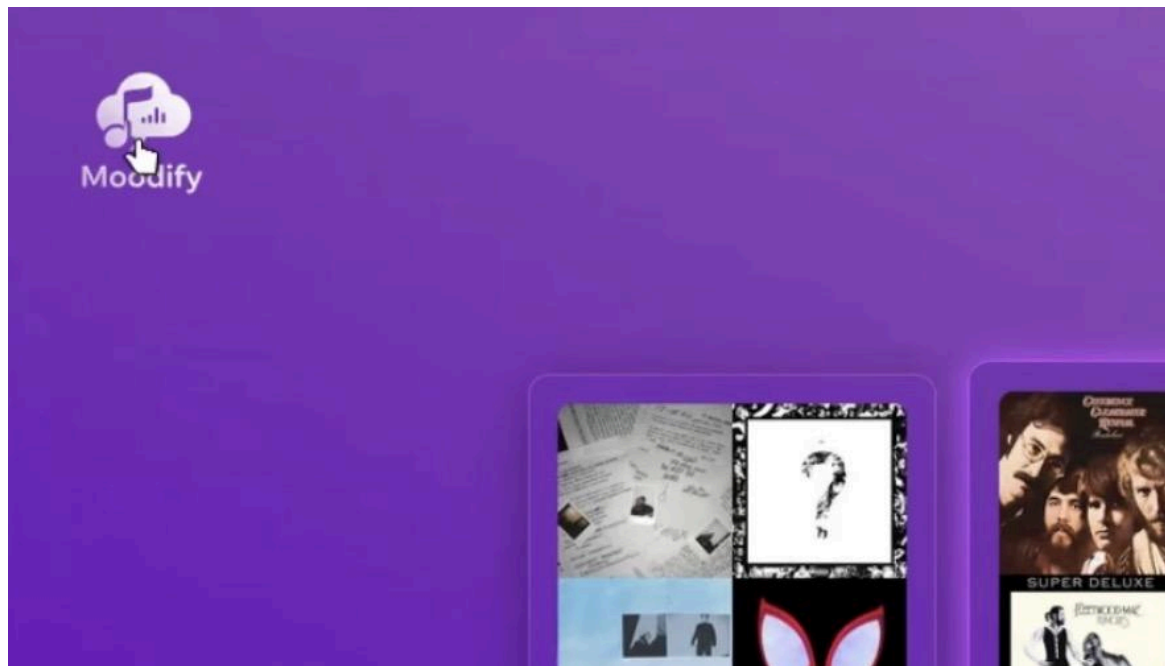
2. Click a playlist to load its preview tracks (up to 10):
 - Track name
 - Artist
 - Album



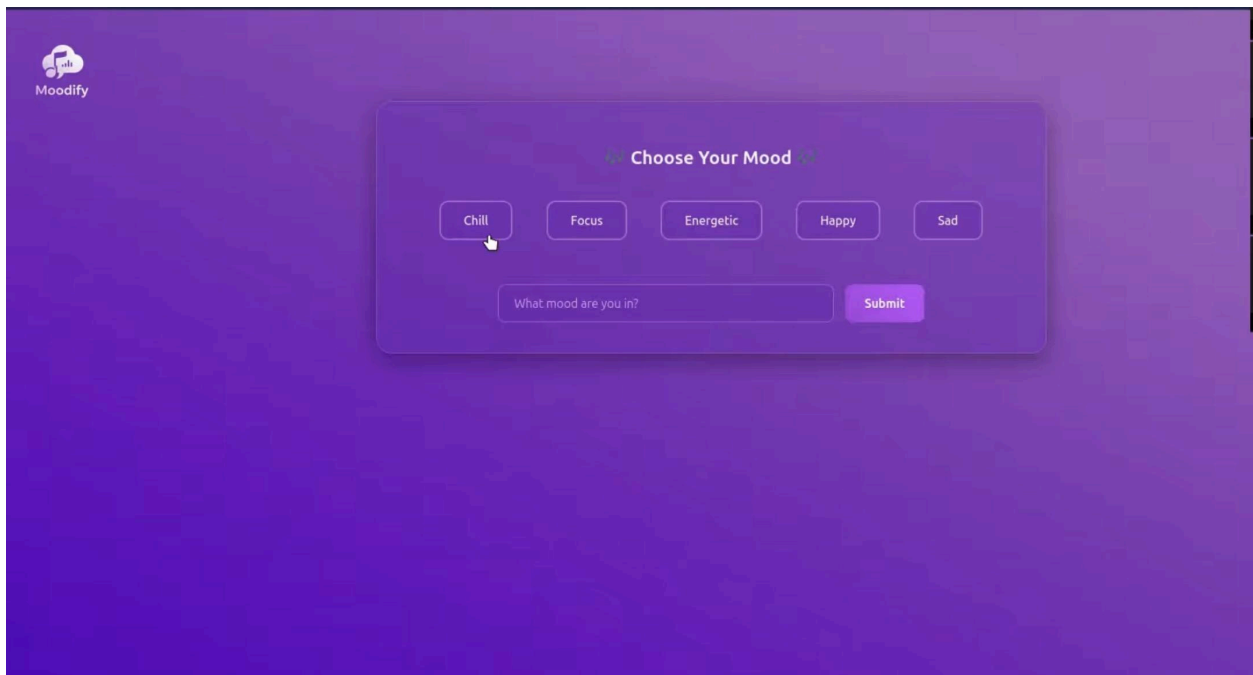
Returning to Home

If you are done looking at the playlists that are generated or want to see more for other moods:

1. Click on the Moodify logo to send you back to the landing page



2. Once clicked, the landing page will launch and users would be able to submit another mood if they want.



C. Admin/Operations Guide (runbook, on-call, backup/restore)

System Startup and Deployment:

- The backend requires a .env file containing PORT, MONGODB_URI, SPOTIFY_CLIENT_ID, and SPOTIFY_CLIENT_SECRET.
- The server is started by running server.js through Node.js using the production start command.
- The server listens on the port defined in the .env file (default 5001).
- The connectDB utility establishes the MongoDB connection.
- The Express app initializes middleware (cors, express.json()) and mounts the API routes (/api/playlists, /api/spotify).

On-Call:

- Database Connection Failure
 - Symptom: Server exits with code 1; console logs "Error: [error message]".
 - Component: config/db.js
 - Remediation:
 - Verify MONGODB_URI value in the environment file.
 - Ensure the MongoDB instance is running and reachable.

- Spotify Authentication Failure
 - Symptom: Console logs "Error retrieving Spotify token"; API requests return 503 or 500.
 - Component: `utils/spotifyAuth.js`
 - Remediation:
 - Review console logs for the specific authentication error.
 - Verify `SPOTIFY_CLIENT_ID` and `SPOTIFY_CLIENT_SECRET` values.
- Spotify API Call Failure
 - Symptom: Backend returns HTTP 500 with "Failed to fetch playlists from Spotify".
 - Component: `routes/spotifyRoutes.js`
 - Remediation:
 - Check the Spotify API service status.
 - Review the backend console for the exact error returned by the API.

Backup/Restore:

- The application itself is stateless; no user-specific data is stored within the server process.
- The primary items requiring backup are configuration files (such as `.env`) and the codebase.
- Backup procedure:
 - Keep a version-controlled repository of all source code.
 - Securely store copies of environment configuration files in an encrypted location (not in Git).
 - Maintain a backup of any deployment-related scripts or operational notes used by the team.
- Restore procedure:
 - Recreate the environment variables from the backed-up `.env` or configuration store.
 - Redeploy the backend and frontend from the version-controlled repository.
 - Verify functionality by starting the backend server and testing the API routes.

D. API Reference

The Moodify backend provides a lightweight REST API for retrieving Spotify playlist data and track information. All API routes operate under the `/api/spotify` namespace and use a server-managed Spotify access token obtained through the OAuth 2.0 Client Credentials flow.

Available Endpoints:

- `GET /api/spotify/token`
 - Returns a valid Spotify access token. The server automatically requests a new token if the cached one is expired. A successful response includes the token string

and the expiration timestamp. Errors generally indicate invalid environment variables or authentication failures with Spotify.

- GET /api/spotify/playlists?mood=
 - Returns up to five playlists matching the mood provided as a query parameter. The mood value must correspond to one of the supported mood categories. Each playlist includes its Spotify ID, name, description, and preview image. Errors may include missing query parameters, invalid mood categories, or upstream Spotify API issues.
- GET /api/spotify/playlists/:id/tracks
 - Returns up to ten tracks from the specified Spotify playlist ID. Each track object contains its title, the primary artist, and a URL to the track. Errors include invalid playlist IDs, missing route parameters, or Spotify service failures.

Data Model:

Playlist objects follow the schema defined in the backend code. Each object includes the mood category, playlist name, Spotify playlist ID, and an array of tracks. Each track within the array includes three fields: name, artist, and URL.

Authentication/Token and Error Handling:

The backend manages Spotify access tokens on the server side. Tokens are stored temporarily in memory and refreshed automatically. No client-level authentication or session management is required. All endpoints return structured error responses containing an error message and HTTP status code. Common error types include authentication failures, Spotify API rate limits or outages, invalid input, and internal server errors.

E. Poster, Slides, and Video Links

Poster Directory:

<https://drive.google.com/file/d/1x8eMc6lXXW8dG12nkwTyzbkUlheNYrd1/view?usp=sharing>

Presentation Slides Directory:

https://drive.google.com/file/d/1C_MekKTuGAoCwS0MIlpXYPqfwn36eyCg/view?usp=sharing

Video Directory:

<https://www.youtube.com/playlist?list=PLzaDCc1oERAj0L6YPexYpLy3S3kciD36m>

F. Scrum Evidence Index

Scrum Ceremonies Directory:

 AllScrumMeetings.pdf