FINAL PROJECT: Spotify Taste Mixer

Description

Develop a widget-based web application with Next.js that generates personalized Spotify playlists based on user musical preferences. The application must allow users to configure multiple preference widgets and visualize custom playlists directly on the web.

Create the following pages by making the necessary requests to the Spotify Web API, documented at [Web API | Spotify for Developers]

Authentication

Spotify OAuth Login: Implement Authorization Code Flow using Spotify's OAuth 2.0

```
https://accounts.spotify.com/authorize?
  client_id={CLIENT_ID}&
  response_type=code&
  redirect_uri={REDIRECT_URI}&
  scope=playlist-modify-public playlist-modify-private
user-read-private user-top-read&
  state={STATE}
```

NOTE: When you complete the OAuth flow, you will receive an access token from Spotify, which you must store in the browser's local cache:

```
// Save data to localStorage
localStorage.setItem('spotify_token', accessToken)
localStorage.setItem('spotify_refresh_token', refreshToken)

// Read data from localStorage
const token = localStorage.getItem('spotify_token')
```

Token refresh: Handle token expiration using the refresh token
Headers for API requests:

```
headers: {
  'Authorization': `Bearer ${token}`,
  'Content-Type': 'application/json'
}
```

Widget Dashboard

Main Interface: Display all preference widgets in a responsive grid layout

Widget Types to implement:

- Artist Widget: Search and select favorite artists: `GET /search?type=artist&q={query}`
- Track Widget: Search and select favorite songs: `GET /search?type=track&q={query}`
- Genre Widget: Select from available genres: `GET /recommendations/available-genre-seeds`
- Decade Widget: Choose preferred musical eras with year filtering
- Mood Widget: Select energy levels and musical characteristics
- Popularity Widget: Choose between mainstream hits and hidden gems

Playlist Generation

Central Recommendation Area:

- Display generated playlist based on widget selections
- Take it in count favorites for playlist generation (optional)

Playlist Management Features:

- Remove individual tracks from generated playlist
- Mark individual tracks as favorite
- Refresh playlist generated
- Add more tracks to existing selection
- Drag and drop reordering (optional)

Spotify Integration

Create Playlist: `POST /users/{user_id}/playlists` (optional)
Add Tracks to Playlist: `POST /playlists/{playlist_id}/tracks` (optional)
Get User Profile: `GET /me` for user information

Goals

1. Practice everything you've learned by creating a professional Next.js application.
2. Implement the basics of React:
    a. Creating reusable widget components
    b. Props and prop drilling
    c. useState and useEffect hooks
    d. Event Management across multiple components
    e. Parent-child communication between widgets and main app
    f. Conditionals and lists for rendering tracks
    g. Application of responsive styles
    h. Assigning dynamic styles and classes
    i. Styled components and CSS modules
3. Integrate external API:
    a. OAuth 2.0 authentication flow
    b. HTTP requests (axios or fetch) to Spotify Web API
    c. Error handling and rate limiting awareness
4. Local Storage Management:
    a. Persist authentication tokens
    b. Save favorite songs
    c. Save widget preferences (optional)
    d. Store playlist generation history (optional)

Troubleshooting Guide

1. Planning

- Design the structure of the application:
    - pages/routes (`/`, `/auth/callback`, `/playlist/[id]`)
- Components (Widgets, PlaylistDisplay, TrackCard, Layout)
- Navigation flow and user journey
- Define component state and props structure
- Choose the libraries to use (optional: React Hook Form, Formik, Axios)

2. Implementation

2.1 Routing

Configure application routes with NextJS App Router or Pages Router
Create page.js and layout components for each route
Handle OAuth callback route for Spotify authentication

2.2 Components

Develop interface components where applicable:

Header with authentication status
Widget container grid
Individual widget components (Artist, Track, Genre, etc.)
Central playlist display area
Track card component with remove/preview functionality
Navigation menu
Loading states and error boundaries
Authentication pages
Modal for playlist creation

2.3 Functionalities

Implement the logic of each component:

OAuth flow handling and token management
Widget state management and selection limits
Search functionality with debouncing
Playlist generation algorithm using search filters
Track removal and playlist modification
Save playlist to Spotify account

2.4 Styles

Style components with CSS modules or styled-components
Implement responsive design for mobile, tablet, and desktop
Use CSS Grid or Flexbox for widget layout
Consider using Tailwind CSS (optional)

3. Integration of Libraries (optional)

Implement React Hook Form for authentication forms (optional)
Use Axios for HTTP requests with interceptors for token refresh
Add loading skeletons or spinners for better UX

4. Testing and Debugging

Test OAuth flow in different browsers
Test responsive design on various screen sizes
Handle API rate limits and error responses
Fix bugs and optimize performance

Navigation

Use Next.js Link component for client-side navigation:

```jsx
<Link href="/dashboard">Dashboard</Link>
```

Use useRouter for programmatic navigation after authentication:

```javascript
const router = useRouter()
router.push('/dashboard')
```

Additional Considerations

- Error Handling: Display user-friendly error messages for API failures
- Loading States: Show loading indicators during API requests
- Rate Limiting: Implement request throttling to respect Spotify's API limits
- Token Expiration: Handle token refresh automatically
- Responsive Design: Ensure widgets work well on mobile devices
- Data Persistence: Save user preferences in localStorage
- Security: Never expose client secret in frontend views
- API Limitations: Explore limits of the API sec

API Endpoints Reference

Authentication

- OAuth Authorization: `https://accounts.spotify.com/authorize`
- Token Exchange: `POST https://accounts.spotify.com/api/token`
- Token Refresh: `POST https://accounts.spotify.com/api/token`

Search & Discovery

- Search: `GET /search?type={type}&q={query}&limit={limit}`
- Get Available Genres: `GET /recommendations/available-genre-seeds`
- Get User's Top Items: `GET /me/top/{type}?time_range={range}&limit={limit}`

Playlist Management

- Get User Profile: `GET /me`
- Create Playlist: `POST /users/{user_id}/playlists`
- Add Tracks to Playlist: `POST /playlists/{playlist_id}/tracks`
- Get Playlist Details: `GET /playlists/{playlist_id}`

Required Scopes

```
playlist-modify-public
playlist-modify-private
user-read-private
user-top-read
```

Important Notes

1. The Spotify recommendations endpoint has been deprecated for new applications.
2. Use search-based discovery with filters instead.
   Implement proper error handling for common issues like expired tokens, rate limiting, and network errors.
3. Ensure your application handles the OAuth flow correctly, including state parameter validation.
4. Test thoroughly with different musical preferences to ensure playlist quality.
5. Consider implementing more functionalities with the Spotify API.
