Project Plan: From Text Prompt to AI-Generated Song

This document outlines a comprehensive plan to build a song generation application. The system will take a user's text prompt, generate lyrics, and then convert those lyrics into an audio file.

1. High-Level System Architecture

Before diving into the details, let's visualize the end-to-end workflow of your application.

1. **User Input (Frontend):** A user types a prompt into a web interface.
   - *Example Prompt:* `"Create a melancholic Lo-fi hip-hop song about walking through a rainy city at night. The lyrics should be clean and reflective."*
2. **Lyrics Generation (Backend):** The prompt is sent to a backend server, which uses an AI model to generate song lyrics that match the prompt's genre, mood, and theme.
   - This is the core step where you can either use your own custom-trained model or a powerful API like Gemini. The system could even generate 2-3 options for the user to choose from.
3. **User Selection (Frontend):** The generated lyric options are displayed to the user. The user selects their favorite version.
4. **Music Generation (Backend):** The chosen lyrics are sent to a Text-to-Music model. This model generates an instrumental track and attempts to align the lyrics with the music, producing a final audio file (e.g., in .wav or .mp3 format).
5. **Final Output (Frontend):** A link to the generated audio file is provided to the user, allowing them to play and download their song.

2. Phase 1: Lyrics Generation (The Core AI Challenge)

This is the most critical part of your project. You have two main paths you can take, and you can even implement both to compare results.

Option A: Build Your Own Lyrics Model (The Deep Learning Path)

This is the more challenging but incredibly rewarding path. It involves training your own neural network. Since you mentioned RNNs and LSTMs, this is a great place to apply them, although we should also consider more modern architectures.

- **Concept:** You will train a model on a massive dataset of existing song lyrics. The model learns the patterns, structure, rhyme schemes, and vocabulary associated with different genres.
- **Key Technologies:** Python, TensorFlow/PyTorch.
- **Model Architectures:**
  - **RNN/LSTM:** The classic choice for sequence data like text. They maintain an internal state to remember previous words, which is crucial for generating coherent sentences. This is a great starting point to build on your existing knowledge.
  - **Transformers (e.g., GPT-style):** This is the current state-of-the-art for language

generation. Models like GPT-2 are based on the Transformer architecture, which uses an "attention" mechanism to weigh the importance of different words in the input, allowing it to capture much longer-range dependencies and context. **Fine-tuning a pre-trained Transformer model is the most effective approach.**

- **Step-by-Step Guide:**
  1. **Data Collection:** You need a large, high-quality dataset of lyrics with genre labels.
     - **Where to find data:** Look for "lyrics datasets" on platforms like **Kaggle**. You can also use APIs like the **Genius API** to fetch lyrics, but be mindful of their terms of service. Your goal is to get thousands of songs per genre you want to support.
  2. **Data Preprocessing:** This is a crucial step. You'll need to clean the text (remove special characters, convert to lowercase), and then **tokenize** it (convert words into numerical IDs). You'll also structure the data into input/output sequences for the model to learn from (e.g., given the first 10 words, predict the 11th).
  3. **Model Training:** You'll write a script to build and train your chosen model (LSTM or Transformer). This involves feeding the processed data to the model over many iterations (epochs) and optimizing it to minimize prediction errors. **Warning:** This step requires significant computational power, often a powerful GPU. You can use services like **Google Colab** which provide free GPU access.
  4. **Generation (Inference):** Once trained, you'll write a function that takes a starting prompt (e.g., "Rainy city night"), feeds it to the model, and then iteratively predicts the next word to generate a full set of lyrics.

Option B: Use a Pre-trained LLM like Gemini (The Practical Path)

This path leverages a massive, pre-existing model to get high-quality results quickly with minimal setup.

- **Concept:** You use "prompt engineering" to instruct a powerful Large Language Model (LLM) to act as a lyricist. The quality of your prompt directly determines the quality of the output.
- **Key Technologies:** A simple API call using Python (requests) or any other language.
- **How it Works:** You will make a POST request to the Gemini API with a carefully crafted prompt.
- **Example of a Strong Prompt:**
  You are an expert songwriter specializing in the [Genre] genre.
  Your task is to write a complete song based on the following details.

  Genre: Hip-Hop
  Theme: The struggle and triumph of an underdog artist.
  Mood: Hopeful, determined, slightly gritty.
  Keywords: concrete, dreams, skyline, microphone, heartbeat.
  Structure:

- Intro
- Verse 1
- Chorus
- Verse 2
- Chorus
- Bridge
- Outro

Please generate the lyrics now.

- **Implementation:** You would simply replace the bracketed [Genre], [Theme], etc., with the user's input before sending the prompt to the API.

3. Phase 2: Music Generation (Text-to-Audio)

Once you have the lyrics, you need to turn them into music. This field is called Audio Generation or Text-to-Music. The key here, as you specified, is to use open-source and free tools.

- **The Challenge:** High-quality text-to-music generation is computationally intensive and a newer field than text generation. Finding a free, easy-to-use API is not possible, but you can run open-source models yourself.
- **Top Open-Source Recommendations:**
    1. **MusicGen (from Meta AI):** This is one of the best open-source models available right now. It can generate music from a text description and can even be guided by a reference melody. You can find the code and pre-trained models on GitHub. You would run this on your backend server.
    2. **Riffusion:** A very creative model that generates music by creating visual spectrograms (a visual representation of audio) with a Stable Diffusion model, and then converting that image back into audio. It's excellent for generating short loops and instrumentals. The project is fully open-source.
    3. **Mustango:** A text-to-music model from the research community that has open-source code available for you to run.
- **How to Integrate:** You will need to set up the chosen model's environment on your backend server (likely a Python environment with PyTorch). Your backend code will take the user-selected lyrics, pass them to the music generation model, and save the resulting audio file.

4. Recommended Roadmap

Trying to do everything at once can be overwhelming. Here is a practical, step-by-step approach to build your project:

1. **Step 1: Build the Core Pipeline (Proof of Concept).**
    - **Frontend:** Create a very simple HTML page with a text area for the prompt and a "Generate" button.
    - **Backend:** Set up a simple backend using **Flask** or **FastAPI** in Python.

   - **Lyrics:** Integrate the **Gemini API** first. It's the fastest way to get high-quality lyrics.
   - **Music:** Choose **MusicGen** or **Riffusion**. Set it up on your backend and get it to generate a short instrumental based on the user's prompt (e.g., "a sad piano melody"). At this stage, don't worry about syncing the lyrics yet.
   - **Goal:** Prove you can go from a text prompt to a generated audio file.
2. **Step 2: Refine and Integrate.**
   - Improve the frontend UI.
   - Work on the music generation step to better incorporate the *mood* of the lyrics. Some models can take lyrics as an input, but the results can vary. A more reliable approach is to generate an instrumental that matches the prompt's mood and then simply layer the lyrics on top (which would require a separate vocal synthesis step, which is even more complex). The simplest start is to just provide the instrumental and the lyrics side-by-side.
3. **Step 3: Build Your Custom Lyrics Model.**
   - Now that you have a working application, you can tackle the ambitious goal of building your own lyrics model.
   - Follow the steps in **Phase 1, Option A**. Start with an LSTM model to learn the fundamentals.
   - Train it and compare its output to the Gemini API's output. You can even offer both options to the user in your app: "Generate with Quick AI" vs. "Generate with Custom AI".

This phased approach allows you to get a working product quickly while giving you the space to dive deep into the model-building process you're excited about.