1. UI ---> Express JS ---> PostgreSQL

Create a JSON containing the parameters, tables, GET or POST type for each type of query

Feed it to gpt and ask it to give the SQL query and a query name for each in a JSON.

2. UI sends the query name along with the other parameters such as user id and more to Node server, then node server maps the query name to sql query and performs insertions of these parameters into the query.
3. The db query is ready and executed and gets the data from the database and returns it to the UI.
4. UI ----> Sends generation request

   Node JS Server ----> Receives the request and responds with status 200 to the UI, along with the user id and the request id so only now the UI can keep making requests based on this user id and request id, then the Node server makes GET requests to Python API.

   Python API ---> sends the tokens to the NoSQL db to be stored

   Node JS Server ---> keeps making db requests for the music chunks from the NoSQL db and stores it in some memory.

   UI ---> keeps polling for the chunks from the Node JS server, which now responds with the chunks.
5. Keep a separate Database for each business.

   --> This will ensure that there is no mismatch and transfer of wrong data when requests are made.

   --> Even by mistake we cannot access the wrong database since we need to enter the DB credentials.

Streaming Process:

1. Node Server ---> Sends POST request with userID, requestID, promptText ---> Python API Server
2. Python API Server ---> Generate and store the chunks to the NoSQL database one by one along with ---> rowId, userId, and which record in the db it stores the chunks to.
3. Node Server ---> Keeps checking with NoSQL db for any music chunks data in that record.
4. UI ---> Keeps asking for chunks for of music for the specific userId, requestId ---> Node Server
5. Node Server ---> Returns the data as it is created in the NoSQL ---> UI
6. Separate worker process for completed or fully generated music to upload or download the music either in one file or multiple chunks if file is large. Queues are mentioned below for handling mulitple requests. (Upload Input queue, Upload Retry queue, Download Input queue, Download Retry queue)
   1. Upload Queue --> upload the completed chunks or file to Google Storage (Similar to S3) through the google api.
   2. Download Queue --> download the file from Google Storage ---> Store into NoSQL
   3. UI --> when the user want to access the music later --> Node Server --> makes request to NoSQL db for chunks or file --> Same process unfolds as stated in steps 1 to 6.

User Stories:

1. Instant Stream
2. Play any stream that is created.

3. Delete a project ---> set of music data
4. Modify the music ---> Versions
5. Download the music file

   V1 --> Assume for now we keep all the music in the NoSQL db (both for chunks and completed music storage) & dont delete it unless the user deletes it.

   V2 --> Google Storage for files (similar to S3)

Storage:

1. RAM --> Present in the computer Chip ---> Fastest processing speed and data bandwidth (rate at which it can transfer data to the CPU). --> More Expensive ---> Cache is a part of RAM --->
2. Database Layer ---> Slower ---> Present in the computer disk --> Lesser expensive

   --> High Quality Hard disk --> Database

   --> Low Quality Hard disk --> S3
3. S3 is object storage --> Raw data ---> no structure
4. File storage has Folder structure
5. When streaming we will use cache since it is faster and more efficient to get the recent chunks from it.
6. Example of Cache --> Redis, Google Cache Service --> Google provides RAM and cache services.
7. Web server is only supposed to handle http requests from the UI
8. Hence there needs to be a separate worker process which will get the object from the S3 and then we will stream it to the UI