Arquitectura del Software — *miNoise*

1. Capa de Usuario (Interfaz 3D)

Componente: Navegador web **Rol:** Permitir la interacción con el sistema a través de una interfaz visual en 3D.
Tecnologías:

- React
- Vite
- @react-three/fiber
- @react-three/drei
- Shader personalizado (*DitherShader*)

Funciones clave:

- Visualizar canciones como puntos/esferas 3D.
- Mostrar información al seleccionar cada canción (nombre, artista, género, año).
- Permitir navegación orbital (OrbitControls) y exploración interactiva del mapa musical.

---

2. Capa de Presentación (Frontend)

Componente: Aplicación React/Vite **Rol:** Renderizar la visualización 3D y consumir los datos del backend Flask.

Entradas:

- Datos JSON entregados por la API Flask (`/import_dataset_hybrid`, `/api/songs`)

Salidas:

- Representación visual de los datos en el *Canvas 3D*.
- Eventos de usuario (clics, rotación, selección).

Comunicación:

- Solicitudes HTTP con `fetch` o `axios`.
- Proxy configurado en `vite.config.js` hacia `http://localhost:5000`.

---

3. Capa de Aplicación (Backend Flask)

Componente: Servidor Flask **Rol:** Gestionar las solicitudes, ejecutar el algoritmo de importación y entregar datos al frontend.

Módulos principales:

- `app.py` — Controlador principal de la API.
- `models.py` — Definición de entidades (User, Song).
- `features_librosa.py` — Extracción de características de audio.

Endpoints relevantes:

- `/import_dataset_hybrid` → Ejecuta el algoritmo principal.
- `/spotify_emerging/<genre>` → Consulta artistas emergentes.
- `/spotify_search_app/<genre>` → Búsqueda de artistas.
- `/api/songs` → Devuelve dataset procesado para visualización.

---

4. Capa de Procesamiento y Análisis

Componente: Algoritmo `import_dataset_logic()` **Rol:** Recolectar, enriquecer, analizar y filtrar información musical.

Flujo interno del algoritmo:

1. **Carga de semillas (`seeds.json`):** Define los géneros base (30 por defecto).
2. **Consulta a APIs externas:**
    - **Spotify API:** obtiene artistas y popularidad.
    - **Last.fm API:** obtiene tags y artistas similares.
    - **YouTube Music (ytmusicapi):** busca artistas relacionados.
3. **Descarga de audio:**
    - Con `yt-dlp`, guarda fragmentos `.wav` en `audio_cache/`.
4. **Extracción de características acústicas:**
    - Con `librosa`: MFCCs, tempo, rolloff, centroid, etc.
    - Genera vectores numéricos representativos por canción.
5. **Normalización y reducción:**
    - Con `StandardScaler` + `PCA (3 componentes)`.
    - Resulta en coordenadas `[PC1, PC2, PC3]`.
6. **Persistencia:**
    - Inserta en la base de datos (SQLAlchemy).
7. **Salida:**
    - Devuelve JSON con lista de artistas y canciones + features.

---

5. Capa de Datos

Componente: Base de datos (SQLAlchemy ORM) **Rol:** Almacenar la información procesada y permitir consultas eficientes.

Motores soportados:

- SQLite (entorno local)
- PostgreSQL (entorno de producción)

Tablas principales:

- `User`: usuarios y tokens.
- `Song`: canciones, metadatos, y vectores PCA.

---

6. Capa de Integraciones Externas

Servicios conectados:

Servicio	Propósito	Librería
Spotify	Búsqueda y metadatos de artistas	`requests`
Last.fm	Tags, artistas similares y oyentes	`requests`
YouTube Music	Identificación de canciones	`ytmusicapi`
YouTube	Descarga de audio en WAV	`yt-dlp`

---

7. Flujo general del sistema

1. Usuario interactúa con la interfaz 3D.
2. React solicita los datos al backend Flask.
3. Flask ejecuta o lee los resultados de `import_dataset_logic()`.
4. El algoritmo:
    - Recolecta artistas desde Spotify y Last.fm.
    - Busca canciones en YouTube Music.
    - Descarga y analiza audio con Librosa.
    - Genera dataset reducido (PCA).
5. Flask entrega los resultados JSON.
6. React renderiza el mapa musical interactivo con shaders.
