Melodynk – Full-Stack Music Library Web App Project Guide

---

1. Project Overview

Melodynk is a music library web application where users can upload, browse, and stream music. The frontend is built using JavaScript, and the backend uses Python (FastAPI). MySQL is used for storing metadata, and audio files are stored in a shared Docker volume.

The project uses multiple Docker containers to simulate a realistic multi-service architecture.

---

2. Project Structure

```
melodynk/
├── docker-compose.yml
├── frontend/
│   ├── index.html
│   ├── app.js
│   └── styles.css
├── backend/
│   ├── Dockerfile
│   ├── requirements.txt
│   └── main.py
├── db/
│   └── init.sql
└── shared_storage/
    └── (audio files will be saved here)
```

---

3. Database Schema (MySQL)

Table: users | id | username | password_hash | created_at |

Table: songs | id | user_id | title | artist | album | filename | uploaded_at |

Relationships: - `songs.user_id` references `users.id`

---

4. Backend (FastAPI)

Endpoints

- `POST /register` → Register a new user

- `POST /login` → Authenticate user
- `POST /upload` → Upload a song (multipart/form-data)
- `GET /songs` → List all songs
- `GET /songs/{id}/stream` → Stream a song file

Dockerfile (backend)

```
FROM python:3.12-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY . .
CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]
```

---

5. Frontend (JavaScript)

- `index.html` → Basic page layout with upload form, song list, and audio player
- `app.js` → Handles fetch calls to the backend API for uploads, listing songs, and streaming
- `styles.css` → Simple responsive styling

Example JS snippet for fetching songs:

```
fetch('http://localhost:8000/songs')
 .then(response => response.json())
 .then(data => {
   // populate song list
 });
```

---

6. Docker Compose

```
version: '3.9'
services:
 frontend:
  build: ./frontend
  ports:
   - "3000:80"
  depends_on:
   - backend

 backend:
  build: ./backend
  ports:
   - "8000:8000"
```

```
    volumes:
      - ./shared_storage:/storage
    depends_on:
      - db

  db:
    image: mysql:8
    environment:
      MYSQL_ROOT_PASSWORD: root
      MYSQL_DATABASE: melodynk
    volumes:
      - db_data:/var/lib/mysql

volumes:
  db_data:
```

---

7. Steps to Run

1. Create project folders as shown.
2. Add Python and JS files in backend/frontend.
3. Initialize database with `init.sql`.
4. Build and run containers:

   ```
   docker-compose up --build
   ```

5. Access frontend at `http://localhost:3000`
6. Upload, browse, and stream music.

---

8. Optional Extensions

- Add user playlists and favorites
- Add search functionality by artist or album
- Add recommendations or most-played lists
- Background service for audio conversion or compression

Melodynk gives you a fully Dockerized, multi-container environment to practice frontend-backend integration, SQL database management, and media handling.