MOOD MUSIC RECOMMENDER

(DISTRIBUTED PARALLEL SYSTEM)

GROUP MEMBERS:

- **UBAIDILLAH AL-AZHAR (00202300047)**
- **DESY NURSALSABILA (001202300003)**

PRESIDENT UNIVERSITY

20243

I. INTRODUCTION

This report details the development and deployment process of the "Mood Music Recommender" web application. The application aims to provide users with music recommendations and inspirational quotes based on their current mood, offering a personalized and uplifting experience. This document covers the application's overview, technology stack, team roles, design, and a comprehensive step-by-step guide to its Docker-based deployment.

II. APPLICATION OVERVIEW

2.1 Purpose

The primary purpose of the Mood Music Recommender is to help users discover music that aligns with their emotional state. By selecting a mood, users receive a curated list of songs, each accompanied by a unique inspirational quote, designed to enhance or complement their feelings.

2.2 Key Features

- **Mood Selection:** Users can choose from predefined moods (Happy, Sad, Chill, Angry, Romantic).
- **Music Recommendations:** Based on the selected mood, the application displays a list of songs. For this initial phase, we add song data with YouTube links.
- **Unique Quotes:** Each recommended song is paired with a distinct inspirational quote fetched from the ZenQuotes.io API.
- **Favorites Management:** Users can add songs to a favorites list, which persists across sessions (stored in a JSON file on the backend). The favorites list can be filtered by mood.
- **Responsive UI:** The frontend is designed to be user-friendly across various devices.

2.3 Technology Stack

- Frontend: HTML, CSS, JavaScript
- Backend: Python with Flask framework
- Data Storage: JSON file (for favorites and history)
- APIs Used: ZenQuotes.io (for quotes)
- Web Server/Reverse Proxy: Nginx (for serving frontend static files and proxying API requests)
- Containerization: Docker and Docker Compose

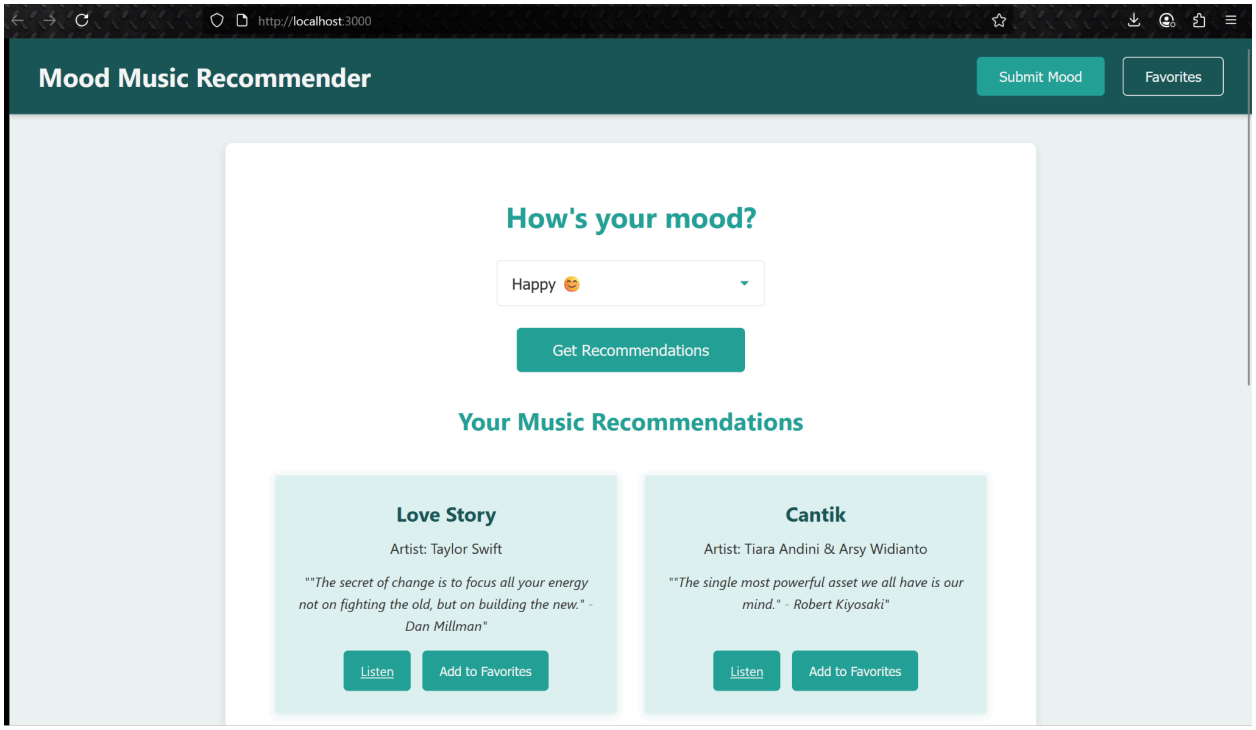
III. TEAM ROLES

- **Desy Nursalsabila:** Responsible for **Backend Development**. This includes setting up the Flask application, handling API endpoints for recommendations and favorites, integrating with ZenQuotes.io, managing JSON data persistence. and configuring the backend Dockerization.
- **Ubaidillah Al-Azhar:** Responsible for **Frontend Development**. This involves designing and implementing the user interface using HTML, CSS, and JavaScript, ensuring responsive design, and managing user interactions on the client side.

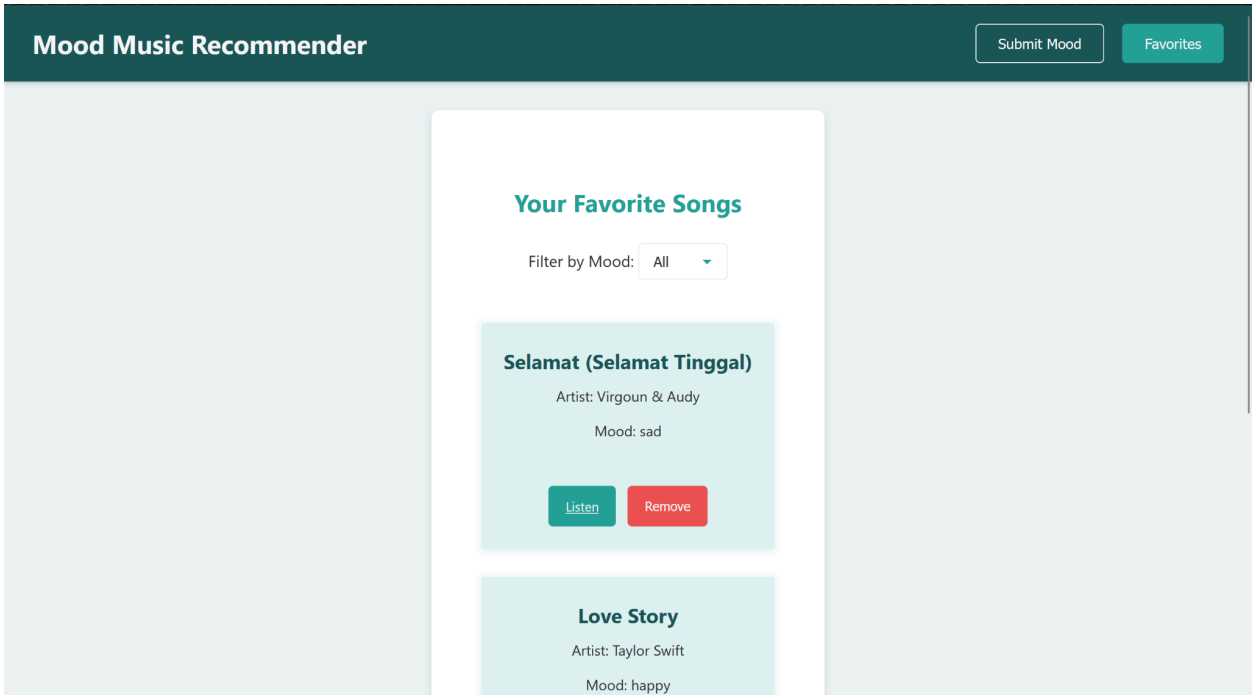
IV. APPLICATION DESIGN & SCREENSHOTS

V. User Interface (frontend)

The application features a clean and intuitive interface. The main page presents a mood selection dropdown and a "Get Recommendations" button. Upon selection, a grid of music cards appears, each displaying the song title, artist, a YouTube link, and a unique quote. A separate "Favorites" section allows users to view and manage their liked songs, with filtering options by mood.



This screenshot illustrates the initial view of the application, featuring the "How's your mood?" dropdown, the "Get Recommendations" button, and a sample grid of recommended songs, each with its title, artist, and a unique quote.



This screenshot displays the "Your Favorite Songs" section, including the mood filter dropdown and a list of favorite songs, showing their title, artist, and the mood they were associated with when added.

4.2 Backend Architecture

The backend is built with Python Flask, serving as the API layer for the frontend.

- **API Endpoints:**
  - /: Serves the index.html file (the main frontend application).
  - /recommend (POST): Receives the selected mood from the frontend, fetches dummy song data, retrieves a unique quote for each song from ZenQuotes.io, and returns the recommendations to the frontend. It also logs the recommendation history to favorites.json.
  - /favorites (GET): Retrieves and returns the list of favorite songs from favorites.json.
  - /add_favorite (POST): Adds a new song to the favorites.json file.
  - /delete_favorite (POST): Removes a song from the favorites.json file.
- **Data Persistence:** A favorites.json file is used to store both the history of recommendations and the user's favorite songs. This file is managed directly by the Flask application.
- **ZenQuotes.io Integration:** The backend makes HTTP requests to the ZenQuotes.io API to fetch random quotes. To ensure unique quotes for each song within a single recommendation request, the backend fetches multiple quotes and assigns them uniquely.

V. STEP-BY-STEP PROJECT DEVELOPMENT PROCESS

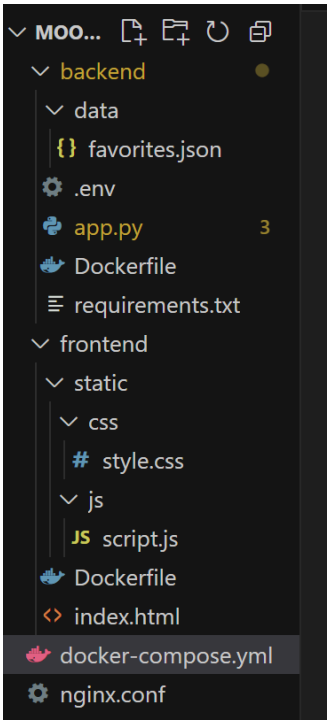
The application is containerized using Docker, with Docker Compose orchestrating two separate services: backend (Flask application) and frontend (Nginx web server).

5.1 Project Planning & Requirements Gathering

- **Define Core Idea:** Establish the main concept of a "Mood Music Recommender."
- **Identify Key Features:** Determine essential functionalities like mood selection, music recommendation display, quote integration, and favorites management.
- **Technology Selection:** Choose the technology stack (HTML/CSS/JS for frontend, Flask/Python for backend, JSON for data, ZenQuotes.io for quotes, Docker for deployment).
- **API Research:** Research suitable APIs for music (initially considered Spotify/Last.fm, then decided on dummy data for simplicity) and quotes (ZenQuotes.io).
- **Team Role Assignment:** Clearly define responsibilities for frontend (Ubaidillah Al-Azhar) and backend (Desy Nursalsabila).
- **Initial UI/UX Sketching:** Conceptualize the layout and user flow (e.g., main page with mood input, recommendations display, favorites page).

5.2 Project Structure

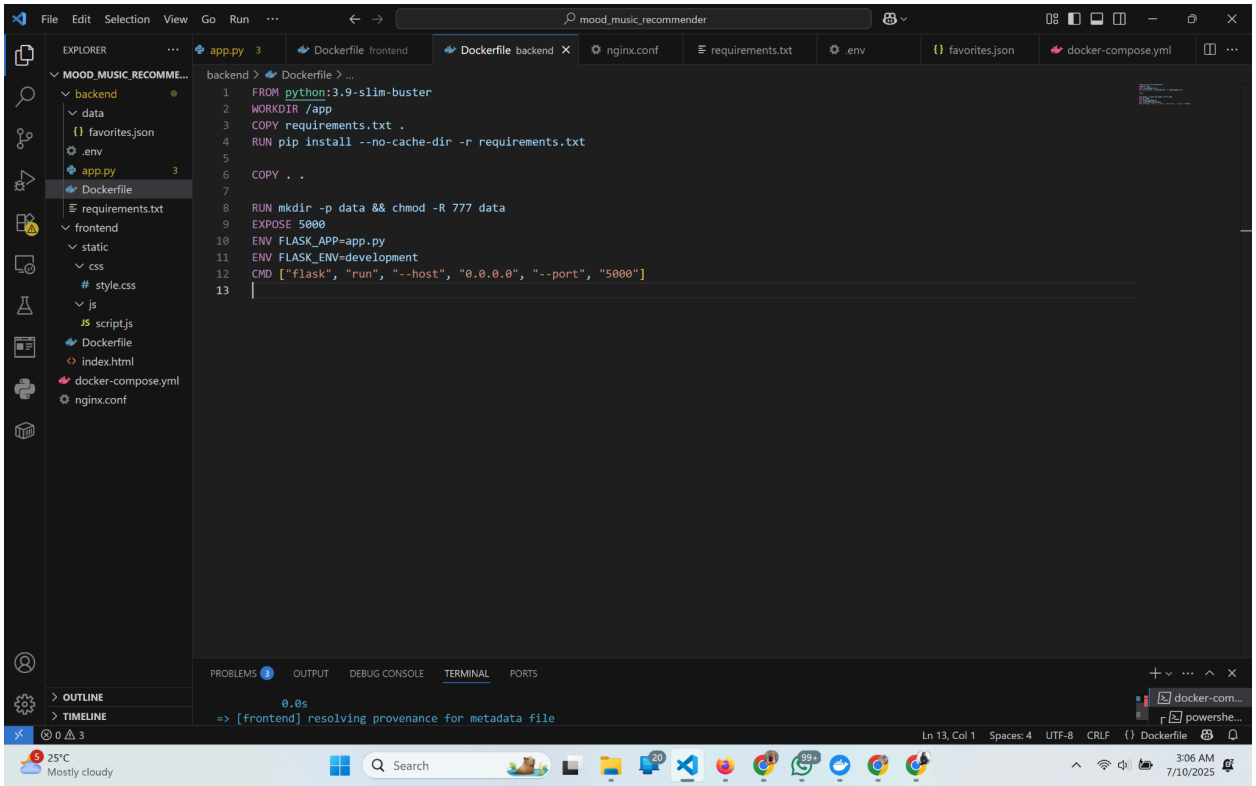
The project follows a modular structure, separating frontend and backend codebases, each with its own Dockerfile:



5.3 Dockerfiles

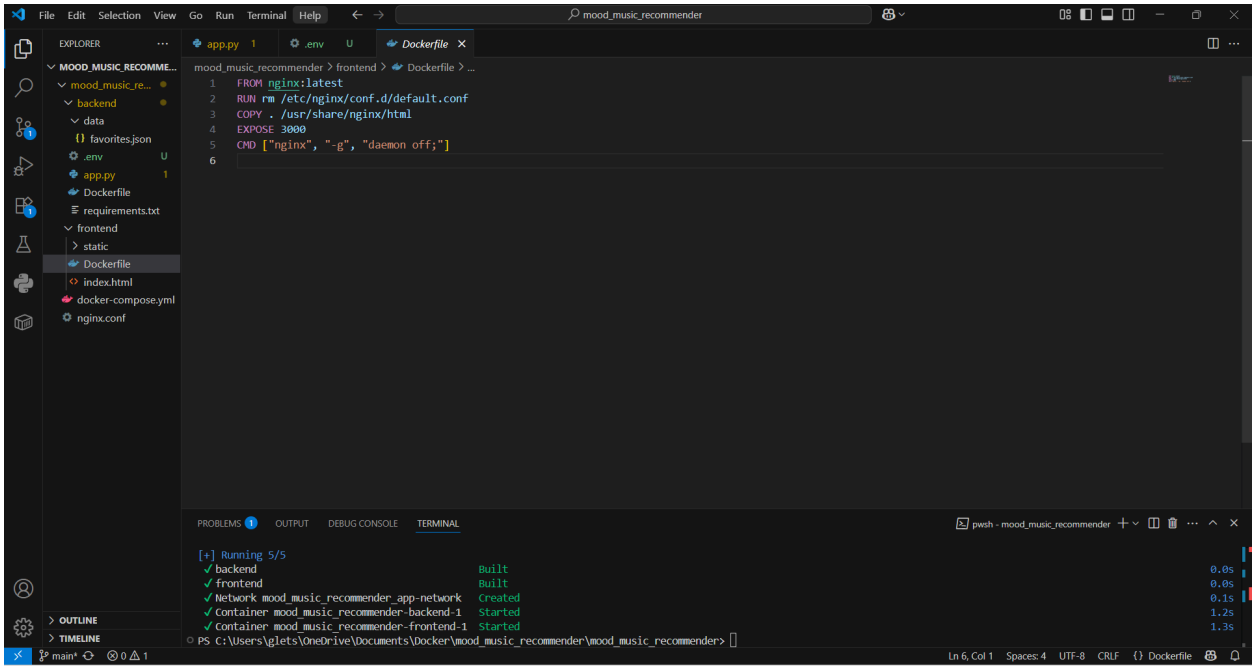
➢ **backend/Dockerfile :**

- Purpose: Builds the Docker image for the Flask backend application.
- Base Image: python:3.9-slim-buster
- Process: Copies requirements.txt, installs Python dependencies, copies the entire backend code, ensures the data directory exists with write permissions, exposes internal port 5000, and defines the command to run the Flask app.



➢ **frontend/Dockerfile :**

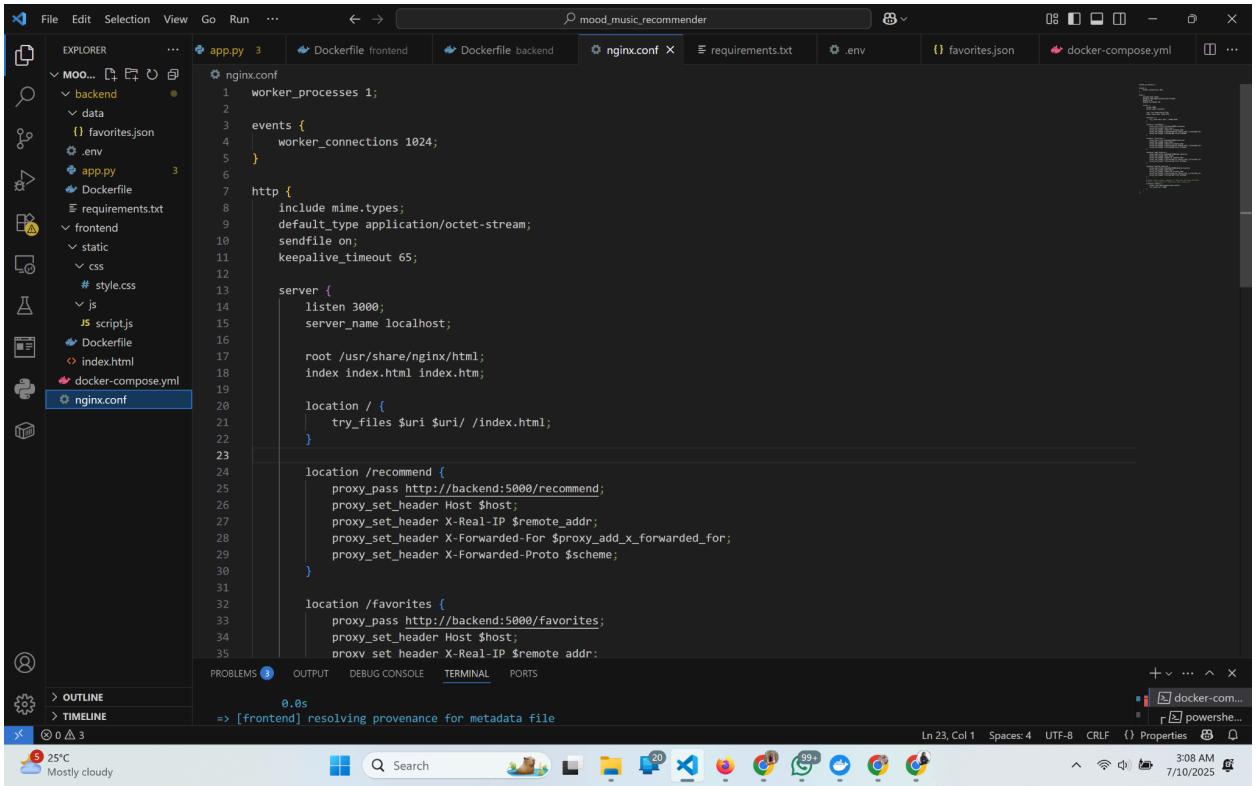
- Purpose: Builds the Docker image for the Nginx web server that serves the static frontend files.
- Base Image: nginx:latest
- Process: Removes default Nginx configuration, copies all frontend static files into Nginx's default web root (/usr/share/nginx/html), exposes internal port 3000, and sets the Nginx start command.



5.4 Nginx Configuration (nginx.conf)

Located in the project root, this file configures Nginx to:

- Listen on port 3000 (exposed to the host).
- Serve static files (HTML, CSS, JS) directly from the frontend/ directory (mounted into /usr/share/nginx/html in the container).
- Act as a **reverse proxy** for API requests (/recommend, /favorites, etc.). When the frontend makes an API call, Nginx intercepts it and forwards it internally to the backend service (which runs on http://backend:5000 within the Docker network). This keeps the backend's port internal and not directly exposed to the host.



5.5 Docker Compose (docker-compose.yml)

This YAML file orchestrates the multi-container application:

- Backend service:
  - build: Specifies to build from ./backend context using Dockerfile inside it.
  - ports: []: Explicitly states that no ports from the backend container are directly mapped to the host (it communicates internally with Nginx).
  - volumes: Mounts ./backend/data from the host to /app/data in the container, ensuring favorites.json data persists even if the container is removed or restarted.
  - env_file: Points to ./backend/.env to load environment variables (even if it's empty for this dummy data setup, it's good practice).
  - networks: Connects to app-network.

```yaml
docker-compose.yml
Run All Services
services:
  Run Service
  backend:
    build:
      context: ./backend
      dockerfile: Dockerfile
    ports:
      - "5000:5000"
    volumes:
      - ./backend/data:/app/data
    env_file:
      - ./backend/.env
    restart: unless-stopped
    networks:
      - app-network

```

➢ Frontend service:

- build: Specifies to build from ./frontend context using Dockerfile inside it.
- ports: - "3000:3000": Maps host port 3000 to container port 3000, making the frontend accessible from the host browser.
- volumes: Mounts the nginx.conf file from the host into the Nginx container's configuration directory.
- depends_on: - backend: Ensures the backend service starts before the frontend service.
- networks: Connects to app-network.

- **networks:** Defines a custom bridge network (app-network) to allow seamless communication between the frontend and backend containers using their service names.

```yaml
docker-compose.yml
services:
  ▷ Run Service
  frontend:
    build:
      context: ./frontend
      dockerfile: Dockerfile
    ports:
      - "3000:3000"
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf:ro
    depends_on:
      - backend
    restart: unless-stopped
    networks:
      - app-network

networks:
  app-network:
    driver: bridge
```

5.6 Environment Variables (backend/ .env)

- A .env file (even if empty in this current setup) must exist in the backend/ folder. This file is used by python-dotenv in app.py and by Docker Compose via env_file to load environment-specific variables.

5.7 Deployment Steps

To deploy the Mood Music Recommender application using Docker Compose:

- **Prerequisites:** Ensure Docker Desktop is installed and running on our system.
- **Project Setup:** Verify that our project structure matches the one described in Section 6.1, with Dockerfiles inside their respective backend/ and frontend/ folders, and nginx.conf and docker-compose.yml in the project root. Also, confirm that backend/.env exists (even if empty).
- **Navigate to Project Root:** Open the terminal or Command Prompt and navigate to the root directory of our mood_music_recommender project.

```
PS C:\Users\Desy Nursalsabila\mood_music_recommender>
```

- **Build and Run Containers:** Execute the Docker Compose command to build the images and start the services:

```
PS C:\Users\Desy Nursalsabila\mood_music_recommender> docker-compose up --build
Compose can now delegate builds to bake for better performance.
 To do so, set COMPOSE_BAKE=true.
[+] Building 22.3s (23/23) FINISHED
r:desktop-linux
 => [backend internal] load build definition from Dockerfile
          0.0s
 => => transferring dockerfile: 337B
          0.0s
 => [backend internal] load metadata for docker.io/library/python:3.9-slim-buster
          2.4s
 => [backend auth] library/python:pull token for registry-1.docker.io
          0.0s
 => [backend internal] load .dockerignore
```

```
          0.0s
 => => naming to docker.io/library/mood_music_recommender-frontend
          0.0s
 => [frontend] resolving provenance for metadata file
          0.0s
[+] Running 2/3
 ✓ backend                                        Built
          0.0s
[+] Running 5/5                                   Built
          0.0s
 ✓ backend                                         Built
          0.0s
 ✓ frontend                                        Built
          0.0s
 ✓ Network mood_music_recommender_app-network      Created
          0.1s
 ✓ Container mood_music_recommender-backend-1      Created
          0.1s
 ✓ Container mood_music_recommender-frontend-1     Created
```

This command will:

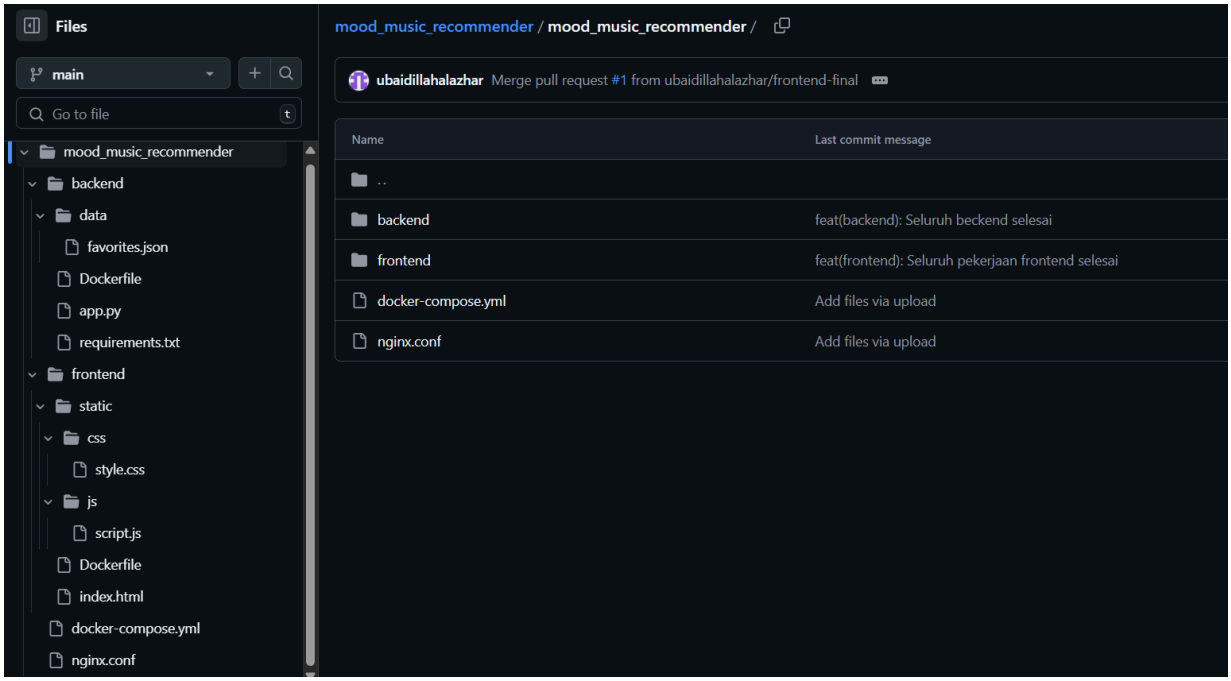
- Build the backend image using backend/Dockerfile.
- Build the frontend image using frontend/Dockerfile.
- Create and start both backend and frontend containers.
- Establish the app-network for inter-container communication.
- Map host port 3000 to the frontend container.

- **Access the Application:** Once the containers are running. The frontend (served by Nginx) will load, and API requests will be transparently proxied to the Flask backend.

VI. GITHUB REPOSITORY

All source code for the Mood Music Recommender application, along with this report, will be hosted on a GitHub repository.

https://github.com/ubaidillahalazhar/mood_music_recommender.git



VII. CONCLUSION

The Mood Music Recommender application has been successfully developed with a clear separation of frontend and backend concerns. The adoption of Docker and Docker Compose enables a robust, modular, and easily deployable architecture. The current setup, utilizing dummy data for music and ZenQuotes.io for quotes, provides a functional prototype for mood-based music recommendations with persistent favorites.

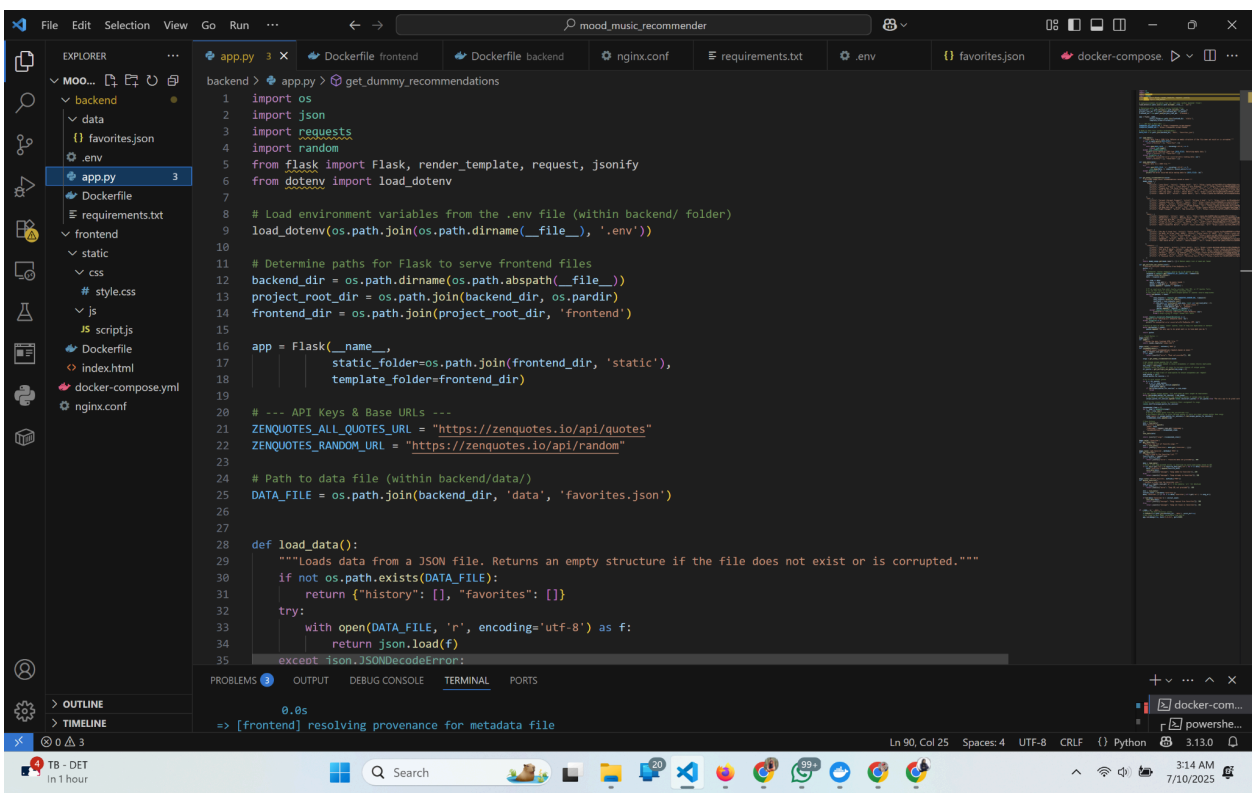
VIII. FUTURE ENHANCEMENTS

Potential future enhancements include:

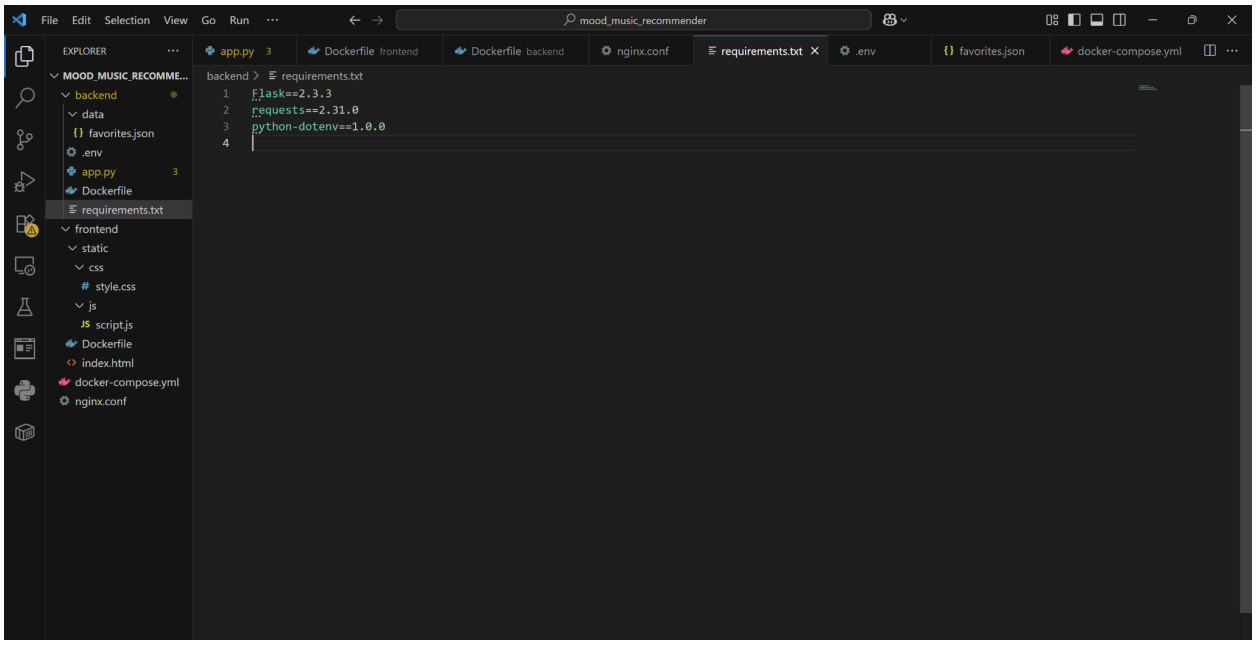
- Integration with a real music API (e.g., Spotify Web API) for dynamic and extensive music recommendations.
- Implementation of user authentication and personalized profiles.
- More advanced UI/UX features and animations.
- Deployment to a cloud platform for public access.

IX. ALL DOCUMENTATIONS

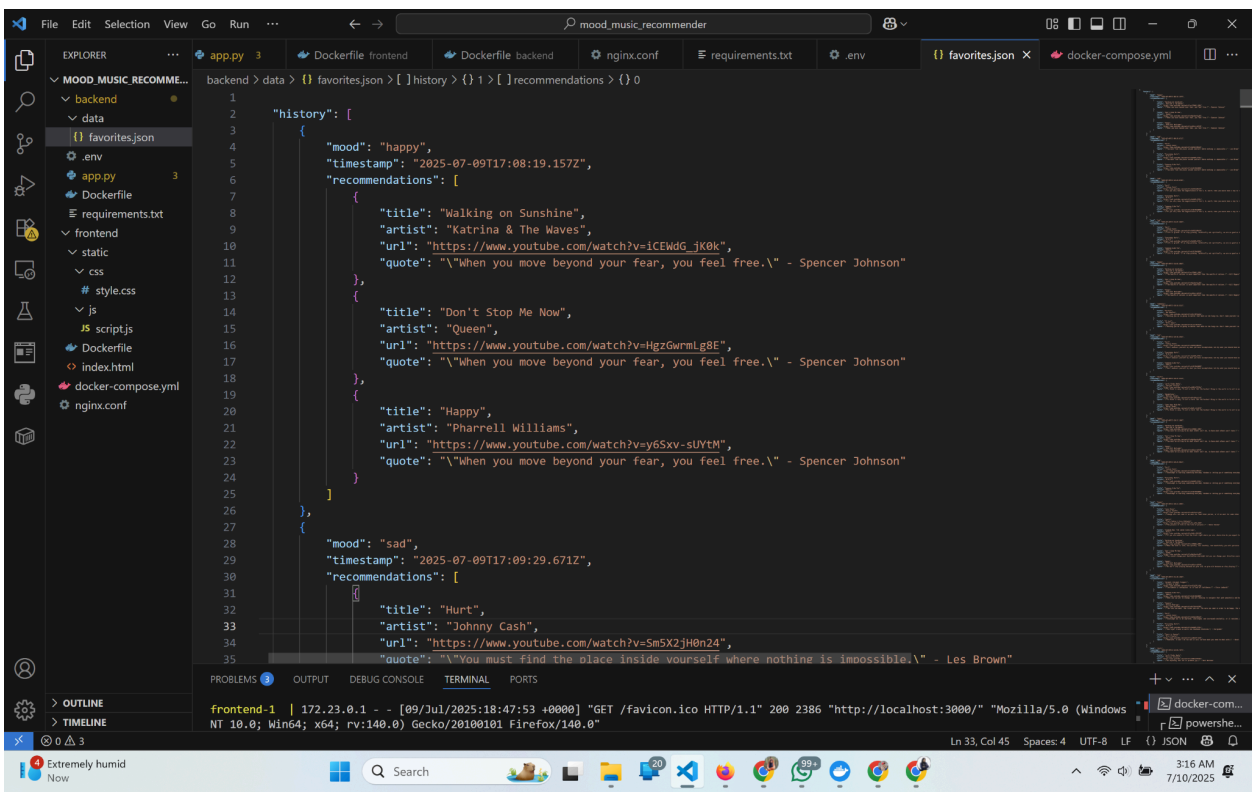
- backend(app.py)



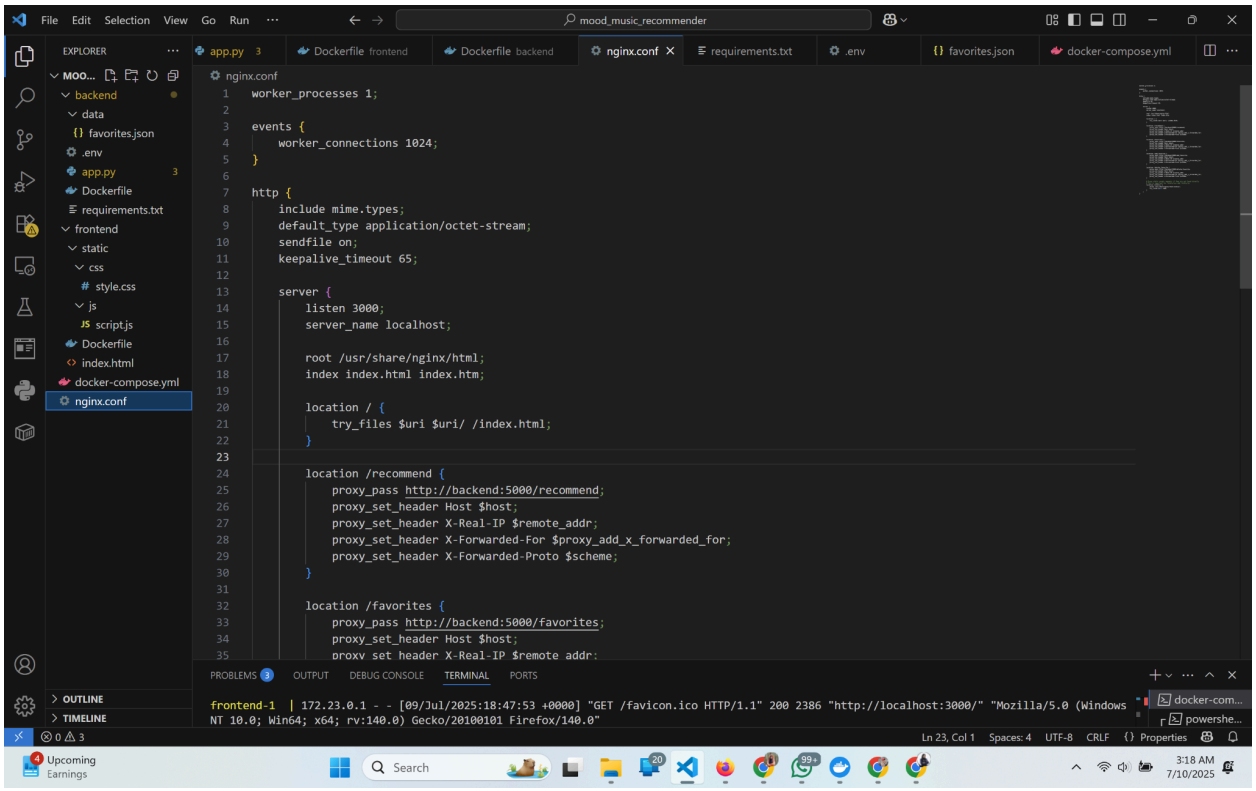
- requirements.txt



- backend/data/favorites.json

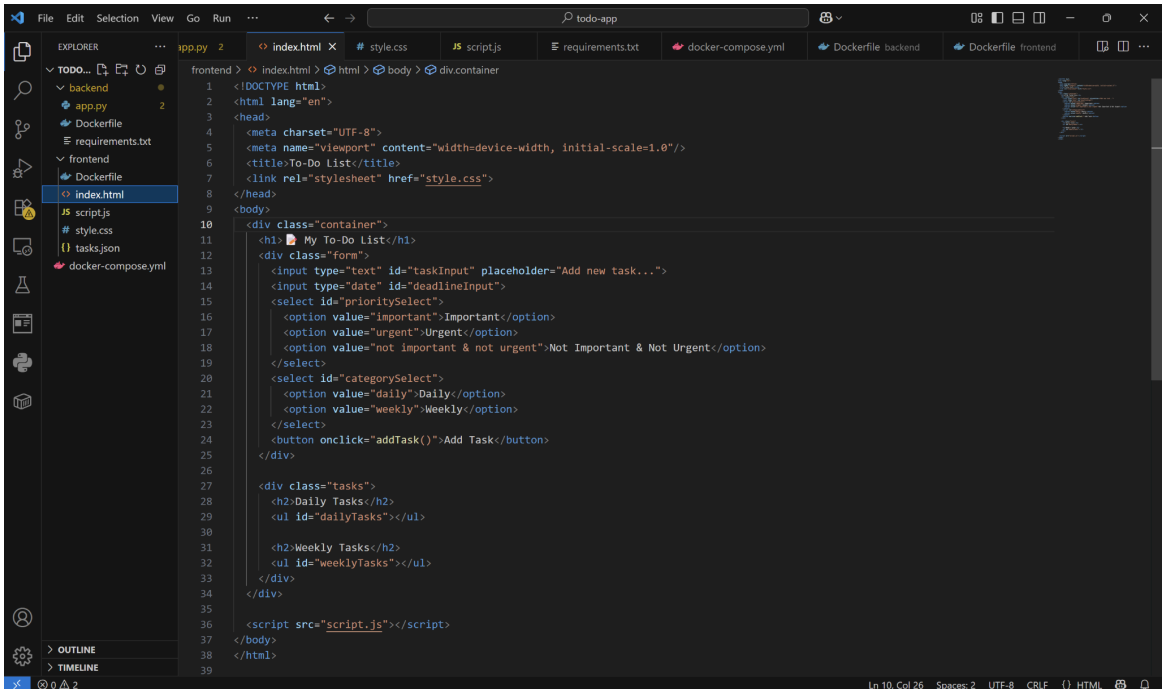


- nginx.conf



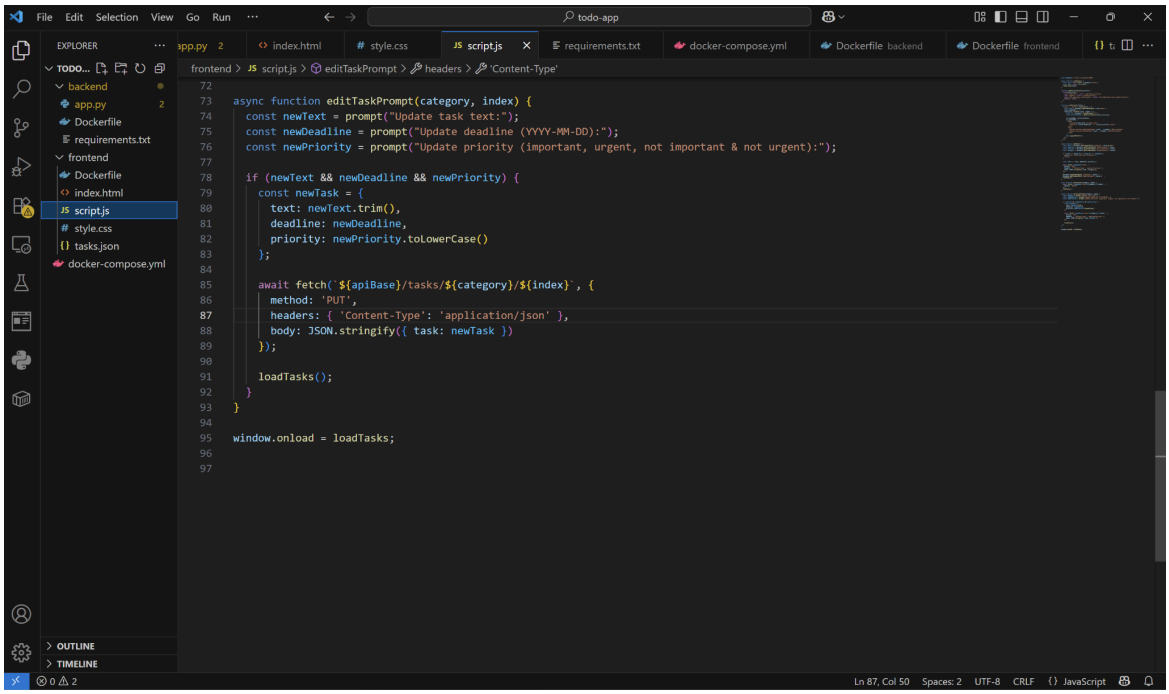
```
worker_processes 1;

events {
    worker_connections 1024;
}

http {
    include mime.types;
    default_type application/octet-stream;
    sendfile on;
    keepalive_timeout 65;

    server {
        listen 3000;
        server_name localhost;

        root /usr/share/nginx/html;
        index index.html index.htm;

        location / {
            try_files $uri $uri/ /index.html;
        }

        location /recommend {
            proxy_pass http://backend:5000/recommend;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
        }

        location /favorites {
            proxy_pass http://backend:5000/favorites;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
```

- Code from index.html (frontend)

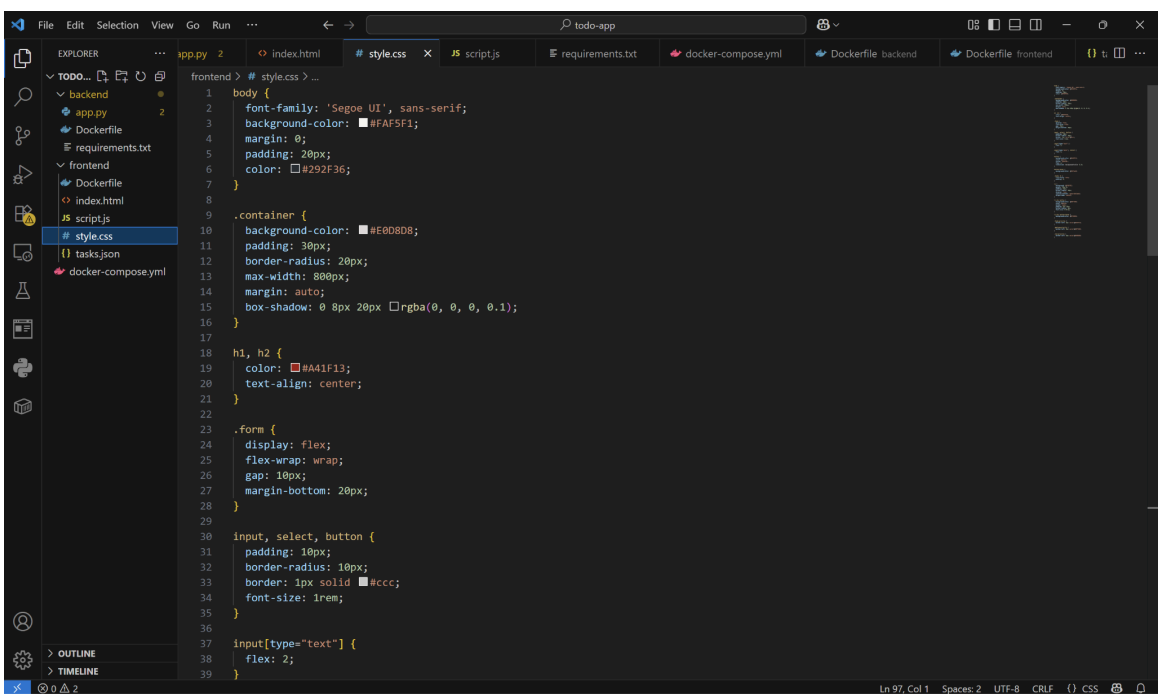
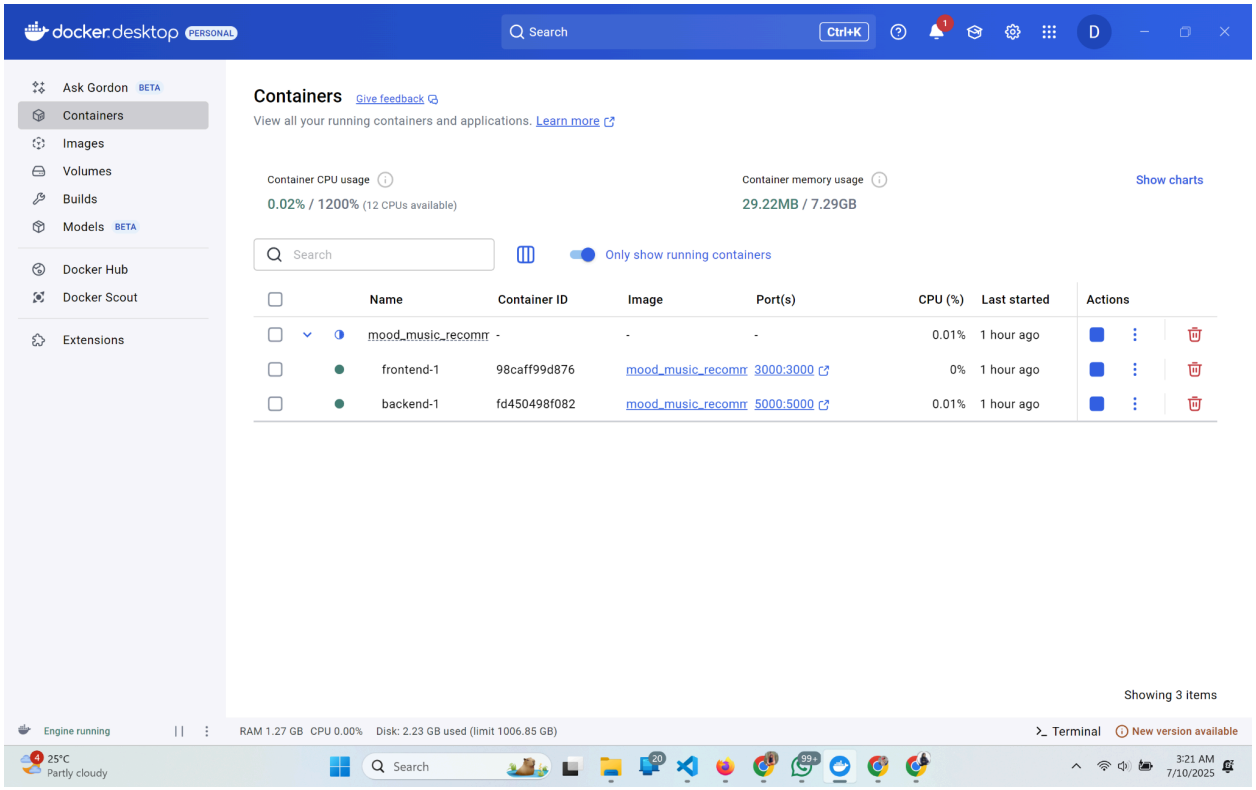


```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
  <title>To-Do List</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <div class="container">
    <h1>📝 My To-Do List</h1>
    <div class="form">
      <input type="text" id="taskInput" placeholder="Add new task...">
      <input type="date" id="deadlineInput">
      <select id="prioritySelect">
        <option value="important">Important</option>
        <option value="urgent">Urgent</option>
        <option value="not important & not urgent">Not Important & Not Urgent</option>
      </select>
      <select id="categorySelect">
        <option value="daily">Daily</option>
        <option value="weekly">Weekly</option>
      </select>
      <button onclick="addTask()">Add Task</button>
    </div>

    <div class="tasks">
      <h2>Daily Tasks</h2>
      <ul id="dailyTasks"></ul>

      <h2>Weekly Tasks</h2>
      <ul id="weeklyTasks"></ul>
    </div>
  </div>

  <script src="script.js"></script>
</body>
</html>
```

- Code from script.js (frontend)



- Code from style.css (frontend)



- Docker Containers