Team CodeBlooded

Daniel Park, Aditya Anand, Jackie Zeng,  Jayden Zhang

SoftDev

P04

TARGET SHIP DATE: {2025-04-25}

# Description

Create an interactive Flask web application that helps users explore and visualize Spotify song trends from 2000–2019 using data analysis and rich, dynamic visualizations. Using D3.js, we will craft interactive data visualizations that reveal relationships between musical features like danceability, energy, popularity, tempo, and more.

# Program Components

## 1. Frontend (Tailwind + D3.js + HTML templates)

- home.html: Landing page with summary stats, intro, and nav.
- explore.html: User can filter data by artist, year, genre, etc.
- visualize.html: D3.js-powered:
    - Correlation matrix
    - Bar charts (e.g., average energy by year or genre)
    - Scatter plot of danceability vs. popularity
    - Interactive time series
- compare.html: Compare two songs, artists, or genres.
- about.html: Info page and links to dataset/API used.

## 2. Middleware (Flask)

- __init__.py: App entrypoint, Flask routing.
- app.py: Routing logic.
- d3_utils.py: Serve data as JSON
- plot_utils.py: Functions to return graphs (plotly/seaborn).
- filter_utils.py: Functions to filter dataset based on user input.
- spotify_data.csv: Preprocessed version of songs_normalize.csv file from dataset.
- templates/: HTML Jinja templates
- static/js/: D3.js scripts

## 3. Backend (SQLite)

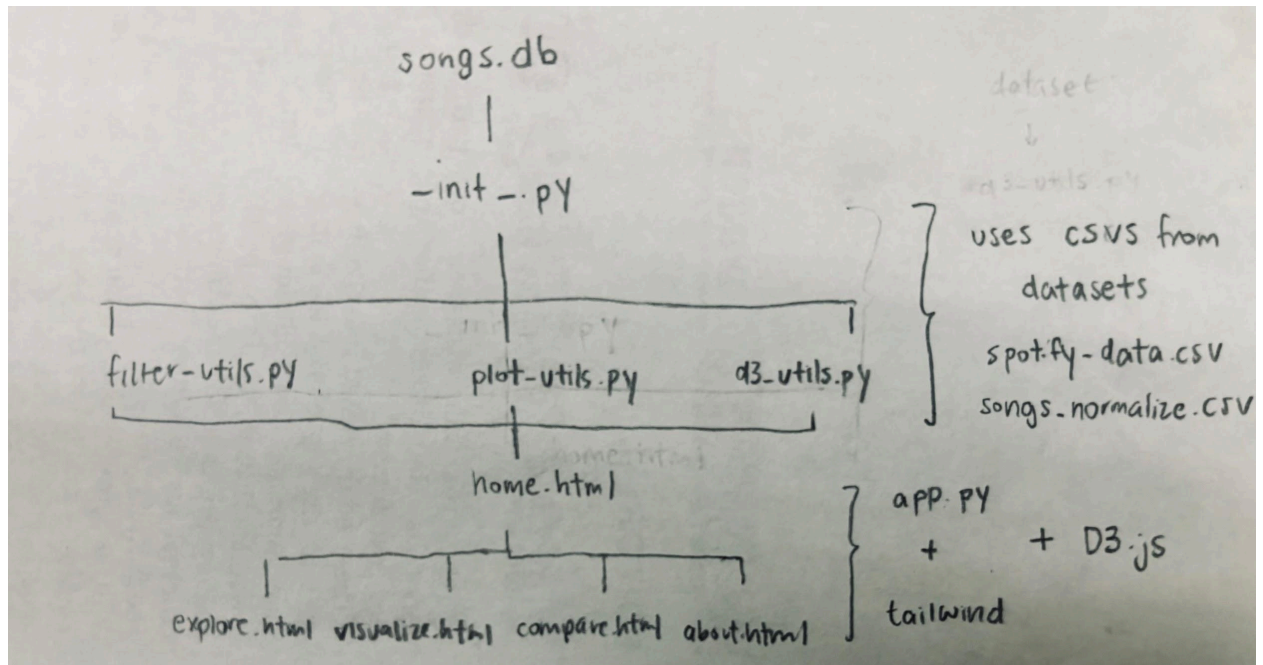- One central table (songs) with all tracks and metadata.

**Dataset**:

- Spotify Dataset:
- Top Hits Spotify from 2000-2019
- Spotify Data Visualization | Kaggle
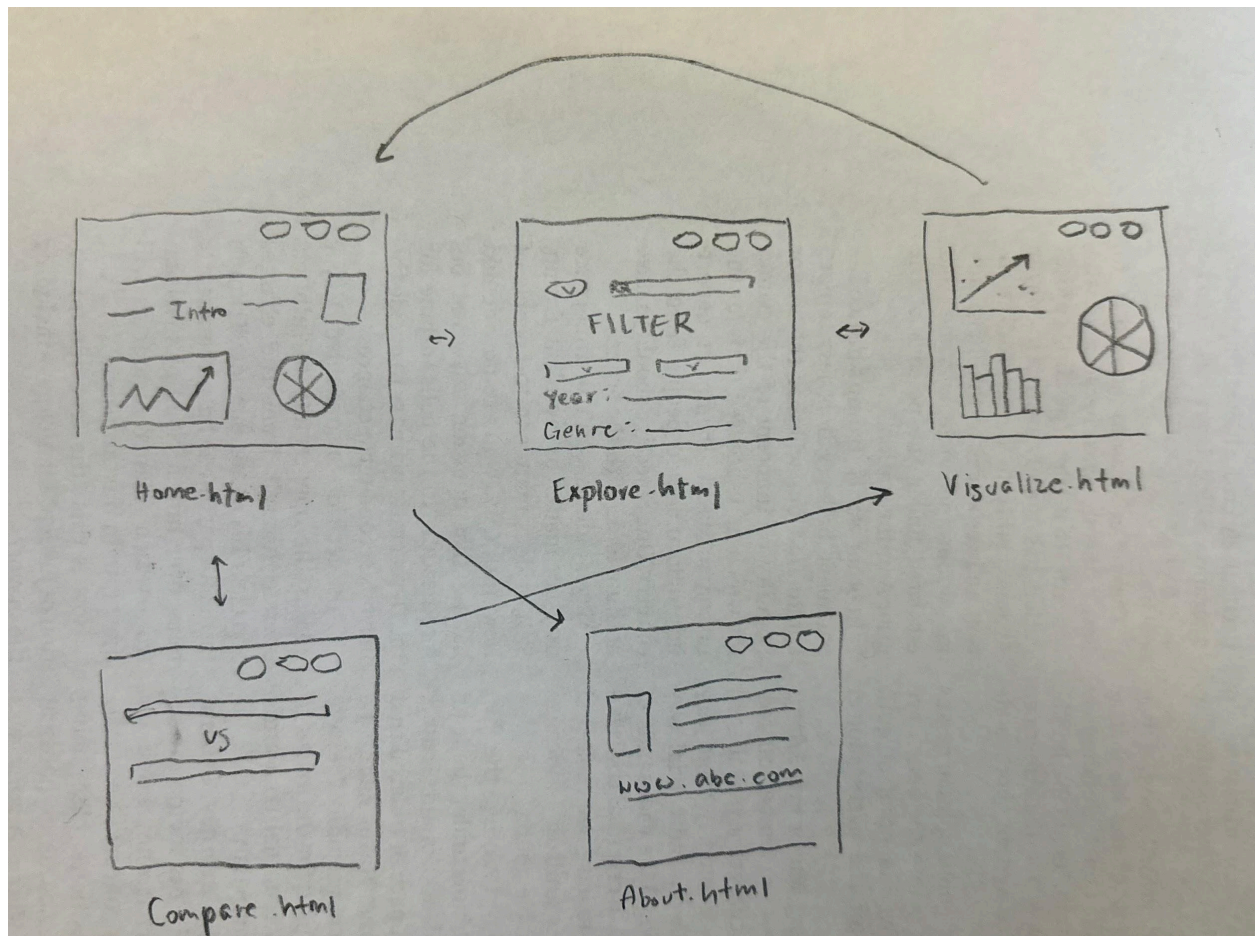- Data Cleaning Algorithm (PANDAS)

# Database Schema

|  | songs.db |  |
|---|---|---|
| artist | TEXT | Song artist |
| song | TEXT | Song title |
| year | INTEGER | Year released |
| popularity | FLOAT | 0–100 |
| danceability | FLOAT | 0–1 |
| energy | FLOAT | 0–1 |
| speechiness | FLOAT | Song artist |
| acousticness | FLOAT | 0–1 |
| instrumentalness | FLOAT | 0–1 |
| liveness | FLOAT | 0–1 |
| tempo | FLOAT | BPM |
| genre | FLOAT | Genre label |
| valence | FLOAT | 0–1 (happy/sad scale) |

# Component Relationships

- Frontend (Tailwind + D3.js) communicates with Flask middleware, which serves data from the SQLite backend.
- Middleware processes user input and queries the database.
- D3.js visualizes processed data dynamically on the frontend.

songs.db
|
__init__.py
|
filter-utils.py    plot-utils.py    d3_utils.py

uses csvs from datasets
spotify-data.csv
songs-normalize.csv

home.html
|
explore.html  visualize.html  compare.html  about.html

app.py
+
+ D3.js
tailwind

## Site Map



Intro

Home.html

FILTER
Year:
Genre:

Explore.html

Visualize.html

vs

Compare.html

www.abc.com

About.html

## Why Tailwind?

- Lightweight and utility-first, enabling faster styling.
- Ensures responsive and consistent UI across all pages.
- Layout (grid, flexbox), responsive breakpoints
- Hover states, transitions
- Consistent navigation bar and visual identity

## Why D3?

- Cause its #gudfam 🗣️
- Very customizable
- Easy-to-visualize graphs for time-stamped data like music

## APIS (stretch goal)

Spotify API: It will let users play samples of the music they are filtering and searching for.

## Task Assignments

| TASK | Daniel Park | Aditya Anand | Jackie Zeng | Jayden Zheng |
|---|---|---|---|---|
| Set up Flask and SQLite3 environment | | X | | X |
| Build User Authentication and Management Backend | | X | X | |
| Build D3 Component | | | X | X |
| Build Database | X | | X | |
| Frontend (HTML Templates) | X | | | X |
| Frontend (Tailwind) | X | | | X |

| Final Testing and Bug Fixing | X | X | X | X |
|------------------------------|---|---|---|---|