SOFTWARE PROJECT FINAL REPORT

Wenzhe Lin, Matthew Lam

December 5, 2024

Music playlist generator

TABLE OF CONTENTS

List of Figures/tables

Matthew Lam
Wenzhe Lin

1. Introduction

1.1. Purpose and Scope

This project aims to create a web-based music playlist generator that allows users to generate personalized playlists based on mood or activities. This project integrates the Spotify API to provide seamless music selection based on user preferences.

1.2. Product Overview

- **Capabilities**:
    - Generate 50 song playlists based on mood, activity, or keywords.
    - Real-time search functionality for music.
    - Responsive UI for seamless user experience.
    - Ability to listen to recommended songs directly on the Spotify platform.
    - Simplistic, modern, easy-to-navigate user interface
- **Scenarios for using the product**:
    - A user selects “Happy” as their mood and receives a curated playlist.
    - A user inputs "Workout" and generates a playlist suitable for exercise routines.
    - A user inputs “I don’t want to be sad anymore” and generates a playlist excluding sad/slow songs.
    - A user interested in a recommended song can listen directly to Spotify’s platform.

1.3. Terms, Acronyms, and Abbreviations

- **API**: Application Programming Interface
- **UI**: User Interface
- **UX**: User Experience
- **Spotify API**: The official API for Spotify music integration
- **Spotipy:** Python library with access to Spotify API
- **Token:** A snippet of string in which access Spotify API

1.4. Scoping

Functional scoping

- Interactive search bar: accept user input and determine if its a valid input or not. If the user input is valid, return a 50-song playlist.
- Viewable generated playlist: if the user input is valid for recommendation generation, return a 50-song playlist curated for the user's needs. If there is an error with generating the playlist then notify the user that an error has occurred.

- Generate button: once the user is done with inputting mood/activity, allow them to begin the generation function.

Technical scoping

- Song category/genre: each genre of recommended songs has set values such as “high energy = 0.8” which is used for retrieving tracks.
- Spotify API integration: The backend of the project must be able to properly communicate with the Spotify API to pull song data, listen on its platform, and convert Spotify IDs.

2. Project Management Plan

2.1. Project Organization

Role organization:

- Backend Development: Wenzhe Lin
- Frontend Development: Matthew
- API Integration: Wenzhe Lin

2.2. Lifecycle Model Used

The project followed the **Waterfall Model**, progressing from requirements to testing in sequential stages.

2.3. Risk Analysis

- **API limitations**: Resolved through error handling and thorough documentation.
- **UI responsiveness issues**: Addressed during testing phases.

Risk table

Risk Index	Risk	Probability	Impact/cause	Mitigation
1	Crash program due to massive data overload	High	This leads to program crashes, data leaks/breaches	Implement proper data handling to ensure proper storage of information
2	Improper API integration	Mid	Failure to retrieve song data	Make sure our information analyzer and usage of web API is correct

3	Performance issues	Low	Poor resources resulting in slow/failure program performance	Efficient sorting and searching algorithms to prevent our code from taking up too much time and space
4	Budget	Low	While most of the tools we have are free, there may be budget issues	Make sure we are using the important tools only to reduce cost as much as we can
5	Failure to meet customer requirements	Mid	Failure to meet what the user wanted from the software	In general, we need to make sure no corners are cut to make sure the consumer is receiving the product they want

2.4. Hardware and Software Resource Requirements

- Hardware: WindowOS/MacOS computer running Windows 10/11 or Mac Big Sur+
- Software: VS Code, Node.js, Git, Spotify API, and a live server for deployment

2.5. Deliverables and Schedule

- **Deliverables**: A functional playlist generator website, API integration, and documentation. A quick presentation with an overview and introduction to the project.
- **Schedule**:
    - Week 1: Requirement Analysis, identify potential initial risks, scoping.
    - Week 2-3: Designing the website
    - Week 4-6: Backend and API Integration, small touch-ups with the front end of the website
    - Week 7: Testing and Deployment/ debugging any errors or fixed parts that did not work

---

3. Requirement Specifications

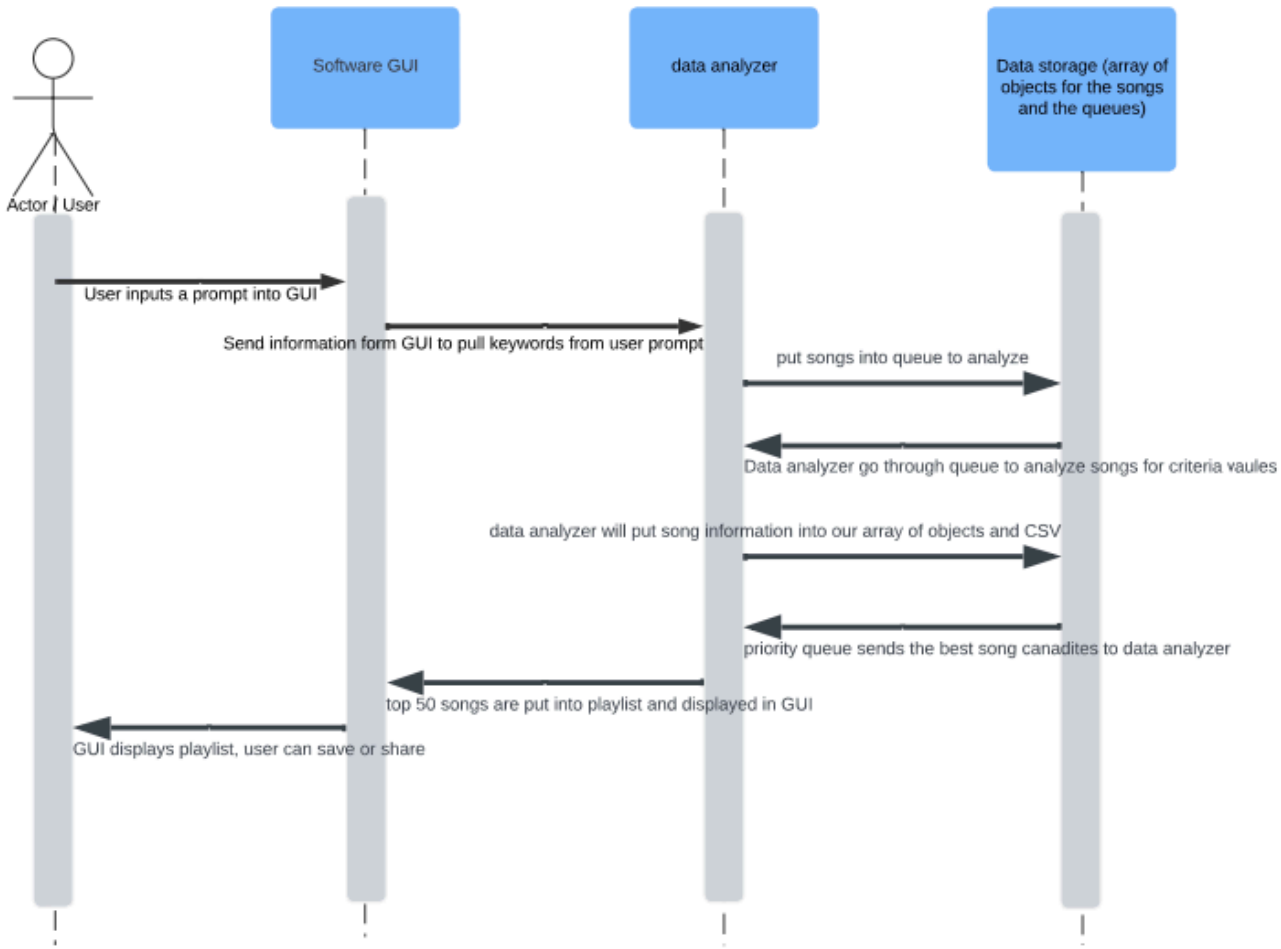
3.1. Stakeholders for the system

- Users seeking personalized music playlists
- Developers integrating music-related APIs
- Music enthusiasts
- Users who are first introduced to music-listening

3.2. Use Cases

1. **Generate Playlist by Mood**: User inputs a mood such as "energetic" or "relaxed," and the website generates a matching playlist.
    - Actors: User, Analyzers
    - Data: Spotify songs from API, Scaler, Analyzer, Playlist
    - Stimulus: User will write a prompt for the software to generate a playlist by mood
    - Response: Generated playlist
2. **Generate Playlist by Activity**: User provides an activity prompt like "running" or "studying," and the website generates a suitable playlist.
    - Actors: User, Analyzers, playlist
    - Data: Spotify songs from API, Scaler, Analyzer, Playlist
    - Stimulus: User will write a prompt for the software to generate a playlist by activity
    - Response: Generated playlist
3. **Listen to the song on Spotify**: The user is interested in listening to a song recommendation on Spotify's platform.
    - Actors: User, Spotify
    - Data: Spotify metadata with the song file for the user to play.
    - Stimulus: The user presses the "listen on Spotify" button under the recommended song to listen on Spotify's platform.
    - Response: take the user to Spotify and play the song they want to listen to.

3.2.1. Graphic Use Case Model



3.3. Rationale for your Use Case Model

Dynamic model best demonstrates the usage of the software. The user interacts with the GUI to input a prompt and from there, our backend(data analyzer, Spotify API, Songs storage) will determine which songs are best fir to be outputted to the user. The graphical model best demonstrates the flow of our intended use cases.

3.4. Non-Functional Requirements

- The system must load playlists within 3 seconds.
- The UI should be responsive and user-friendly on supported devices(windows/mac computers)
- The reliability of the program must be trusted to properly do its job such as generate good recomendations. This includes its security to ensure user privacy.

- Logging and debugging prints to ensure the ability to troubleshoot and monitor errors

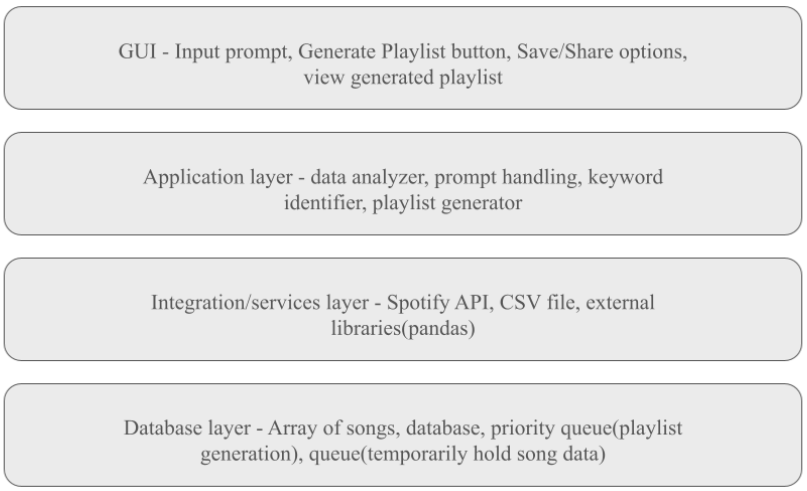
---

4. Architecture

4.1. Architectural Style(s) Used

The architecture follows a client-server model, client: The user interface (UI) is web-based, allowing users to input keywords and view their generated playlist. Server: The backend will process user inputs, perform keyword matching, and fetch corresponding songs from the database.

4.2. Architectural Model



4.3. Technology, Software, and Hardware Used

- **Frontend**: HTML, CSS, JavaScript
- **Backend**: Node.js, Python

- **API Integration**: Spotify API, Spotipy
- **Database**: JSON (for local storage of preferences)

4.4. Rationale for your Architectural Style and Model

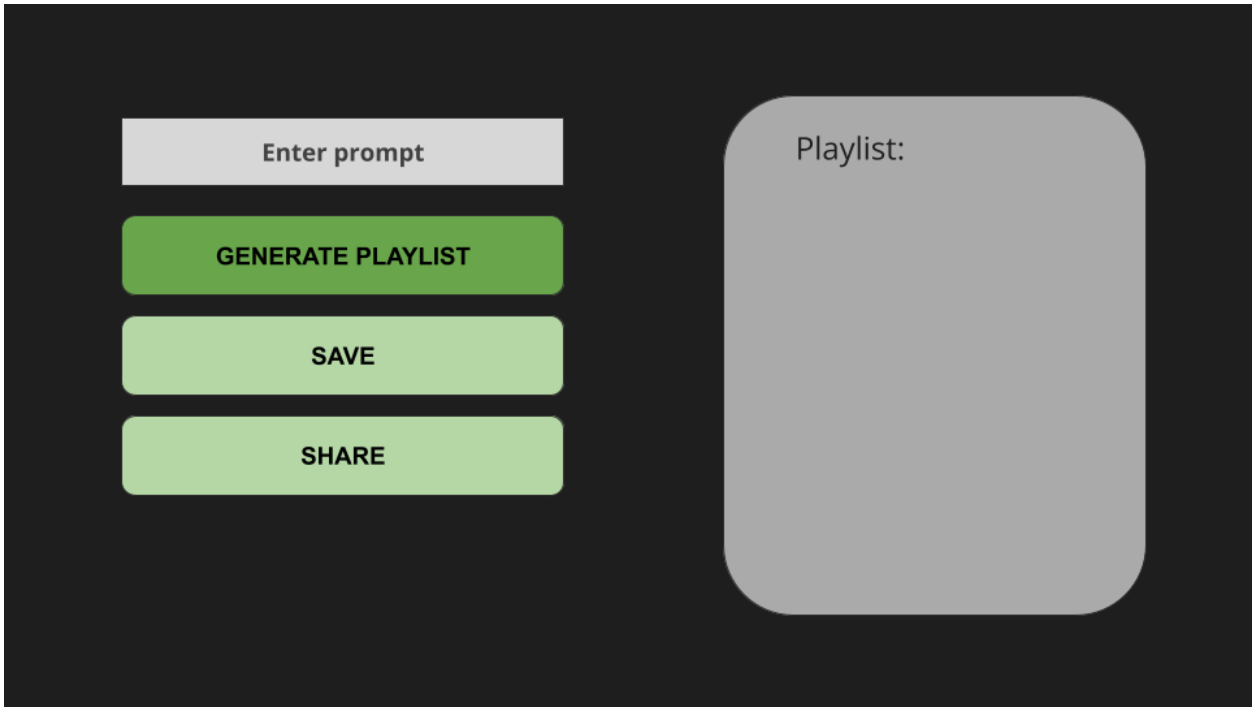
The Client-server architecture provides separation of frontend, backend/server, and API integration function. It demonstrates how we plan to design our project while describing each layer on its role within our system.

5. Design

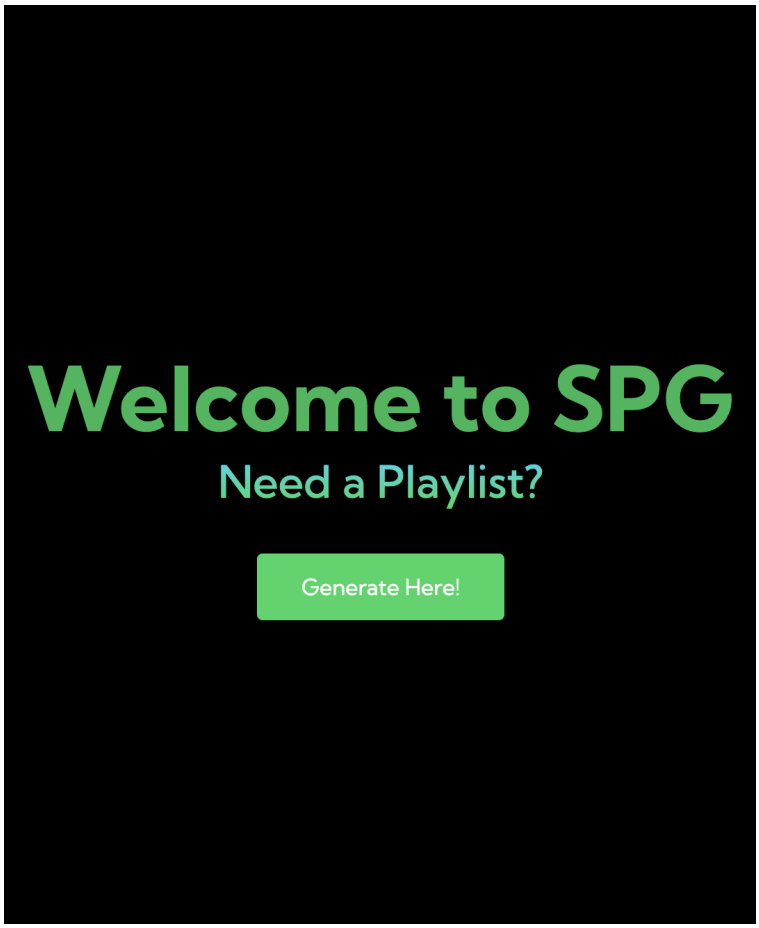
5.1. User Interface Design

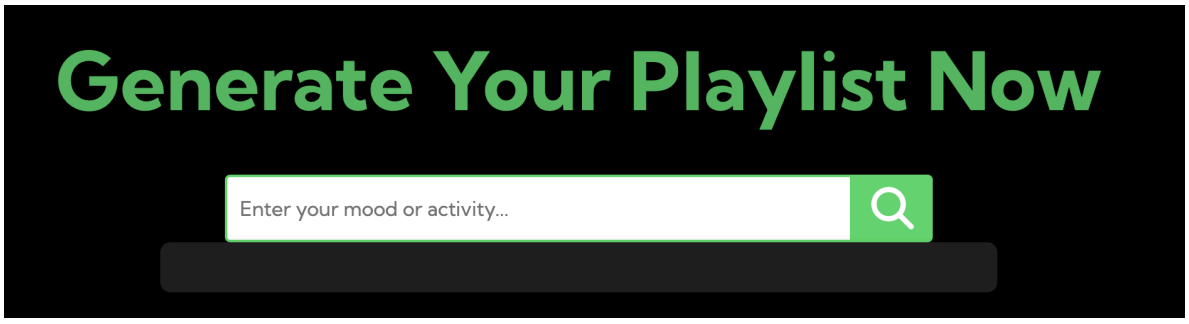
- **Home Page**: Includes a search bar and buttons for mood selection.
- **Playlist Page**: Displays a generated playlist with options to play, shuffle, or save.
- **Navigation and interactive buttons**: “enter prompt”, “generate”, “save”, “share”, “play”

5.1.1 initials UI design



5.1.2 final UI design





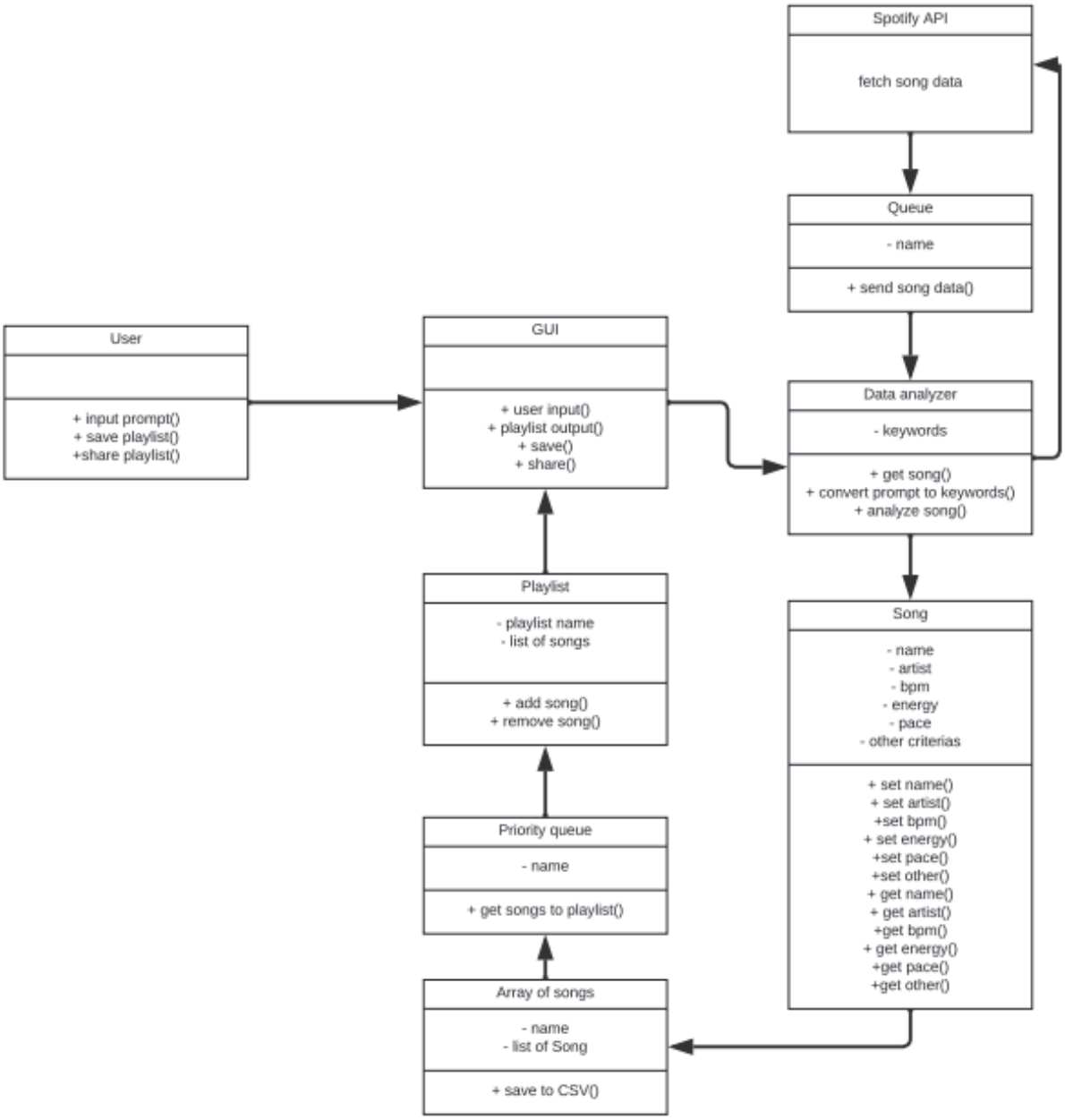
*Some other interactive buttons only once the user presses generate

This design is much more simplistic and easy to use, while maintaining all out needed functions.

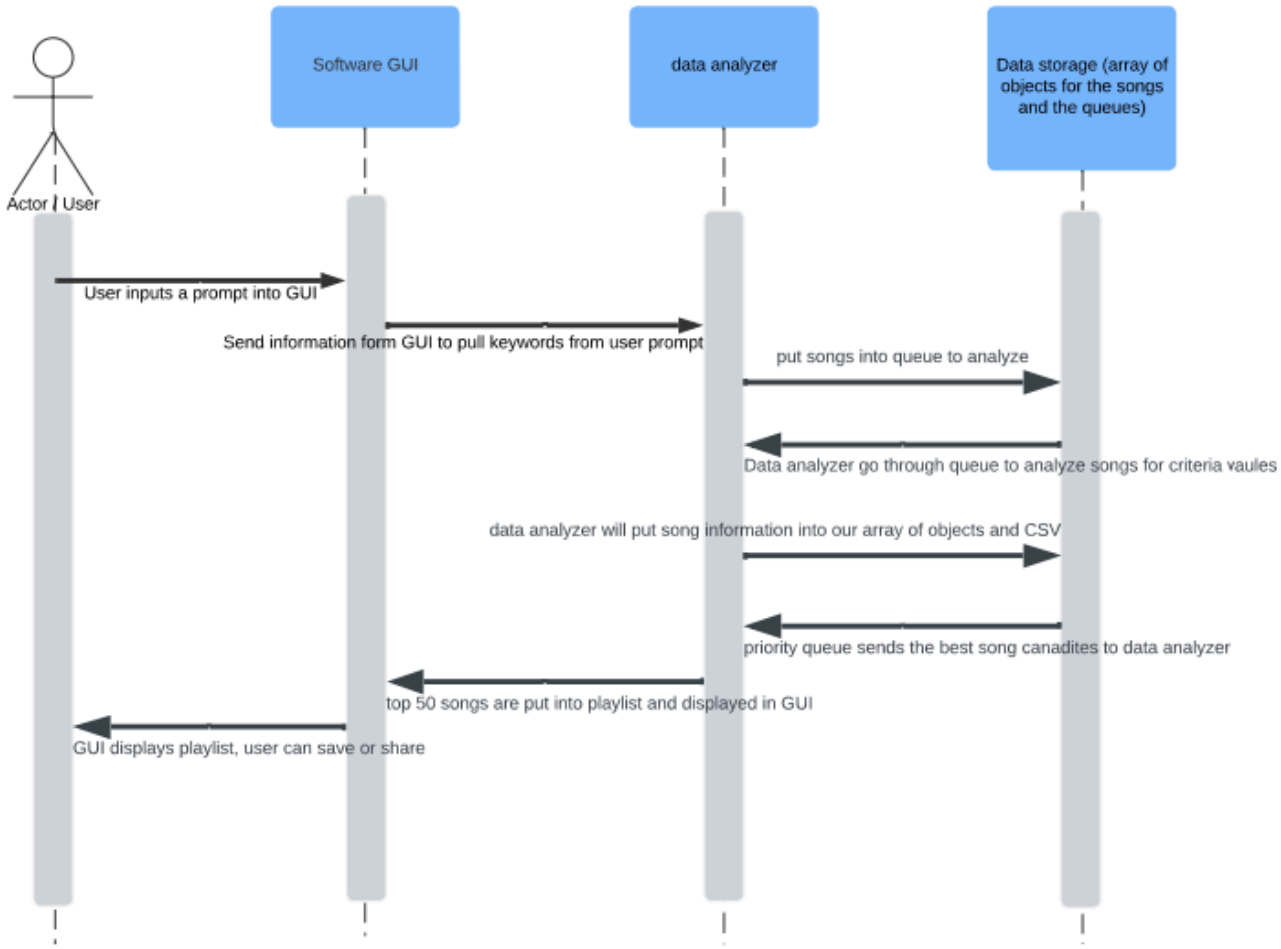
5.2. Components Design

- **Static Models**: UI components such as search bar and buttons.
- **Dynamic Models**: API request-response handling and playlist rendering.

5.2.1 static model



5.2.2 dynamic model



5.3. Database Design

- **Structure**:
    - User input data (mood, activity, or keywords)
    - Generated playlists (song IDs, names, artists)

      Stores information with JSON, temporary hold information.

      (No large database is needed for this project)

5.4. Rationale for your Detailed Design Models

The design ensures intuitive navigation and efficient playlist generation. These models best demonstrate what our intended design looks like, in terms of the layer of architecture, the functionality of each component, and how the user should interact with our software.

5.5. Traceability from Requirements to Detailed Design Models

Each requirement maps directly to a design element (e.g., the mood selection maps to the button component).

---

6. Test Management

6.1. A Complete List of System Test Cases

- **Test Case 1**: Search for the keyword “Workout”
- **Test Case 2**: Click “Generate playlist”
- **Test Case 3**: enter an API request and it returns errors
- **Test Case 4**: The search input is empty
- **Test Case 5**: The user input contains incoherent words, ex: “asdfg”
- **Test Case 6**: User inputs contradicting keywords, ex: “happy”+“sad”
- **Test Case 7**: Navigational buttons taking us to expected pages (ex: home goes to homepage)
- **Test Case 8**: 50-song playlist generates
- **Test Case 9**: Request sent to generate a song playlist

6.2. Traceability of Test Cases to Use Cases

- **Test Case 1**: A playlist containing high-energy songs suitable for a workout. All songs should have at least 120 bpm.
- **Test Case 2**: A playlist with 50 songs is generated and a “Save to Spotify” button
- **Test Case 3**: Error message displayed to the user. Prompt the user then try again later.
- **Test Case 4**: Warning message: "Please enter a mood or activity to generate a playlist"
- **Test Case 5**: Warning message: "Please enter a mood or activity to generate a playlist"
- **Test Case 6**: Warning message: "Input contradicted, may result in unexpected results" Prompt user if they want to proceed
- **Test Case 7**: If we click the home button then we expect to be on the home page, some go for other interactive buttons such
- **Test Case 8**: The expected output is to be able to view the generated playlist
- **Test Case 9**: The back end handles reading song data from Spotify API. print onto CSV to verify the successful reading

6.3. Test Results and Assessments

- **Test Case 1**: Succesful
- **Test Case 2**: Succesful
- **Test Case 3**: Succesful
- **Test Case 4**: Succesful
- **Test Case 5**: Succesful
- **Test Case 6**: Succesful
- **Test Case 7**: Succesful
- **Test Case 8**: Succesful
- **Test Case 9**: Succesful

6.5. Defects Reports

No critical defects were found; minor UI alignment issues were fixed.

---

7. Conclusions

7.1. Outcomes of the Project

- A functional playlist generator website with Spotify API integration.
- User feedback showed high satisfaction with the mood-based playlist feature.

7.2. Lessons Learned

- The importance of error handling when working with APIs.
- Effective project management using the Waterfall Model.
- Better time management
- adaptability to Spotify API updates (or other services in the future)

7.3. Future Development

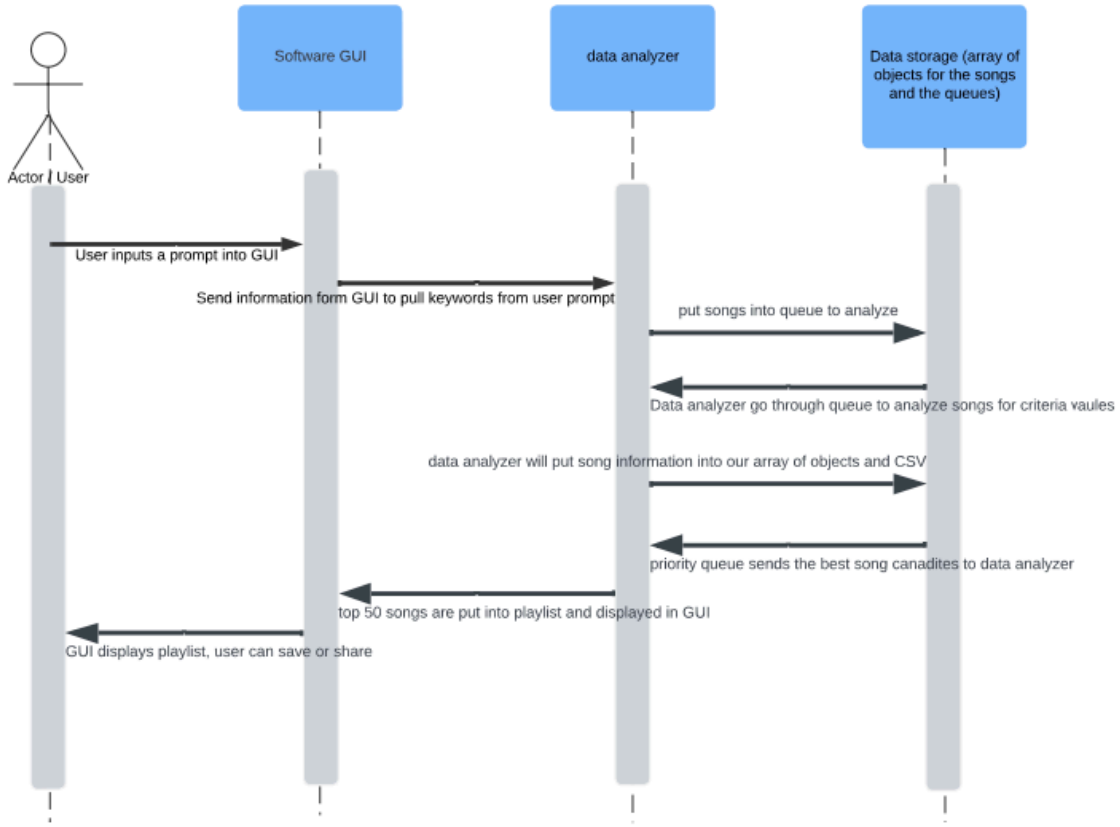
- Add user accounts for saving playlists.
- Combine all the 50 generated songs into a playlist so users can click one button that takes them to Spotify with that playlist, instead of clicking each song individually
- Expand to support other music APIs (e.g., Apple Music).
- Create another tab called playlists that holds the previously generated playlists so users can access

Appendices

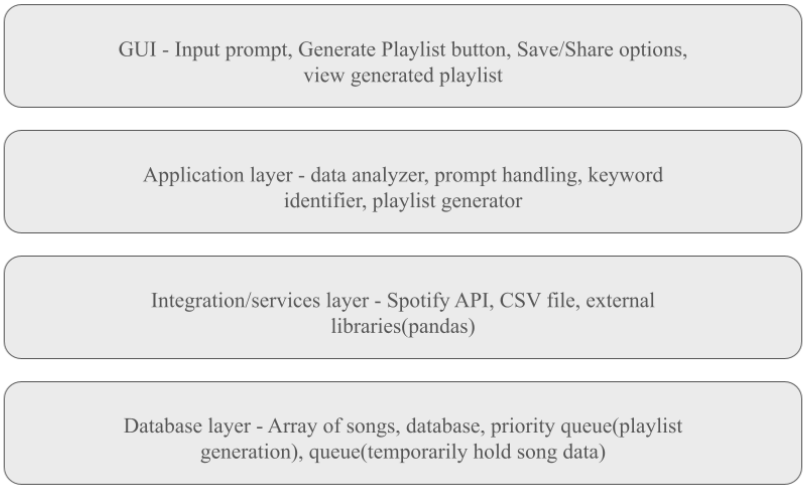
Risk table 2.3

Risk Index	Risk	Probability	Impact/cause	Mitigation
1	Crash program due to massive data overload	High	This leads to program crashes, data leaks/breaches	Implement proper data handling to ensure proper storage of information
2	Improper API integration	Mid	Failure to retrieve song data	Make sure our information analyzer and usage of web API is correct
3	Performance issues	Low	Poor resources resulting in slow/failure program performance	Efficient sorting and searching algorithms to prevent our code from taking up too much time and space
4	Budget	Low	While most of the tools we have are free, there may be budget issues	Make sure we are using the important tools only to reduce cost as much as we can
5	Failure to meet customer requirements	Mid	Failure to meet what the user wanted from the software	In general, we need to make sure no corners are cut to make sure the consumer is receiving the product they want

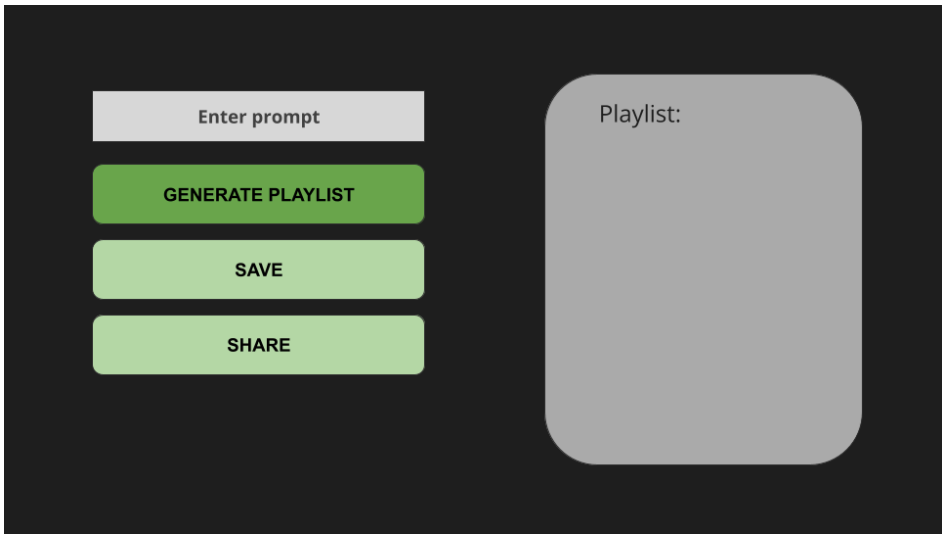
Dynamic model 3.2.1



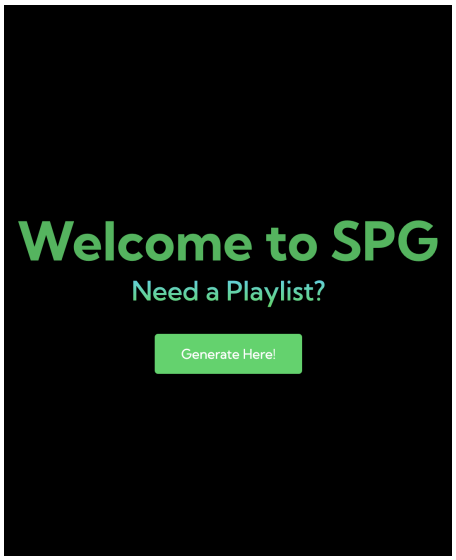
Architectural model 4.1

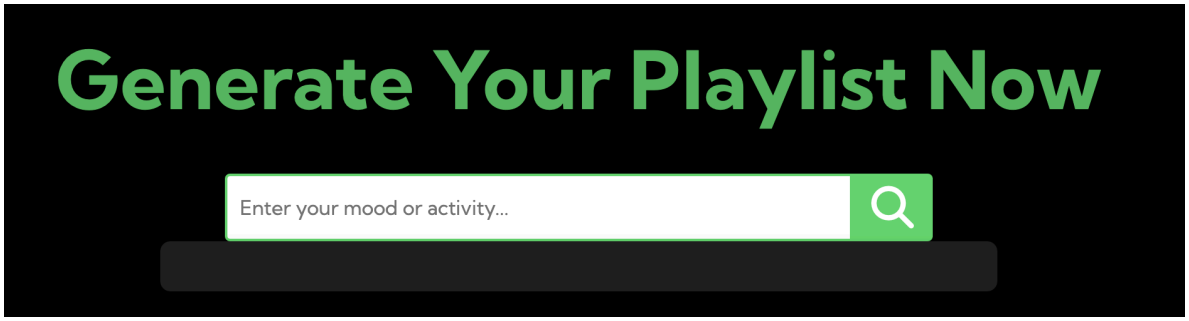


initials UI design 5.1.1

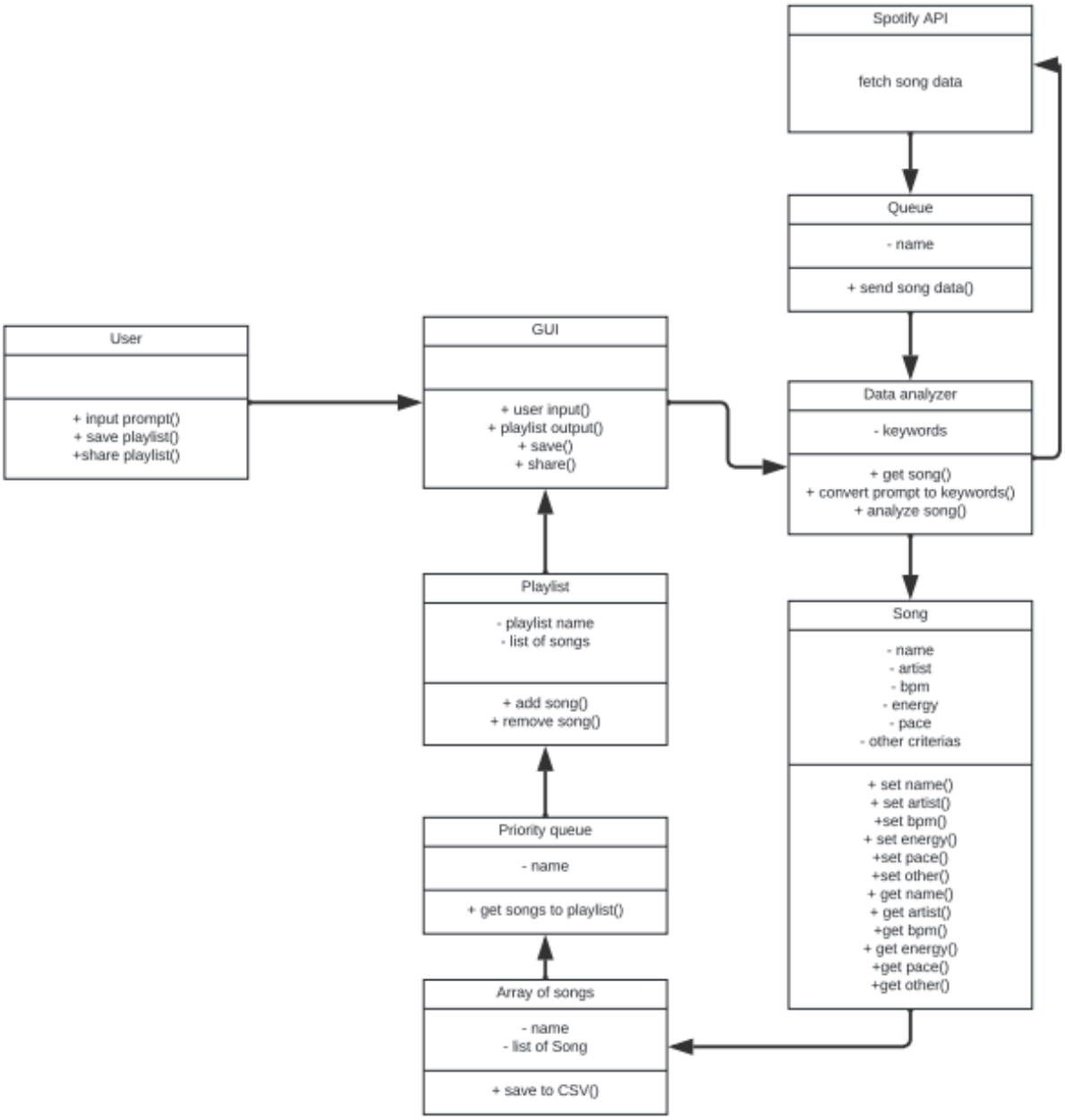


Final UI design 5.1.2





Static model 5.2.1



Test case table

ID	Test Inputs	Expected output	Description
TC-001	Search for the keyword “Workout”	A playlist containing high-energy songs suitable for a	Test the playlist generator for activity-based input.

		workout. All songs should have at least 120 bpm.	
TC-002	Click “Generate playlist”	A playlist with 50 songs is generated and a “Save to Spotify” button	Verify mood-based playlist generation.
TC-003	API returns errors	Error message displayed to the user. Prompt the user then try again later.	Test error handling for failed Spotify API responses. Notify the user there has been an unexpected error at the moment.
TC-004	The search input is empty	Warning message: "Please enter a mood or activity to generate a playlist"	Verify input validation for the search bar.
TC-005	The user input contains incoherent words, ex: “asdfg”	Warning message: "Please enter a mood or activity to generate a playlist"	Test to properly handle unexpected inputs
TC-006	User inputs contradicting keywords, ex: “happy”+“sad”	Warning message: "Input contradicted, may result in unexpected results" Prompt user if they want to proceed anyways	In cases where the user inputs contradicting keywords, to prevent unexpected results, display a warning and prompt the user if they want to continue anyway
TC-101	Navigational buttons taking us to expected pages	If we click the home button then we expect to be on the home page, some go for other interactive buttons such	Make sure our front end takes us to the expected page when the user interacts with the navigational buttons
TC-102	50-song playlist generates	The expected output is to be able to view the generated playlist	Make sure our front end properly connects with the backend to generate the viewable

			50-song playlist
TC-201	Request sent to generate a song playlist	The back end handles reading song data from Spotify API. print onto CSV to verify the successful reading	To ensure our back end is properly communicating with the API, we should log the song and its data once we process it.

Risk table

Risk Index	**Risk**	**Probability**	**Impact/cause**	**Mitigation**
1	Crash program due to massive data overload	High	This leads to program crashes, data leaks/breaches	Implement proper data handling to ensure proper storage of information
2	Improper API integration	Mid	Failure to retrieve song data	Make sure our information analyzer and usage of web API is correct
3	Performance issues	Low	Poor resources resulting in slow/failure program performance	Efficient sorting and searching algorithms to prevent our code from taking up too much time and space
4	Budget	Low	While most of the tools we have are free, there may be budget issues	Make sure we are using the important tools only to reduce cost as much as we can

5	Failure to meet customer requirements	Mid	Failure to meet what the user wanted from the software	In general, we need to make sure no corners are cut to make sure the consumer is receiving the product they want

Requirement traceability matrix

Requirements ID	Description	Components	Data Structures
1	The user inputs a keyword or mood	Song database component	Input data (keywords)
2	The system will then generate a playlist based on what the user inputs	Playlist generator component	Song data (mood or keywords) priority queue
3	Users will be able to view the generated playlist	User interface component	Playlist array
4	The system gets the songs to generate the playlist from the database	Song database component	Song record structure, array of songs (list of dictionary)
5	For now, the playlist will have a maximum of 50 songs	Playlist generator component	Playlist array
6	The system should be able to work with a larger database if we continue the work on this project and improve it.	Backend infrastructure	Priority queue, Array of songs.

7	The system should be able to handle invalid, or insufficient inputs	User input component	Input data