PES UNIVERSITY, BANGALORE

Department of Computer Science and Engineering

B-TECH (CSE)

V Semester

UE20CS303 – SOFTWARE ENGINEERING

PROJECT REPORT ON MUSIC RECOMMENDATION SYSTEM

SUBMITTED BY:

RENITA KURIAN – PES1UG20CS331

RICHA SHAHI – PES1UG20CS334

RIMJHIM SINGH – PES1UG20CS337

RIYA JHA – PES1UG20CS344

PES UNIVERSITY, BANGALORE

Department of Computer Science and Engineering

TABLE OF CONTENTS

PES UNIVERSITY, BANGALORE

Department of Computer Science and Engineering

PROPOSAL OF THE PROJECT

The project aims to create a Machine Learning model that recommends songs to the user based on the user's favorite tracks and genres. The interface is a user-friendly GUI that enables the user to enter her choices and get her personalized tracks. It also offers features such as selection on the basis of genres. The ML model is trained using Spotify dataset and also uses Spotify developer API to extract choicest tracks for the user.

PES UNIVERSITY, BANGALORE

Department of Computer Science and Engineering

SOFTWARE REQUIREMENT SPECIFICATION

1. INTRODUCTION

1.1 DOCUMENT PURPOSE

The product whose software requirements are specified in this document is MusicWreck.
The purpose of this document is to present a detailed description of the product, MusicWreck. This document is intended to

- ➢ Explain the purpose and features of the product, MusicWreck
- ➢ The constraints under which the product must operate
- ➢ How the product would respond to different users' requests?

The document's primary goal is to help the reader get a better understanding of the project. The document is intended for the developers of the software, the end users of the product who have been identified in the later sections, and to the professors who would review the project.

1.2 PRODUCT SCOPE

The software being developed is Machine Learning based Music Recommendation System. The product would recommend songs to the user based on her preferences of genres and songs she like. The product would –

- ➢ Recommend songs to the user based on genres.
- ➢ User can also select certain songs from the dropdown option to get a personalised playlist
- ➢ Would be provided with a Spotify playlist with all these songs

The points specified above would make a competent music recommendation system for the user according to her likelihood of musical tracks.

1.3 INTENTED AUDIENCE AND DOCUMENT OVERVIEW

1.3.1 Intended Audience:

This document is primarily intended for the:

- ➢ Developers of this software
- ➢ Software engineers who would work on further development of the project
- ➢ The professors who would review the document and finally,
- ➢ Clients that is novice or professional event managers, volunteers.

1.3.2 Document Overview:

The first chapter, that is the Introduction section of the document is intended to introduce the reader to the product, MusicWreck.

The second chapter, Overall Description section of SRS document provides an overview of the overall functionality of the product. It describes the informal requirements.

The third chapter, Specific Requirements section, of SRS document is written primarily for the developers and describes in technical terms the details of the functionality of the product.

The second and the third chapter of the document describe the same software product, but are intended for different audiences and thus use different language.

1.4 DEFINITIONS, ACRONYMS AND ABBREVIATIONS

1	Employer	Employer is an individual who has contacted the event organiser.
2	Event Manager	Event Manager is an individual who is responsible for the whole event and can view the entirety of the event being planned on the software. He/She is usually the lead event organizer.

3	HTTPS	HTTPS stands for Hypertext Transfer Protocol Secure. This protocol is a widely used communications protocol for secure communication over a computer network, with especially wide deployment on the Internet.
4	Spotify	Organisation that’s going to provide all the dataset and playlist interface for the user.
5	SRS	SRS stands for Software Requirement Specification. It is a document that completely describes all of the functions of a proposed system and the constraints under which it must operate.
6	Team Head	Team head is an individual who is responsible for all the actions undergoing under his/her team.
7	UI	UI stands for User Interface. It is defined as the space where interaction between humans and machines occurs.
8	View	View means to display and look at data on screen.
9	ML model	The Machine Learning Model being used for the recommendation system.
10	Dataset	The dataset being used for building the recommendation system taken kaggle.

11	API	Application Programming Interface to use HTTP requests to use Spotify Developers Features.

1.5 References and Acknowledgments

1.5.1 References:

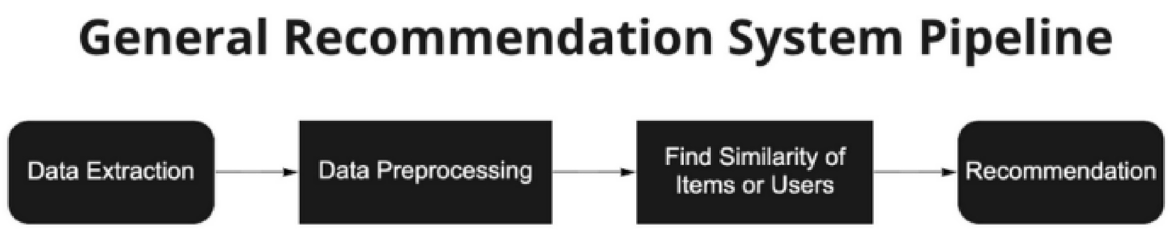
- ➢ **https://docs.google.com/document/d/e/2PACX-1vRIW_TuZ6z0ASjAoxgJgmzjGYLCDx019tKvphaTwK_Za7fnMKywUuXI0- s5wr0nQI_gprm6J6y7L9rL/pub**
- ➢ **https://jinja.palletsprojects.com/en/3.0.x/templates/**
- ➢ **www.python.com**
- ➢ **https://developer.spotify.com/documentation/web-api/quick-start/**

2. Overall Description

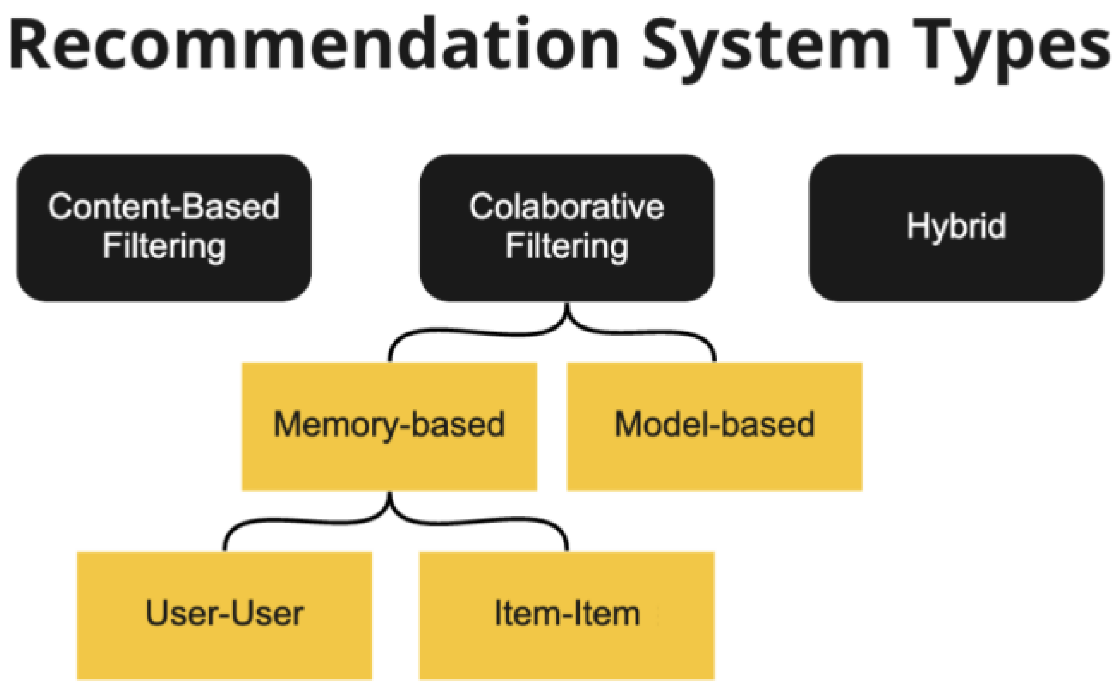
2.1 Product Perspective

The product MusicWreck is a music recommendation system product. It mainly is trained on a Spotify dataset. The user is suggested with song recommendation after she picks her most liked songs. By using music recommender system, the music provider can predict and then offer the appropriate songs to their users based on the characteristics of the music that has been heard previously. In a nutshell, recommendation systems recommend things that the people might like based on your own watch history or you and friends watch history as a collective.

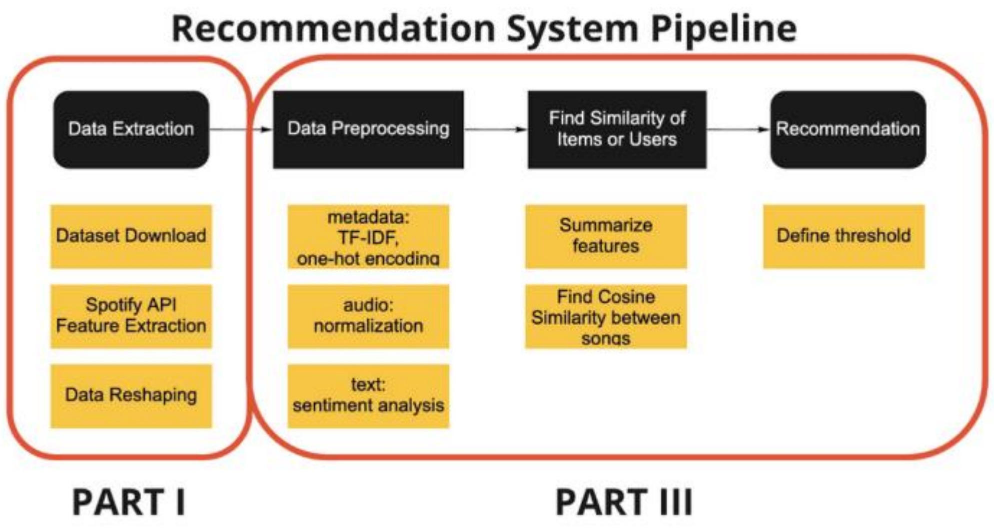
2.2 Product Functionality


General Recommendation System Pipeline
Data Extraction
Data Preprocessing
Find Similarity of Items or Users
Recommendation

General Recommendation System Pipeline. (Image by author)



Different Types of Recommendation System. (Image by author)

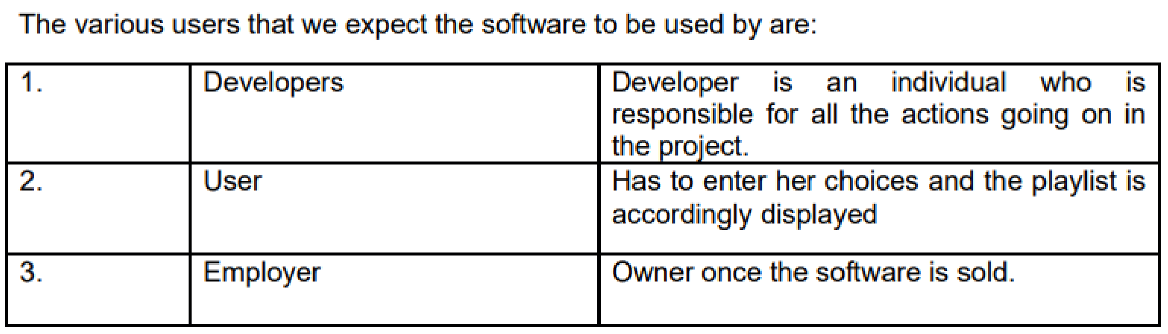


Recommendation System Pipeline for this project. (Image by author)

2.3 Users and Characteristics

The system will support four types of user privileges:

- Developer
- User
- Employer



The various users that we expect the software to be used by are:

1.	Developers	Developer is an individual who is responsible for all the actions going on in the project.
2.	User	Has to enter her choices and the playlist is accordingly displayed
3.	Employer	Owner once the software is sold.

2.4 Operating Environment

The software will be designed to work on any version of Windows, Linux (kernel 2.7 and above) and Mac platform. The software is completely web based and runs on popular web browsers namely firefox, chrome, internet explorer (IE8 and above). These web browsers are preferred since they support HTML.

2.5 Design and Implementation Constraints

We have to design different pages for different types of users such as developers and target audience. The implementation part is yet to be done. But, we have a clear picture as to how our pages would look. The communication protocol will be http and smtp. There are a number of tools which can be used for its implementation. The maximum number of users at a time is yet to be decided we will try to deploy it at an online hosting website or make an installable python executable.

2.6 User Documentation

No tutorials have been developed as of now.

2.7 Assumptions and Dependencies Assumptions

The user is familiar with internet and web based software like social networking sites. The browsers which the user is using is either Google Chrome 10.0 and above or Mozilla Firefox 4.0 and above.

Dependencies

- Spotify Dataset
- Spotify developers API
- Streamlit or Flask

- Other modules and packages to be decided based on implementation

3. Specific Requirements

3.1. External Interface Requirements

3.1.1. User Interfaces

The user interface design is simple and clear. Users can simply choose enter a song and obtain recommendations

3.1.2. Hardware Interfaces

Not applicable.

3.1.3. Software Interfaces

The software is operating system independent. It would run on Linux, Windows and Mac.

3.1.4. Communications Interfaces

A web browser is a basic necessity for the software to be deployed. Playlist is shared via SMTP protocol or User has to enter Spotify credentials to get it

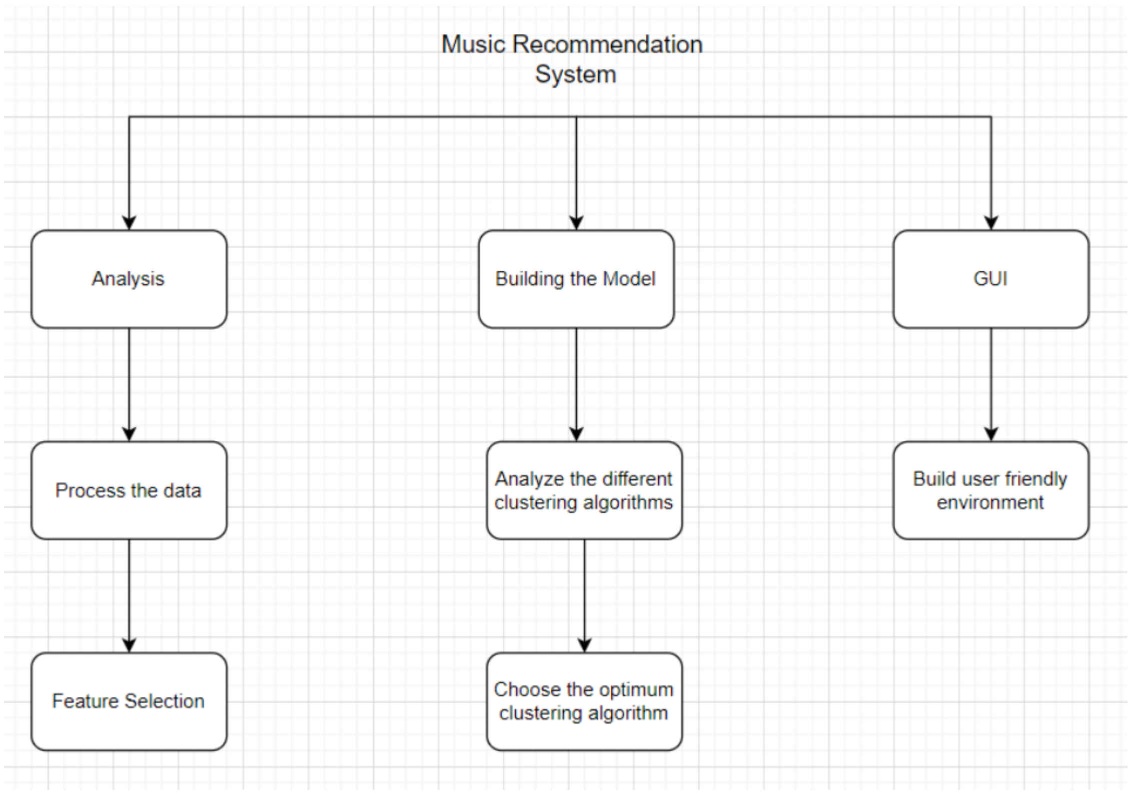
PES UNIVERSITY, BANGALORE

Department of Computer Science and Engineering

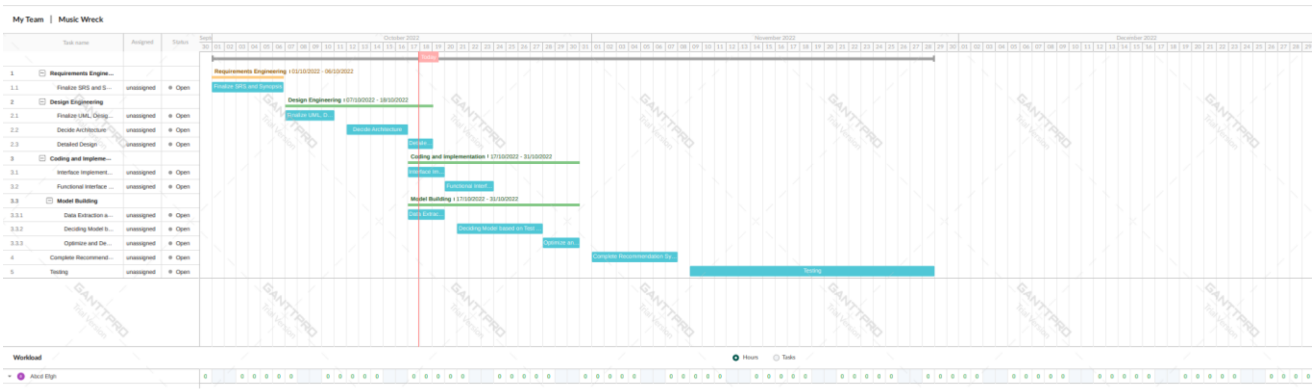
PROJECT PLAN

We have decided to go with Agile methodology with client Server Tiered architecture. We chose to use JIRA for Agile project management and tracking project development. The work was equally distributed using WBS and multiple sprints. Each sprint was focused on making the product better by adding features iteratively.

Work Breakdown Structure:


Music Recommendation System
Analysis
Building the Model
GUI
Process the data
Analyze the different clustering algorithms
Build user friendly environment
Feature Selection
Choose the optimum clustering algorithm

Gantt Chart:

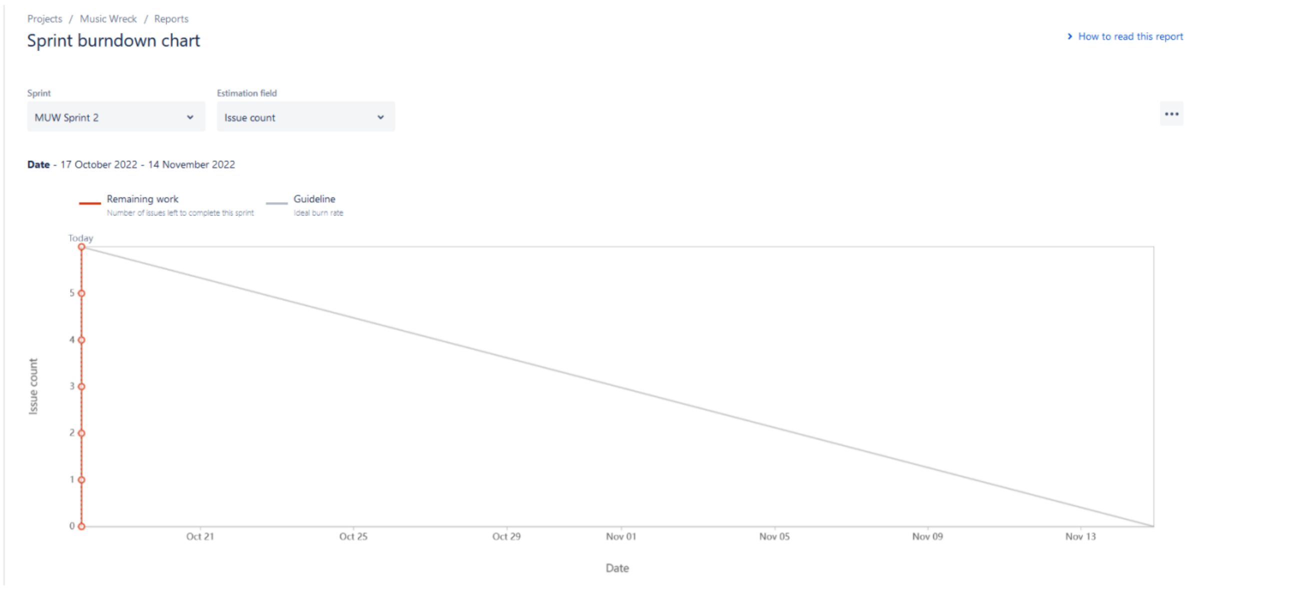


My Team | Music Wreck

	Task name	Assigned	Status
1	Requirements Engine...		
1.1	Finalize SRS and S...	unassigned	Open
2	Design Engineering		
2.1	Finalize UML, Desig...	unassigned	Open
2.2	Decide Architecture	unassigned	Open
2.3	Detailed Design	unassigned	Open
3	Coding and Impleme...		
3.1	Interface Implement...	unassigned	Open
3.2	Functional Interface ...	unassigned	Open
3.3	Model Building		
3.3.1	Data Extraction a...	unassigned	Open
3.3.2	Deciding Model b...	unassigned	Open
3.3.3	Optimize and De...	unassigned	Open
4	Complete Recommend...	unassigned	Open
5	Testing	unassigned	Open

Requirements Engineering | 01/10/2022 - 06/10/2022
Finalize SRS and Synopsis
Design Engineering | 07/10/2022 - 18/10/2022
Finalize UML, D...
Decide Architecture
Detaile...
Coding and Implementation | 17/10/2022 - 31/10/2022
Interface Im...
Functional Interf...
Model Building | 17/10/2022 - 31/10/2022
Data Extrac...
Deciding Model based on Test ...
Optimize an...
Complete Recommendation Sy...
Testing
Today
October 2022
November 2022
December 2022
Workload
Hours
Tasks
Abcd Elgh

Sprint Burndown Chart:



Projects / Music Wreck / Reports

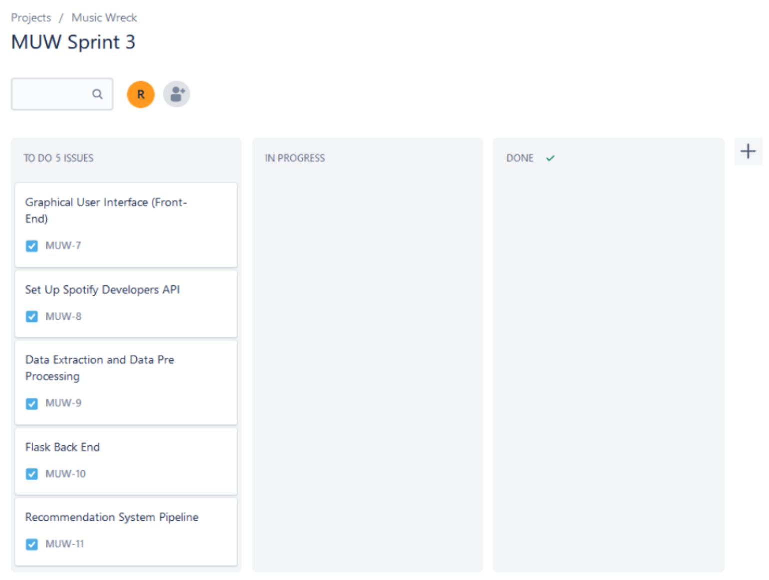
Sprint burndown chart

Sprint: MUW Sprint 2

Estimation field: Issue count

Date - 17 October 2022 - 14 November 2022

Sample screenshots from JIRA for sprint Reports:


Projects / Music Wreck

MUW Sprint 3

TO DO 5 ISSUES

Graphical User Interface (Front-End)
MUW-7

Set Up Spotify Developers API
MUW-8

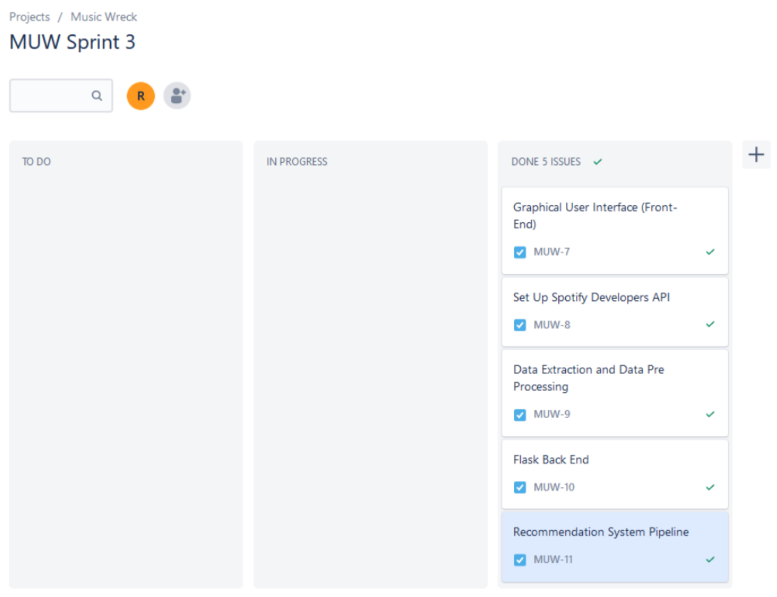
Data Extraction and Data Pre Processing
MUW-9

Flask Back End
MUW-10

Recommendation System Pipeline
MUW-11

IN PROGRESS

DONE ✓


Projects / Music Wreck

MUW Sprint 3

TO DO

IN PROGRESS

DONE 5 ISSUES ✓

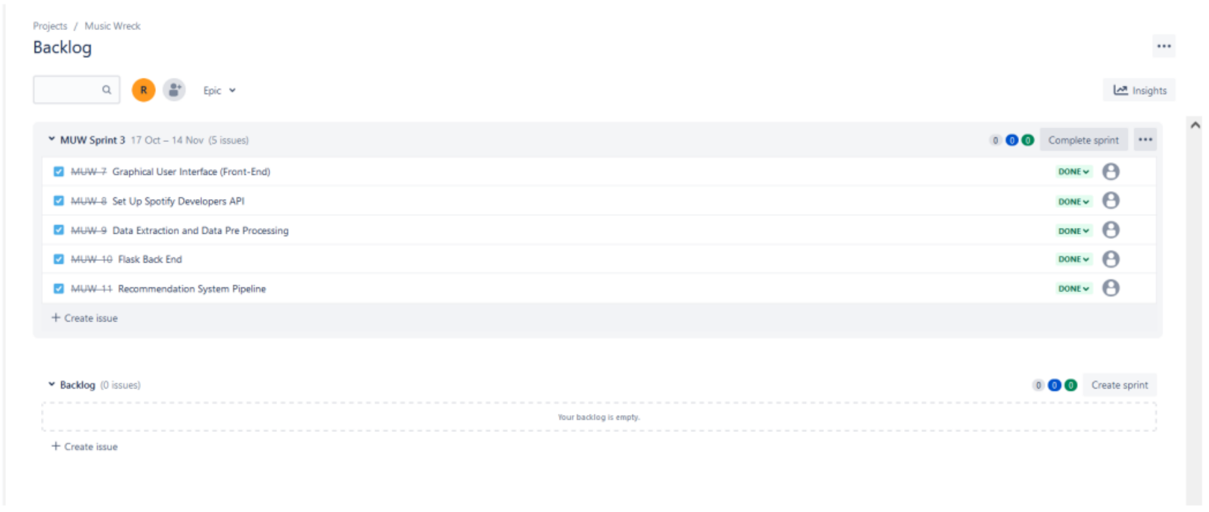
Graphical User Interface (Front-End)
MUW-7 ✓

Set Up Spotify Developers API
MUW-8 ✓

Data Extraction and Data Pre Processing
MUW-9 ✓

Flask Back End
MUW-10 ✓

Recommendation System Pipeline
MUW-11 ✓


Projects / Music Wreck

Backlog

MUW Sprint 3 17 Oct – 14 Nov (5 issues)

~~MUW-7~~ Graphical User Interface (Front-End) DONE
~~MUW-8~~ Set Up Spotify Developers API DONE
~~MUW-9~~ Data Extraction and Data Pre Processing DONE
~~MUW-10~~ Flask Back End DONE
~~MUW-11~~ Recommendation System Pipeline DONE

Backlog (0 issues)

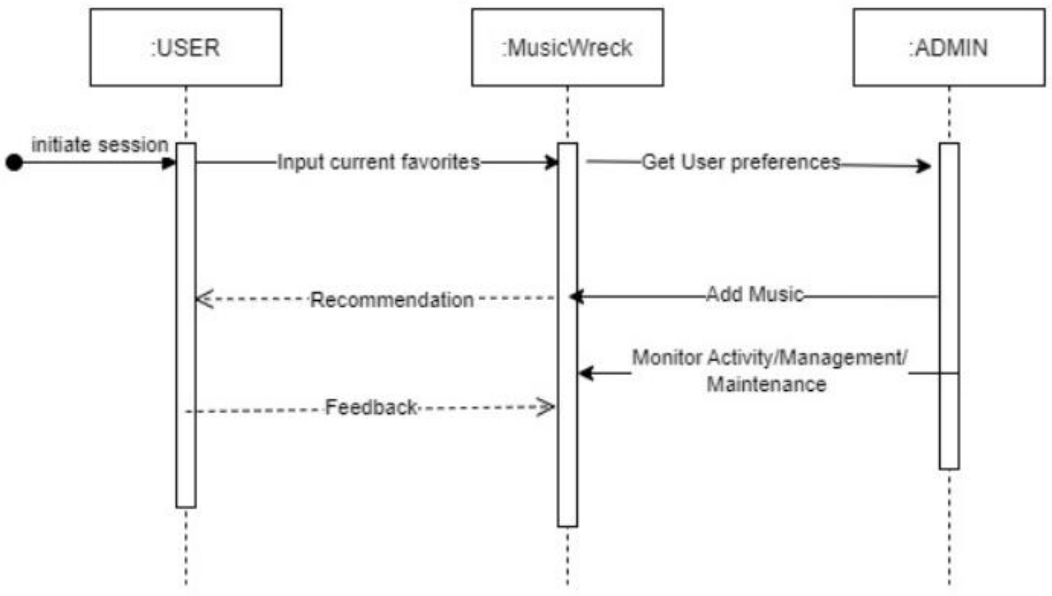
Your backlog is empty.

PES UNIVERSITY, BANGALORE

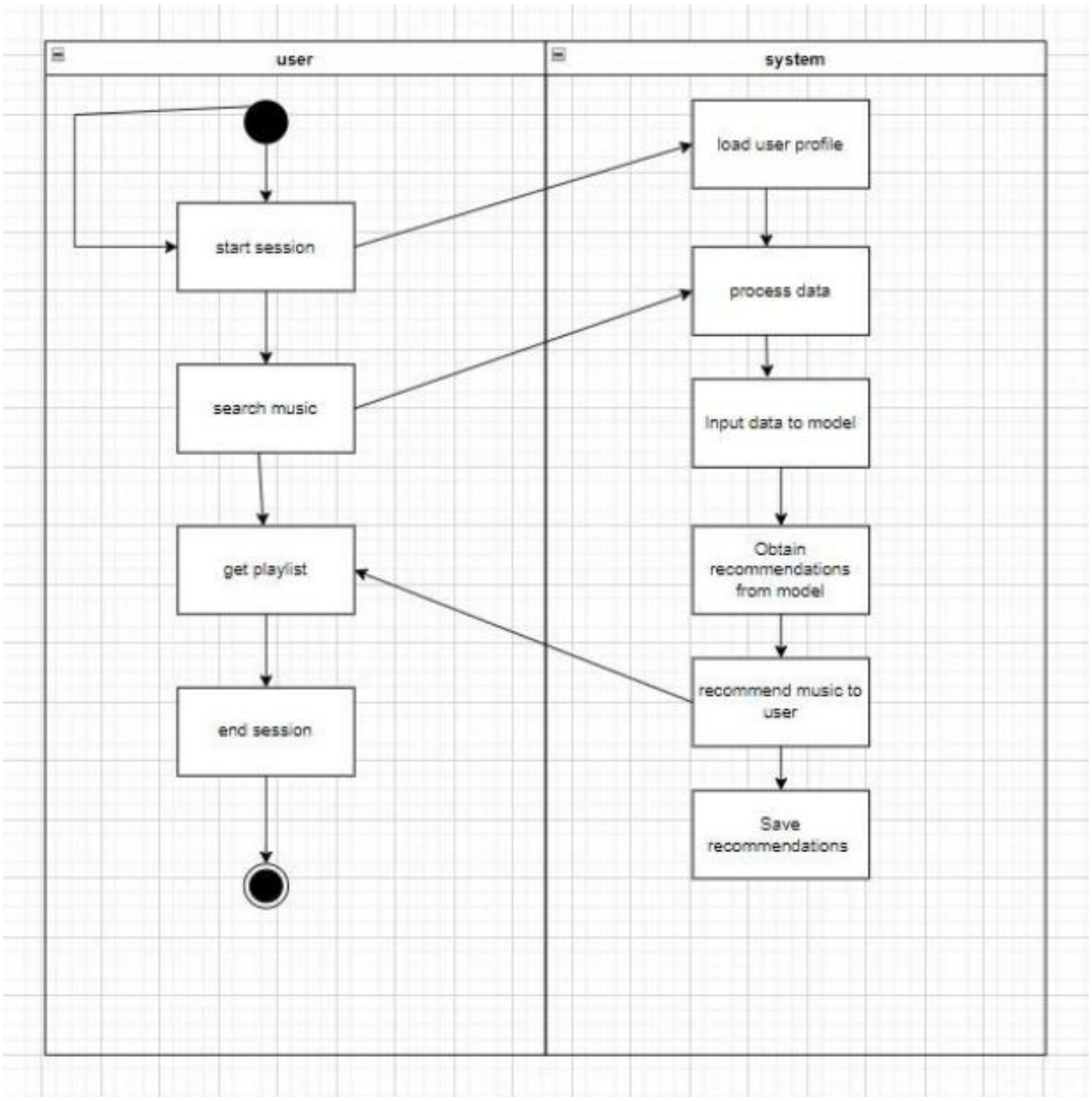
Department of Computer Science and Engineering

DESIGN DIAGRAM

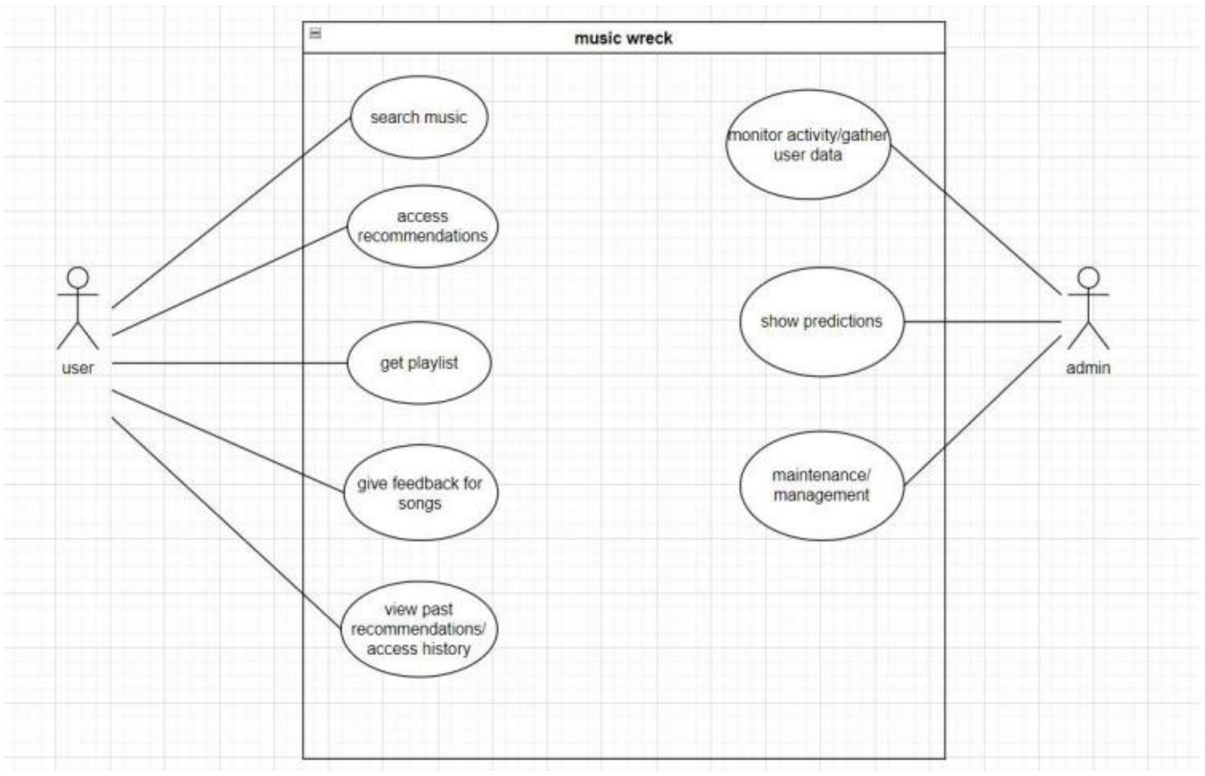
UML Diagram:



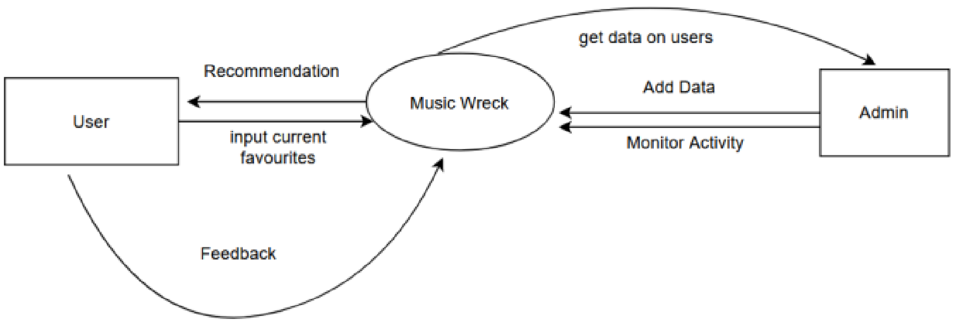
Activity Diagram:



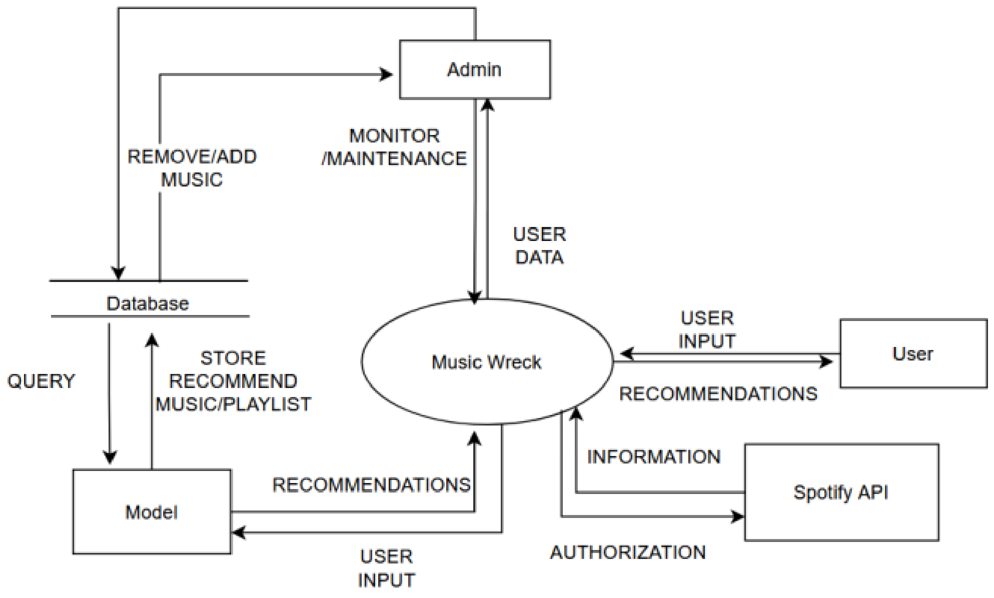
Use Case Diagram:



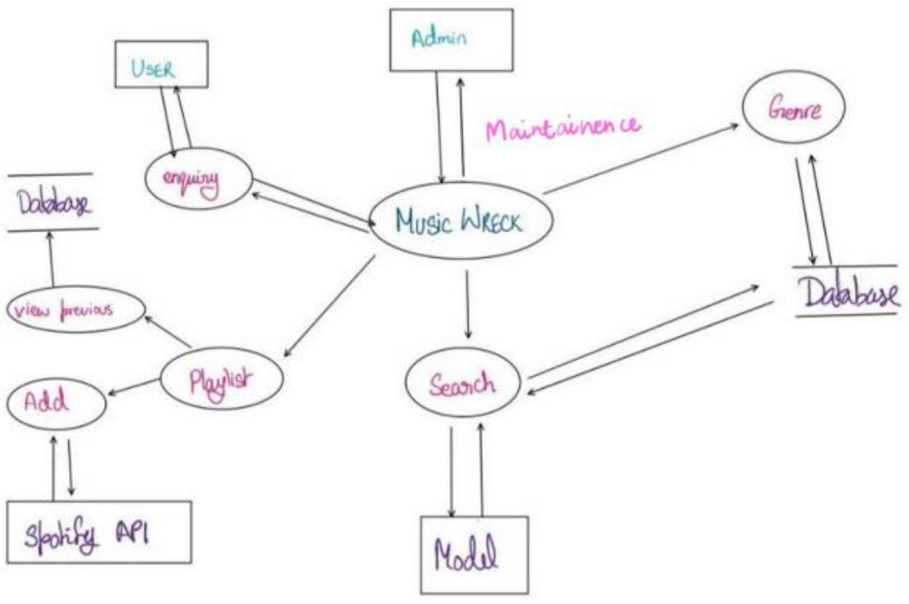
Top Level DFD:



First Level Decomposition:



Second Level Decomposition:



PES UNIVERSITY, BANGALORE

Department of Computer Science and Engineering

SOURCE CONTROL MANAGEMENT USING GITHUB

We chose to use Git for our source control management tool. All members were given access to the project repository in order to fix bugs or add new features. Members working on a new feature would open a new branch, make the relevant changes and then send in a pull request to merge once they were confident with the feature functionality.

PES UNIVERSITY, BANGALORE

Department of Computer Science and Engineering

TEST CASES

Test Suite:

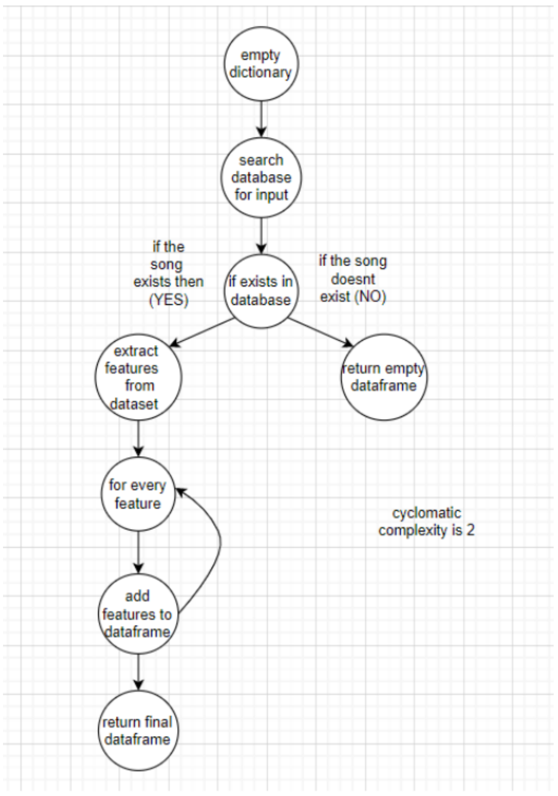
1. Take input from user
2. Find relevant song data from the input given by users using the Spotify API
3. Get recommendations from model
4. Output the recommendations obtained by the model to the frontend

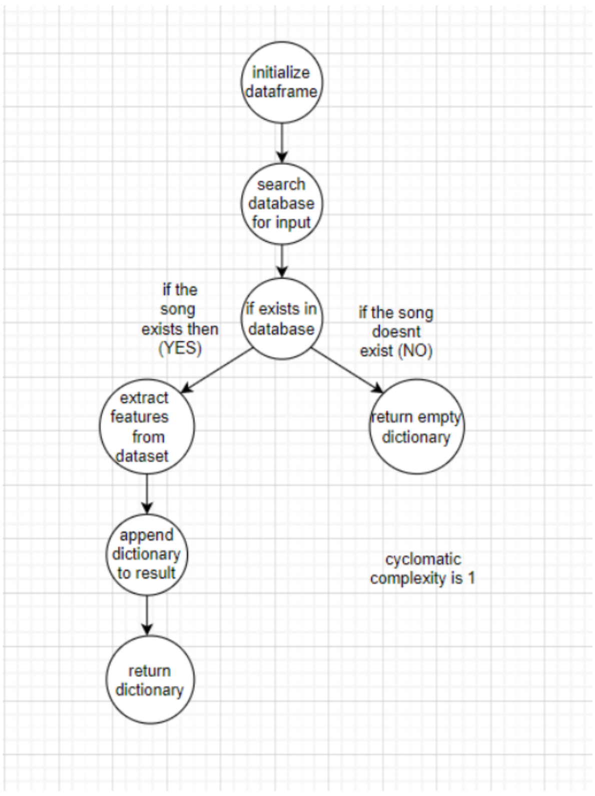
Test Strategy:

We have used four testing strategies -

1. Static testing:

   Initially we were outputting a data frame after extracting features from the API for which we were using a ‘for’ loop. In order to optimize the code and minimize complexity we are now outputting a dictionary instead of a ‘for’ loop. As we see from the screenshot below, the number of nodes are 7 & edges are 6. Hence the complexity is reduced.

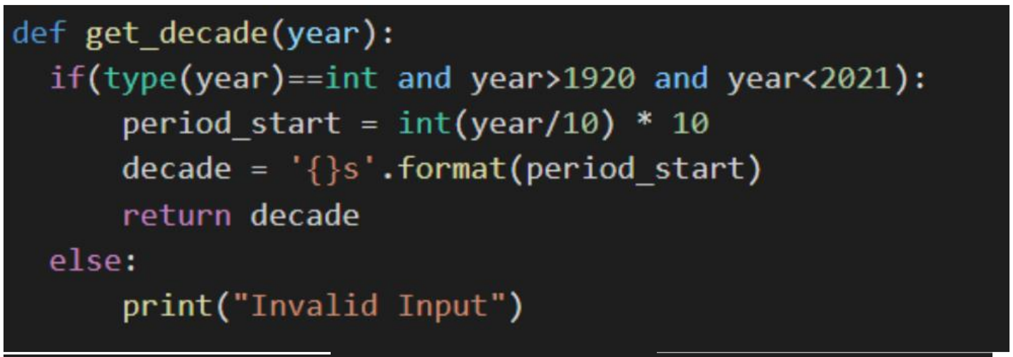

empty dictionary
search database for input
if exists in database
if the song exists then (YES)
if the song doesnt exist (NO)
extract features from dataset
return empty dataframe
for every feature
add features to dataframe
return final dataframe
cyclomatic complexity is 2
Initial Design


initialize dataframe
search database for input
if exists in database
if the song exists then (YES)
if the song doesnt exist (NO)
extract features from dataset
return empty dictionary
append dictionary to result
return dictionary
cyclomatic complexity is 1
Optimized Design

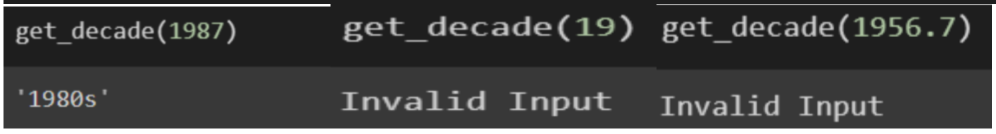
Complexity = E – N + 2
Where E = edges, N = nodes

For initial design, complexity = 2
For optimized design, complexity = 1

2. Mutation testing:



```
def get_decade(year):
  if(type(year)==int and year>1920 and year<2021):
      period_start = int(year/10) * 10
      decade = '{}s'.format(period_start)
      return decade
  else:
      print("Invalid Input")
```



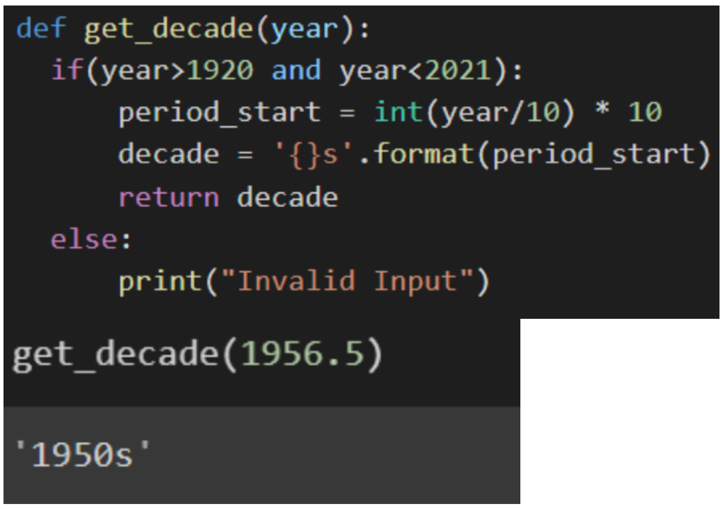
```
get_decade(1987)
'1980s'
```

```
get_decade(19)
Invalid Input
```

```
get_decade(1956.7)
Invalid Input
```

The get_decade function takes in year as input and outputs the decade to which it belongs. The above screenshots show the code and output for various test cases of the function.

Mutation 1 -



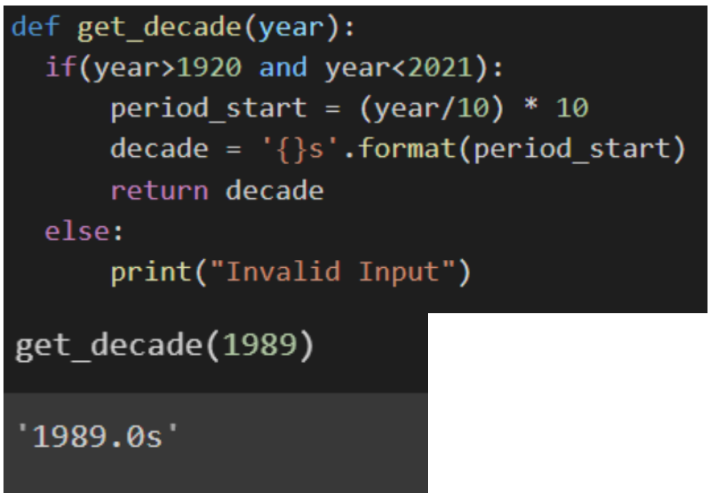
```
def get_decade(year):
  if(year>1920 and year<2021):
      period_start = int(year/10) * 10
      decade = '{}s'.format(period_start)
      return decade
  else:
      print("Invalid Input")
```

```
get_decade(1956.5)
```

```
'1950s'
```

In the above mutated code the condition to check for data type of year is removed. From the above screenshot it is seen that the wrong output is obtained. Year cannot be of type float but the function inputs float value as a valid input for year.

Mutation 2 -



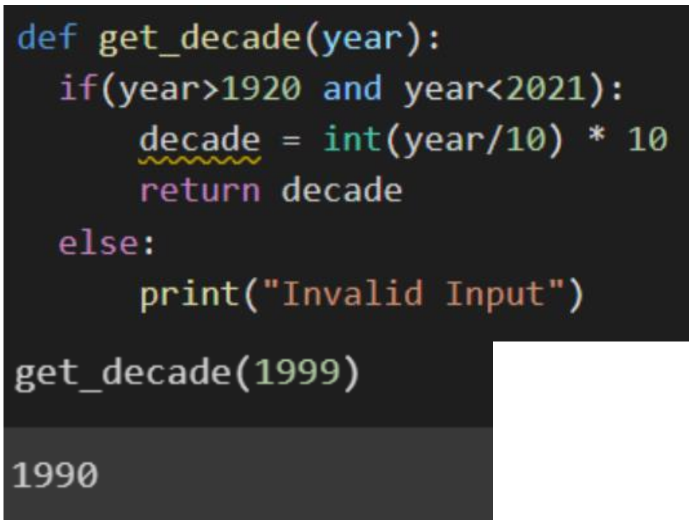
```
def get_decade(year):
  if(year>1920 and year<2021):
      period_start = (year/10) * 10
      decade = '{}s'.format(period_start)
      return decade
  else:
      print("Invalid Input")
```

```
get_decade(1989)
```

```
'1989.0s'
```

In the above mutated code the explicit type conversion to int is omitted. This causes the function to give invalid output. The function returns a float value as year but year is supposed to be of type int.

Mutation 3 -


```
def get_decade(year):
  if(year>1920 and year<2021):
      decade = int(year/10) * 10
      return decade
  else:
      print("Invalid Input")
```
```
get_decade(1999)
```
```
1990
```

In the above mutated code a line of code is removed from the original function being tested. It can be seen that the output gives out a year. However, it cannot be clearly determined that the result is the decade to which the year belongs. 1950s indicate a decade whereas 1950 is a year.

3. Unit testing:

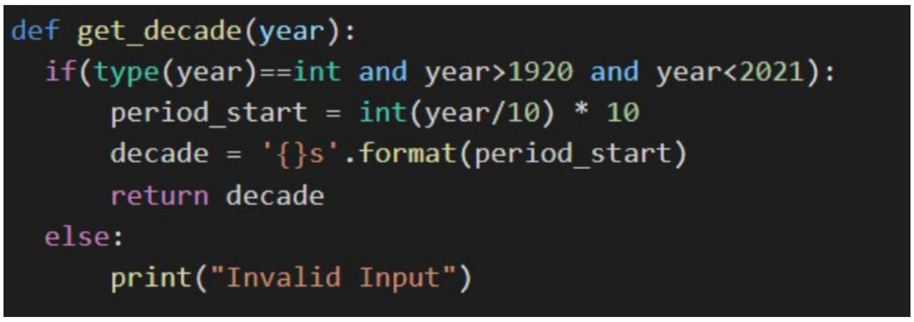
The project can be divided into different units such as
➢ Pre-processing of data
➢ Building the model
➢ Finally obtaining recommendations using Spotify API

The Pre-processing of data can be further divided into smaller units such as
➢ Filtering out essential features
➢ Grouping data by decade as opposed to year

The Model building Unit can be further divided into smaller units such as
➢ Creating pipeline
➢ Training the model

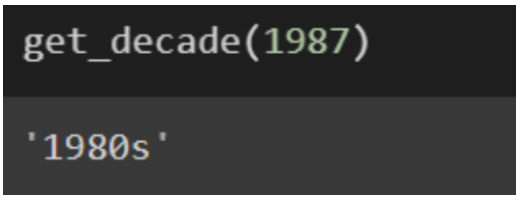
Obtaining recommendation Unit can be further divided into smaller units such as
➢ Getting input from user
➢ Cross Checking input with Spotify API to obtain relevant features
➢ Recommend based on model output

Test Case for pre-processing:



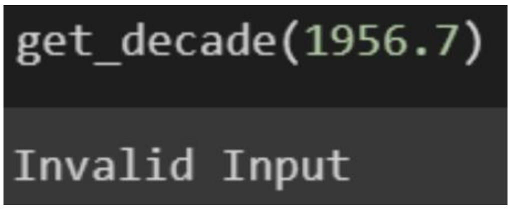
```
def get_decade(year):
  if(type(year)==int and year>1920 and year<2021):
      period_start = int(year/10) * 10
      decade = '{}s'.format(period_start)
      return decade
  else:
      print("Invalid Input")
```

The above screenshot shows the function get_decade used to preprocess the data. The function takes in year as input and outputs the decade to which it belongs.



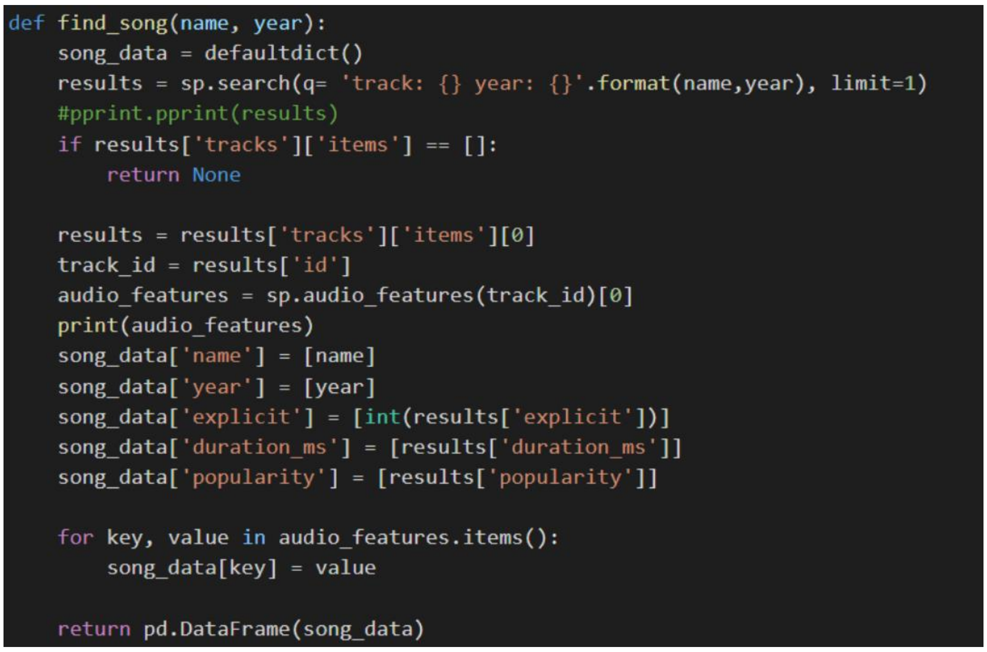
```
get_decade(1987)

'1980s'
```

The above screenshot shows an example of a test case for valid data. In this case, the year 1987 is given as input and the function outputs 1980s.



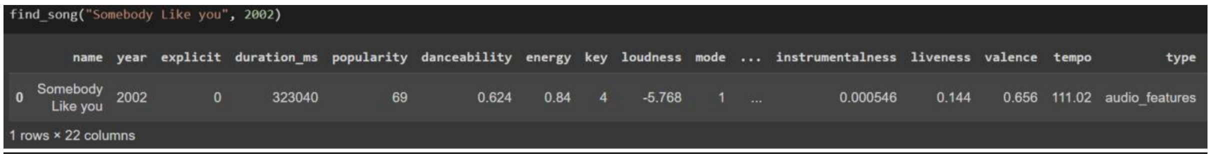
```
get_decade(1956.7)

Invalid Input
```

The above screenshot shows an example of a test case for invalid data. The year input is given as 1956.7 which is a float value and hence results in invalid input.

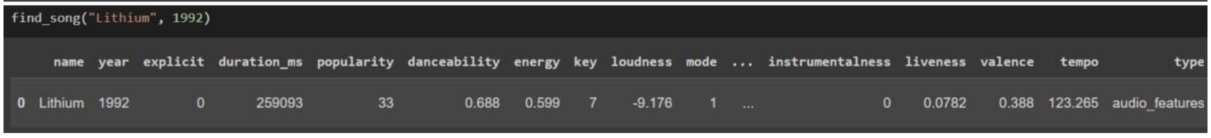


```
def find_song(name, year):
    song_data = defaultdict()
    results = sp.search(q= 'track: {} year: {}'.format(name,year), limit=1)
    #pprint.pprint(results)
    if results['tracks']['items'] == []:
        return None

    results = results['tracks']['items'][0]
    track_id = results['id']
    audio_features = sp.audio_features(track_id)[0]
    print(audio_features)
    song_data['name'] = [name]
    song_data['year'] = [year]
    song_data['explicit'] = [int(results['explicit'])]
    song_data['duration_ms'] = [results['duration_ms']]
    song_data['popularity'] = [results['popularity']]

    for key, value in audio_features.items():
        song_data[key] = value

    return pd.DataFrame(song_data)
```

The above function find_song is used in the recommendation unit to obtain song details from spotify API. The function inputs the song name and year and outputs a dataframe with track details.

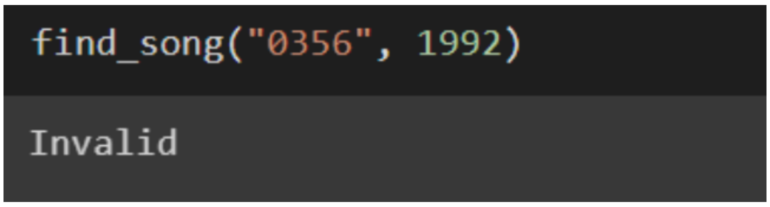


```
find_song("Somebody Like you", 2002)

          name  year  explicit  duration_ms  popularity  danceability  energy  key  loudness  mode  ...  instrumentalness  liveness  valence   tempo            type
0  Somebody     2002         0       323040          69         0.624    0.84    4    -5.768     1  ...          0.000546     0.144    0.656  111.02  audio_features
   Like you
1 rows × 22 columns
```



```
find_song("Lithium", 1992)

      name  year  explicit  duration_ms  popularity  danceability  energy  key  loudness  mode  ...  instrumentalness  liveness  valence    tempo            type
0  Lithium  1992         0       259093          33         0.688   0.599    7    -9.176     1  ...                 0    0.0782    0.388  123.265  audio_features
```

The above screenshots are examples of valid test cases which result in data frames with track info such as popularity, liveness, energy etc.



```
find_song("0356", 1992)

Invalid
```

The above screenshot shows an example of a test case with invalid data.

4. Integration testing:

   We have also tested all the units together to make sure all of them are working properly in synergy.

Test Plan:

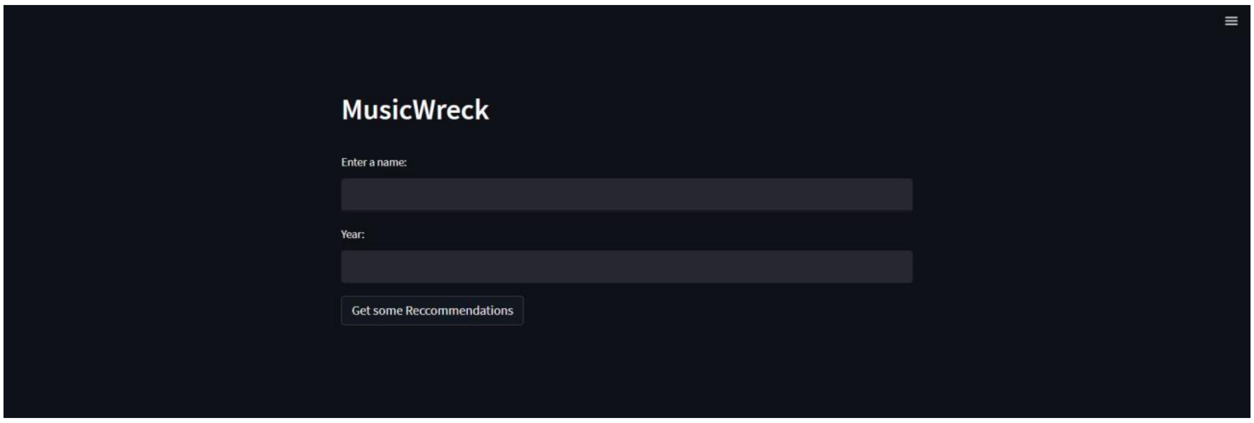
- ➢ This product is aimed to give users a user-friendly experience with no complicated technology knowledge required.
- ➢ In order to produce a bug free product that provides a user-friendly environment, we have chosen to use four testing types - unit, integration, static and mutation.
- ➢ Testing is performed by developers after each and every sprint.

PES UNIVERSITY, BANGALORE

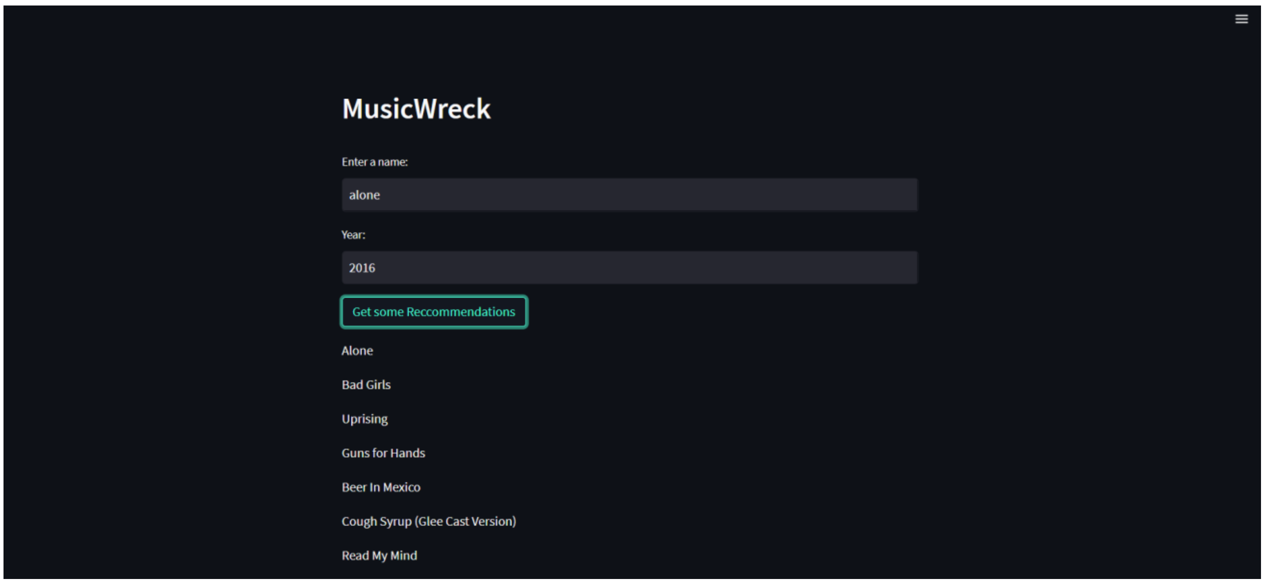
Department of Computer Science and Engineering

SCREENSHOTS OF THE OUTPUT

Sample screenshots of UI



Screenshot of home page

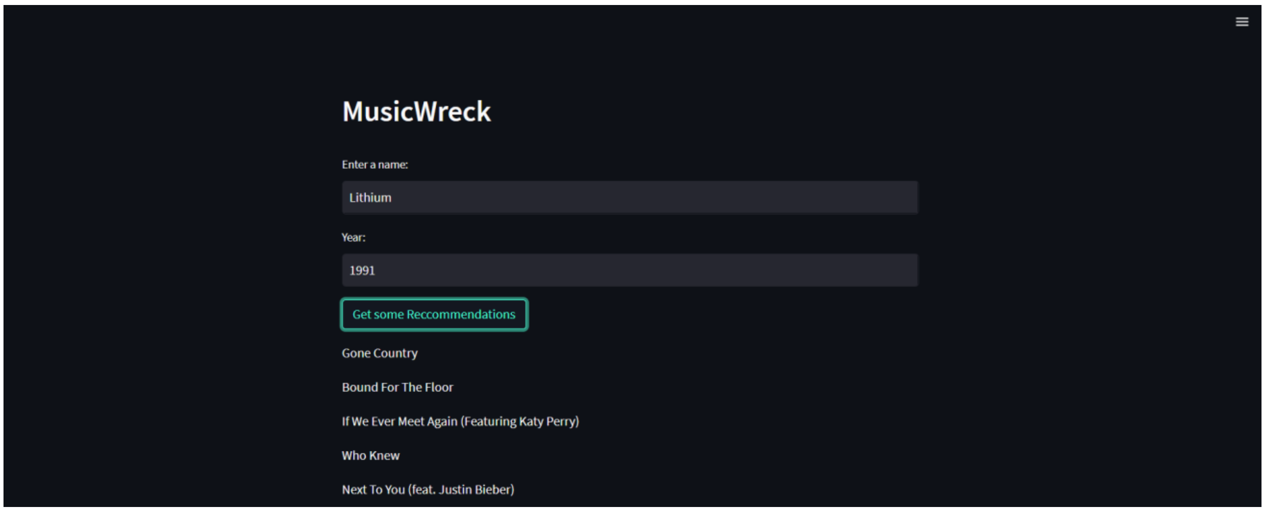


alone

2016

Alone
Bad Girls
Uprising
Guns for Hands
Beer In Mexico
Cough Syrup (Glee Cast Version)
Read My Mind

Screenshot of page with sample results 1



Lithium

1991

Gone Country

Bound For The Floor

If We Ever Meet Again (Featuring Katy Perry)

Who Knew

Next To You (feat. Justin Bieber)

Screenshot of page with sample results 2