

Automating Spotify Playlist Creation using

Machine Learning

Interim Report

TU856

BSc in Computer Science

**Alexandros Tsiogas**

**C20336236**

**Supervisor: Brendan Tierney**

School of Computer Science

Technological University, Dublin

**31/10/2023**

Abstract

This project aims to create a new way for listeners to find music that they enjoy, using machine learning. This project explores the shortcomings of the music recommendation systems that are currently popular and aims to address such shortcomings through a new recommendation system. The project focuses on addressing perceived issues in the most widely used music streaming platform – Spotify.

This system will be presented through a web application that connects to a user’s Spotify account and accesses their data. The user is presented with a list of their Spotify playlists and may select one, from which a series of recommendations will be generated by a machine learning model. Prior to recommendation generation, a user is given the option of pre-emptively changing their recommendations, by increasing or decreasing variables about the mood and sound of the music. This aims to address the apparent issue of a user’s mood not being considered when they are being recommended music by Spotify’s current system.

Declaration

I hereby declare that the work described in this dissertation is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

Signed:

Alex Tsiogas\_\_\_\_

Alexandros Tsiogas

31/10/2023

Acknowledgements

Body text

Table of Contents

[1. Introduction 7](#_Toc151554788)

[1.1. Project Background 7](#_Toc151554789)

[1.2. Project Description 8](#_Toc151554790)

[1.3. Project Aims and Objectives 8](#_Toc151554791)

[1.4. Project Scope 9](#_Toc151554792)

[1.5. Thesis Roadmap 9](#_Toc151554793)

[2. Literature Review 11](#_Toc151554794)

[2.1. Introduction 11](#_Toc151554795)

[2.2. Alternative Existing Solutions 11](#_Toc151554796)

[2.3. Technologies 14](#_Toc151554797)

[2.3.1 Programming Languages 14](#_Toc151554798)

[2.3.2 Web Frameworks 15](#_Toc151554799)

[2.4. Recommender Systems 16](#_Toc151554800)

[2.5. Existing Final Year Projects 17](#_Toc151554801)

[2.5.1 Project 1: Moodify 17](#_Toc151554802)

[2.5.2 Project 2: Football Data Mining, Result Prediction, and Visualization 18](#_Toc151554803)

[2.6. Conclusions 20](#_Toc151554804)

[2.6.1 Requirements 20](#_Toc151554805)

[2.6.2 Technologies 20](#_Toc151554806)

[3. System Design 21](#_Toc151554807)

[3.1. Introduction 21](#_Toc151554808)

[3.2. Software Methodology 21](#_Toc151554809)

[3.2.1 Agile 21](#_Toc151554810)

[3.2.2 Scrum 22](#_Toc151554811)

[3.2.3 Kanban 23](#_Toc151554812)

[3.3. System Architecture 24](#_Toc151554813)

[3.4. Frontend 25](#_Toc151554814)

[3.4.1 Use Case Diagram 25](#_Toc151554815)

[3.4.2 Flowchart 26](#_Toc151554816)

[3.4.3 Wireframes 26](#_Toc151554817)

[3.5. Backend 29](#_Toc151554818)

[4. Testing and Evaluation 30](#_Toc151554819)

[4.1. Introduction 30](#_Toc151554820)

[4.2. Plan for Testing 30](#_Toc151554821)

[4.2.1 Integration testing 30](#_Toc151554822)

[4.2.2 Unit Testing 31](#_Toc151554823)

[4.2.3 User Acceptance Testing 31](#_Toc151554824)

[4.3. Plan for Evaluation 32](#_Toc151554825)

[4.4. Conclusions 32](#_Toc151554826)

[5. Prototype Development 34](#_Toc151554827)

[5.1. Introduction 34](#_Toc151554828)

[5.2. Prototype Development 34](#_Toc151554829)

[5.3. Other Sections 34](#_Toc151554830)

[5.4. Conclusions 34](#_Toc151554831)

[6. Issues and Future Work 35](#_Toc151554832)

[6.1. Introduction 35](#_Toc151554833)

[6.2. Issues and Risks 35](#_Toc151554834)

[6.2.1. Scope Creep 35](#_Toc151554835)

[6.2.2. Unexpected Evaluation Results 35](#_Toc151554836)

[6.2.3. Integration Risks 36](#_Toc151554837)

[6.2.4. External Risks 36](#_Toc151554838)

[6.3. Plans and Future Work 36](#_Toc151554839)

[6.3.1. Database 36](#_Toc151554840)

[6.3.2. Frontend 37](#_Toc151554841)

[6.3.3. Recommender System 37](#_Toc151554842)

[6.3.4. Testing and Evaluation 38](#_Toc151554843)

[6.3.5. GANTT Chart 38](#_Toc151554844)

[6.3.5.1. Until Interim Submission 38](#_Toc151554845)

[6.3.5.3. Until Final Submission 39](#_Toc151554846)

[Bibliography 40](#_Toc151554847)

# 1. Introduction

This chapter describes the project background and a high-level description of the project.

## Project Background

In a time where online music streaming services are more popular than ever, Spotify dominates the market. Stats gathered in 2022 show that Spotify have by far the greatest market share in this domain at 30.5% - over twice that of the nearest competitor, Apple Music (1).

Despite Spotify’s massive usage, listeners seem to be engaging less with their music recommendation functionality. Spotify offers several ways for users to explore new music, such as their “Radio” or “Daily Mix”, and yet surveys have found that more listeners are finding new music through traditional means. In a 2019 survey of over 500 music listeners, 45.9% of responses listed “Active search myself” as their main source of new music, while Spotify and other streaming services accounted for only 26.1% of responses (2).

A pie chart with text

Description automatically generated

Figure : Survey results on how people find new music (2).

These statistics raise questions over why Spotify’s dominance in the music streaming domain cannot be replicated in the music recommendation domain.

The shortcomings of Spotify’s recommendation systems are a matter of clear frustration for users. A 2015 study (3) found that only 3% of Spotify users find that Spotify generated recommendations always match their taste, while 68.75% of users stated they match their taste only sometimes. The fact that users’ moods aren’t considered when they are being recommended music was highlighted as “one of the most important drawbacks” of the recommendation system. 66.7% of users interviewed chose “mood” as the main influence factor on the music they want to listen to.

It is abundantly clear that Spotify is failing to satisfy its user’s desires to find new music that matches their mood, leading to many users searching elsewhere for new music. This project will attempt to address these shortcomings by developing a recommendation system that can better account for the mood of a user.

## Project Description

This project will take the form of an online web application. The app will be presented through web pages designed with HTML and CSS, and the underlying functionality will be made in the Python web framework “Flask”.

Upon entering the web app, users will be presented with a Spotify login screen that prompts them to log in with their username and password. Once they successfully log in, the application is granted access to their saved playlists. This is achieved through the Spotify Web API, which enables the creation of applications that can interact with Spotify's streaming service. The user will then be presented with a list of their saved playlists, from which they can select a playlist as a basis for recommendation creation. The user may then alter some variables about the mood of the music they desire before recommendations are generated. The contents of the playlist and the mood values can then be passed through a machine learning model, which eventually outputs a series of recommendations. These recommendations can be saved directly to a user’s library through the user interface, via the Spotify Web API.

## Project Aims and Objectives

The main objective of this project is to produce a web application and machine learning model that are capable of producing music recommendations that account for user mood. In success, these recommendations will be superior to those produced by Spotify – this sentiment will be measured in the end evaluation. The milestones that will be reached throughout the project are as follows:

* Extensively analyse the strengths and shortcomings of the current Spotify recommendation algorithm and determine how my application can improve on it.
* Successfully build a Flask web app that grants access to a user’s Spotify library.
* Implement an aesthetically pleasing front end, with a visually pleasing UI that is easy to navigate and offers a good user experience.
* Create functionality through the Spotify Web API that allows the manipulation of a user’s library via the web application.
* Creation of elements on the web page that allow a user to change variables about the mood of their recommendations, which will be processed by the machine learning algorithm.
* Creation of a machine learning algorithm that can select songs from Spotify’s database to recommend to a user based on their selected playlist and their mood values.

## Project Scope

This project is at heart a machine learning project, focused on designing a machine learning model that can accept direct user input and existing user Spotify data, and producing recommendations based on these. The retrieval, cleaning and processing of user data are all within the scope, as is the start-to-finish construction of a suitable recommender system, including all testing and evaluation.

Furthermore, a user-friendly web application that serves as user interface for interaction with the recommender system will be developed, tested, and evaluated.

This project aims to be compatible with Spotify only, due to the popularity of the service and the availability of the Spotify Web API. Compatibility with any other online music streaming service is considered out of scope.

The application aims to be presented in a web application – no mobile application is in scope.

While the machine learning algorithm will account for user mood, the application will play no role in determining the mood of the user - the user will describe their own mood through on-screen elements.

## Thesis Roadmap

**Chapter 2 – Literature Review**

This chapter will describe the research conducted and literature reviewed thus far in the project.

**Chapter 3 – System Design**

This chapter will describe the planned design of the final system, including diagrams describing architecture and visuals. It will also introduce the methodologies to be used in development.

**Chapter 4 – Testing and Evaluation**

This chapter will describe a plan for the eventual testing and evaluation of the finished project.

**Chapter 5 – Prototype Development**

This chapter will present the prototype that has been created and will detail all work completed and milestones achieved thus far in the development process.

**Chapter 6 – Issues and Future Work**

This chapter will outline any issues faced in the development process so far and will discuss potential ways of resolving these issues going forward. It will also identify a plan of future work needed to complete the project, which will be structured as a GANTT chart.

# Literature Review

## 2.1. Introduction

This chapter describes all research conducted and literature reviewed thus far in the project.

## 2.2. Alternative Existing Solutions

The shortcomings of the Spotify recommendation system are an issue that developers have been trying to address a great deal in recent times, as is evident in the sheer volume of web and mobile applications that have been developed that utilise the Spotify API to build on the application’s core functionality.

**Discz**

One such application is the mobile app “Discz” - a massively successful music recommendation app that operates using the Spotify API. The app allows users to swipe through songs and give their sentiments on them, in a binary “like” or “dislike”. The app uses the users swiping data to learn about the user’s taste, and gradually improve its recommendations.

A screenshot of a phone

Description automatically generated

Figure 2: Discz' swiping functionality

The Discz app uses a machine learning algorithm in its recommendations, which proves the feasibility of my project, and means the app can serve as a comparison to my finished product, and a source of research and comparison for my initial design. The popularity of ‘Discz’ is highlighted in a 2022 Rolling Stone Article, which finds that “more than 15 million songs have been discovered and saved on the app to date” (4). This figure clearly shows the desire that music listeners have for better ways to find new music.

The Discz app serves as an inspiration and a comparison for some facets of my app. One such way is that it eliminates the need for the user to manually add songs to a playlist. The flow of the recommendation system is easy to follow - a user repeatedly swipes left or right, and once they are satisfied with what they have selected, all the songs can be added directly to a newly created playlist. This simple recommendation flow is something I found to be very intuitive, and it aids in streamlining the playlist creation process. It is something I have drawn inspiration from in the design of my own app, where a user will be able to generate recommendations and export them to their Spotify library with the click of a button.

There are some areas where Discz succumbs to the same shortcomings of the Spotify recommendation system – it does not directly account for user mood. The recommendations are not mood based, and the user gets no direct say over what they are recommended. As previously discussed, users want to have their mood considered when they are being recommended music, hence why mood-based recommendation is a primary goal of my project.

**MagicPlaylist**

One application that does make some attempt to account for user mood is MagicPlaylist. Much like my application, this is a web app that offers music recommendations, using the Spotify Web API. The homepage presents the user with a number of “moods”, upon which they can have a playlist generated.

A collage of two people

Description automatically generated

Figure : The "moods" feature of MagicPlaylist

A screenshot of a computer

Description automatically generated

Figure : Recommendations generated by MagicPlaylist.

The strongest merits of this app lie in its UI and UX. The UI is exceptionally intuitive throughout, with every “mood” framed in a large clickable button with a relevant background image.

A screenshot of a computer

Description automatically generated

Figure : The MagicPlaylist UI

The landing page is tidy and concise, comprising of just a search bar and the “mood” buttons. The capabilities of the website are described elegantly in the header ­- “*Type your favourite songs and create perfect playlists. Magic!”* - and a user can easily intuit the how to generate a playlist from here, making user experience straightforward. Comparing it to similar applications in the music recommendation space, MagicPlaylist is one of the most accessible for users. In my research, it served as an inspiration for the design of my UI, and the general flow of user interaction with the website.

The application does however lack somewhat in the mood-based recommendations. The user plays no role in defining the moods - they are predetermined within the application. This detracts from the user’s control over what they are recommended, as none of the predetermined moods may match exactly what they desire. I aim to grant the user more control over describing their mood, by allowing them to fine tune multiple “mood variables”, that their suggestions will be based upon.

Due to it being so close in nature to my application, MagicPlaylist may well serve as a comparison in my final evaluation.

## 2.3. Technologies

This section describes technologies that were researched and considered for use in the project, including those that were ultimately chosen to be used.

### 2.3.1 Programming Languages

This section describes the programming languages researched for the application.

##### 2.3.1.1 Java

Java is a general-purpose class-based, object-oriented programming language, designed to have as few implementation dependencies as possible (5). As of 2019, it was reported to be one of the most used programming languages among developers, particularly in developing client-server web applications, with a reported 9 million developers (6). Its widespread popularity in web applications and the large amount of documentation available make java a viable option.

A white background with black text

Description automatically generated

Figure : Some simple Java code (7).

##### 2.3.1.2 Python

Like Java, Python is a widely used general- purpose programming language, used in a range of applications including web applications (8). Python has a greater focus on readability and is less verbose than Java *(see Figure 6).* Python was chosen as the main programming language for this project due to its readability and compatibility with numerous packages, including Spotipy. Spotipy and its importance to my project will be described in the next section.

A white background with black text

Description automatically generated

Figure : Some simple Python code (7).

###### 2.3.1.2.1 Spotipy

Spotipy is a lightweight Python library for the Spotify Web API. It essentially eliminates the need to make direct calls to the Spotify API, instead compressing these calls into built in python functions that can be used to access Spotify data (9). Using Spotipy, data can easily be retrieved from a user’s Spotify library in the form of python arrays and dictionaries. This allows for easy manipulation of the data by other parts of the application that are written in Python, namely the machine learning model. By writing the application in Python and using the Spotipy library, I avoid any potential language interoperability problems (10). Research was conducted on Spotipy through the official Spotipy docs (9) and through past final year projects that utilised it (see section 2.5.1).

### 2.3.2 Web Frameworks

This section describes viable web frameworks researched for the application.

##### 2.3.2.1 Django

Django is a popular python-based web framework. The Django website describes it as being a framework that “encourages rapid development and clean, pragmatic design” Django is generally preferable for large, full-stack projects that require incorporated complex features (11).

##### 2.3.2.2 Flask

Flask is a more lightweight web framework written in Python. It is one of the most widely used web frameworks and has become commonly accepted as the perfect tool for quick and simple solutions. It is generally preferred for smaller and more lightweight web applications (12).

Flask’s more lightweight nature and its popularity with smaller scale applications make it the most viable option for my application. As most of my project’s complexity will fall under the machine learning aspect, the web application will be relatively small-scale. Flask was chosen as the web framework to be used in the app.

## 2.4. Recommender Systems

This section details the research conducted into machine learning and viable models for my application. The two main approaches to recommender systems are **Collaborative** **Filtering** and **Content-Based** **Filtering**, which were the two methods researched.

##### 2.4.1 Collaborative Filtering

Collaborative filtering is a common approach to designing recommender systems. It essentially anticipates the taste of a user by comparing known data about them with that of other users, often by utilising the k nearest neighbours’ algorithm (13).

A table of names and numbers

Description automatically generated with medium confidence

Figure : A table of data to about peoples’ ratings of movies (13)

For example, in figure 7, a collaborative based filtering algorithm should predict a favourable score for Carol for Harry Potter, as her scores closely mirror those of Joe.

Collaborative filtering is only possible if sentiment data is available for a large group of users, which will not be the case for my app, which will deal just with the data of a single user. For this reason, collaborative filtering was ruled out upon research.

##### 2.4.2 Content-Based Filtering

Content-based filtering operates similarly to collaborative filtering, but instead of recommending based on the sentiments of other users, it uses a user’s own data from the past. At a high level, the system essentially gets a feeling for the user’s taste using available data and makes recommendations that appeal to these tastes. When implemented, a content-based filtering system could recommend a user items that are similar to items they have expressed positive opinions about in the past (13).

A screen shot of a white background

Description automatically generated

Figure : A user's list of favourite movies.

Given a user’s list of favourite movies like that in figure 8, a Content Based Filtering System could recommend movies the user may like, from a larger list of movies. This can be manipulated in such a way that it could recommend a user songs that have similar audio features to songs already in their playlist.

Upon research, the content-based filtering approach was chosen as the approach for my application. Details about the planned implementation were described in section (SECTION HERE)

KNN & Cosine Similarity

## 2.5. Existing Final Year Projects

As part of my initial research, I reviewed Final Year Projects from years past, that pertain to my project. Two of those projects will be discussed here, as they have served as a source of inspiration for my own project.

### 2.5.1 Project 1: Moodify

**Project Title:** Moodify

**Student:** Louis Miguel Chavez

**Overview:**

This project is an Android application that classifies songs by mood, based on their lyrics and audio features, using AI and the Spotify API

The sentiment and music mood classifiers were made through Python, in Jupyter Notebook – other components were made in PyCharm. The Android application was made in Android Studio, using Java for functionality and XML for the layouts and visual components. A DigitalOcean Managed Cluster was used as the final database functionality.

The main strengths of this project, in my view, are the originality and accuracy of the mood classifier - it offers interesting insight into the listening habits of users. The functionality allowing for the classification of a song’s mood through its lyrics is equally original. I also find the front-end to be aesthetically pleasing and accessible for users of all technical skill levels. The only evident weakness of this project is that it requires users to manually upload playlist files for analysis, as opposed to having the client automatically extract them from the device – this is something I aim to avoid in my project.

**Similarities to my Project and Other Findings:**

This project and mine share appear to share clear similarities in the fact that they deal with the concept of ‘mood’ in music – however, they do so in very different ways. The main premise of this project is classifying the mood of a playlist using machine learning, while my project is more concerned with recommendation systems, and plays no role in determining user mood.

Where the projects are similar is in their use of machine learning models to classify music. This project contains a complex machine learning algorithm for sentiment classification, used to determine the mood of songs through sound and lyrics. While my project will not be used to determine the mood of a song, there is much I can learn from how the creator of this project constructed his machine learning model. A key piece of technology used by this student was **Spotipy**. As discussed in section 2.3, Spotipy is a Python library that can be used to interact with the Spotify Web API. I discovered Spotipy through this final year project and after conducting more research on it, decided to use it in my own.

### 2.5.2 Project 2: Football Data Mining, Result Prediction, and Visualization

**Project Title:** Football Data Mining, Result Prediction, and Visualization

**Student:** Yahia Ragab

**Overview:**

This project is a mobile and web application that allows users to view in depth stats on the English Premier League, and to view predictions of the outcome of matches, created with machine learning.

A huge amount of historical data relating to the English Premier League was mined and stored. A sophisticated machine learning model was developed to allow for predictions to be made, and a detailed front-end was designed that visualises statistics in digestible ways.

Python was used for creating the machine learning model, with use of libraries like NumPy, Pandas and Matplotlib for data visualisation. The web app was presented as a Django application, and Docker was used for deploying cloud components. MYSQL was used as the database management system.

This project contains a machine learning algorithm written in Python, similar in nature to the one I plan on creating in my app. The model is clearly highly sophisticated, and after rigorous testing seemed to yield very accurate results, which I believe is a huge strength of the project. The amount of data made available for viewing on the app is similarly impressive, and the way in which it is visualised is understandable and useful.

The front-end design is quite cluttered and perhaps not very user friendly and lacks some aesthetic appeal – it could have done a more thorough design phase that focused more on user experience. The project is, however, of a very high standard and displays many strengths which I could draw inspiration from.

**Similarities to my Project and Other Findings:**

This project differs from mine in subject matter, as it deals with football scores and results prediction. However, it utilises a machine learning model constructed in Python – something I plan on creating for my project too. What I gained most from researching this project was insight into how to evaluate machine learning models. This project was evaluated through a survey with potential users via a Usability Feedback Testing Form. In this form, users were asked to assess the accuracy of the results they received from the model, as well as their general sentiments about how useful it is. Information of this nature would be very useful to me in evaluating my final product, hence I took inspiration from this project when planning my evaluation – this will be discussed in greater detail in section 4.3.

## 2.6. Conclusions

After performing thorough research on existing literature in the domain of my project, I am prepared to enter the design and early development stages of my project. Many key decisions have been informed by information gathered through reviewing.

### 2.6.1 Requirements

While I always planned on making a Spotify music recommendation system, research on the biggest shortcomings of the Spotify recommendation algorithm influenced my decision to shift towards a mood-based recommendation system. Research on existing solutions to this issue revealed that while it’s an issue that’s been tackled often by developers, a perfect solution doesn’t exist and there is plenty of room for improvement in the domain.

### 2.6.2 Technologies

The decisions made on technologies to be used were heavily informed by comparative research conducted on viable options.

Python was chosen as a primary programming language after research, which influenced the choice of Flask as a web framework.

Researching past final year projects also aided in understanding how these technologies can be integrated into a larger system.

Content-based filtering was chosen as the approach to recommender system over collaborative filtering. It was seen as the most suitable option, as only user data will be available to the application, and not data of multiple users.

# 3. System Design

## 3.1. Introduction

This chapter describes the overall design of the project, including the visual style and layout of the frontend, and the structuring of the backend. It will also touch on the software methodologies adhered to during the development process.

## 3.2. Software Methodology

This project has combined multiple methodologies, using valuable and suitable elements of various common methodologies to establish an efficient and productive way of working.

The project has been performed iteratively, using the concept of “Sprints” commonly associated with the Agile and Scrum software development methodologies. The project adopted further principles commonly associated with Agile to enhance the flexibility and efficiency of work. Features of the Kanban methodology were also used, which focuses on visualising work and maximizing project efficiency. Each of these methodologies and the way in which they were used will be explained in this section.

## 3.2.1 Agile

Agile is a widely used software development methodology that defines rule that should be adhered to in the software development lifecycle (Figure 10). When adhered to, these rules make for more flexible development, focussed on continuous improvement and delivering working software that satisfies user needs (14)(15).

A diagram of a software company

Description automatically generated with medium confidence

Figure : The Agile Principles (15).

Some of the main principles of Agile, as described in the Agile Manifesto (15), are that changing requirements are welcomed, and working software is the primary measure of progress. These requirements are usually driven by the Product Owner – a member of an Agile team who puts forward the project objectives and represents the desires of the shareholders (16).

To adhere to the Agile framework, I undertook the role of Product Owner for this project. I did this by defining the requirements and deliverables of the project at an early stage. Changing requirements were welcomed, as they occurred due to development issues or external feedback from my supervisor. Keeping these defined requirements as the primary focus in development made for more efficient development. It also yields working software that can be evaluated throughout the project lifecycle, not only the interim stage.

Adherence to the Agile principles was further upheld by use of the Scrum methodology.

## 3.2.2 Scrum

Scrum is a software development framework commonly used by teams employing the Agile methodology. The main premise of Scrum is to develop software iteratively, to optimize predictability and control risk (17). This iterative development process is upheld through “Sprints” – these are 2/3 week long cycles of development, preceded by thorough planning of the work, and concluded with a retrospective analysis of work done and issues faced (17).

A diagram of a scrum

Description automatically generated

Figure : The Scrum Process (17)

Figure 11 illustrates the Scrum process of taking in requirements, ordering them by priority, completing a 2–3 week sprint of work, and reviewing the sprint. Scrum teams typically comprise of several members, such as developers, a Scrum Master, and a Product Owner.

For this project, I essentially assumed all these roles, and worked in one-week sprints. At the beginning of each week, I defined the tasks to be completed for the week ahead. At the end of each week, I performed a short retrospective, reflecting on the tasks completed and those that weren’t completed, identifying any issues or risks. Working in this way allows for greater predictability and makes risks easier to identify and mitigate.

## 3.2.3 Kanban

Like scrum, kanban is a development framework used to implement agile, but with a large emphasis on dividing up and visualising work into work items. These work items are visualised on a kanban board. Every work item involved in the project is represented on this board, usually in a “To do”, “Doing” or “Done” state. This allows all team members to understand the state of all work at any given time, and gives a clear idea of where the project is in its lifecycle and how much work remains (18).

A kanban board made in Azure DevOps was used in this project, to keep track of work easily and to better implement an agile style of work.

A screenshot of a computer

Description automatically generated

Figure : Project Kanban Board.

While scrum and kanban are often seen as two distinct options for development frameworks, it is possible, and often beneficial, to borrow parts of both in the development process (19). Kanban boards can make the scrum processes of planning work and reflecting on work done periodically easier, as they offer a visual description of work done and work remaining. This project made use of both kanban boards and scrum ceremonies & sprints.

## 3.3. System Architecture

This diagram illustrates a high-level architecture of the whole system, detailing how the server (flask application) interacts with the client and the requests made between them. It also describes the requests made by the app to the Spotify Web API. Note that for the prototype build, a large Spotify dataset from which recommendations are derived is stored locally in the server device’s file system – this is going to be exported to a database in the final application. This is discussed further in section 6.3.1.

A diagram of a software company

Description automatically generated

Figure : Software Architecture Diagram

## 3.4. Frontend

## 3.4.1 Use Case Diagram

This Use-Case Diagram illustrates the functionality of the system and how the user can interact with this functionality through the front-end.

A diagram of a software application

Description automatically generated

Figure : Application Use Case Diagram

Once they access the web app, users are prompted to log in via a Spotify authentication page. They can then select a playlist from a list of their playlists and have a recommendation playlist generated. After this they can then choose to save this playlist to their library or have it regenerated with new recommendations. The general flow of this functionality is described in a flowchart in section 3.4.2.

## 3.4.2 Flowchart

This flowchart describes the flow of all of a user’s possible interactions with the web app.A diagram of a software process

Description automatically generated

## 3.4.3 Wireframes

The following wireframes illustrate the initial design of the application screens. Each wireframe was designed using Nielsen’s Heuristics as a reference point and guide. These are 10 principles used for interface design, commonly adopted by developers and web designers. A particular focus was placed on keeping the design aesthetic and minimalist, per Nielsen’s 8th Heuristic (20).

A screenshot of a music player

Description automatically generated

Figure : Application Landing Page

Figure 11 shows the initial landing page of the web app, displayed after a user has authenticated their Spotify account from this page, a user has the option to access the homepage (the current page), an “about” page, or to log out of their Spotify account. The large button in the centre of the screen labelled “Choose a Playlist” will direct a user to the playlist selection screen.

A screenshot of a music player

Description automatically generated

Figure : "Select a Playlist" Top Half of page.

Figure 12 is the top half of the “Select a Playlist” screen. It displays each of the user’s playlists in a scrollable list, retrieved from the Spotify API. The user can select one playlist as the recommendation subject, via a radio button.

A screenshot of a computer

Description automatically generated

Figure : "Select a Playlist" Bottom Half of page.

Figure 13 is the bottom half of the “Select a Playlist” screen. Here the user may alter the mood variables via sliders which are encapsulated in an area labelled “Change the Mood”. This area also contains a button that generates the recommendations and redirects the user to the recommendations page.

A screenshot of a computer

Description automatically generated

Figure : "Your Recommendations" Screen.

Figure 14 illustrates the “Your Recommendations” screen. The generated playlist is displayed here as a scrollable list of songs. A user can choose to regenerate these recommendations, recommendations will refresh to a new set. Otherwise, they can save the recommendations directly to their Spotify library as a playlist.

## 3.5. Backend

# 4. Testing and Evaluation

## 4.1. Introduction

This section will describe the plans that have been made for the eventual testing and evaluation of the final application.

Testing and evaluation are a crucial part of the project development process. Thorough testing will be performed on the final application to discover and remove bugs, and to assess the general usability and performance of the application. This will involve end to end integration testing, unit testing, and black box user acceptance testing (UAT).

A thorough evaluation will be performed on the system too, to assess the accuracy of the recommendations generated by the recommender system. The evaluation methods will be heavily based on user feedback through surveys, and comparison with existing music recommendation systems.

## 4.2. Plan for Testing

The plan for testing can be divided into three distinct sections:

* Integration testing
* Unit testing
* User Acceptance Testing

### 4.2.1 Integration testing

Integration testing is carried out to validate the interoperability of software systems. It assures that subsystems of a larger system can interact with each other in the expected (21). By nature, integration testing is carried out not only at the end of the development process, but throughout it.

Whenever a component of the system is complete, it will be tested using unit tests and then integrated with other completed components to test its interoperability with them. This particular method of integration testing is called “incremental” integration testing (22). This incremental method of testing will allow defects to be easily identified in smaller components before they are integrated into the larger system, which can avoid complications later in the development process. Incremental testing also lends itself to the iterative nature of the agile methodology, particularly the principle which call for continuous attention to technical excellence and working software being the measure of progress (see Figure 10).

Key areas of my project which will benefit from integration testing include the module containing recommender model, the module containing the flask application, and the HTML templates. Testing the interoperability of these at the earliest possible stage will reduce the risk of bugs occurring later in development.

### 4.2.2 Unit Testing

Unit testing is performed on a small “unit” of code in isolation, to validate that it produces expected results for all possible inputs (23). Unit test cases are generally written up and executed by developers as the first part of the testing phase of the development process. Important functionality of my code will be unit tested thoroughly, using pytest.

#### 4.2.2.1 Pytest

Pytest is a python testing tool that allows small, readable unit tests to be written on python code (24). Pytest tests assert the expected result of a function or piece of code after execution (Figure 19) and return a pass or fail based on the real results (Figure 20).

A screenshot of a computer screen

Description automatically generated

Figure : Pytest test case (24)

A computer screen shot of a error

Description automatically generated

Figure : Pytest test failing (24)

Pytest will be used for my unit testing, as it is an easy way to test functionality of python code, that can be scaled up to support complex functionality. Unit test cases will be drawn up based on python code, and then tested extensively using pytest.

## 4.2.3 User Acceptance Testing

User acceptance (UAT) is the final stage of testing in application software development. It involves user testing of the system to find out if the solution is fit for use and meets their expectations (25). For my UAT, the expectation will be that the application can perform the functionality that was laid out in the project proposal.

Users will be presented with a description of the promised functionality of the system and will be asked to test this functionality. Feedback will be captured through a test scenario form (Figure 21) which will describe the functionality that needs to be tested. Users will test this functionality and mark a pass or fail on each test scenario.

A screen shot of a computer

Description automatically generated

Figure : Example Test Scenario Form (26)

Testing in this way will identify any discrepancies with how the system works and its intended functionality, while also offering users exposure to the system and the opportunity to offer feedback.

## 4.3. Plan for Evaluation

Evaluation of the system will primarily focus on evaluating the recommendation model and assessing the quality of recommendations. Evaluation will be carried out through user surveys. Surveys are the most popular method of software evaluation – studies have found that over 26% of software development projects use user surveys as their main evaluation method (27). Surveys can be an ideal way of garnering user opinions and getting a quantifiable measure of satisfaction with the system.

A group will be given access to the application, for use with their own Spotify account. They will be asked to generate recommendations for some of their playlists, and to fill in a survey that captures their sentiments about the recommendations. Discovering the general feelings of users about their recommendations will allow us to quantify the accuracy of the recommendation system, and the overall success of the project. Asking questions that identify areas of dissatisfaction can also highlight potential areas of future work.

As this project is aiming to address the shortcomings of the current Spotify recommendation system, it appears logical to compare the recommendations of my application with those of the Spotify application. Users will be asked to listen to Spotify recommendations of the same playlists, using Spotify’s “radio” functionality. Data will be gathered through a survey about which recommendations the user find to be more accurate, which will again serve as a measure of project success and may highlight areas for improvement.

## 4.4. Conclusions

Testing will play a key role throughout the development of this application. Incremental testing methods will be utilised to identify and resolve any problems at the earliest opportunity, and to assure the final product functions as expected for users in UAT.

Evaluation will play the equally important role of assessing user satisfaction with the system.

Testing and evaluation plans have been developed that will hopefully reduce the risk of problems occurring at the UAT stage and yield positive feedback from users.

# 5. Prototype Development

**As least 2 pages, but as many as you like (but lots of code samples).**

## 5.1. Introduction

This section will detail all development that has taken place in the project up to the interim stage. Code snippets and images of output will be used to describe the prototype application.

## 5.2. Prototype Development

The current prototype has delivered some of the main parts of the promised functionality. It is presented as a Flask web application, that displays the UI through simple HTML pages. It contains a functioning recommender system that utilises the Spotify Web API to gather user data and recommend songs. The development of this prototype will be explained in detail in the following sections.

### 5.2.1. Flask Application

#### 5.2.1.1. Spotify Authentication

The underlying functionality of the application is contained in a single python file. Upon accessing the application, a user is directed to a login screen - this is handled in a Flask app route

A screen shot of a computer

Description automatically generated

This route uses a function called “create\_spotify\_oauth()” to gather necessary information for login, and then uses this information to get the authorization URL using the spotipy function “get\_authorize\_url()”. The user is then directed to this URL.



A screenshot of a computer

Description automatically generated

#### 5.2.1.2. Retrieving User Data

The code for viewing playlists is defined in a route called /viewPlaylists, which the user is redirected to through a button on the frontend. This route contains a function view\_playlists() that creates an instance of the spotipy class called “sp” and uses its method “current\_user\_playlists()” to retrieve all a user’s playlists. These are then sent to the HTML template to be displayed.

A screenshot of a computer program

Description automatically generated

When a user chooses one of these playlists through the frontend, a list of all its tracks is displayed. This is handled through a route “/playlist”, containing a method “playlist\_page()”. This method takes the playlist id and using spotipy, retrieves the playlist and all its tracks. The songs are added to an array which are displayed on the frontend.

A screen shot of a computer code

Description automatically generated

A screen shot of a computer program

Description automatically generated

If the user chooses to generate recommendations for this playlist, they are redirected to the route “/getRecommendations”. This route contains a function “recommendations\_page()” which retrieves the mood variables that were defined through frontend interaction, via a POST request. It then calls the “get\_recommendations()” function and passes the playlist id and the mood variables. This function generates recommendations using the recommender model, which are displayed on the frontend.

A screen shot of a computer screen

Description automatically generated

### 5.2.2. Frontend

### 5.2.2.1. Jinja

When developing my application’s frontend, it became apparent that I needed some way to easily display data contained in python variables to the screen. Variables for songs and playlists are contained in Python arrays and dictionaries and need to eventually be presented to the user. This wasn’t easy to do in JavaScript as one may usually do, as JavaScript doesn’t have great interoperability with Python applications.

Upon research, I found that Jinja provided similar functionality to JavaScript, but is far more compatible with Python. The official Jinja documentation describes it as a templating engine that provides placeholders in templates that allow code writing similar to Python (28). Using Jinja in my HTML templates allowed for Python variables to easily be interpreted by the HTML template and displayed to the screen. Figures 22 and 23 show an example of this in my application.

A white background with black text

Description automatically generated

Figure : Python variable shown on screen using Jinja.

A black screen with text and symbols

Description automatically generated

Figure : Jinja code in the HTML template.

### 5.2.2.2. HTML Templates

The prototype contains 5 distinct HTML pages –

* home.html
* error.html
* playlist\_list.html
* playlist.html
* recommendations.html

The purpose and functionality of each one will be described in detail in this section.

**home.html**

This page serves as the landing page for the web app, where a user will be directed once they have completed Spotify authentication. In the prototype, it has little functionality other than presenting a link to the “view playlists” page, but it will eventually serve as a home page with information about the website and website navigation tools.

A white background with black text

Description automatically generated

Figure : home.html in the browser

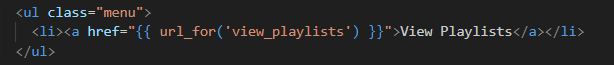


Figure : home.html code for linking to view\_playlists route

**error.html**

Users are redirected to this page when the flask application throws an error – these are usually due to Spotify authentication issues or users trying to access deleted playlists.

A white background with black text

Description automatically generated

Figure : error.html in the browser.

A black background with white text

Description automatically generated

Figure : passing error text in python

A black screen with text and symbols

Description automatically generated

Figure : displaying error text in HTML/Jinja

**playlist\_list.html**

A screenshot of a computer

Description automatically generated

Figure : playlist\_list.html in the browser.

This page presents the user with a list of their public playlists as clickable links – these are retrieved from the list of playlists that were passed to this template, from the main python code.

A screen shot of a computer code

Description automatically generated

Figure : HTML/Jinja functionality to display playlists.

Clicking on any one of these links redirects the user to the app route “playlist\_page”, passing the id of the selected playlist.

**playlist.html**

This template presents a list of the songs in the selected playlist, and the mood sliders. The songs are retrieved from the variable “psongs” that was passed to the template. The sliders are contained within a HTML form, so they can be easily submitted to the recommendations page. It also contains the “generate recommendations” button, which redirects the user to the route “getRecommendations”, passing the playlist id and the variables chosen from the mood sliders.

A screenshot of a music list

Description automatically generated

Figure : playlist.html in the browser.

A computer screen shot of text

Description automatically generated

Figure :HTML/Jinja functionality to display songs.

A screen shot of a computer code

Description automatically generated

Figure : HTML Sliders

**recommendations.html**

This page presents 40 song recommendations in a HTML table, with 2 columns for artist and song

A white text on a white background

Description automatically generated

Figure : recommendations.html in the browser.

A computer screen shot of a program code

Description automatically generated

Figure : HTML/Jinja functionality to display recommendations.

These recommendations are retrieved from the recommendations dictionary, which as passed to the template in the python code.

### 5.2.3. Recommender System

This section discusses the development of the song recommender system. As discussed in section 2.4, the recommender system implements content-based filtering. At the interim stage, this was done using cosine similarity – K nearest neighbours or other algorithms will be implemented going forward, to enhance recommendations.

The recommender system was initially developed in a Jupyter Notebook before being implemented into the flask web app – images of output below are taken from the Jupyter Notebook.

The first step in this algorithm, like many machine learning algorithms, is data preprocessing. The user’s songs are extracted from the selected playlist, and the required features are gathered for each song. They are then added to a pandas dataframe. Duplicate song ids are dropped, so no song has its audio features considered more than once.

A computer screen shot of a program

Description automatically generated

Figure : getting playlist features.

A screenshot of a computer screen

Description automatically generated

Figure : playlist features dataframe.

A similar dataframe “all\_songs\_features” is then created, using data from the “all songs dataset” – this dataframe describes the features of all the songs in our large song dataset. The two dataframes “all\_songs\_features” and “playlist\_features” contain the same features, so they can be easily compared.

A second dataframe is used also created called “all\_songs\_df”. This is another similar dataframe, but containing song titles will be used to draw recommendations from.

A computer screen with text

Description automatically generated

Figure : creating new dataframes for all songs.

A screenshot of a black screen

Description automatically generated

Figure : all\_songs\_df contents.

A screenshot of a computer

Description automatically generated

Figure : playlist\_features content.

As each row of the the “all\_songs\_features” dataframe essentially represents a vector, the next task is getting a single vector that represents the entire user playlist. This is needed to allow cosine similarity to be calculated between this vector and every song in the matrix. This is done by simply dropping the id column and adding the remaining features together. Playlistfeatures now represents our entire playlist, but encapsulated in a single vector, or “song” that can be compared to other songs.

A screenshot of a computer screen

Description automatically generated

Figure : dropping duplicate ids.

A screenshot of a computer

Description automatically generated

Figure : playlistfeatures contents.

The values set by the user in the mood sliders are then applied to the playlist features, so they can be accounted for in the recommendations.

A screen shot of a computer code

Description automatically generatedA computer screen with numbers and text

Description automatically generated

Figure : applying the mood sliders adjustments.

Figure : contents of playlistfeatures before & after mood is applied.

We can now perform cosine similarity between the dataframe of all songs, and our playlist\_features vector. We return the 40 highest ranking songs, based on similarity, to a dictionary.

A computer code on a black background

Description automatically generated

Figure : calculating cosine similarity.

recommendations\_dict is then simply passed to the HTML template to be displayed.



Figure : passing recommendations to the HTML template.

# 6. Issues and Future Work

## 6.1. Introduction

This chapter will describe any issues any potential risks that have been identified in the project so far, and methods of risk mitigation and management that have been put in place to deal with these risks. It also outlines a plan for completing the remaining work in the project, including a GANTT chart.

## 6.2. Issues and Risks

Multiple potential risks to the delivery of the project were identified during development, and appropriate mitigation techniques have been put in place to reduce risk. These risks will be discussed in this section.

### 6.2.1. Scope Creep

Project scope creep is essentially the tendency of projects to extend beyond its initial boundaries (29). If a project’s requirements increase over its lifecycle, it can become a threat to project delivery, as not all requirements may be delivered on time.

To eliminate the potential for scope creep in my project, I outlined a very clear project scope (see section 1.4) and set of requirements (see section 1.3) that will be strictly adhered to and not expanded throughout project lifecycle. The creation of a GANTT chart also creates a timeline to accompany project scope, withing which all development, testing and evaluation should be completed.

Clearly defining project scope and strictly adhering to the timeline set out by the GANTT chart should eliminate any risks associated with project scope.

### 6.2.2. Unexpected Evaluation Results

An obvious risk of my application is that it doesn’t yield positive results in the evaluation, i.e., users don’t find their recommendations to be satisfactory. To mitigate this risk, testing and evaluation has been allocated three weeks of time – if evaluation in the first week gives unsatisfactory results, the second week will be reserved to learn from user feedback and refine the application before a second evaluation in the third week. Thoughtful planning has allowed time for this process, all before the submission deadline. By doing this, the risk of submitting an application that wasn’t received well by users is mitigated.

This is however a contingency plan, and should the evaluation produce satisfactory results on the first attempt, the following two weeks will be used for more thorough evaluation and other housekeeping.

### 6.2.3. Integration Risks

When developing multi-tiered software, there is an inherent risk that different technologies will not integrate as expected, which could cause setbacks in the project lifecycle as these issues are addressed.

This risk has been mitigated using thorough research, as technologies were chosen that are often used together and have a track record of good interoperability. For example, Flask was chosen as the web framework due to its compatibility with the Python language, which the recommender system will be written in. Both are also compatible with Spotipy, the Python library for the Spotify Web API.

Furthermore, time has been allocated in the GANTT chart for integration testing after each major component has been developed, which will remove the potential for problems relating to integration late in the project lifecycle.

### 6.2.4. External Risks

External risks is an umbrella term used to encompass any risks not related to project development, such as lost files or broken hardware.

These two risks have been mitigated by using cloud storage for important project files – all the code and data have been saved to OneDrive, allowing them to be accessed case the local versions become inaccessible.

All code has also been uploaded to a remote GitHub repository which allows code to be reverted to previous versions. In a situation where important code is deleted or the application stops functioning, the project can be rolled back to a previous version that is known to be functional.

## 6.3. Plans and Future Work

The main areas of work are yet to be completed are:

* Implement the database.
* Improve the frontend.
* Enhance the recommender system.
* Testing and evaluation

### 6.3.1. Database

As described in the prototype development, the Spotify songs dataset is currently stored locally on the machine where the app resides. This is only suitable for the app while it’s in development, as it requires every user of the app to have this large dataset saved to their device. To address this issue, this dataset needs to be exported to an external database. By doing this, users will be able to use the full functionality of the application without having to interact directly with the dataset.

Going forward, research will be conducted on viable DBMS solutions, and a suitable option will be chosen and implemented.

### 6.3.2. Frontend

In the prototype development, a large portion of development time was allocated to having the complex functionality of the application working, such as the Spotify authentication and the basic recommender system – less time was allocated to the frontend development, which was considered a lower priority goal. For this reason, the current frontend is very basic, and does not resemble the design illustrated in section 3.4.3. The frontend will be improved greatly in the remainder of the development process, so it reflects the design that has been set out, and to deliver the desired user experience.

Some more frontend elements also need to be implemented to allow the full functionality of the application to work – the “mood” sliders need to be implemented to allow the mood-based recommendation functionality to work as intended.

For these reasons, a substantial amount of time will be allocated to frontend development going forward.

### 6.3.3. Recommender System

The recommender system developed at the interim stage does not deliver all the promised functionality yet – namely, it doesn’t account for mood. Generating mood-based recommendations is the primary functionality of he project and is seen as the highest priority goal. For this reason, it will be the first area of work tackled after the interim stage, to allow ample time for coding, testing, and addressing any issues that may arise in the development process. The coding will consist of implementing frontend elements that allow users to alter the mood of recommendations, and underlying code that adds these variables to the features derived from the user’s playlist before they are passed into the recommender system.

The accuracy of the recommendations is also not yet at the desired level. While the generated recommendations are accurate with regards to audio features, they often fall into wildly different genres, or are even in different languages, than the user’s taste. To address this issue, the recommender system will be enhanced to account for the genre of music. The genres of songs are not currently being represented by a feature in the recommender system – this will be amended by using “one hot encoding” on the genres. One hot encoding involves representing categorical values, like “genre” in our case, as numerical values so they can be interpreted easily by machine learning models (30). By accounting for genre in generating recommendations, the system will hopefully achieve a greater level of accuracy. One hot encoding will be thoroughly researched, and eventually implemented going forward.

### 6.3.4. Testing and Evaluation

Testing and evaluation have not yet been dealt with at the interim stage. As more components are developed, time will be allocated for incremental integration testing, the benefits of which were discussed in section 4.2.1. The testing and evaluation plan outlined in section 4 will be completed at appropriate times in the development cycle. The times at which testing will take place are detailed in the GANTT chart.

### 6.3.5. GANTT Chart

The following GANTT charts outline a planned timeline of work, up to the interim stage, and from the interim stage until final submission.

### A graph with different colored squares Description automatically generated with medium confidence6.3.5.1. Until Interim Submission

### A screenshot of a computer Description automatically generated6.3.5.3. Until Final Submission

# Bibliography

1. Statista [Internet]. [cited 2023 Oct 31]. Music streaming services subscribers market shares 2022. Available from: https://www.statista.com/statistics/653926/music-streaming-service-subscriber-share/

2. Dachs P. Hear what I’ve found — A survey on music discovery [Internet]. The Sound of AI. 2019 [cited 2023 Oct 31]. Available from: https://medium.com/the-sound-of-ai/hear-what-ive-found-a-survey-on-music-discovery-3869b6b6cbe4

3. Ding Y, Liu C. Exploring drawbacks in music recommender systems : the Spotify case [Internet]. 2015 [cited 2023 Oct 31]. Available from: https://urn.kb.se/resolve?urn=urn:nbn:se:hb:diva-8690

4. Leight E. It Started as a College Project. Now It’s the Music Industry’s Hottest Recommendation Tool [Internet]. Rolling Stone. 2022 [cited 2023 Nov 1]. Available from: https://www.rollingstone.com/music/music-features/discz-tiktok-recommendation-app-1324785/

5. Guttag BL with J. Program Development in Java - Abstraction, Specification, and Object-Oriented Design. USA, Addison Wesley; 2000.

6. Business Insider [Internet]. [cited 2023 Nov 7]. The 10 most popular programming languages, according to the “Facebook for programmers.” Available from: https://www.businessinsider.in/the-10-most-popular-programming-languages-according-to-the-facebook-for-programmers/articleshow/66253927.cms

7. Java Vs Python - Key Differences Between Java And Python [Internet]. [cited 2023 Nov 7]. Available from: https://www.softwaretestinghelp.com/java-vs-python/

8. Python (programming language) ~ Information Technology ~ 2420 ~ kelas-karyawan-bali.kurikulum.org [Internet]. [cited 2023 Nov 7]. Available from: http://kelas-karyawan-bali.kurikulum.org/IT/en/2420-2301/Python\_3721\_kelas-karyawan-bali-kurikulumngetesumum.html

9. Welcome to Spotipy! — spotipy 2.0 documentation [Internet]. [cited 2023 Nov 7]. Available from: https://spotipy.readthedocs.io/en/2.22.1/

10. Chisnall D. The Challenge of Cross-language Interoperability: Interfacing between languages is increasingly important. Queue. 2013 Oct 8;11(10):20–8.

11. Django Project [Internet]. [cited 2023 Nov 7]. Django. Available from: https://www.djangoproject.com/

12. Copperwaite M, Leifer C. Learning Flask Framework. Packt Publishing Ltd; 2015. 250 p.

13. B.Thorat P, M. Goudar R, Barve S. Survey on Collaborative Filtering, Content-based Filtering and Hybrid Recommendation System. Int J Comput Appl. 2015 Jan 16;110(4):31–6.

14. Atlassian. Atlassian. [cited 2023 Nov 11]. What is Agile? Available from: https://www.atlassian.com/agile

15. agile-manifesto-download-2019.pdf [Internet]. [cited 2023 Nov 11]. Available from: https://www.agilealliance.org/wp-content/uploads/2019/09/agile-manifesto-download-2019.pdf

16. Sverrisdottir HS, Ingason HT, Jonasson HI. The Role of the Product Owner in Scrum-comparison between Theory and Practices. Procedia - Soc Behav Sci. 2014 Mar;119:257–67.

17. Sachdeva S. Scrum Methodology. Int J Eng Comput Sci [Internet]. 2016 Jun 6 [cited 2023 Nov 11]; Available from: http://ijecs.in/issue/v5-i6/11%20ijecs.pdf

18. Atlassian. Atlassian. [cited 2023 Nov 11]. Kanban - A brief introduction. Available from: https://www.atlassian.com/agile/kanban

19. Kniberg H, Skarin M. Kanban and Scrum - Making the Most of Both. Lulu.com; 2010. 122 p.

20. Gonzalez-Holland E, Whitmer D, Moralez L, Mouloua M. Examination of the Use of Nielsen’s 10 Usability Heuristics & Outlooks for the Future. Proc Hum Factors Ergon Soc Annu Meet. 2017 Sep 1;61(1):1472–5.

21. Tsai WT, Bai X, Paul R, Shao W, Agarwal V. End-To-End Integration Testing Design. 2001. 166 p.

22. Educative [Internet]. [cited 2023 Nov 15]. What is incremental testing? Available from: https://www.educative.io/answers/what-is-incremental-testing

23. Olan M. Unit testing: Test early, test often. J Comput Sci Coll - JCSC. 2003 Jan 1;19.

24. pytest: helps you write better programs — pytest documentation [Internet]. [cited 2023 Nov 15]. Available from: https://docs.pytest.org/en/7.4.x/

25. Ganesh K, Mohapatra S, Anbuudayasankar SP, Sivakumar P. User Acceptance Test. In: Ganesh K, Mohapatra S, Anbuudayasankar SP, Sivakumar P, editors. Enterprise Resource Planning: Fundamentals of Design and Implementation [Internet]. Cham: Springer International Publishing; 2014 [cited 2023 Nov 15]. p. 123–7. (Management for Professionals). Available from: https://doi.org/10.1007/978-3-319-05927-3\_9

26. Rana K. Test Scenario | Definition, Template & Examples [Internet]. ArtOfTesting. 2019 [cited 2023 Nov 15]. Available from: https://artoftesting.com/test-scenario-examples

27. Paz F, Pow-Sang J. A systematic mapping review of usability evaluation methods for software development process. 2016 Jan 1;10:165–78.

28. Introduction — Jinja Documentation (3.1.x) [Internet]. [cited 2023 Nov 25]. Available from: https://jinja.palletsprojects.com/en/3.1.x/intro/

29. Ajmal M, Khan M, Al-Yafei H. Exploring factors behind project scope creep – stakeholders’ perspective. Int J Manag Proj Bus. 2019 Nov 22;ahead-of-print.

30. Gefferth A. Medium. 2023 [cited 2023 Nov 18]. One Hot Encoding scikit vs pandas. Available from: https://towardsdatascience.com/one-hot-encoding-scikit-vs-pandas-2133775567b8