The Story of Us,

Taylor Swift Mix

The app that generates a Taylor Swift Playlist that perfectly sums up your last relationship.

MAY 2022

CFG – SOFTWARE 2 - PROJECT 2
Authored by: [Purvi Thakkar, Aude Vernet, Caroline Sautter]

REPORT ROADMAP

INTRODUCTION

Aim and Objective of the Project:

Aim and Objective of the Project:

Time and generations have changed immensely. Before 1990, we bought cassettes to listen to music. From 1995-to 2000, we downloaded songs from the internet to listen to music.

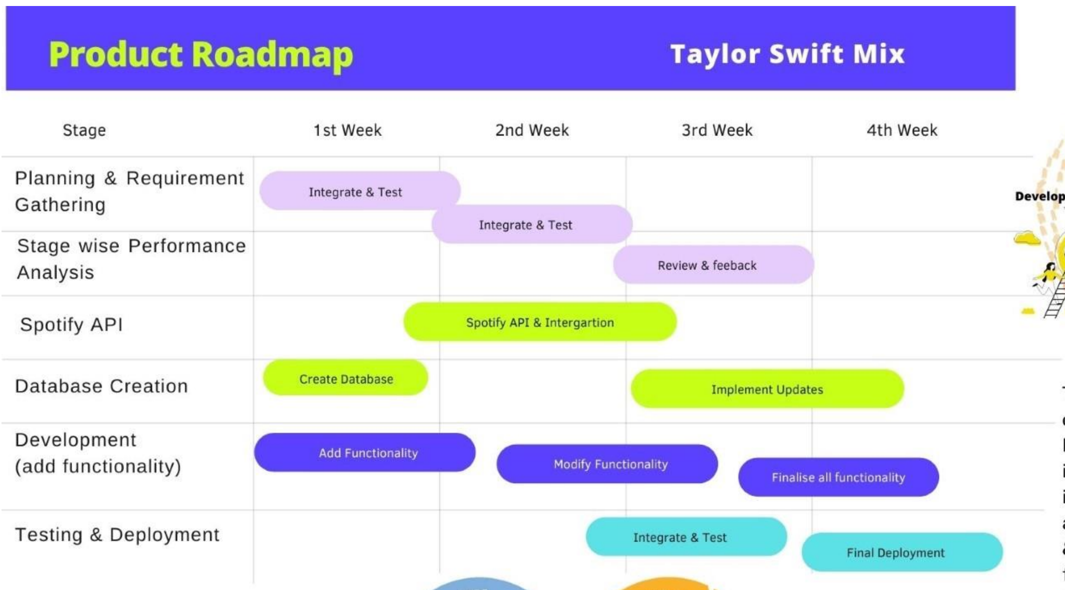
In the 2010s, we started using the applications to listen to music. According to Goldman Sachs, the music industry will increase from $62 billion in 2017 to $131 billion in 2030. This clearly explains we have little time to pick the right song/singer we want to hear in the right mood at the right time from the 100 million songs worldwide.

Taylor Swift is one of the most famous artists, partly due to her vast discography of famously relatable songs written about different sorts of relationships. You can create a perfect playlist based on only Taylor Swift songs for any relationship, past or present.

We at Taylor Swift Mix have developed an application that will take users' input to create a playlist that perfectly sums up their feelings about a past or current relationship, using Spotify API. Our listeners and creators want to deliver innovation to create the best user experience possible.

This app speeds up the process of creating that perfect playlist and is also fun to use, with a young, modern user interface and tone.

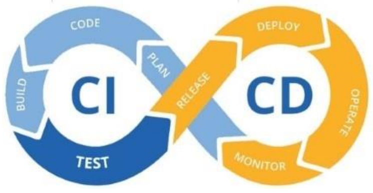
This report contains the technical and non-technical requirements for the use of the app, instructions, design and an overview of how we worked together to build the app. We give details of our working methods, plans, challenges faced, and reflections on the process.



Product Roadmap — Taylor Swift Mix

Stage	1st Week	2nd Week	3rd Week	4th Week
Planning & Requirement Gathering	Integrate & Test	Integrate & Test		
Stage wise Performance Analysis			Review & feeback	
Spotify API		Spotify API & Intergartion		
Database Creation	Create Database		Implement Updates	
Development (add functionality)	Add Functionality	Modify Functionality		Finalise all functionality
Testing & Deployment			Integrate & Test	Final Deployment

The team was focused on Continuous Integration which involved frequent integration of code into a shared repository. (CI & CD process was followed several times a day)



SPECIFICATIONS AND DESIGN

Requirements Technical and Non-Technical

Our project consists of the following aspects:

- A front end was created using Flask and some HTML
- An SQL database containing categorized Taylor Swift songs
- Python code collects user responses to a set of questions, which generates an SQL query depending on their answers.
- Connection to the SQL database, passing the query and retrieving a list of songs.
- Use of the Spotify API. This includes user authentication for the app to be able to create and modify playlists on their own personal accounts. The list of songs is then passed to the API to generate the playlist.

We aim to use what we learned on the Nanodegree to develop this project. Therefore, we mainly used the programming language we had learned over the last few weeks; MySQL and python. However, we also incorporated elements of HTML to develop a simple front end for our project. The following requirements are used:

Tools and libraries

We have used the following technologies to develop our web application:

- **FrontEnd:**
  - HTML
  - CSS
  - Bootstrap - version 5.0
- **Back-End:**
  - Python 3.9
  - Flask
  - Requests
  - mysql-connector-python
- **Database:**
  - MySQL
- **API:**
  - Spotify API

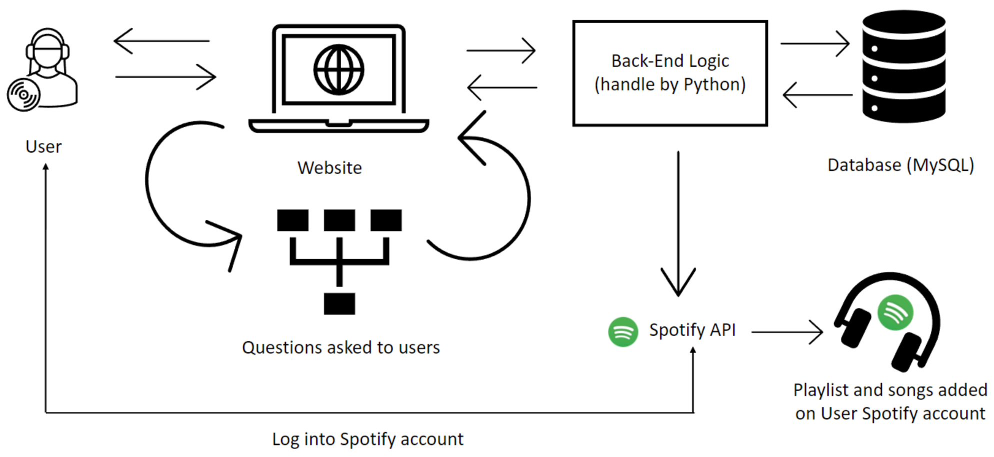
We used the following tools during the project:

- MySQL workbench
- PyCharm & Visual Studio Code
- GitHub
- Spotify API
- Trello board
- Slack for communication
- Postman

Design and Architecture

- **Simple front-end website. This contains:**
    - A landing page with buttons to start the questions.
    - A page for each question; there are 4 questions in total.
    - A page displays the playlist with buttons to authenticate users' Spotify accounts.
    - An ""about"" page
    - A feedback page to capture user reviews.
    - A simple design. The tone of the application should reflect the fan base it is aimed at, e.g., young and well established online.
- **Questions for users and collection of answers**
    - There is multiple choice, created using Flask submit buttons.
- **Back-end logic**
    - Takes the answers given by users and turns them into variables.
    - These are passed into a SQL query.
    - Able to connect to a SQL database and pass the query to the database.
- **Database in MySQL**
    - Normalized database that contains Taylor Swift songs, categorized by vibe, subject and time
- **Spotify API**
    - The application will redirect the user to a Spotify authentication page to allow the app to modify the user playlist.
    - After authentication, the user will be redirected to our web application.
    - Appropriate measures will need to be taken to ensure that this is secure. Spotify has its own methods for protecting its users, so by linking to them, we can take advantage of these features.
    - The application uses the API to create a playlist on the user's Spotify account based on the SQL query

System design



User Workflow:

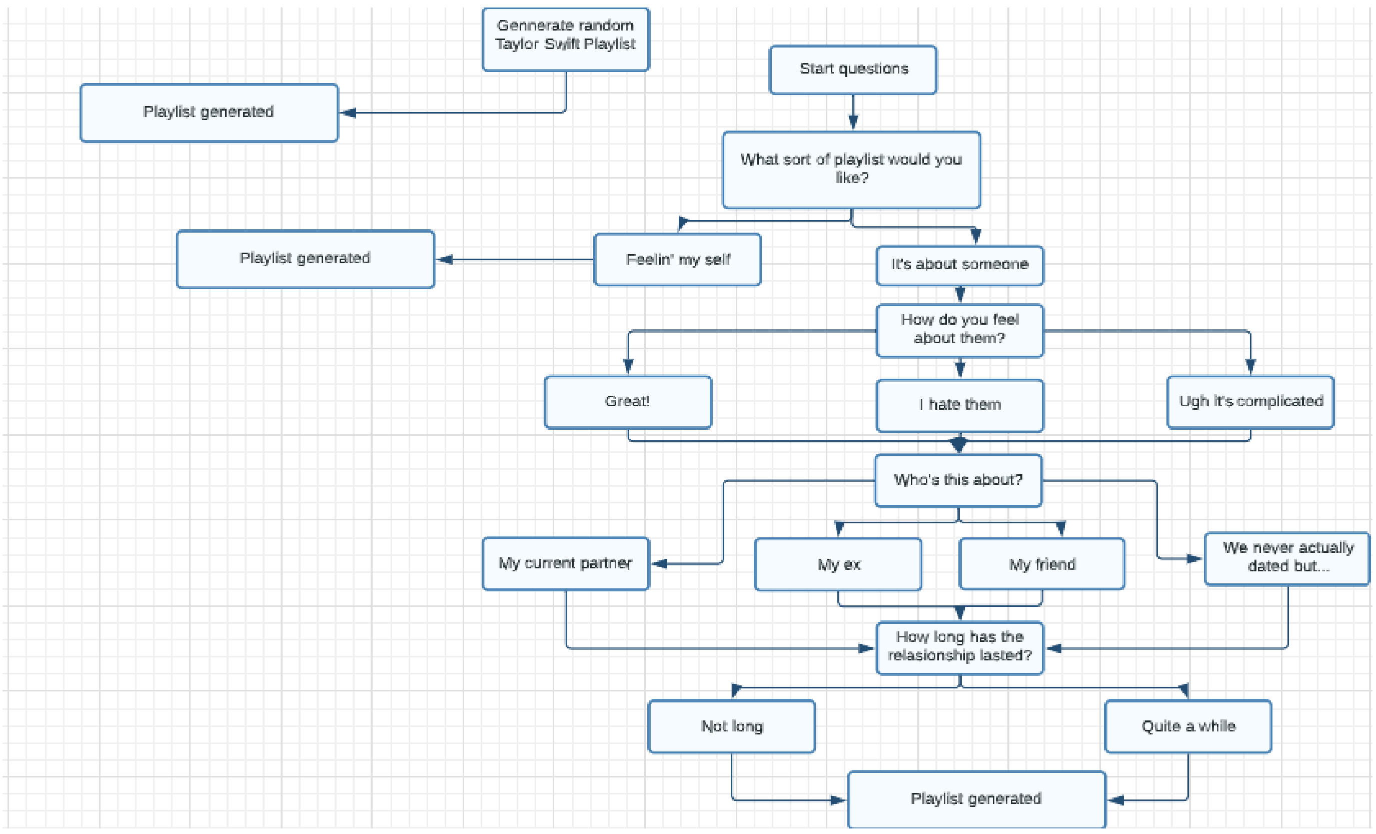
1. Go to the website
2. Answer series of question
3. View your playlist
4. Connect to Spotify (or another music platform stream)
5. New playlist added on user Spotify account
6. Enjoy the music!

Application workflow:

1. Questions display
2. Get answer from user
3. Python handle the logic according to answers
4. Python connects to database
   4.1. Create query
   4.2. Fetch data from database
   4.3. Return list of songs
5. Python calls Spotify API
   5.1. User redirected to Spotify authentication
   5.2. User logs into their Spotify account
   5.3. Create a playlist and add songs to the users Spotify account
   5.4. Send back the user to the website
6. Display list of songs to user on website

Question Workflow

The user experience was very important to us, so the questions that we asked the users had to be right. Below is a flow chart of how the user flows through the question process to generate a playlist.



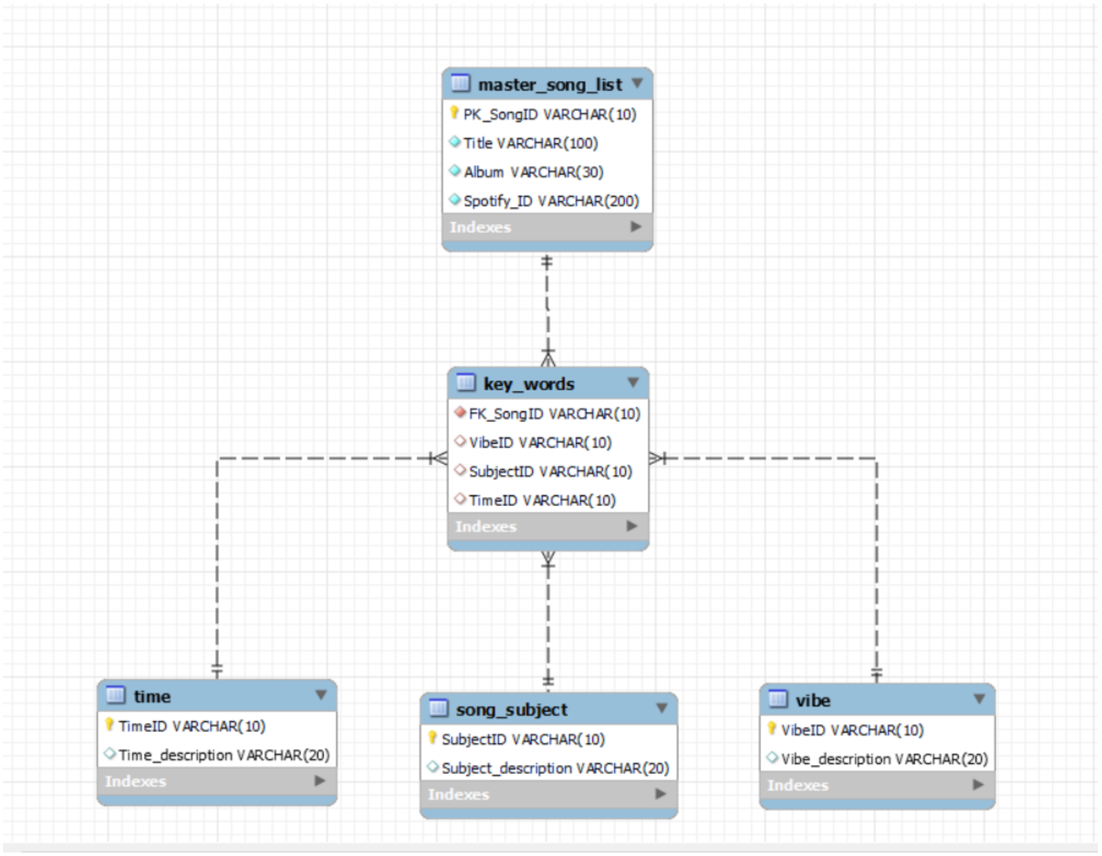
We incorporated all the lessons of the nanodegree in our project. This firmly informed our design decisions when we were selecting features. For example, we wanted to learn more about the various APIs, so we decided to get to grips with the Spotify one. We also needed to use Python and MySQL, which is why we chose MySQL to create the database.

Backend

Where possible, we tried to follow SOLID Principles of object-oriented programming. Following the Single Responsibility Principle, we aimed to create classes that only did one thing. For example, we started separate classes for database connection (connect to the SQL database) and database management (manage queries).

We also were conscious of the Open-Closed Principle. We used child and parent classes to allow the app to expand if needed. For example, we created a Taylor Questions Class as a child of a Questions class. This means that if we want to add other artists and ask different questions in different formats, we can without changing the original class.

We created a normalized SQL database using primary and foreign key constraints, allowing us to list one song in multiple categories. We made sure data dependencies made sense by only storing related data in the same table (e.g. the types of ""vibe"", ""subject"" and ""time"" were stored in different tables). The ""time"" table has not yet been integrated into the front end, but we are planning to do so in the near future.

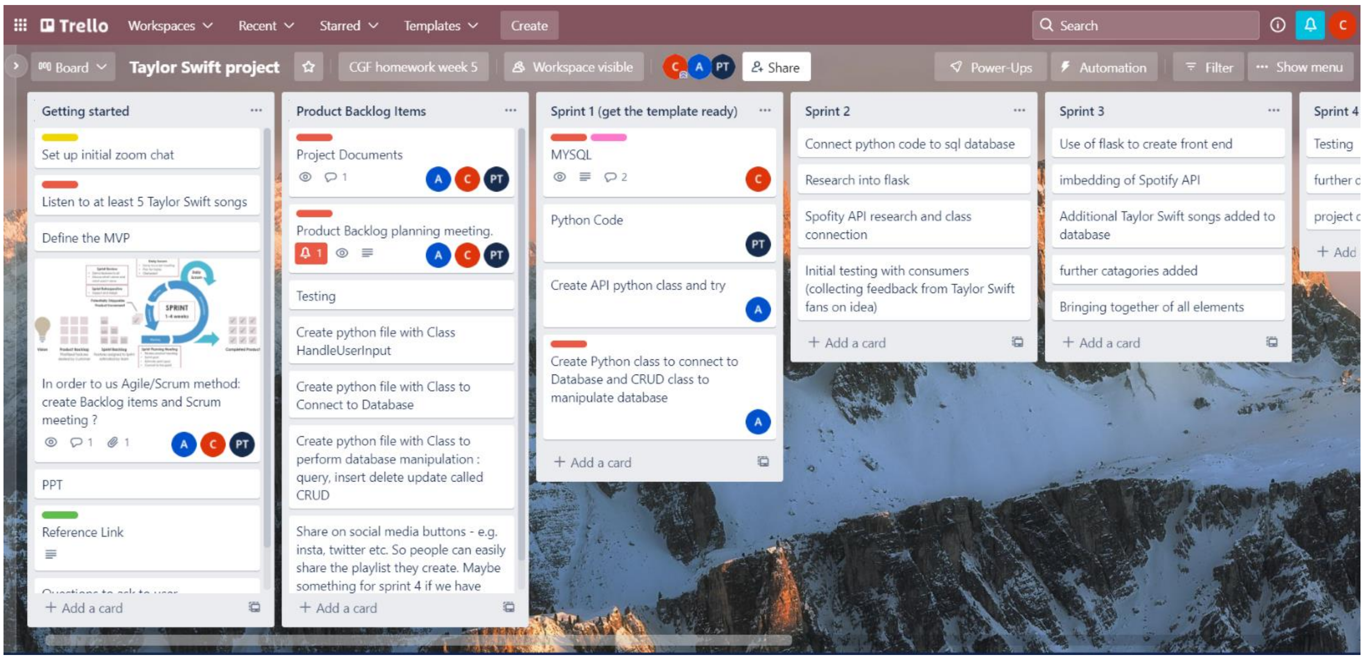


Frontend

Although not the focus of the course, we knew we wanted to create an app that would appeal to the target audience, not just technically but also on a surface level. Therefore, when designing the front-end, we ensured the tone and look of the app were fun, colloquial and modern to appeal to Taylor Swift fans.

IMPLEMENTATION AND EXECUTION

We followed agile methodology and split the app's development into sprints, which we planned on our Trello board. We also use the Software Development Life Cycle approach to implement your project.


Taylor Swift project

Getting started
- Set up initial zoom chat
- Listen to at least 5 Taylor Swift songs
- Define the MVP
- In order to us Agile/Scrum method: create Backlog items and Scrum meeting ?
- PPT
- Reference Link

Product Backlog Items
- Project Documents
- Product Backlog planning meeting.
- Testing
- Create python file with Class HandleUserInput
- Create python file with Class to Connect to Database
- Create python file with Class to perform database manipulation : query, insert delete update called CRUD
- Share on social media buttons - e.g. insta, twitter etc. So people can easily share the playlist they create. Maybe something for sprint 4 if we have

Sprint 1 (get the template ready)
- MYSQL
- Python Code
- Create API python class and try
- Create Python class to connect to Database and CRUD class to manipulate database

Sprint 2
- Connect python code to sql database
- Research into flask
- Spofity API research and class connection
- Initial testing with consumers (collecting feedback from Taylor Swift fans on idea)

Sprint 3
- Use of flask to create front end
- imbedding of Spotify API
- Additional Taylor Swift songs added to database
- further catagories added
- Bringing together of all elements

Sprint 4
- Testing

Sprint one

Plan:

- Flesh out the idea for the project, including minimum viable product and backlog of possible future features.
- Undertake a SWOT analysis of the group and team members, including opportunities for professional development and assign work accordingly.
- Establish templates of all the main backend features, including SQL database, database connection, and backend logic of playlist generation. Create a group GitHub repo.

Actions:

The sprint started very well. We quickly established our critical aims for the project and what the minimum viable product should look like. This was captured on our Trello board, the primary tool for all our planning and project management. By the end of the week, we had set up the SQL database with two of Taylor Swift'sSwift's albums and categorized all the songs by subject and vibe. We had a user input program in python and had written the code to connect the python program to the database.

We also established our team steams strengths and weaknesses. Purvi is very familiar with Agile, so she gave the group a training session on Agile working methods, which we then used to plan our project. Aude is the most experienced coder in the group, so she leads on the most challenging aspects of the code (such as the Spotify API). She then taught what she had learned to the rest of the group. Caroline has previous experience with SQL and leads the creation of the SQL database. As the group's biggest Taylor Swift fan, she gave initial ideas for song categorization and questions, which were further developed by the group. She then created a simple python program to show how the questions should appear to users. She also has a strong understanding of the fan base, so she was able to guide the direction of the product.

Challenges:

Once, the challenge we faced was with the Spotify API. We found it to be more complicated than the previous APIs we had used. Therefore, we could not complete the code to connect the API to the program in week one. This thus became a priority in sprint two.

Other challenges we faced were:

- How to structure files and folders.
- To follow naming convention across the code.
- Choosing the correct libraries and extensions
- Integrating different extensions, installing them, and running them successfully in all team members' systems.

Sprint two

Plan:

- Connect the python code to the SQL database
- Research Flask in preparation for front end development
- Further research of Spotify API and writing code to use it
- Initial testing with potential consumers

Actions:

In sprint two, we wanted to combine the different aspects that had been developed in stage one, which was successful. We also were able to make a start on the code to use the Spotify API, including user authentication, creation and management of playlist. A significant aspect of week two was research and learning. Aude's familiarity with coding enabled her to delve into the Spotify API first. She then taught what she had learned to the rest of the group. The group also researched Flask ahead of the front-end build in. Purvi furthered her project management by firming up our working methods and capacity planning, which she wrote up for the week 2 homework. Caroline's responsibility was to articulate the aim and direction of the product for the week 2 homework and pull it together into a succinct paper.

We also tested the idea of the product with potential users via social media. From this, we gathered feedback that the songs needed further categorization to fully achieve the aim of creating a playlist that adequately summed up a relationship. We put this into action in sprint 3.

Challenges:

The additional week two homework had not been planned for at the start of the project, so this sprint's time management was slightly complex. However, it did help with fully rounding out our idea for the project and what we would need to do to accomplish it.

Sprint three

Plan :

- Use Flask and HTML to create front-end
- Add additional songs to the database and an extra layer of categorization.

Actions:

This sprint was focused on bringing in Flask. Caroline took this as a development opportunity and had a go at creating an initial web form on Flask, while Aude used her prior knowledge to build the main app.py file. Aude was able in to incorporate features that had not been learned on the course, so it was agreed that Aude'ss code would be used as a base, with Caroline'scode copied in where relevant.

Challenges:

This week we faced several significant challenges. Purvi's commitments outside of the course were hectic for the sprint, but she managed to work on the feedback buttons on Flask and the front-end on HTML. Caroline faced technical difficulties with Flask, meaning that the original plan for her to merge the code Aude had written was significantly delayed. This was, however, resolved at the start of Sprint 4.

Sprint four

Plan:

- Add additional questions to the user interface
- Testing and bug fixing
- Report drafting
- Further work on the front end

Actions

The final sprint was mainly focused on testing, bug fixing and report drafting. However, we were also able to upgrade the user interface. As the most experienced coder, Aude led much of the coding for the app, teaching and delegating tasks to the rest of the team, where they went beyond the Nanodegree course content. Purvi used Aude's suggestion to develop a feedback page, while Caroline added the additional user question and further worked on the HTML templates. The additional user question will be added at a later stage.

Although we had been debugging as we went while writing the code, this was the sprint that we thoroughly considered testing, and we developed unit tests and used third-party apps where we could. Further details are addressed in the sprint four challenge and testing and evaluation sections below.

While Aude's focused on the code itself, Purvi and Caroline developed a first draft of the paper together. Aude then shared her comments and provided further contributions.

Challenges:

We faced significant challenges this week, in particular with the Spotify API. As the API was using the free development platform created by Spotify, we realized that we would be unable to share the app without manually adding new users to the development dashboard, which is limited to 25 people. Being able to distribute the app is something we aim to do in future but was out of the scope of the current project due to time constraints and budget.

We also faced challenges with testing, as the type of testing we wanted to do to properly test the app was very complicated and far beyond what we had learned on the course. Further details of this can be found in the next section. The next step will be to deploy/launch the app on Heroku and request Spotify to review and test the app for the app to have higher API rate limits and no limits on the amount of Spotify users that can connect.

Other features of the app that had started working on but were unable to include in time for the initial project deadline, such as the additional user question. However, we plan to include this at a later stage.

TESTING AND EVALUATION

Throughout the app's development, we invoked manual testing to ensure functionality. As we knew what the expected outcomes of our functions and methods should be, we were able to test functions using print statements and type ().

We used unit testing to test some aspects of the app. To do this, we imported the unit test library in python.

Once the app's front end was developed, we could test particular cases by pretending to be the end-user. Due to the previously discussed limitation on the number of users of the Spotify API using the development dashboard, we did not have to worry about too many users trying to access the app. However, this is something we will consider in future if we distribute wider. This meant that our edge cases were more related to individual use of the app.

The features of our app that needed testing are the code for database connection and management and Flask code to create the front end and pass the variable to the SQL query. Unit testing for database connections and flask apps is a level of complication above what we have learned on the course, as you would usually use third-party apps for this. While we did not have time to do this level of testing within the time frame, this is our plan for testing in future:

- Postman API testing of the Spotify API
- Unit testing database functions using a mock database.
- User acceptance testing with Taylor Swift fans recruited through social media.
- Future testing could also include the use of the Selenium package.

Challenges faced during testing

- **Scale:** It has a single source that handles requests one at a time. When trying to serve multiple bids, it took more time than anticipated.
- **Modules:** Using more modules means installing more modules, which could be a significant breach in security. After downloading more modules, third party modules are also a part of the process and development between the web framework and the developer. This also results in increased time to fix issues and errors and a security risk if a malicious module is included.

System limitations

Our most significant system limitations were with the Spotify API.

As the app is still in development and has not been deployed yet, the app is still on ""Development Mode"". Therefore, the API rate limit is calculated based on the number of calls our app makes to Spotify in a 30-second window.

Access to the API via the app is authorized for only 25 persons, which must be added to the Spotify API development dashboard. At the moment, we are adding users manually. However, we can automate this task by using Python and Selenium packages.

CONCLUSION

We thoroughly enjoyed developing this app and have learned much through our process. Caroline now has a greater understanding of Flask, an aspect she struggled with upon first learning. She also solidified her knowledge of normalization and SQL by building the SQL database. Her knowledge of Agile and the software life cycle have been greatly improved thanks to Purvi, and Aude's level of expertise and excellent teaching ability has helped her to tackle coding challenges that she did not think she could.

Purvi learned Flask and API working with Aude and Caroline. Aude's helping nature and deep knowledge of python and API have elevated my knowledge. Caroline is an expert on Databases and SQL. She is proactive in her approach. Aude and Caroline are great team members to work with.

Aude learned to use Spotify API and gained a deep understanding of oauth02. She learned to teach her knowledge to the rest of the team. She has also been known to apply Agile principal thanks to Purvi and improved her organizational skills and enthusiasm thanks to Caroline.

We worked well together as a group as we had complementary skill sets. As mentioned, Aude is the most experienced code, with excellent knowledge of SOILD programming practices, Flask, APIs and SQL. This meant we could incorporate elements into the project beyond what we had learned in the course. She was an excellent teacher, and her explanation of the various aspects of the project improved Purvi and Caroline's programming skills.

Purvi is very experienced with Agile and project management in tech. She ensured the team used agile working methodologies and demonstrated how to use various planning tools. This was especially helpful in the early stages of the project as she ensured our planning was rigorous and met the app's needs.

Caroline's understanding of SQL and Taylor Swift's discography meant she was the perfect candidate to create and occupy the SQL database. Her knowledge of the fan base also proved helpful for the product design. Additionally, her organization and admin skills ensured that the group stayed on track.

Overall, this has been an excellent experience, and we would like to thank Prakhar, Anete, Anyi and the Code First Girls team for this fantastic opportunity.