The Story of Us – Taylor Swift Mix

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

Meet The Team

Aude Vernet **Caroline Sautter** **Purvi Thakkar**

The Product

We are building a playlist generator, based on Taylor Swift's discography. It will be a web application that takes user input describing a relationship, and generates a playlist of songs that describe that relationship.

The user input will take the form a series of multiple-choice questions about the relationship the user wants to create a playlist about. The app then compares these answers to an SQL database of Taylor Swift songs that have been categorized based on the relationships they describe, and generates a custom playlist for that relationship.

The dynamic website will use Python and Flask frameworks, and MySQL as a database.

What does it do or what kind of problem does it solve?

Taylor Swift is one of the most popular artists of all time, in part due to her huge discography of famously relatable songs written about different sorts of relationships. We have created this app based on the premise that you can therefore create a perfect playlist for any relationship, past or present, based on only Taylor Swift songs.

This app serves two functions. Firstly, to automate the creation of this playlist and secondly, to be a fun activity. Therefore, both user experience of answering the questions and the final playlist generated are important to us. We've deliberately given the application a colloquial voice with fun and "online" personality to add to the entertainment value of the process.

The app is aimed at Taylor Swift's fan base, which is predominantly young people who are very active on social media. We therefore want to create something that these fans will want to share online. This will affect the look of the app and where we take it. For example, one feature in our pipeline will be to add buttons that allows the user to share the playlist they create directly onto their social media accounts (Instagram, Facebook, Twitter, TikTok etc).

Key Features

The minimum viable product should be able to do these things:

- The application is able to display on web browser a series of questions
  - The application should have a landing page that welcomes users to the application. This page should then display a series of multiple-choice questions.
  - This should be an easy to use with a simple but fun design. The tone of the application should reflect the fan base it is aimed at e.g. young and well established online.
- The application can capture the user answer via web browser
  - The application should take the answers that the user has been given and generate a MySQL query that, once connected, can select the appropriate songs from the mySQL database.
- The application is able to connect the database
- The application is able to read and fetch data form the database
  - The generated MySQL query should be passed to the database and fetch the appropriate songs based on the user's input.
- The application is able to connect to the user Spotify
  - The application will redirect the user into a Spotify authentication page to allow the app to modify user playlist
  - After authentication, the user will be redirected to our webapplication
  - 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 is able to use Spotify API
- The application is able to create a playlist on Spotify
- The application is able to add songs to a Spotify playlist
- The application is able to display the playlist to the user.
- The application is able to capture user's feedback via a form on the website. The feedback is then saved to a database, which can be analyzed later on for improvement.

Additionally, the application should be scalable. There are a number of features that we are considering adding after the MVP has been created, depending on user feedback. These include:

- Further multiple-choice questions to enable the playlist to be even more specific to the user's particular relationship.
- Ability to expand the database and add further songs. We are initially starting with a small number of Taylor Swift albums so want to ability to include her entire back catalogue and any further releases in time.
- In app ability to share the created playlist on social media platforms.
- Ability for the user to modify the playlist in app once it has been created.
- Ability to link to other APIs, such as YouTube, so that the user can create a playlist on different platforms

Architecture diagram

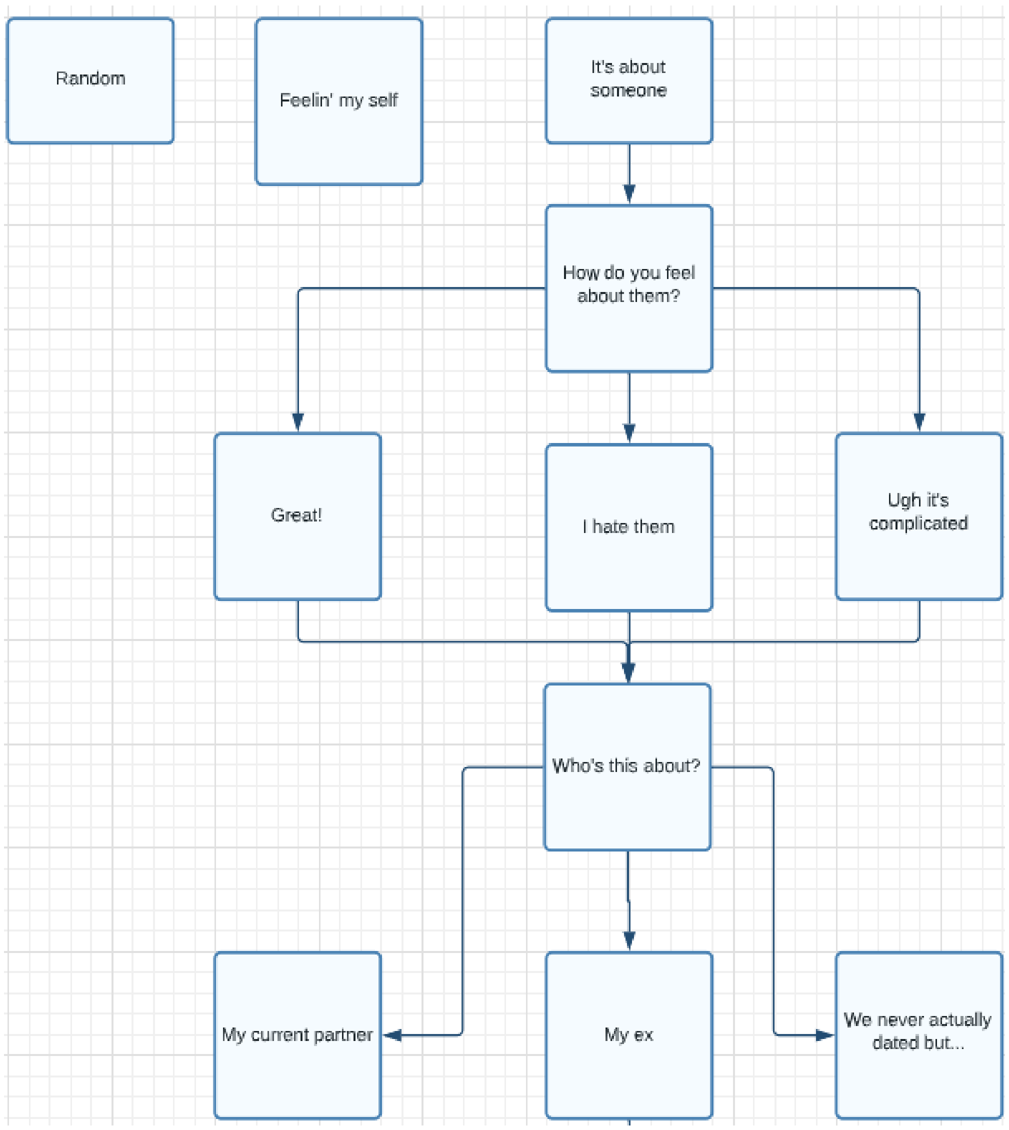
User Workflow:

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!

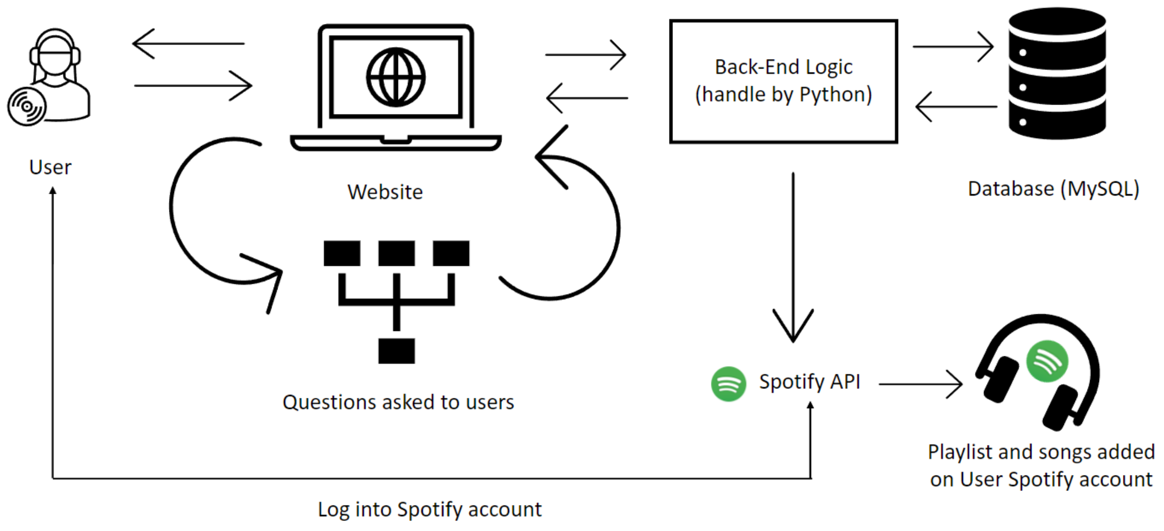
Application workflow:

- Questions display
- Get answer from user
- Python handle the logic according to answers
- Python connects to database
    - a) Create query
    - b) Fetch data from database
    - c) Return list of songs
- Python calls Spotify API
    - d) User redirected to Spotify authentication
    - e) User logs into their Spotify account
    - f) Create a playlist and add songs to the users Spotify account
    - g) Send back the user to the website
- Display list of songs to user on website

Questions workflow:



System diagram:


User
Website
Questions asked to users
Back-End Logic
(handle by Python)
Database (MySQL)
Spotify API
Playlist and songs added
on User Spotify account
Log into Spotify account

Team approach

Our ways of working

We have adopted a collaborative approach where 3 team members worked on our strengths and knowledge to deliver a successful project.

Everyone in our team is using:

- **Miro Board:** to share their ideas for discussion.
- **Trello Board:** to collaborate and organize the Taylor Swift project. The objective of using the Trello board was to know what we as a team are working on, who's working on what, and what is the stage of this each task. This helped in removing silos & enabling smooth communication.
- **Github:** to host our code for version control and collaboration. Github helped our team work together on Taylor Swift Project.

We used the following tools:

- **MYSQL:** MYSQL is our preferred database management system as it stores application data and is a robust transactional database engine.
- **Python**
- **SpotifyAPI**

We plan to conduct all Scrum ceremonies in our project. (Sprint Planning, Daily Scrum, Sprint Review and Sprint Retrospective). Sprint Planning and Daily Scrum are attended by everyone where the progress of the assigned task is discussed and any potential roadblocks are raised. These ceremonies are helping our team in:

- Building strong working relationships through effective cooperation and teamwork.
- Helped team members to learn and share knowledge with each other.
- Making everyone collectively responsible for the outcome
- We as a team are focused
- on the quality outcome of this project.

Workload distribution

We applied **capacity-based** sprint planning (in hours): Capacity-based sprint planning is an estimation of hours to complete each backlog item. With the hours in mind, the team will select tasks that fit in the sprint. We follow the below steps to calculate our capacity:

1. **Team member's availability**: We calculated no of hours each member can allocate per week to the project
2. **Sprint Duration**: Because we had to deliver the project in 1 month, we decided to allocate 7 days per sprint.
3. **Other factors**: We accounted for any full-time, part-time work, other commitments, and holidays that impact work hours during the planning process.
4. **Other Work/Project:** We also considered other priorities like Theory assignments and also the weekly homework that will take the team from being active for a certain time.
5. **Calculate the capacity**: Actual hours that the team can focus on the sprint goal without any impediments.

After careful consideration, our 3-member team learned that we will require 30-hour teamwork to finish the project.

Let's calculate the total capacity:

Team's Total Capacity (TTC) = Number of Team Members * Time (hours) * Days
Team's Total Capacity (TTC) = 3 members * 0.6 hours * 20 days

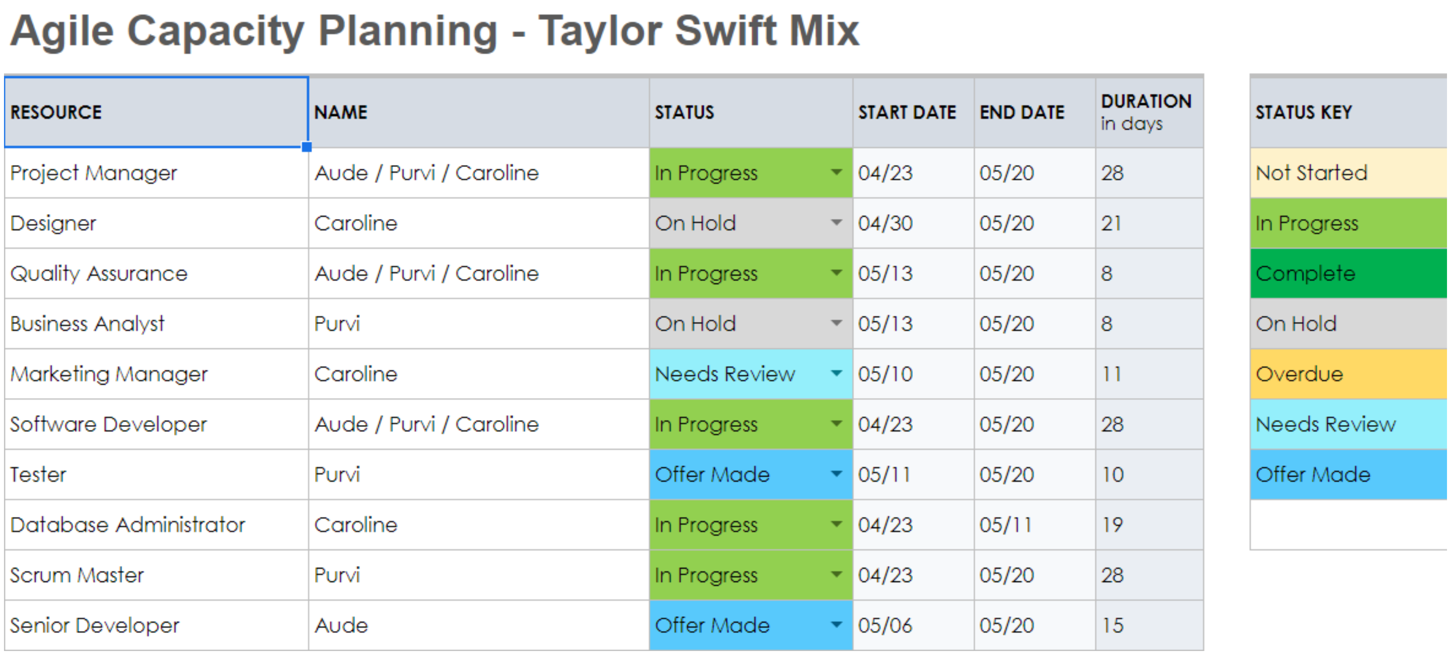
Team's Total Capacity (TTC) = 36 hours of capacity

Team works a 9-hour week, we are planning for a one-week sprint, the capacity is 36 hours: 3 x 0.6 x 20 days = 36 hours of capacity.

So, we use estimation technique, to analyze the tasks, allocate time, and determine what is feasible.

- **Task 1:** Planning / Research = 2
- **Task 2:** Designing / Coding = 10
- **Task 3:** Coding / Testing = 10
- **Task 4:** Review / Retrospection = 8

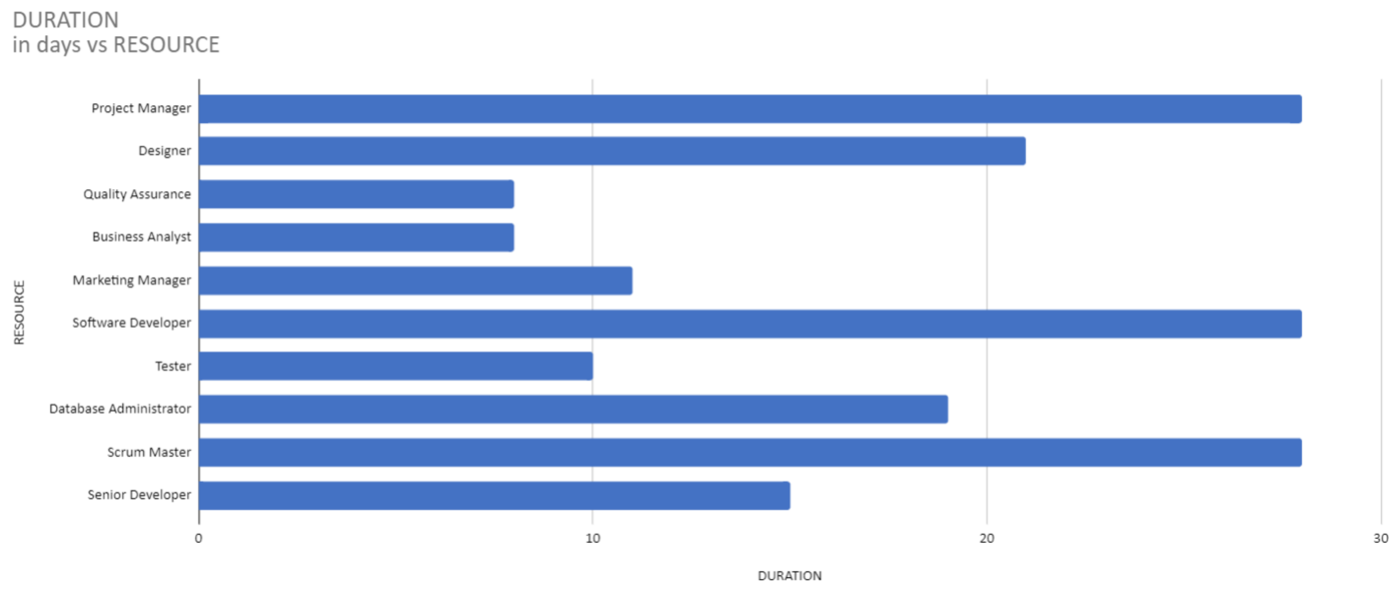
The estimated hours for 4 tasks are 30 hours. We have extra hours 6 hours in case a fallback option or we decide to add new functionality to the application.



Agile Capacity Planning - Taylor Swift Mix

RESOURCE	NAME	STATUS	START DATE	END DATE	DURATION in days
Project Manager	Aude / Purvi / Caroline	In Progress	04/23	05/20	28
Designer	Caroline	On Hold	04/30	05/20	21
Quality Assurance	Aude / Purvi / Caroline	In Progress	05/13	05/20	8
Business Analyst	Purvi	On Hold	05/13	05/20	8
Marketing Manager	Caroline	Needs Review	05/10	05/20	11
Software Developer	Aude / Purvi / Caroline	In Progress	04/23	05/20	28
Tester	Purvi	Offer Made	05/11	05/20	10
Database Administrator	Caroline	In Progress	04/23	05/11	19
Scrum Master	Purvi	In Progress	04/23	05/20	28
Senior Developer	Aude	Offer Made	05/06	05/20	15

STATUS KEY
Not Started
In Progress
Complete
On Hold
Overdue
Needs Review
Offer Made



Testing our product

Objective of Agile is to generate working software at the end of each iteration. After each sprint, we have a retrospective meeting to understand what could have been done better.

We use Unit Testing and Agile Testing

Design, development and testing occur iteratively throughout the sprint and not at the end of the sprint. All the developers and testers work together to reach the sprint goal.

We follow the pair programming method for our testing.

- The developer of a section of code will pair with another team member to act as a tester to write unit tests.
- Our Business Analyst in the team may review test cases and provide feedback to the team.

Our focus:

- We focus on fixing all the defects in the same iteration without discriminating between critical, major or minor flaws. If any spillover out the sprint must be prioritized into the product backlog for the team to pick them up in the coming iteration. (Fixing the bugs as they surface is the best method to improve quality)
- Software developed during several consecutive sprints is integrated into the release and deployed to the production.
- During release planning, we also discuss the risks during unit testing. A mitigation plan is set in place if the threats occur in future.
- A high percentage of test results produces a better quality of code.
- Our team members follow a supportive working environment in the team and encourage others to find ways to accomplish their goals.
- We also focus on adjusting the product backlog to accommodate changes.

Our Strategy for a Good Unit Test

- **Automatic**: Group of tests and checking results for PASS/FAIL should be automated.
- **Thorough**: Bugs tend to bunch around certain code regions, ensuring that the test passes all critical paths and scenarios.
- **Repeatable**: Tests should produce the same results every time.
- **Readable**: Well factored methods classes with revealing names, No duplication, tests with good names, file name.
- **Independent**: Tests must be reproducible and independent of external factors such as the environment or running order. Also, when a test fails, it should specify the location of the problem.
- **Professional**: Mind the exact standard of good design for your test code.