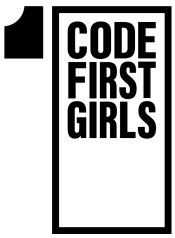
CFG Nanodegree Project Report

Feels: The Spotify Playlist Mood Generator



CFG Nanodegree Summer 2021 Cohort

Fatma Elasouad, Mariyam Lulat, Saman Kashif, Aanisah Suhail, Chidera Onyeukwu, Rabia Tanweer

August, 2021

Contents

List of Figures

1 Introduction

What this project entails is the creation of a flask application that will determine and analyse the mood of the user and consequently recommend a playlist to the user. The architecture of this application will involve a three-tier client server model. The way the user's mood will be determined will be by a questionnaire.After the playlist is confirmed, users will be able to rate the playlist there will also be a random generator where popular playlists will be recommended if the user desires. This project plays a very useful role in the form of music as entertainment as it allows users to both be exposed to new music and enjoy the process of listening to music simply based of their mood.

2 Background

This flask application will have many features meaning many things have to be taken into consideration. To begin with we decided that the most efficient way to go about determining the users mood was to divide potential moods into four broad moods which is then include specific moods so for example loving would be the umbrella for playful, adoration and passionate etc. In terms of accessibility, we need large text options for visually impaired users,semantic HTML tags for screen readers and accessibility machines and an option for colour theme to allow for higher contrast. In terms of documentation we ensured that we documented code within the python files for developers. Security considerations included that users must enter the correct username and password and the system will check the database and see there previously liked playlists and data collected from users must be stored securely with only necessary data stored to the database. After the mood of the user has been determined the flask application will present and embedded playlist alongside all of these features the system will allow users to choose predetermined playlists based on their mood as in the user will be given options between a predetermined mood or to take a quiz on the homepage of the app. These in short are all the features and functions that our application will offer.

3 Specifications and Design

3.1 Requirements

The system requirements were discussed and analysed by the team in order to determine what are the key functional requirements for the running of the system and the non-functional requirements required to run the app efficiently.

3.1.1 Functional Requirements

- The system must present questions and collect answers for the 2 question quiz
- The system must create and send a request for playlist links from Spotify with the chosen mood
- The system must accept playlists to present to the user from Spotify response
- The system will present an embedded playlist on the screen
- The user will have the option to rate the playlist on screen
- The system will store all rated playlists links to a database, with its associated mood, and add to its popularity count too
- The system will also allow users to choose predetermined playlists based on their mood
- The system will choose and present the most popular playlists from the database in order of popularity based on the chosen mood
- The user should be able to choose between a predetermined mood or to take a quiz on the homepage of the app
- The user should be able to choose answers regarding how they are feeling to generate a playlist
- The user should be able to listen to the playlist from our system as an embedded playlist on the web app

3.1.2 Non-Functional Requirements

- **Accessibility**
  - Large text options for visually impaired
  - Semantic HTML tags for screen readers and accessibility machines
  - Option for colour theme to allow for higher contrast
- **Documentation**
  - Well documented code within the python files for developers
  - Ensuring documented software engineer planning processes
- **Testability**
  - Extensive unit testing
  - Handle any unexpected error pages on site
- **Security**
  - Data collected from users must be stored securely with only necessary data stored to the database
- **Portability**
  - The system must be portable to other devices of different widths and operating systems

3.2 Architecture

After analysing many different options regarding the architecture choice for the system, according to the key functionality and requirements needed from the initial scope of the project, it was decided that the best type of architecture design for the project would be the client-server design. As the system would represent a web application that will be used by clients through the internet, but also will need access to a database to store rated playlists, as well as send requests to Spotify within itself, it was decided that the three tier client-server model would be best suited for this system. The design is based on the three main components. Firstly, the client side that will be interacting with the Flask web app. Secondly, the Flask app that will be responsible for presenting the front end using Jinja2 as HTML templates as well as sending the API requests to Spotify and managing calls to the database, and finaly the database component itself that will communicate with the database manager in the app, all shown in 1.

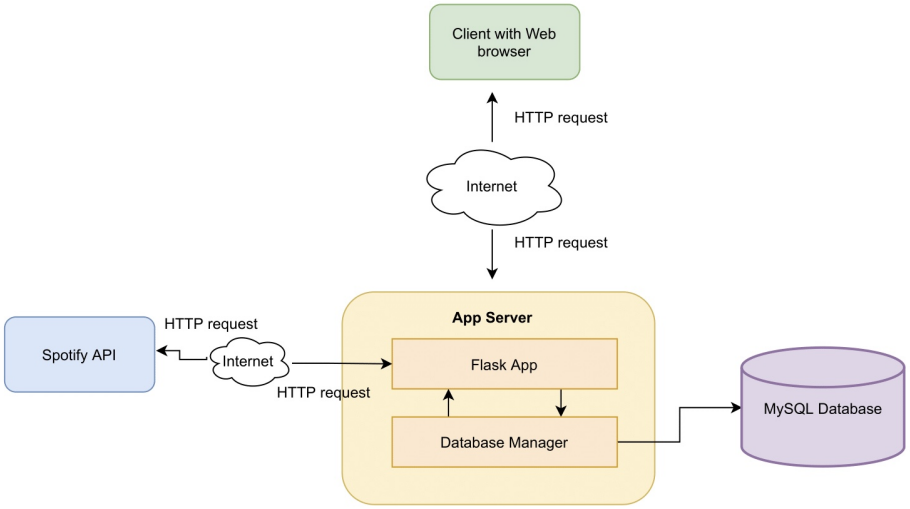


Figure 1: Three tier client-server model design

3.3 User Stories

In order to understand the use cases and to start creating tasks to build the project, we began by creating three key user personas in order to create the user stories as shown in the appendix, figures 5, 6, 7. The user stories were then created based on the persona's and the intended functionality of our app as seen in figure 2.

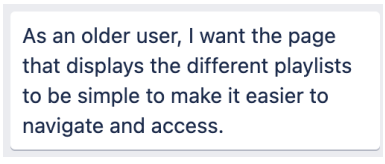


Figure 2: Example of a user story

These stories were grouped into three main categories; (1) the mood quiz playlist's generator, (2) a way to rate and access previously rated playlists, and finally (3) an intuitive and clean design for the app. These categories were then reflected into the key tasks placed in the backlog, whereby each card included a list of tasks to be completed and one to two members were assigned to those tasks as seen in figure 8 within the appendix.

3.4 Project Structure

The project structure itself contains 2 main folders, the testing suite and the src folder. Within the src folder we will be creating a specific main python file that will include the main run method in order to run the whole app, it also contains the environment file which holds the project secrets used within the application and is a file GitHub ignores in order to preserve the secrets. In the website folder, it holds the majority of the functionality of the app itself. In there, the templates and the static folders are required for jinja2 to load the HTML files and for the Flask app to render the templates, therefore it is structured in that way. The next two folders represent the other two main components, which are the database component which includes a data manager that will convert and bundle the data extracted by the feels database module that will interact directly with the database; where in our case we are utilising CSV files to mock the database tables. Finally, the API folder represents the API handler for the app in communicating back and forth between the Spotify API and that module is used directly in the Flask app in order to extract the information and data from the response and present it back to the user on the screen. A larger version of the structure can be viewed in the appendix here 9

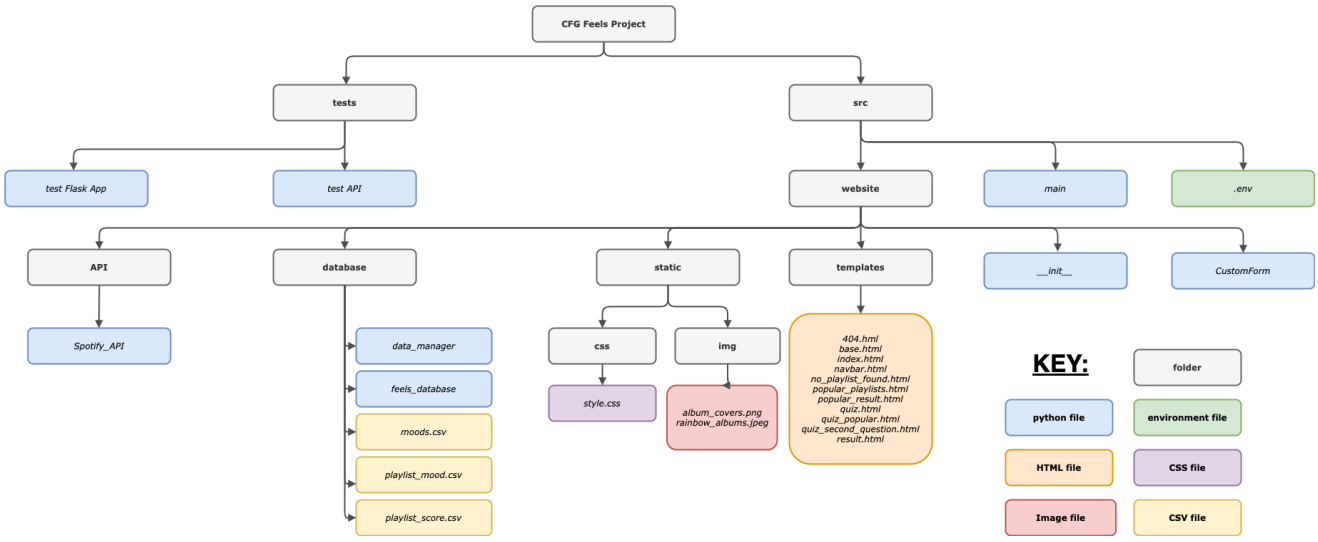


Figure 3: Structure of the project

3.5 Database Design

The database was designed based on the discussed need of the app in terms of data required to be displayed on the screen, specifically the quiz answer options, which needed to be a table on its own that can be added and altered to directly in the table that would then be used to fill the answer fields for the quiz forms, and the second set of tables within the database included the storing of the playlists from our service. The two tables designed were the playlists mood and the playlists score. The playlist mood table provides a way to store the playlist name, its distinguishing URL, and the main mood it falls under but most importantly the sub mood it falls under. In this way, if we wanted to present that playlist to our users again through our popular rated feature, then we will have access to the necessary information stored in our database. The playlist score table is used to store the rating of the playlist by the user using our service. This score will be updated if the same playlist is displayed to another user who also votes on the same playlist, and this table when joined with the playlist mood table is then used to find the most popular rated playlists based on a specific sub mood. By defining the tables required, we were able to create an entity relationship diagram (ERD), as shown in figure 4, that would inform the creation of the database for the app. As we were mocking the database through the use of CSV files, and its access by using primarily the Pandas library and the CSV library, each table represents a CSV implementation.

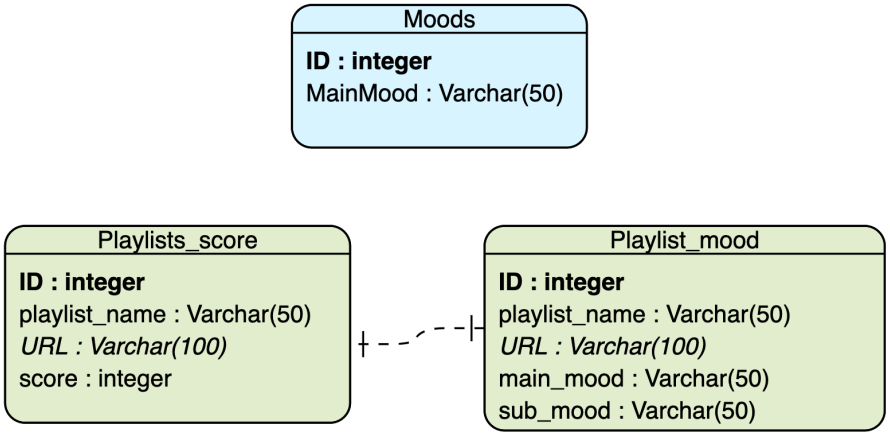


Figure 4: ERD of database

4 Implementation and Execution

4.1 Development Approach

We took an agile approach, creating user stories to identify what features would be required from the user's perspective. This helped us to better comprehend the project's requirements and break them down into smaller tasks that could be completed. We divided our tasks on a Trello board and used it to keep track of what had been completed, difficulties that had emerged, and what needed to be done. Alongside this, a wireframe was also then created for us, as a team, to envision how we wanted our front-end to look.

4.1.1 Spotify API

In order to create the Spotify API, spotipy was installed to use. This enabled us to use the search query and to find playlists in Spotify according to the user's mood. The challenge in this was understanding how Spotipy was used and how to use the different features that Spotipy allowed.However, once receiving help from another member of the team and some guidance as to how it is used, the API was then created - allowing a sense of accomplishment to be felt.

4.1.2 Quiz

The Quiz was created using a list of major emotions such as "Happy, Sad, Loving." Then another list was created, this time with sub-moods associated with the main moods. This enabled us to design the quiz by using OOP principles to create functions in which, if a user selected a specific primary mood, it would then ask them to specify that mood using the sub-mood.

4.1.3 Database

If a user already knew how they felt, a python file was written that allowed them to obtain playlists that were deemed popular based on the ratings that users had previously provided, i.e. the top playlists would be returned. The database was also used to keep track of the ratings for the many playlists that were provided, as well as to update the score if a specific playlist received a new rating. As a result, the database was used to store user-favourite playlists as well as the scores assigned to the playlists. A function for adding new moods and sub-moods was also created.
However, if the number of new sub-moods added did not equal the length of the existing rows in the CSV files, it would automatically fill them with NaN values. This was an issue that we encountered, but we were able to fix it by adding a line of code that filters out NaN values so that when the list is returned, the NaN value is not displayed in the output.
Additionally, pandas was used to access the CSV mocking our database. We used this library to read and write into the CSV file. It was also used to extract data, manipulate it, and then save it again, such as when updating the playlist scores CSV file. Another use case for pandas is the ability to extract data, filter out the information that is required, and then output the results into a list or dictionary. Pandas was also used to simulate a 'join' function similar to that learned in SQL; this was done to find the top three rated playlists.
Pathlib library was used to create the paths for the CSV files to create consistency when establishing the file paths that are used in the feels-database python file.

4.1.4 Flask and HTMl

The flask framework was used to build the web app, and jinja2 was used to create HTML templates alongside the Flask framework to pass variables and other data between the app and the frontend. Flask was used to create an initial create-app function that contains all of the necessary routes functions where each route loads up a page and variables are passed into it by either adding the variable to the route path itself, passing it through to the jinja HTML file so it can be output to the screen, or saving it to the session and calling it throughout the app if access was needed to specific data. Flask was also used to handle some error pages. Libraries such as Werkzeug were used to redirect the function from this library to redirect between routes in the functions in flask app.

Dotenv was used to load the environment that holds the secret keys and other secrets we used throughout the app. The OS library was used to get the variables saved in the environment.
The flask-wtf library was used to create specific forms called FlaskForm in order to create classes that inherit from this class to create our own custom forms.
Wtforms was used in conjunction with flask-wtf to create form fields such as input fields, selection fields, and even a submit field that generated a submit link. It also works to easily collect submitted data and validate it on the frontend page all at once, requiring less code from our end and allowing us to use it to pass variables around in the flask app backend. The first quiz question form was used, the second quiz question was used to get the submood from the other form, and the third form is a rating form that will allow the user to rate the playlists.

4.1.5 Team Member Roles

Aforementioned, the duties were divided using the Trello board. One team member (Fatma) worked on the HTML and integrating the API and the Quiz to the front-end using Flask. Another member of the team (Saman) worked on the Spotify API, while two others (Aanisah and Mariyam)worked on the Quiz. We set a week aside to finish these duties. We decided to go on to creating the database (Mariyam and Fatma) and unit testing (Chidera and Fatma) for the project after this was accomplished. Finally, Aanisah created the instructions in the readme file, Chidera and Fatma both worked on creating more unit tests for the API and the flask and Saman worked on making the web more responsive and accessible. The logo design was created by another team member (Rabia). While doing so, we reviewed each other's code and refactored it to make it easier to comprehend and implement, particularly when it came to integrating it all. Alongside this, we also split the sections of the report and worked on that too.

4.2 Implementation Process

4.2.1 Issues and Solutions

While working on this project, we encountered a few difficulties with implementation,development, and code writing. When developing the Spotify API, we had to first determine whether an authorization token was required, and if so, how to renew it after it expired. We decided we didn't need one because we weren't asking the user to directly log in to their Spotify account, enabling the issue to be resolved.
Another issue was determining how to test the sessions in a flask project. This was resolved by reading up on the flask unit testing documentation, watching tutorials, and following a similar flask unittest implementation. Fatma also learned how to make a Flask client and how to test for specific things like response codes if a route has successfully loaded.
Another challenge was saving and passing variables between routes in a flask app. However, this was resolved by utilising sessions, where everything stored in the sessions is cleared when you return to the homepage, otherwise it collects information as you pass through the app to determine the next page to be directed to. Variables are passed into HTML files via rendering templates. The route is also used to pass variables in order to obtain the top first, second, or third most popular playlist by returning a number of 1, 2, or 3 that will be used to call on the top three dictionary, which holds the key 1, 2, or 3 with details corresponding to the top playlist such as its name, URL, and score.
Finally, the last issue we came across was trying to figure out how to run the project locally whilst being able to save the client-id and client-secret (credentials that are needed in order for the project to run) as secrets on GitHub.

4.2.2 Changes

Although we intended our website to welcome the user using their name on our page, e.g."Welcome (name)", we had to get rid of this functionality.
The popular song database was an idea that was formed and transformed over time and it wasn't until the end of the project where we were able to implement that feature. An ambition for this feature is a login page that will check to see if it is the administrator that is logging in and if it is,then they should be able to add and update the moods in the database alone - this section of the site should only be accessible to the administrator with the login details.

4.2.3 Successes

- Learning to pass variables between the flask app and the rendered HTML page, showing some HTML based on a variable in the JINJA HTML page and to have it change as the variable changes and learning to create Flask specific unittest.
- Being able to mock the behaviour of a database using CSV files and pandas when with the bugs it throws in saving NaN types - learning to filter out those values when extracting data from the CSV file
- Learning how Spotipy worked and how to use the different features in the API
- Learning to build custom forms and being able to incorporate that into the main flask app and easily validate the data entered and collected from it and using that to collect the data. For instance, the answers and passing it as a variable within the flask app to move to the next page.

5 Testing and Evaluation

5.1 Testing Strategy:

Our testing strategy was focused on two buckets of the traditional testing pyramid - unit testing and UI testing. Our test suites consists of unit tests of varying granularities. We used the standard unittest module to execute out text cases in order to increase our test coverage as well as speed of test execution.

5.1.1 Unit Test Areas:

-Status codes: we wrote multiple test cases to check for a 200 response code on each onour web app page (index page and quiz pages). -Data: we wrote test cases to check that the response from the API was returned as HTML data as this was a requirement for our index page content. -Loaded forms: we created forms to allow the user to select their "primary" and "secondary" moods as well as rate playlists. Part of our test suite included test cases to check that each of these forms load correctly on the page. This was particularly important to ensure that the correct "secondary" mood was loaded based on the chosen "primary" mood

5.1.2 User Acceptance Testing (UAT):

After the application was developed and tested, we tested the webapp as an end-user to ensure everything was working as per our specification. This included testing some of the key user stories we created during our planning stage.

5.1.3 System Limitations:

1. High degree of randomisation: Every user gets a random playlist based on their current mood. The high degree of randomisation (0-1000) means that there are low cumulative scores for each playlist. This makes it difficult to rank popular playlists as not aggregate date due to low chances of users rating a playlist more than once. Proposed solution: (a) Limit degree of randomisation (e.g.. 0-100) or allow users to rate playlists from saved playlists as this increases likelihood for users to rate same playlist more than once. (b) Add feature that randomises saved playlist in our database so that users can rate. 2. App Deployment: App is deployed by installing a Dotenv which can deter app usability. Future work on app can include app being hosted on a cloud application platform like Heroku.

5.1.4 Evaluation

We evaluated our flask application UI using the Nielsen Heuristic model, as shown in figure 5.1.4, on the page below.

Criteria	Pass / Fail	Comments
Visibility of system status	Fail	There is no feature on web app that keeps the user informed of what is happening on the page. Recommendations: include loading circle to let user know that system is handling their request
Match between system and real world	Pass	Highly recognisable symbols/icons e.g. star symbol to rate playlists were used in the app
User control and freedom	Pass	Navigation buttons are present and visible across pages including "homepage" button that takes user back to homepage as an "emergency exit"
Consistency and standards	Pass	Web app adheres to standard website layouts e.g. placements of navigation bar and visible action buttons. Recommendations: Use emoticons to illustrate mood options
Error Prevention	Pass	Multiple implementations of try/except blocks whilst building web app to handle errors. Flask app was sufficiently tested to prevent errors.We also handle 404 errors and the "case" where Spotify request fails, user is taken to 404 page.Recommendations: More robust error-handling e.g. try statements with else clause. Specify type of error
Recognition rather than recall	Pass	Options selected by user are shown frequently and visible e.g. home button in consistent location across pages to reduce user memory load. Recommendations: More instructions where necessary
Flexibility and efficiency of use	Fail	Application does not cater to both novice and expert users. Recommendations: Include login feature that allows user to create their own profiles and tailor frequent actions e.g saving playlists. Include accelerators or shortcuts to speed up interaction for more experienced users

Criteria	Pass / Fail	Comments
Aesthetic and minimalist design	Pass	Only relevant information is displayed on each page. Overall web app design is minimal with few icons and a filtered background image on home page
Help users recognise, diagnose and recover from errors	Fail	No mention of how users can diagnose and recover from errors. Recommendations: Include error messages in plain language (no code) to indicate problems and suggest easily implementable solutions
Help and documentation	Fail	Detailed ReadMe file explaining how to use the install, run and deploy application including written tutorial on how to use app.Recommendations: Include video demo/tutorial detailing app features for increased accessibility for users. Include visible help and information buttons throughout the app for users who are using it

6 Conclusion

This project resulted in the successful development of a flask application that determines the user's mood with a questionnaire and recommends a random Spotify playlist to the user based on that mood. The application also allows users to be recommended a popular playlist based on the mood they have selected if they already know which mood/type of music they would like to listen to. The application consists of the Spotify API, a database which is provisionally created with CSV files utilising the pandas and csv library, and HTML templates and CSS files used alongside the flask framework to build the frontend of the application. This project has allowed us to create an application which is the ideal companion to the Spotify application. It allows users from all walks of life the option to discover new music,9 rediscover old favourites, or simply taking the decision-making part of playing songs out of their hands, so they can just enjoy the music

A Appendix A

A.0.1 User Personas

John Paul Adam III

- a 27 year old millennial, works as a consultant for an up an coming tech company based on central London, reserved and quiet - has close relationships over wanting to please everyone
  - John has no time on hands, very busy and commutes to work everyday
  - would like:
    - a way to quickly and easily get a taste of new music as he is too buys to 'feel out' new music
    - wants to be introduced to new musics that he isn't in touch with, from old artists he used to love, with anymore, sick of all the new music
    - also wants to learn new music that he doesn't know that fit the style that he is familiar with and prefers
    - likes fast pace music- start the day, make time feel like its passing faster
    - to use music to drown people out in order to have alone time
  - disklikes:
    - he hates slow people, e.g. on the train, walking behind slow people

Figure 5: John User Persona

Rosie Donald

- a 35 year old lady who is a housewife, and enjoys being a mum, likes architecture, family, soccor mum, involved parent, head of pta, organised, loves hosting events, her home is immaculate, kids in a variety of extra curricualar. kept busy with her life and responsibitlies, up early and and likes to keep a regimened routine in the evenings
  - would like:
    - easy and intuitive systems to listen to music
    - wants to listen to music while running errands or doing work
    - enjoys listening to happy music to keep her upbeat and energised
    - likes to have the perfect music for the different events she hosts e.g. late night dinner party vs bbq cookout
  - dislikes:
    - rude language
    - untimely and unorganised 'behaviour' - customer service
    - learner drivers

Figure 6: Rosie User Persona

A.0.2 Trello Board - User Stories

Amellia Clarke

- 16 year girl in sixth form, studying english and history and likes painting, makeup and fashion, and art, very social - social butterfly
  - taste:
    - alternative vibes (craves that vibe)
    - enjoys watching and learning about new bands that are up and coming on the scene
    - loves to dye her hair
    - thrifts her clothes
    - economical and environmental, and vegan
    - listens to unreleased lana del ray music
  - would like:
    - a quirky new way to find new music
    - find new musics
    - wants to show people the music she founds
    - likes to find new chilled playlists
    - as an emotional teen she likes to have music on all the time that fits her mood
    - dislikes:
      - ariana grande. - pop mainstream
      - fast fashion
      - hats the structure and rigidity of college

Figure 7: Amelia User Persona

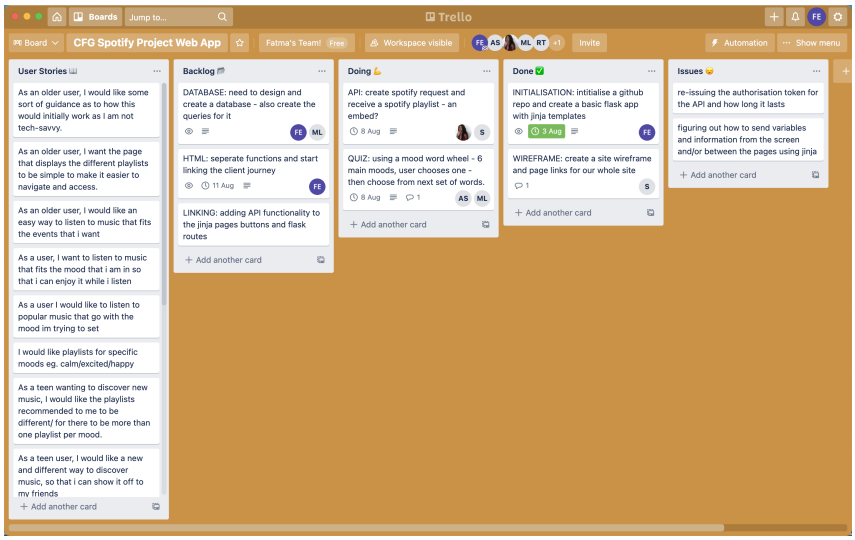


Figure 8: User Stories, Backlog and Tasks

A.0.3 Project File Structure

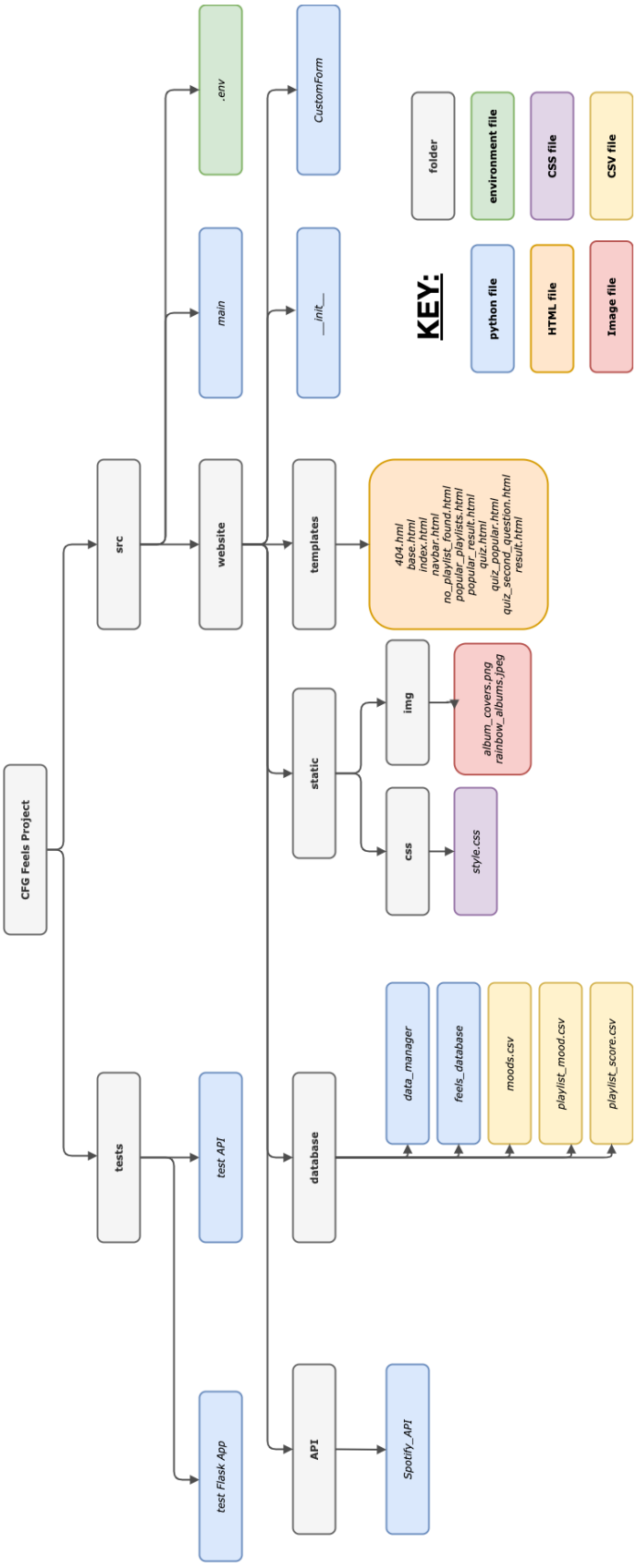


Figure 9: Structure of the project