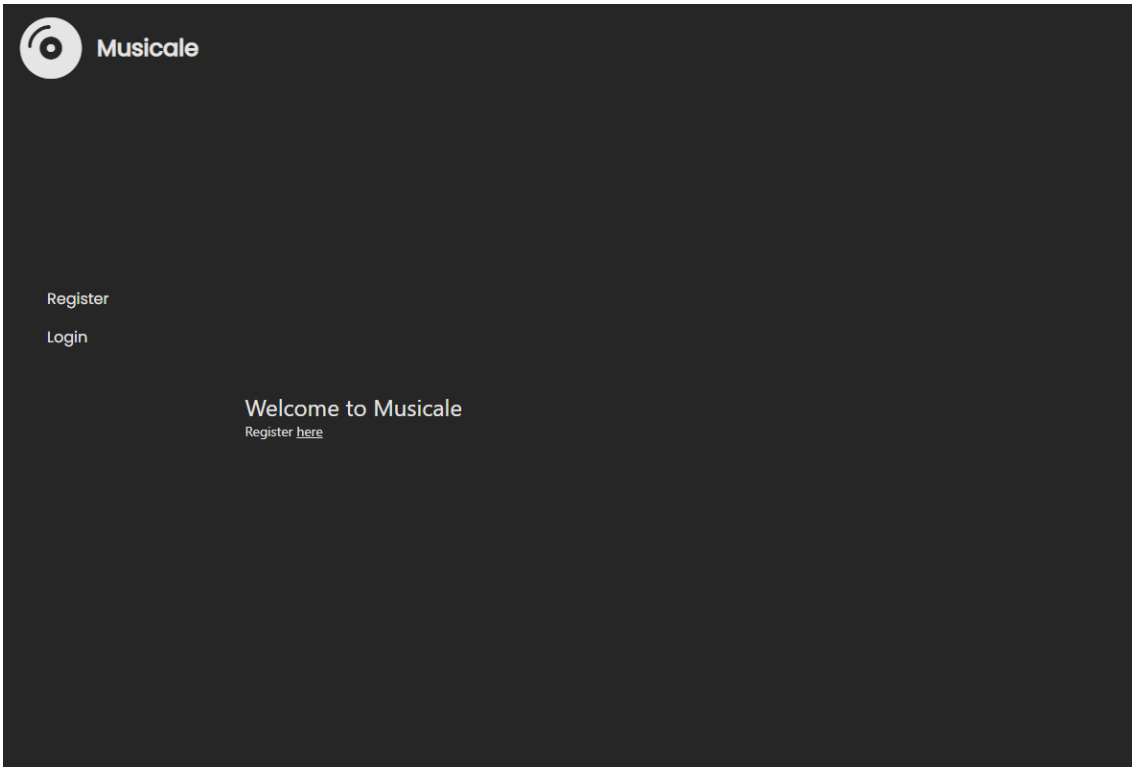
CS235_2022_assignment-khar453_nwu928 Cool New Feature Design Report

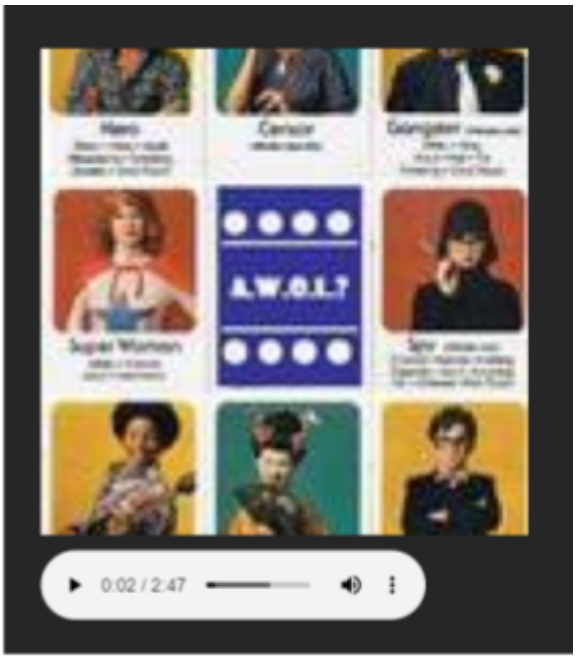
The web application Musicale has existing features of registration and signing in which is powered through authentication to ensure only authenticated (registered) users can make changes to their playlists and reviews. After being authenticated, the user can then browse tracks, add a track to their own playlist which is visible to the unique user only and rate and review an individual track. If a user is not authenticated on the application, clicking on the 'Musicale' text logo will redirect them to the home page to register.



The additional functionality that has been added on top of these existing features is the ability to play the audio of each track within the app. This feature retrieves the link to the music archive in the raw_tracks_excerpt.csv file and adds the audio functionality of the track into the Musicale web app as a mp3 so that users can listen to the track within the app itself before adding it to a favourite playlist and writing a review on it. It can be accessed by clicking on the track image when an authenticated user is browsing tracks or when clicking on the track that is in their playlist.

The benefit of this feature is that previously, users would not be able to listen to the tracks before adding it to the playlist which means they had little to no idea of whether they liked the track or not. There was also no functionality that linked the external website that the track is retrieved from so the user can listen to it themselves, albeit externally. This feature now

allows users to listen to the track instead of using the platform as solely a track library. A user must be authenticated to see this feature in action.



Screenshot of the play feature in action

We applied the Single Responsibility principle throughout by splitting the Flask application into multiple blueprints, including review, authentication, home and tracks. Each blueprint has a single responsibility (detailed by its name, for example the review blueprint is responsible for containing all the details about reviewing a track) and the content of each blueprint/module is cohesive. This principle allows each module to have responsibility over the functionality. The new feature was added into the review module which takes responsibility over the audio functionality of this feature, including pause, play, download, playback speed and volume functionality. Each class/module is in charge of one thing which allows for the benefit of avoiding code duplication and creating robust classes/modules.

The app also applies the Repository pattern which provides an abstraction of the database away from the Domain Model. This follows the Dependency Inversion principle which states that both high-level and low-level modules should depend on abstractions rather than each other. The abstraction in this case is the Abstract repository interface. Although the application does not currently have a database, creating the Abstract repository interface (repository.py) and the Memory Repository which implements this interface (memory_repository.py) allows the benefit of ease of future extension to a database in SQL. This means that the backend of the application can be changed in the future without having to change the repository interface compared to if the Domain Model was directly connected

to the database. In the future extension of a database, a new Database repository would implement the Abstract repository interface and this repository would directly interact with the SQL database.