Intermediate Submission 3 Group 49

Miguel Lebrun, Kiril Aleksiev, David Mandado, Marko Spyrou, Victor Handzhiev, Stoyan Meshov
Eindhoven University of Technology, The Netherlands
2111152, 2108399, 2138549, 2106426, 2159384, 1978276, Group 49

1 INTERMEDIATE SUBMISSION 3

1.1 Changes to Use Case and Requirements

We had some important architectural changes to fit the constraints of the project of no external dependencies and other minor changes:

- Implemented custom 'Database' with CSV files (Comma separated values) due to external dependency constraints.
- Eliminated Spotify API dependency, replaced by hard-coded Database due to project constraints
- We renamed "Friends" to "Following" to match our one-way relationship model (this makes many database processes far less expensive and complex).

1.2 Project Structure

We are following the MVC (Model-View-Controller) architectural pattern as it brings many useful benefits for our use case. This means the project is separated into **models** for Data representation, **controllers** for the GUI's logic and **views** for GUI related classes. Our rationale was that it provides:

- **Testability.** Controllers and services are decoupled from view code, letting us unit-test logic in isolation.
- **Separation of concerns** Considering our use case and the large amount of GUI work that would need to be done, we found that MVC provided good enough separation to make development easy as opposed to other architectures (MVP separates too much logic which becomes cumbersome is larger apps, MVVM is overkill for our use case).

In addition to this, we have **services** which act as the backend. For the database we have 2 classes Table and Database. In the class Table we define methods for reading and writing into csv files, so that it will be easy to work with our database. In the Database class we define the tables we need for out project and the connections between them. We also have classes for User, Review, Reviewable objects (Song, Artist, Album) which are used to model the data loaded from our database. Finally there are **utility** classes for GUI purposes mostly

1.3 Project Status

The development process is overall within schedule and we have the main use case practically implemented:

Completed features: All major pages, Login, signup, search, review submission, follow/unfollow, star rating UI, CSV persistence, minor testing.

Work In Progress: Recommendation algorithms, which are designed on paper and needs to be implemented.

To do:

- Implement algorithms
- Introduce tests for controllers and database
- Publish recommendations to the home page
- Add more robust validation to all inputs (login, signup, search, friend search, fix commas in a review breaking csv file)
- UI Improvements, Like button for reviews, generating more data for mock db.

In terms of data we currently have 4 sample songs in songs.csv which can be searched for and a few sample users and reviews in the respective csv files. To keep the workload manageable and on schedule, we are prioritizing the algorithms, connecting them to the UI, as well as the testing, input validation, and then everything else.