University of Nottingham
School of Computer Science

Interim Report

Recommending Music Through Social Connections Based on Music Taste

Author:
Daniel Jackson

Supervisor:
Max Wilson

December 8, 2016

Contents

1 Introduction

From discussion amongst friends to the wide-scale promotion of radio stations, compilation albums and event planners, music recommendation has played an integral role in the development of music and it's ability to reach listeners around the world. Recently, automated music recommendation systems have been developed for popular web pages and applications including Spotify, Soundcloud, GooglePlay and Apple Music. Millions of users access these services every day [1] in order to discover new music and they are fast becoming a regular tool in people's day-to-day lives.

I believe however that these systems are still unable to produce consistent and reliable recommendations for their users. From personal experience alone I have found that although these systems do produce a small number of agreeable recommendations, the majority of presented songs are not recommendations that I would make myself. I do find however that a small number of friends, each with similar tastes to my own, are able to produce much more dependable recommendations than any existing systems.

The aim of this project is to understand why recommendation systems are not yet completely reliable, the reasons that sharing music in close friendship groups is typically more reliable, and how these two ideas can be combined to produce a novel approach to music recommendation.

The reliable recommendations produced by close friendship groups can be considered a result of a shared social environment in which similar music tastes can develop. If a friendship group shares similar tastes in music then it can be assumed that the recommendations from its members are likely to be enjoyed by its other members. As a result it can also be considered likely that the same pattern will apply to groups of people who have a shared taste in music but are not considered friends. If these groups could be collected in order to share music with one another then it is likely that reliable recommendations could be shared between members of the groups.

The focus of this project will be to develop a new form of music recommendation which brings together people with similar taste and produces recommendations through those connections. In order to achieve this I will produce a mobile application which focuses primarily on the social connections between people to power its recommendation. A user will be entered into a social network of music lovers, each broadcasting their own recommendations, and the users who regularly like one another's suggestions will be considered to have similar tastes. The users with similar tastes can then be considered likely to produce reliable recommendations for one another. It is important to note that recommendation of this kind will not require the consideration of any external data linked to music e.g. artists, albums and genres.

To produce such an application I must first identify existing work which relates to the problems I have encountered, understand why these patterns exist and recognise how others have attempted to solve them before me. I will then use these ideas to develop a new and novel approach to music recommendation which can be incorporated into a system that challenges the problems found in existing systems.

2 Motivation

Many of the largest web pages and online services integrate features for recommendation into their products. Spotify, Soundcloud, GooglePlay and Apple Music are some of the most popular music streaming services and provide recommendations to their users in a number of ways. Spotify offers a range of features including Discover, Radio, Discover Weekly and Daily Mix which each produce personalised playlists of songs for their users. Soundcloud offers Stations which, rather than building set playlists, plays tracks one at a time based on similarities to a singular selected track. Alternatively, GooglePlay and Apple Music provide a Recommended for You service which base their recommendations on the personal preferences and purchase history of each user. Amazon and Netflix are two of the largest and most used web services which incorporate the use of recommendation features which do not focus explicitly on the recommendation of music but employ similar techniques to their music counterparts.

Computational recommendation is a process which has yet to be fully explored and yet poses great benefits for both the social development of music and the economic development within companies who gain from the reliable recommendation of products to their users. Research into new recommendation techniques is still ongoing from the continued development of traditional collaborative filtering, content-based filtering and knowledge-based recommendation techniques to the more recent and abstract approaches including deep learning. However, there currently exists a gap in music recommendation applications which focus solely on producing recommendations through social connections based on similar tastes. This gap broadens further when considering how these social connections should be produced; there remains to be a mobile application which collects users of similar tastes through the repeated approval of recommendations appearing in a continuous feed of music. I hope to fill this gap with a novel music recommendation application which develops new possibilities for social recommendation within these systems.

3 Related Work

Mesnage et al. [2] examined whether more reliable music recommendations can be obtained by taking into account the tastes of a user's social contacts. In order to test this theory a facebook application Starnet was developed which would generate recommendations based either on the positive ratings of friends, positive ratings of others in their network or randomly selected tracks.

Starnet used a technique coined social shuffle which recommended tracks by diffusing discoveries throughout a social network. If a user received a recommendation which they decided to rate 4 stars then the track would be sent to connections in their network. If a member of that network also gave the song a high rating then it would be sent to their connections and so on.

The results of the study found that social recommendations were preferred over non-social recommendations and concluded that people tend to share music tastes with the friends they are connected to within social networks.

Similarly, Konstas, Stathopoulos & Jose [3] investigated the role of social network relationships in developing a music recommendation system. The paper evaluates a Random Walk with Restarts (RRWR) model on a dataset of music and found that the incorporation of friendship and social tagging can improve the performance of an item recommendation system.

However, Starnet and the studies into the RWWR model were limited in that their social recommendations were produced by friends of the user in their social networks. My project takes a similar approach to these recommendation techniques but will not be limited to the existing friendships between users. The application I aim to produce will provide a means for such recommendation techniques to be tested on large, unrelated sets of users with the focus of shared music taste forming the social connections over existing friendship.

Shardanand [4] proposed the idea of Social Information Filtering (SF) which filters items based upon other users whose tastes are similar to your own, regardless of whether you are friends or not. Over time, the system would record a user's traits based on their past history and this information could be compared to the profiles of other users. Each comparison would then be assigned a weight to show the degree of similarity between the two profiles. The results of the study showed that social information filtering techniques were effective in producing reliable recommendations and that finding new music through users with similar tastes is an effective approach to recommending music.

The filtering technique Shardanand proposed is comparable to the approach I am taking. However, rather than evaluating the similarities between users through the comparison of user profiles I will be tracking their similarities dynamically as users rate one another's music recommendations. Each time the SF approach would like to produce recommendations it must compare with a number of random profiles and decide which is the most similar. My technique will continually track the most similar profiles to each user and as users rate one another's recommendations this recorded similarity will change over time.

Ma et al. [5] studied recommendations produced through social connections they call "trust relationships" which are distinguished from "social friendships" as relationships that do not require both users to confirm the connection. Trust-aware recommender systems are based on the assumption that users have similar tastes with users they trust. The paper found that good recommendations may not always be true in social recommender systems since the tastes of one user's friends may vary significantly and so implementing trust-aware techniques can help to avoid this problem. My application will develop on the idea of trust based recommendation by considering trusted users to be the users who regularly share music that you like. I will introduce the idea of this technique into my own specialised social network and track the trust between individual users as they make use of the system.

4 Design

4.1 Data Storage

To retain a certain degree of simplicity and efficiency my system should only store necessary data and the data it does store should be stored in the most simple form possible. In order to produce recommendations the system must store both users and the tracks that they will be recommending. A value representing the similarity in music taste between each user should also be stored within the system so that new recommendations can be retrieved from the most similar users. Data in my system must be stored in such a way that allows me to easily track the connections between the users whilst representing differences in music taste between those it links together. It must also be able to search through these connections in order to find new users from which recommendations can be retrieved.

Standard relational databases are defined by sets of rows and columns in which the row can be perceived as an object and the columns represent the properties of that object [7]. Relational databases are optimized for highly structured data with pre-determined columns. To represent connections in data the rows can reference rows in other tables by referring to their primary-key attributes via a foreign-key column. The joins between these rows are computed at query time by matching the primary and foreign keys and these operations are both compute and memory intensive [8]. An alternative to relational databases is the graph database. Graph databases focus on these implicit relationships and define them explicitly. Data is stored conceptually as a graph by representing information as nodes and connections between those nodes. Each node contains a list of it's relationships with other nodes and these relationships can hold additional attributes. By storing these relationships within the node itself the queries have direct access to the connected node, eliminating the need for an expensive search/match computation. Graph databases are good for storing data in which the connections are an important aspect of the information.

Since my system relies on the connections between users to produce its recommendations the graph database will be the most appropriate system for storing data. Using a graph database will allow me to easily build a social network of connected users and navigate through those connections.

4.2 User Interface

The most prominent component of my system will be the interface that each user interacts with. In order to encourage intuitive interaction I will be aiming to keep this component of the system as simple as possible. Throughout my design process I focused on simplicity and the encouragement of intuitive interaction. It was decided that the feed of music recommendations would be the centrepiece and most important aspect of the application and so it should not be cluttered by other features. Users would scroll this feed vertically and decide whether they liked the recommendation or not. If a recommendation is liked then it should be possible to access the profile of the user who made that recommendation so that a conversation can be initiated. The act of clicking a song would play a preview of that song and the current track being played would be displayed at the bottom of the screen where it could be shared.

As users will be required to provide feedback on the recommendations that appear in their feed I explored the potential approaches to receiving this feedback from the user. A star rating system allows users to rank each song to a certain scale, though my system is only interested in whether a user likes a song or not and not the degree of how much they like it. Like and dislike buttons could be added to the songs so that users can provide this feedback but the buttons would have to be made to a specific size.

I found that one of the most effective ways for a user to like or dislike a specific item is through the act of swiping an item left or right through a touch screen interface. The swiping functionality allows the entire width of the screen to act as input for the users decision unlike a button.

The dating app Tinder is one of the most well known examples of this functionality and offers a platform in which users match with other users who they are interested in based on pictures, bio's and personal interests. In order to produce a match, user's swipe left on users they are not interested in and right on users they are interested in; when two users both swipe right on one another they are matched and direct communication can be initiated.

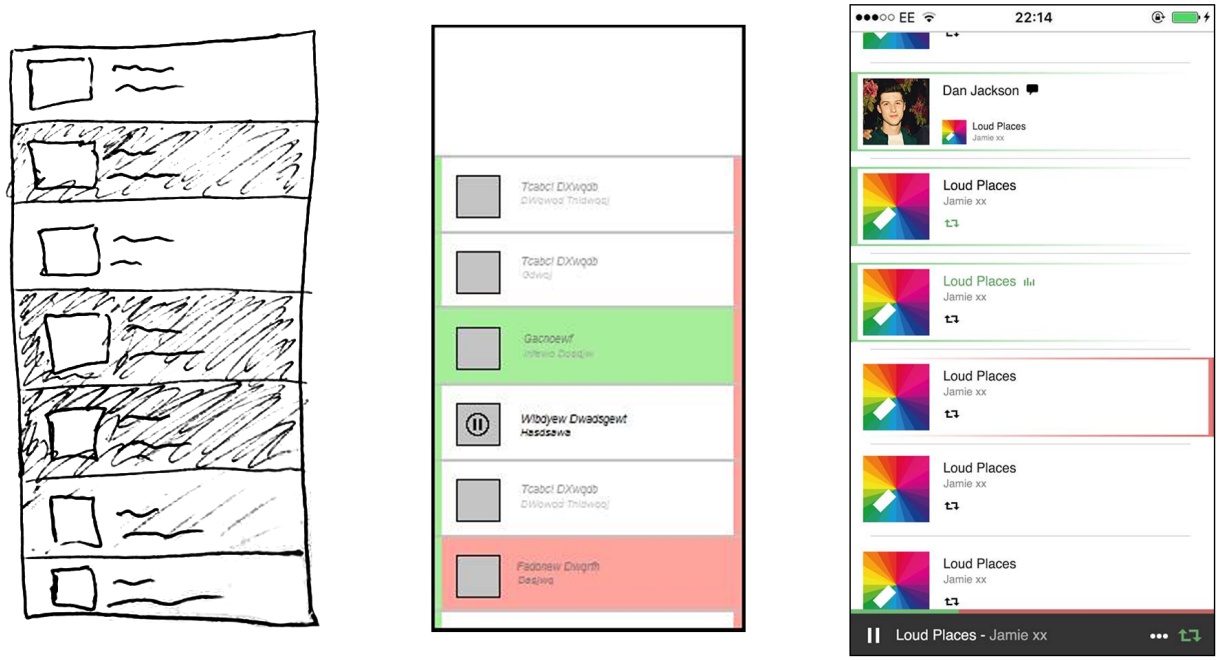
The simplicity of the swiping motion plays a large part in the effectiveness of the Tinder application; achieving what can be considered an important decision in a very short amount of time with very little effort plays a large role in the reusability of the application.

In his study on the principles of learning and behaviour, Domjan [6] explored the ideas proposed by behavioural psychologist B.F Skinner in 1957. Skinner found that when a behaviour is followed some form of positive reinforcement there is an increased probability that we will repeat the behaviour in the future. This pattern is reflected by the swiping feature on Tinder and could explain why users find the application so appealing; matches represent a reward for the user as they swipe through other user's profiles. Developing on this idea Skinner also studied whether we are more likely to keep gambling if we never win, always win or only win a proportion of the time; he found that the most effective results were produced by a reinforcement pattern in which only a proportion of the responses is rewarded and where this proportion is not fixed. Tinder also plays on this idea as not every right swipe results in a match though the user is periodically rewarded.

It is clear that there are many links between the matching functionality of tinder and the matching of music tastes that I am hoping to achieve. The functionality of swiping will be both a simple solution for the liking of disliking of recommendations as well as an intuitive interface which is likely to increase the appeal and re-usability of my application. This functionality could be achieved in my application by introducing a swiping system which rewards users for swiping on recommendations; though it should be noted that it seems a 100% success rate is not ideal and the possibility of a bad recommendation may actually be integral to the re-usability of this system. Swiping right on a recommended song could match the swiper with the user who originally shared that recommendation and then encourage communication between the two. By building a social network of matches the system could help find new users who are likely to share music tastes and produce good recommendations.

My initial interface designs were to include:

- A feed of recommendations shared by other users.
- An ability to like/dislike recommendations.
- A swiping functionality.
- An ability to share other users recommendations.
- A user profile.
- A messaging service.
- A music player to preview recommended songs.



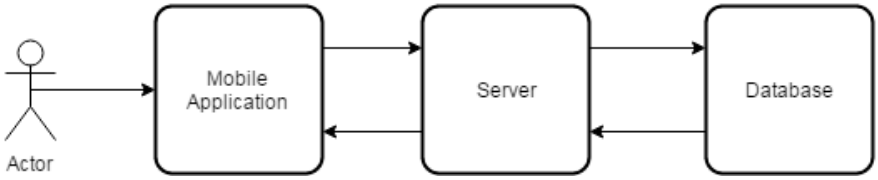
To maximize re-usability I decided to utilise the findings of Skinner by rewarding users for swiping within the application. The green borders around liked songs would reinforce a positive response and access to user profiles and direct messaging would only be available after a positive right swipe.

Since the initial designs were proposed developments have been made. The messaging and profile features are to be considered low priority as they are not integral to the recommendation of music; these features will be considered significant to the re-usability of the system but not the functionality and will only be implemented if all other features have been successfully implemented beforehand. It has also been decided since my initial designs that a share button is not required and the act of liking a song should automatically share it to the close connections of the user; these changes will be reflected in later iterations of the application design.

4.3 Remote Server

In order to keep the mobile application as simple and efficient as possible I will perform most necessary work from a remote server. It is not necessary for communication with the graph database to be performed from the mobile device itself and so the server component of the system will handle all communication between the components. The server component will receive requests from the mobile application to find new recommended songs and should perform work on the database itself before returning a list of results including only the most necessary and simple information.

4.4 Diagram



5 Implementation

5.1 Database

It was decided throughout my design process that a graph database should be used to store the users and their connections in my system. The consideration of implementing my own bespoke graph database led to the conclusion that such an approach would not fit within my time plan and would require reproducing functionality which could be gained from existing graph database implementations. Neo4j is an existing implementation of a graph database management system that offers access to a free community edition for small projects and includes all functionality required for my project. The system runs queries on its graph database through a query language named Cypher which provides an easy way for my system to search connections between users; these queries can be received externally through a REST API.

The Neo4j database can be also be accessed from a web browser as shown in Figure 1 and provides a graphical representation of your data and the results of your queries.

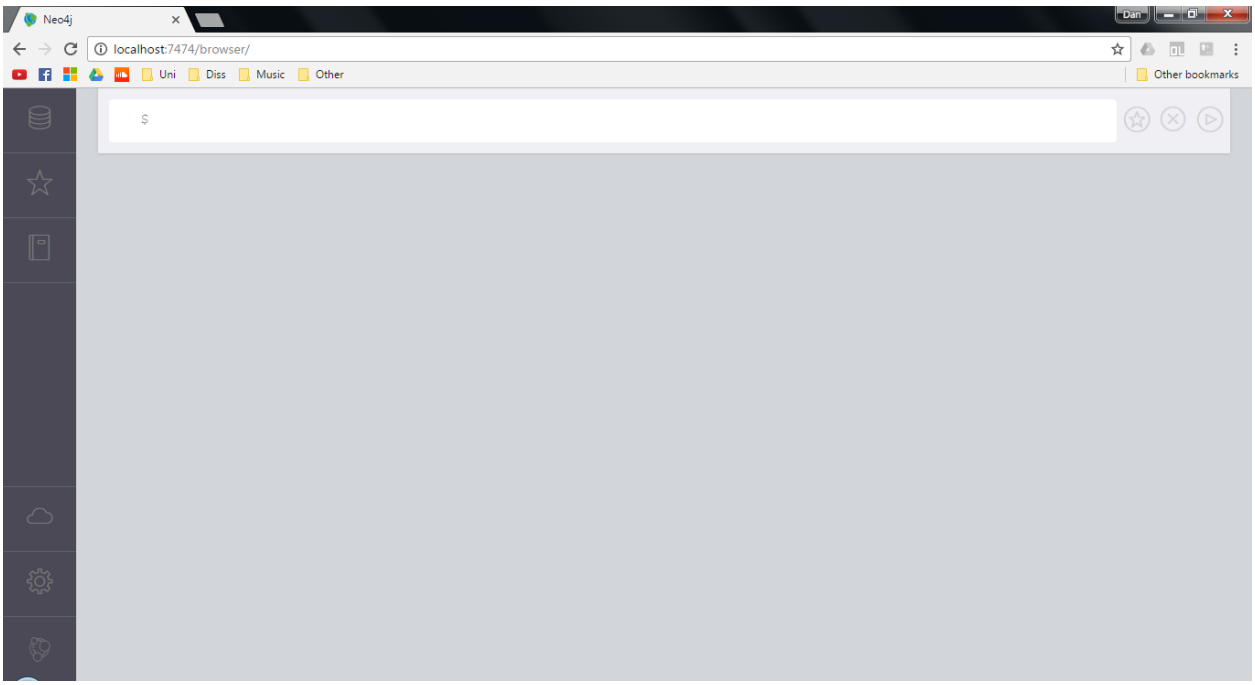


Figure 1: Neo4j web interface.

Running the following query:

```
CREATE
(user1:User { name: 'User1' }),
(user2:User { name: 'User2' }),
(user3:User { name: 'User3'}),
(user4:User { name: 'User4' }),
(user1)-[:CONNECTED]->(user2),
(user2)-[:CONNECTED]->(user3),
(user2)-[:CONNECTED]->(user4)
```

creates four new user nodes and defines relationships between them. The resulting data is represented within the web interface as shown in Figure 2.

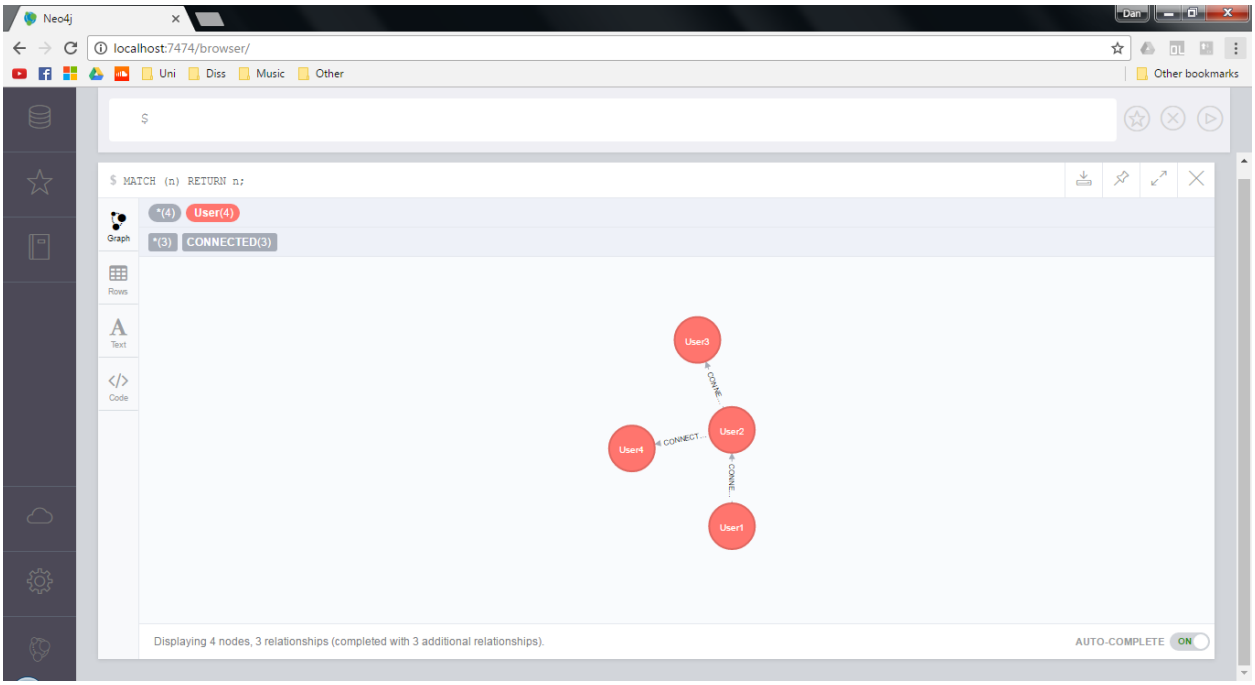


Figure 2: Neo4j web interface with nodes and relationships.

5.2 Mobile Application

As highlighted in the software requirements found in Appendix A the mobile application has been developed for the android operating system for KitKat version 4.4. One of the key reasons for this decision was the fact that android applications are developed in Java which is the language which I am currently most experienced with. I have also developed applications for android before and so am familiar with the framework and Android Studio. Version KitKat 4.4 was chosen because currently 25% of all android devices run 4.4 and the majority of the remaining devices run at least 4.4 or above.

5.3 Server

As the mobile application was to be developed in java it was decided that the implementation of the server component should also be developed in java for consistency and familiarity. Similarly, as Neo4j communicates through the REST API it was decided that the mobile application should communicate with the server component through REST requests. As a result it was decided that the server should run as a Jersey RESTful web service which allows for REST requests to be received from the mobile device.

5.4 Prototype

As proposed in my time plan I began the implementation of a prototype based on the initial research and design stages. This prototype was developed to provide basic functionality which could prove each component of the system was able to communicate successfully and the application could produce simple recommendations through the proposed approach.

The prototype application I have produced is able to retrieve lists of music data from an external source and display individual tracks to the user. By retrieving external music data I am able to provide an initial set of tracks to recommend for testing purposes. Last.fm is a music recommendation website which offers a public API that allows anyone to access their database of music. The API receives queries via URL connections and results are returned in the form of Json. My prototype is able to connect to this API and the code shown in Figures 3, 4 and 5 retrieves the liked tracks of a specified user and stores them in a list of custom Track objects.

```
private HttpURLConnection openURLConnection() throws IOException {
    final URL url = new URL("http://ws.audioscrobbler.com/2.0/?" +
            "method=user.getlovedtracks" +
            "&user=tegna" +
            "&api_key=705f6c8ad7cf18e713403234b1150754&format=json");
    return (HttpURLConnection) url.openConnection();
}
```

Figure 3: Connecting to the API via a URL query.

```
private List<Track> getTracksFromResults(final String result) {
    final List<Track> resultTracks = new ArrayList<>();
    try {
        final JSONArray jsonTracks = getJsonTracksFromResults(result);
        for (int i = 0; i < jsonTracks.length(); i++) {
            final JSONObject jsonTrack = jsonTracks.getJSONObject(i);
            final Track track = getTrackFromJson(jsonTrack);
            resultTracks.add(track);
        }
    } catch (final JSONException e) {
        Log.d("Error", e.getMessage().toString());
    }
    return resultTracks;
}
```

Figure 4: Receiving the Json results and storing them as a list of Track objects.

```
private Track getTrackFromJson(JSONObject track) throws JSONException {
    final JSONObject artist = track.getJSONObject("artist");
    final JSONArray images = track.getJSONArray("image");
    final JSONObject coverImage = images.getJSONObject(2);
    final Bitmap coverImageBitmap = getCoverImageBitmap(coverImage.getString("#text"));
    return new Track(track.getString("name"), artist.getString("name"), coverImageBitmap);
}
```

Figure 5: Parsing Json results into a Track object.

The implemented interface also allows users to swipe the tracks in their feed to the left or right and updates the colour of the item accordingly. The Track objects are displayed using an android ListView which makes use of an ArrayAdapter; the ArrayAdapter shown in Figure 6 takes a Track object and displays it's values in a ListView item.

```
public class TrackAdapter extends ArrayAdapter<Track> {

    public TrackAdapter(Context context, ArrayList<Track> tracks) { super(context, 0, tracks); }

    @Override
    public View getView(final int position, final View convertView, final ViewGroup parent) {
        View adapterView;
        if (convertView == null) {
            adapterView = LayoutInflater.from(getContext()).inflate(
                    R.layout.item_track, null);
        } else {
            adapterView = convertView;
        }
        final Track track = getItem(position);
        final TextView trackName = (TextView) adapterView.findViewById(R.id.trackName);
        trackName.setText(track.name);
        final TextView artistName = (TextView) adapterView.findViewById(R.id.artistName);
        artistName.setText(track.artist);
        final ImageView coverImage = (ImageView) adapterView.findViewById(R.id.albumCover);
        coverImage.setImageBitmap(track.coverImage);
        return adapterView;
    }

}
```

Figure 6: ArrayAdapter to display Track objects in the UI.

When the user swipes an item within the ListView the code in Figure 7 handles the appropriate response. Currently, if the user swipes to the right then the background of the item is set to green and if the user swipes left then it is set to red. In future developments of the application the system at this point will also send information to the server about the track which has been swiped and appropriate work will be performed.

```
@Override
public boolean onFling(MotionEvent e1, MotionEvent e2, float velocityX, float velocityY) {
    if (e2.getX() - e1.getX() > 50) {
        onSwipeRight(e1);
        return true;
    } else if (e1.getX() - e2.getX() > 50) {
        onSwipeLeft(e1);
        return true;
    }
    return false;
}

private void onSwipeRight(final MotionEvent e1) {
    final View swipedTrackView = getSwipedTrackView(e1);
    swipedTrackView.setBackgroundResource(R.drawable.green_gradient_selector);
}

private void onSwipeLeft(final MotionEvent e1) {
    final View swipedTrackView = getSwipedTrackView(e1);
    swipedTrackView.setBackgroundResource(R.drawable.red_gradient_selector);
}

private View getSwipedTrackView(final MotionEvent e1) {
    int id = trackListView.pointToPosition((int) e1.getX(), (int) e1.getY());
    return trackListView.getChildAt(id);
}
```

Figure 7: Handling gestures performed on the ListView items.

The resulting user interface shown in Figure 8 demonstrates the ListView displaying track data and the visual updates displayed as a result of tracks being swiped.

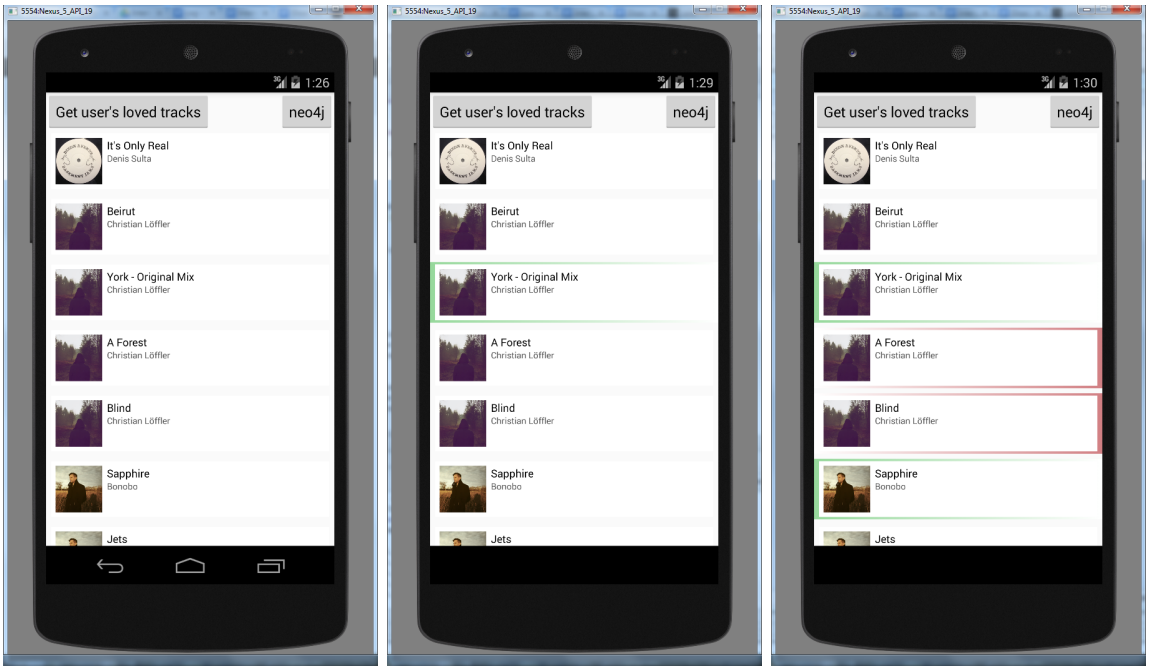


Figure 8: Android user interface.

A Jersey server implemented as a prototype can receive REST requests from the application and relay them on to the Neo4j database. The queries are sent to the database as URL's using code shown in Figure 9.

```
final Driver driver = GraphDatabase.driver("bolt://localhost", AuthTokens.basic("neo4j", "neo4j"));
final Session session = driver.session();
session.run("CREATE " +
        "(user1:User { name: 'User1' }),(" +
        "(user2:User { name: 'User2' }),(" +
        "(user3:User { name: 'User3'}),(" +
        "(user4:User { name: 'User4' }),(" +
        "(user1)-[:CONNECTED]->(user2),(" +
        "(user2)-[:CONNECTED]->(user3),(" +
        "(user2)-[:CONNECTED]->(user4)");
StatementResult result = session.run("MATCH (n) RETURN n;");
final List<Record> records = new ArrayList<>();
while (result.hasNext()) {
    final Record record = result.next();
    records.add(record);
}
session.close();
driver.close();
```

Figure 9: Querying the Neo4j database.

The implementation of these prototype components has shown that what I aim to achieve in my final product will be possible using the chosen technologies and so the implementation of a final system can begin.

6 Progress

6.1 Project Management

Throughout my project I have been following an agile development methodology which incorporates the use of sprints and scrum boards. Work has been split into sprints of two week periods and progress tracked on incremented scrum boards. The scrum boards are assigned with work 'To-Do' at the beginning of each sprint and as work progresses the individual tasks are moved through the stages 'In Progress' and 'Done'. Each task is assigned a label and due date dependant on its work category and estimated workload where tasks are primarily based on particular software requirements from the specification above. This approach has been working effectively on tasks which fit within the sprint's available time frame allows me to easily track the progress of all aspects of the project. The scrum boards allow me to visualise how each part of the system is developing independently and keep track of which requirements are yet to be completed.

My initial development stage, following the research and design stages, was the production of a basic prototype to underpin all basic functionality that my system requires to operate. The prototype was produced in order to ensure that all fundamental aspects of the project were achievable and to help me introduce developments to my initial designs. This has been a very effective development approach as the implementation of the most basic features has allowed me to understand the challenges I will face in implementing the full system and provided me with an opportunity to develop my initial ideas into designs into new approaches will be more effective.

The Gantt Chart proposed in my project plan seen in Figure 10 has been the most effective factor in tracking the progress of my work. In my initial project plan I highlighted the main obstacles I would face and the justifications for the assignment of tasks throughout my time plan. I expected that coursework, exams and revision would hinder productivity and so blocked those expected areas out accordingly. I also highlighted that my autumn semester would include 70 credits worth of modules whereas my summer semester would include just 50, predicting that the busiest time of the year would be throughout the first semester of work.

As the project has developed and work has been completed the Gantt Chart was updated accordingly to reflect the true progress of the project and the most up to date version can be seen in Figure 11.

As highlighted in my project plan most of the research and planning stages were assigned at the very beginning of the project as this was the period in which the least obstacles were expected. Despite the large quantity of work assigned early on in the project most of these tasks were completed within their given time frames. However, the research and planning tasks related to recommendation algorithms have been pushed back as their priority was considered lower than other aspects of the project research. Pushing these tasks back has caused a problem as some research tasks are still outstanding and work on these aspects of the system will be held up until that research can be completed.

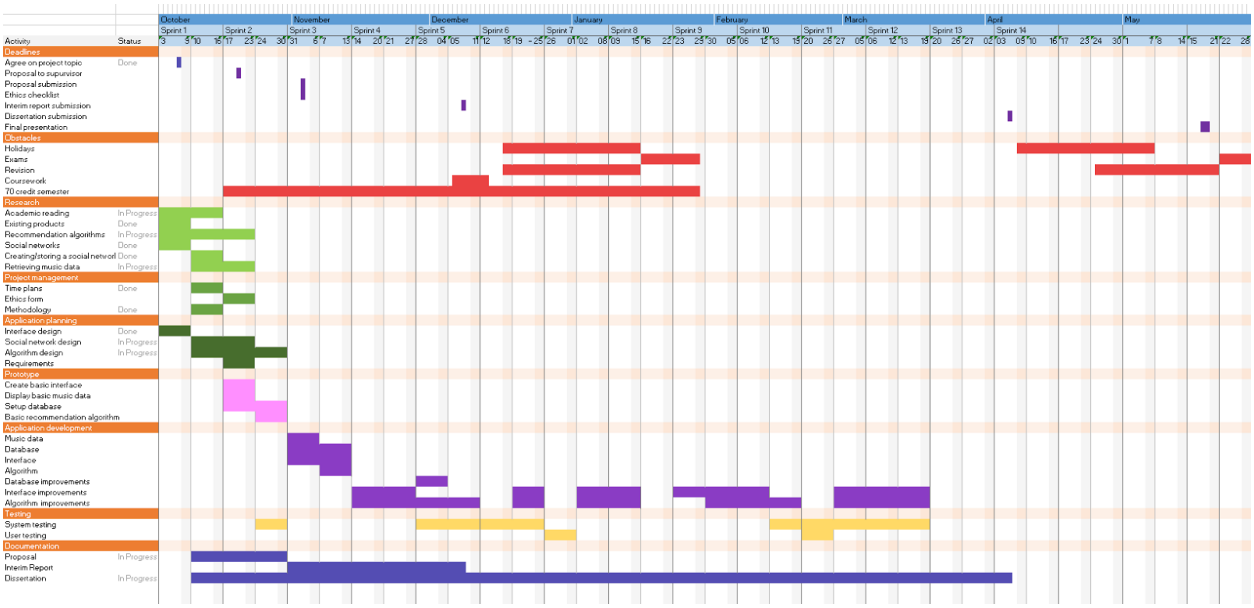


Figure 10: Gantt chart produced at the start of the project.

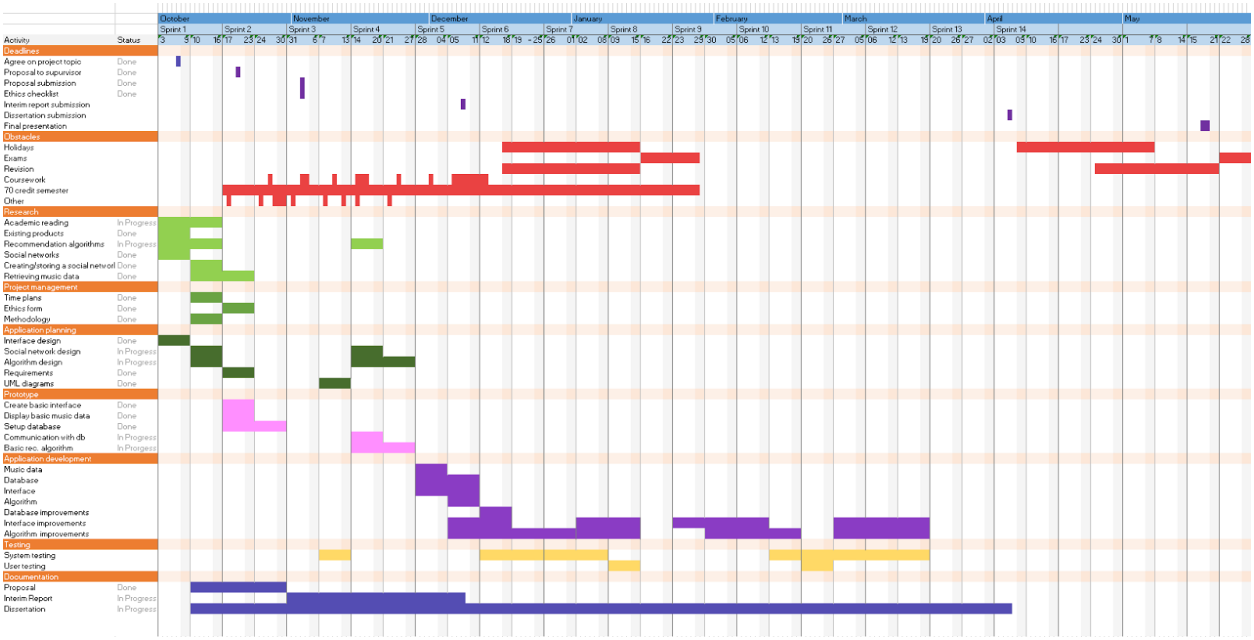


Figure 11: Current state of the gantt chart.

Sprint 2 successfully produced a basic prototype which was able to display music data in a mobile interface and provided swiping functionality on individual tracks. However, as predicted in my initial plan, the main hindrance for the development of the project were time constraints produced as a result of coursework within the 70 credit autumn semester. Prototype work in Sprint 3 was held back by large amounts of coursework and so tasks related to communication with the database and introduction of a recommendation algorithm were put on hold with a lower priority to the writing of this interim report. As a result of this the tasks which were not completed for

their assigned sprint have been pushed back to their following sprint and in some cases pushed back once more. It can be seen in the updated time plan that work on the prototype is still in progress when the implementation of the full system should already have started and as a result shifting all implementation tasks backwards on the chart. The delay of each individual task between sprints is being tracked on the work log shown in Figure 12 which highlights periods of time in which the project has been worked on and periods justifying which work was not completed.

Although each delay has been justified within the work log and though it was predicted within the

Log

Date	Task
Thu 27th Oct	Working on PEC coursework - step 2 assignment
Fri 28th - Sun 30th Oct	Halloween weekend
Thu 3rd Nov	Working on PEC coursework - step 3 assignment
Thu 3rd - Fri 4th Nov	Working on MDP coursework 1 - mobile application development
Sat 5th Nov	Implementation of swiping
Mon 7th Nov	PEC coursework 1 - assigned: deadline 12th December
Mon 7th Nov	Meeting: Interim prototype to be done by 21st November Risk assessment of private data
Thu 10th Nov	Working on PEC coursework - step 4 assignment
Fri 11th Nov	Interim report prototype
Mon 14th	Meeting: cancelled
Tue 15th Nov	UML Diagrams
Tue 15th - Thu 17th Nov	Working on MDP coursework 2 - mobile application development
Thu 17th Nov	Working on PEC coursework - step 5 assignment
Fri 18th Nov	Interim report

Sprint Delays

Planned For	Task	Completed In
Sprint 2	Recommendation algorithm research	Sprint 4
Sprint 2	Social network design	Sprint 4
Sprint 2	Communicate with database	Sprint 4
Sprint 2	Basic music recommendation algorithm	Sprint 4
Sprint 2	Prototype testing	**Sprint 3**
Sprint 2	Algorithm design	Sprint 4
Sprint 2	Fix scrolling of ListView	Sprint 4
Sprint 2	UML Diagrams	**Sprint 3**
Sprint 3	Interface development	Sprint 4
Sprint 3	Retrieving more music/user data	Sprint 4
Sprint 3	Database development	Sprint 4
Sprint 3	Recommendation algorithm iplementation	Sprint 4

Figure 12: Work log and tasks which have been delayed.

plan that these delays would exist, it remains a problem that work on the project has been pushed back so drastically on the project's time plan. To compensate for the shift in work towards the back end of my project I will have to re-assign future work more regularly than initially planned. The 50-credit summer semester will mean that I have more time to work on this project but the predicted workload was underestimated in my initial plan. As I have been following an agile methodology the work which is not yet completed has not caused a need to update the methodology in use though the work assigned to later sprints will need to be reconsidered. The gantt chart will continue to be updated throughout the project and assigned work shifted backwards accordingly.

6.2 Contributions and Reflections

The project so far has established that there is definite potential in the ideas proposed within my initial project plan. It can be seen from the ongoing research of related work and implementation that the recommendation techniques and proposed application will be a new and unique attempt at solving the problem of music recommendation. The prototype implementation has already shown that the aims and objectives of the project are realistic and despite the setbacks of time constraints the project is well under way. I believe that the approach to the mobile application's simple, intuitive and effective interface has been an innovative solution to the proposed problems laid out in my initial plan and offers a novel approach to the control of recommendations by users. I also

believe that the approach I am taking to produce my recommendations is an innovating strategy and has shown that the implementation of such approaches will prove a novel solution to music recommendation.

I am very happy with the initial stages of my project and the direction in which it is heading. However, I am unhappy with the speed at which it is progressing and hope that this can be increased towards the end of the project. If I could start this project again I would be more active in the initial research stages of the project and be more strict on the deadlines that I set myself. Sticking more strictly to deadlines would have meant that at this point in time I would be much further ahead in the implementation of my system.

7 Appendices

7.1 Appendix A - Software Requirements

Software Requirements

High Medium Low priority
Non-functional requirements = N
Functional requirements = F

ID	Description	
Operating system		
1.1	The user wants to be able to download the application from the Android app store	N
1.1.1	The application must be developed for the Android operating system	F
1.1.2	The application must run on Android operating system version KitKat 4.4 and above	F
1.1.3	The application must be downloadable from the Android app store	F
1.1.4	The user wants the application to be free	N
Usability		
2.1	The user wants the application to present music recommendations quickly	N
2.1.1	The application must load within two seconds of opening	F
2.1.2	The application must generate recommendations in less than one second	F
2.2	The user wants the application to be responsive	N
2.3	The user does not want the application to crash	N
Recommendations		
3.1	The user wants to receive reliable music recommendations	N
3.1.1	The application must find users with similar music tastes to the current user	F
3.1.2	The application must retrieve music recommendations from users with similar tastes	F

Interface		
4.1	The user wants the interface to be intuitive and easy to use	N
4.2	The application must display a feed of music recommendations	F
4.2.1	The feed must display at least four recommendations on screen at any given time	F
4.3	The user wants to see information about a recommended song	N
4.3.1	The application must display the name of a recommended song	F
4.3.2	The application must display the artist who created a recommended song	F
4.3.3	The application must display the album cover of a recommended song	F
4.3.4	The application must display song information simply and clearly	N
4.4	The feed must be presented on the homepage of the application	F
4.5	The applications design scheme must be consistent throughout	N
Interactivity		
5.1	The application must respond to the touch screen input of a user	F
5.2	The user wants to be able to scroll through their feed of recommendations	N
5.2.1	The feed of music must be scrollable	F
5.3	The user wants to be able to like and dislike songs	N
5.3.1	The application must allow users to *like* songs	F
5.3.2	The application must allow users to *dislike* songs	F
5.3.3	The user wants the liking and disliking process of songs to be intuitive	N
5.4	The user wants to be able to listen to recommended songs	N
5.4.1	The application must be able to play 30 second previews of songs	F

Personal		
6.1	The user wants to be able to create their own personal account	N
6.11	The application must be able to store user account information	F
6.1.2	The application must have a user login screen	F
6.1.3	The login screen must be secure	F
6.1.4	The application must have a user logout screen	F
6.2	The user wants access to playlists from their accounts on external systems	N
6.2.1	The application must be able to access information from external music libraries	F
6.2.2	The application must be able to store external music data	F
6.3	The user wants to see all of the recommendations they have previously liked	N
6.3.1	The application must store all previous likes by a user	F
6.3.2	The application must be able to display a list of all previous user likes	F
6.4	The user wants to recommend songs to other users	N
6.4.1	The application must allow users to recommend specific songs to other users	F
Database		
7.1	The application must be able to connect to a database of users	F
7.1.1	The user database must be able to connect users together	F
7.2	The application must be able to connect to a database of music	F
7.2.1	The music database must store information about individual songs	F

References

[1] Statista, "Number of paying Spotify subscribers worldwide from July 2010 to September 2016 (in millions)", [Online] Available: https://www.statista.com/statistics/244995/number-of-paying-spotify-subscribers/. [Accessed: 8 December 2016]

[2] Cedric Mesnage, Asma Rafiq, Simon Dixon, Romain Brixtel "Music Discovery with Social Networks" October 2011

[3] Ioannis Konstas, Vassilios Stathopoulos, Joemon M Jose "On Social Networks and Collaborative Recommendation" July 2009

[4] Upendra Shardanand "Social Information Filtering for Music Recommendation" September 1994

[5] Hao Ma, Dengyong Zhou, Chao Liu, Michael R. Lyu, Irwin King "Recommender Systems with Social Regularization" February 2011

[6] Michael Domjan "The Principles of Learning Behaviour" January 2014

[7] Justin J Miller "Graph Database Applications and Concepts with Neo4j" March 2013

[8] Neo4j, "From Relational to Neo4j", [Online] Available: http://neo4j.com/developer/graph-db-vs-rdbms/. [Accessed: 8 December 2016]