Table of Contents

Analysis

Background to the Problem

Digital streaming within music has become increasingly popular over the past decade, with over 1,000,364,000 active users on the most popular streaming platforms (excluding YouTube). An automatic music recommendation system is an information filtering system that seeks to predict the 'rating' or 'preference' that a user would give to an item, which in this case is a song. Recommendation models can be trained with various data factors, with the design including the two main types of filtering: collaborative filtering and content filtering. Collaborative filtering is the popular choice, using similar user listening profiles to predict what tracks a user might react positively towards. To recommend the most accurate songs to its users, services will use a mixture of filtering methods. Not only is it about finding songs similar to user tastes, but also suggesting music that the user has no previous relationships with, a problem commonly known as 'cold start'. Since most music is now sold and streamed digitally, streaming platforms are continuously updating their algorithms to recommend the most suitable songs to their users.

I will be developing a new recommendation system for my client, student Daisy Thomas. Daisy uses these automatic music recommendation tools to discover new music, with Spotify being the service primarily used. Spotify is an audio streaming and media services provider founded in 2006 by Daniel Ek. It is the world's largest music streaming service provider, with over 356 million monthly active users, including 158 million paying subscribers, as of March 2021. Recommendations are made from a user's listening activity and user-created playlists. From the playlist, Spotify analyses the features of the songs as well as the user's listening history to recommend 6 songs they think the user will like.
The current system my client uses lacks functionality in terms of being able to keep track of recommended songs, and often recommends the same songs after being told the user does not think it is a good recommendation. The client has expressed that she finds that there is a limited selection to the variety of music she is offered and that most recommendations are songs by similar artists or songs she has already added to playlists.

Interviews with Client and Prospective Users

While Daisy Thomas is the primary client, she has informed me that she would like the system designed with another student's (Alexandra Smith) feedback in consideration as well. To understand what they were looking for in this new system, I conducted an interview with each of them.

Daisy Thomas **(DT)** – Primary Client
Alexandra Smith **(AS)** – User of the System

What is the current tool(s) that you use for music recommendation?
DT: I use Spotify and its recommendation features after switching from Amazon Music. I also use the website *Moodify* to get other recommendations.
AS: I use Spotify, I listen to my 'discover weekly' as well as the artist radio playlists it provides me.

Why do you use this tool?
DT: I find that it is organised and helpful and gives me recommendations that reflect my listening tastes.
AS: I find it recommends me a wide range of new and exciting music.

What features of this tool do you find makes it better than others?
DT: The organisational features of where your playlist lies compared to the recommendations so I can filter through music I know I already like and the new songs. It also has a feature where it can blend two user's listening tastes to create one mixed playlist of recommended songs for both users.
AS: The quantity and variety of music, it feels like there is a little man recommending me the songs I will like most.

Are there any drawbacks to this service?
DT: It automatically refreshes once a week, and I feel the pressure that I have missed out of a song I may have really liked, and that this will affect my future recommendations.
AS: Sometimes it can get stuck in a loop and keep recommending the same songs.

How do you think that the music recommendations are accurate?
DT: There is a lot of influence from the artists I already listen to and solo work from collaborations of artists I listen to. Also, they have similar genres and similar sounding voices, and it also features music from similar eras to what I already listen to.
AS: I like the songs and believe they are of a similar taste to the songs I generally listen to.

What do you believe are the most important features in a music recommendation tool?
DT: It should be based on the listening activity of other users who listen to the same artists as you.
AS: Variety and innovation.

What would you want your recommendations to be based on?
DT: Based on your whole listening activity and then also having options based off specific playlists.

AS: I think the option of choosing where from would be best, but from a specific playlist or a specific song are fine.

How would you like these recommendations to be formatted? (i.e., in a playlist, a list of songs....)
DT: In a playlist arranged by genre or that have similarities.
AS: A playlist so I can easily listen to them. Either where each song flows into one another so there is a similarity between them as they differentiate, or just a variety of songs.

How many songs do you think would be sufficient for this tool?
DT: 20? Maybe give the user the option to choose how many songs they would like.
AS: Between 15 and 25 if you can maintain accuracy.

From the interviews, I have concluded that Spotify has the most efficient automatic music recommendation service available to my client, and that this is what I am trying to improve. The prospective users have also confirmed these findings, while also giving me more information on how the data would be best organised and presented. I have also gained insight into how my recommendation system should be presented to the user, with the presentation of the findings important to my client. I also have an idea of the factors that my client would like the model to prioritise when recommending.

Observation of Existing Services

Through observing the two prospective users using the services they had mentioned in the interview, I gained a deeper understanding of the relationship between the services and user interaction and was able to identify any problems that arose during the process.

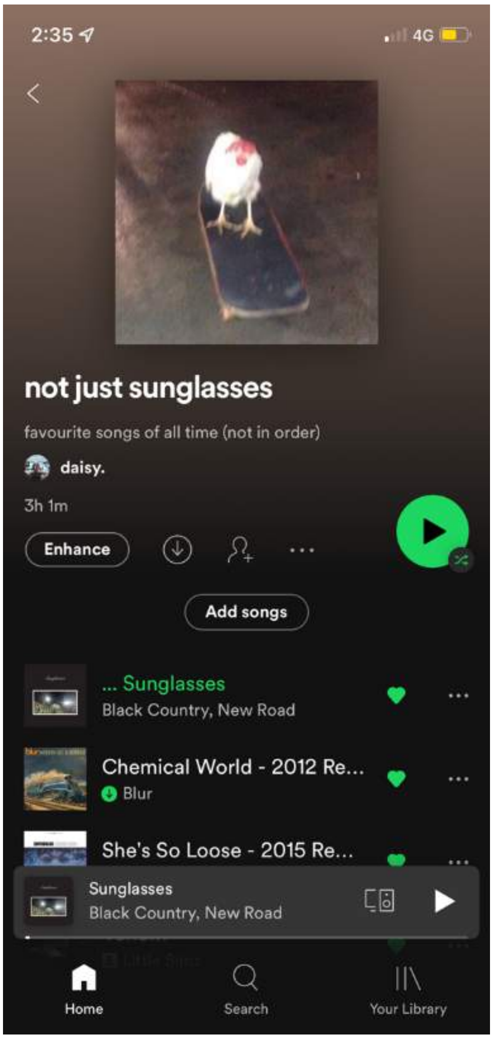
To create a successful music recommendation system, research into the different methods companies use to implement existing systems is crucial. Through listening habits and the actual structure of the song itself, many companies strive to achieve the most efficient recommendation system. A variety of machine learning algorithms are used to select the perfect song to recommend to the user. Both collaborative and content-based filtering is used to select suitable music, with each technique focusing on certain data to compare against other songs.

Collaborative filtering relies on listening data. Using matrix manipulation, a weight matrix factorization (WMF) algorithm is applied against a rating matrix (R) and uses two matrices to predict which song is more likely to be listened to. Content-based filtering relies on the actual data of the song itself. The data could include and is not limited to, the raw audio data, audio data concerning tempo and length, or the descriptive metadata about the artist and track name. Metadata like a song play count and the tempo of the song are important factors when it comes to training models, and access to this data is very important. Spotify also uses these algorithms to recommend music to its users.

From the interview, there are specific features within Spotify that the prospective users use most frequently. From this selection, the two most popular features of the service that I will be observing are:

1. User-generated playlists
2. ‘Discover Weekly’

User-Generated Playlists



not just sunglasses

favourite songs of all time (not in order)

daisy.

3h 1m

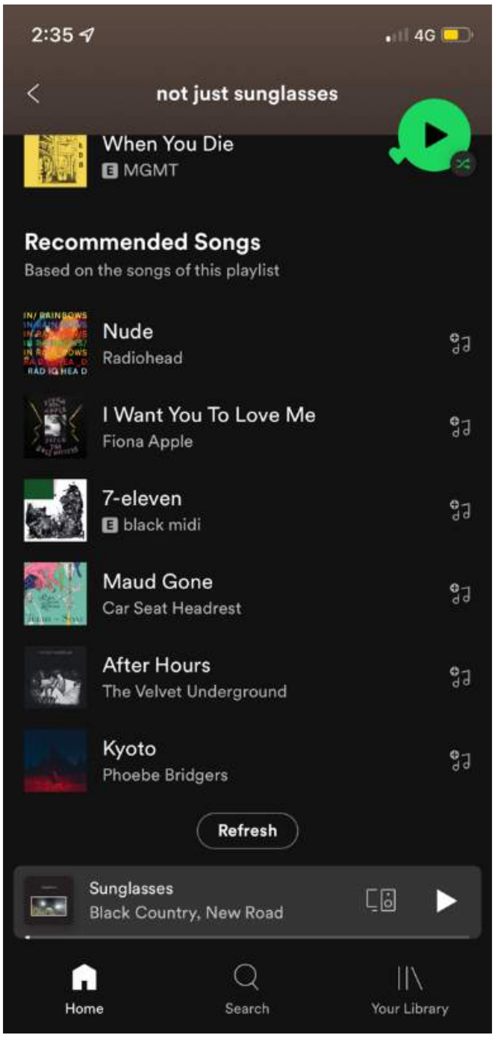
... Sunglasses
Black Country, New Road

Chemical World - 2012 Re...
Blur

She's So Loose - 2015 Re...

Sunglasses
Black Country, New Road

Figure 1



not just sunglasses

When You Die
E MGMT

Recommended Songs
Based on the songs of this playlist

Nude
Radiohead

I Want You To Love Me
Fiona Apple

7-eleven
E black midi

Maud Gone
Car Seat Headrest

After Hours
The Velvet Underground

Kyoto
Phoebe Bridgers

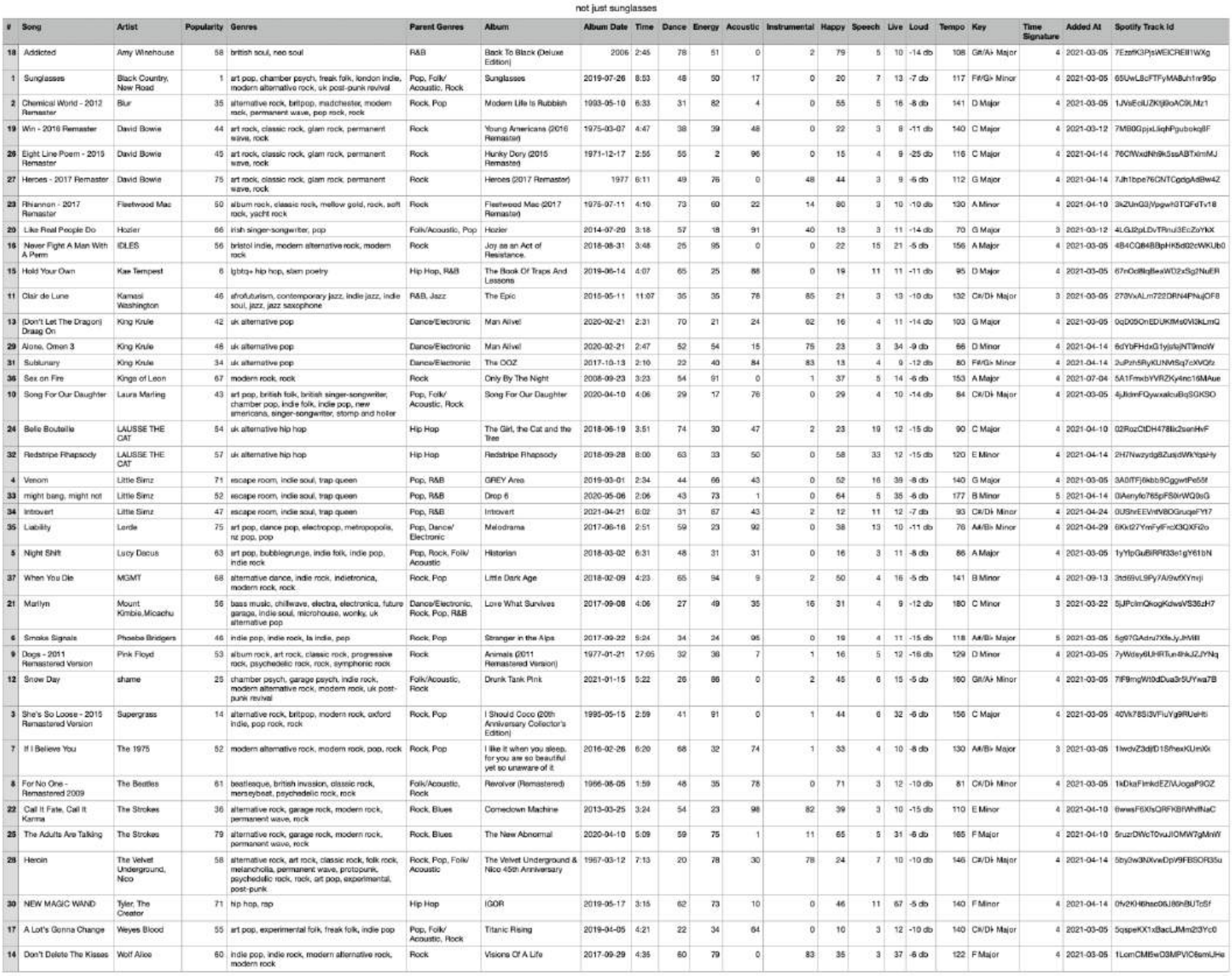
Sunglasses
Black Country, New Road

Figure 2

First, I looked at one of the client's Spotify playlists. Spotify allows users to create their own playlists and personalise them with their own cover art, name, and description [figure 1]. Once a user has added songs to their playlist, a 'Recommended Songs' section underneath the playlist will appear [figure 2] with 6 tracks that Spotify believes are like the songs in the playlist and that the user will enjoy. The user also has the option to refresh these recommended songs to get a new set of tracks. Whilst analysing the client using this feature, I realised that every time the user re-enters the playlist, the list of recommended songs refreshes, even if the user has not had a chance to properly look at them.

I then looked at the data of the songs within the playlist to understand why Spotify recommended these songs. Using the website 'https://www.chosic.com/spotify-playlist-analyzer/', I captured the information about the playlist, in the format of a .csv file [figure 3]. It is formatted in the same way as Spotify gives the data to a system, it is easier to visualise it using a pre-made tool rather than creating a new tool.

The data is acquired from a collection of user-generated data as well as using the company The Echo Nest, a music intelligence and data platform for developers and media companies owned by Spotify.



not just sunglasses

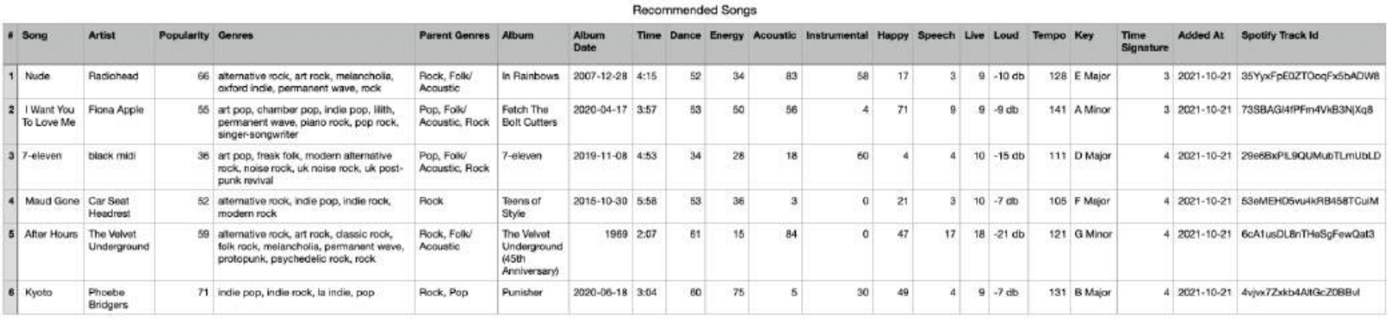
#	Song	Artist	Popularity	Genres	Parent Genres	Album	Album Date	Time	Dance	Energy	Acoustic	Instrumental	Happy	Speech	Live	Loud	Tempo	Key	Time Signature	Added At	Spotify Track Id
18	Addicted	Amy Winehouse	58	british soul, neo soul	R&B	Back To Black (Deluxe Edition)	2006	2:45	78	51	0	2	79	5	10	-14 db	108	G#/A♭ Major	4	2021-03-05	7EzofK3PjsWEICREII1WXg
1	Sunglasses	Black Country, New Road	1	art pop, chamber psych, freak folk, london indie, modern alternative rock, uk post-punk revival	Pop, Folk/ Acoustic, Rock	Sunglasses	2019-07-26	8:53	48	50	17	0	20	7	13	-7 db	117	F#/G♭ Minor	4	2021-03-05	65UwL8cFTFyMABuh1rr95p
2	Chemical World - 2012 Remaster	Blur	35	alternative rock, britpop, madchester, modern rock, permanent wave, pop rock, rock	Rock, Pop	Modern Life Is Rubbish	1993-05-10	6:33	31	82	4	0	55	5	16	-8 db	141	D Major	4	2021-03-05	1JVsEclUZKtj9oACtXLMz1
19	Win - 2016 Remaster	David Bowie	44	art rock, classic rock, glam rock, permanent wave, rock	Rock	Young Americans (2016 Remaster)	1975-03-07	4:47	38	39	48	0	22	3	8	-11 db	140	C Major	4	2021-03-12	7MB0GpjxLiqhPgubokq8F
26	Eight Line Poem - 2015 Remaster	David Bowie	45	art rock, classic rock, glam rock, permanent wave, rock	Rock	Hunky Dory (2015 Remaster)	1971-12-17	2:55	55	2	96	0	15	4	9	-25 db	116	C Major	4	2021-04-14	76CfWxdNh9k5ssABTxlmMJ
27	Heroes - 2017 Remaster	David Bowie	75	art rock, classic rock, glam rock, permanent wave, rock	Rock	Heroes (2017 Remaster)	1977	6:11	49	76	0	48	44	3	9	-6 db	112	G Major	4	2021-04-14	7Jh1bpe76CNTCgdgAdBw4Z
23	Rhiannon - 2017 Remaster	Fleetwood Mac	50	album rock, classic rock, mellow gold, rock, soft rock, yacht rock	Rock	Fleetwood Mac (2017 Remaster)	1975-07-11	4:10	73	60	22	14	80	3	10	-10 db	130	A Minor	4	2021-04-10	3kZUnG3jVpgwH3TQFdTv18
20	Like Real People Do	Hozier	66	irish singer-songwriter, pop	Folk/Acoustic, Pop	Hozier	2014-07-20	3:18	57	18	91	40	13	3	11	-14 db	70	G Major	3	2021-03-12	4LGJ2pLDvTRnul3EcZoYkX
16	Never Fight A Man With A Perm	IDLES	56	bristol indie, modern alternative rock, modern rock	Rock	Joy as an Act of Resistance.	2018-08-31	3:48	25	95	0	0	22	15	21	-5 db	156	A Major	4	2021-03-05	4B4CQ84BBpHK5d02cWKUb0
15	Hold Your Own	Kae Tempest	6	lgbtq+ hip hop, slam poetry	Hip Hop, R&B	The Book Of Traps And Lessons	2019-06-14	4:07	65	25	88	0	19	11	11	-11 db	95	D Major	4	2021-03-05	67nOd8lqBeaWD2xSg2NuER
11	Clair de Lune	Kamasi Washington	46	afrofuturism, contemporary jazz, indie jazz, indie soul, jazz, jazz saxophone	R&B, Jazz	The Epic	2015-05-11	11:07	35	35	78	85	21	3	13	-10 db	132	C#/D♭ Major	3	2021-03-05	273VxALm722DRN4PNujOF8
13	(Don't Let The Dragon) Draag On	King Krule	42	uk alternative pop	Dance/Electronic	Man Alive!	2020-02-21	2:31	70	21	24	62	16	4	11	-14 db	103	G Major	4	2021-03-05	0qD05OnEDUKfMs0Vi3KLmQ
29	Alone, Omen 3	King Krule	46	uk alternative pop	Dance/Electronic	Man Alive!	2020-02-21	2:47	52	54	15	75	23	3	34	-9 db	66	D Minor	4	2021-04-14	6dYbFHdxG1yjsfajNT9mdW
31	Sublunary	King Krule	34	uk alternative pop	Dance/Electronic	The OOZ	2017-10-13	2:10	22	40	84	83	13	4	9	-12 db	80	F#/G♭ Minor	4	2021-04-14	2uPzh5RyKUNMSq7cXVQfz
36	Sex on Fire	Kings of Leon	67	modern rock, rock	Rock	Only By The Night	2008-09-23	3:23	54	91	0	1	37	5	14	-6 db	153	A Major	4	2021-07-04	5A1FmxbYVRZKy4nc16MAue
10	Song For Our Daughter	Laura Marling	43	art pop, british folk, british singer-songwriter, chamber pop, indie folk, indie pop, new americana, singer-songwriter, stomp and holler	Pop, Folk/ Acoustic, Rock	Song For Our Daughter	2020-04-10	4:06	29	17	76	0	29	4	10	-14 db	84	C#/D♭ Major	4	2021-03-05	4jJldmFQywxalcuBqSGKSO
24	Belle Bouteille	LAUSSE THE CAT	54	uk alternative hip hop	Hip Hop	The Girl, the Cat and the Tree	2018-06-19	3:51	74	30	47	2	23	19	12	-15 db	90	C Major	4	2021-04-10	02RozCtDH478lk2senHvF
32	Redstripe Rhapsody	LAUSSE THE CAT	57	uk alternative hip hop	Hip Hop	Redstripe Rhapsody	2018-09-28	8:00	63	33	50	0	58	33	12	-15 db	120	E Minor	4	2021-04-14	2H7NwzydgBZusjdWkYqsHy
4	Venom	Little Simz	71	escape room, indie soul, trap queen	Pop, R&B	GREY Area	2019-03-01	2:34	44	66	43	0	52	16	39	-8 db	140	G Major	4	2021-03-05	3A0lTFj6kbb9CggwtPe55f
33	might bang, might not	Little Simz	52	escape room, indie soul, trap queen	Pop, R&B	Drop 6	2020-05-06	2:06	43	73	1	0	64	5	35	-6 db	177	B Minor	5	2021-04-14	0Aenyfo765pFS8rWQ0sG
34	Introvert	Little Simz	47	escape room, indie soul, trap queen	Pop, R&B	Introvert	2021-04-21	6:02	31	67	43	2	12	11	12	-7 db	93	C#/D♭ Minor	4	2021-04-24	0UShrEEVntV8OGruqeFYt7
35	Liability	Lorde	75	art pop, dance pop, electropop, metropopolis, nz pop, pop	Pop, Dance/ Electronic	Melodrama	2017-06-16	2:51	59	23	92	0	38	13	10	-11 db	76	A#/B♭ Minor	4	2021-04-29	6Kkt27YmFylFrcX3QXFi2o
5	Night Shift	Lucy Dacus	63	art pop, bubblegrunge, indie folk, indie pop, indie rock	Pop, Rock, Folk/ Acoustic	Historian	2018-03-02	6:31	48	31	31	0	16	3	11	-8 db	86	A Major	4	2021-03-05	1yYlpGuBIRRf33e1gY61bN
37	When You Die	MGMT	68	alternative dance, indie rock, indietronica, modern rock, rock	Rock, Pop	Little Dark Age	2018-02-09	4:23	65	94	9	2	50	4	16	-5 db	141	B Minor	4	2021-09-13	3td69vL9Py7A/9wfXYmjl
21	Marilyn	Mount Kimbie,Micachu	56	bass music, chillwave, electra, electronica, future garage, indie soul, microhouse, wonky, uk alternative pop	Dance/Electronic, Rock, Pop, R&B	Love What Survives	2017-09-08	4:06	27	49	35	16	31	4	9	-12 db	180	C Minor	3	2021-03-22	5JPclmQkogKdwsVS36zH7
6	Smoke Signals	Phoebe Bridgers	46	indie pop, indie rock, la indie, pop	Rock, Pop	Stranger in the Alps	2017-09-22	5:24	34	24	95	0	19	4	11	-15 db	118	A#/B♭ Major	5	2021-03-05	5g97GAdru7XfeJyJHMlI
9	Dogs - 2011 Remastered Version	Pink Floyd	53	album rock, art rock, classic rock, progressive rock, psychedelic rock, rock, symphonic rock	Rock	Animals (2011 Remastered Version)	1977-01-21	17:05	32	38	7	1	16	5	12	-16 db	129	D Minor	4	2021-03-05	7yWdsy6UHRTun4hkJZJYNq
12	Snow Day	shame	25	chamber psych, garage psych, indie rock, modern alternative rock, modern rock, uk post-punk revival	Folk/Acoustic, Rock	Drunk Tank Pink	2021-01-15	5:22	26	86	0	2	45	6	15	-5 db	160	G#/A♭ Minor	4	2021-03-05	7IF9mgWt0dDua3r5UYwa7B
3	She's So Loose - 2015 Remastered Version	Supergrass	14	alternative rock, britpop, modern rock, oxford indie, pop rock, rock	Rock, Pop	I Should Coco (20th Anniversary Collector's Edition)	1995-05-15	2:59	41	91	0	1	44	6	32	-6 db	156	C Major	4	2021-03-05	40Vk78Si3VFiuYg9RUeHti
7	If I Believe You	The 1975	52	modern alternative rock, modern rock, pop, rock	Rock, Pop	I like it when you sleep, for you are so beautiful yet so unaware of it	2016-02-26	6:20	68	32	74	1	33	4	10	-8 db	130	A#/B♭ Major	3	2021-03-05	1lwdvZ3dijD1SfhexKUmXx
8	For No One - Remastered 2009	The Beatles	61	beatlesque, british invasion, classic rock, merseybeat, psychedelic rock, rock	Folk/Acoustic, Rock	Revolver (Remastered)	1966-08-05	1:59	48	35	78	0	71	3	12	-10 db	81	C#/D♭ Minor	4	2021-03-05	1kDkaFlmkdEZiVJogaP9OZ
22	Call It Fate, Call It Karma	The Strokes	36	alternative rock, garage rock, modern rock, permanent wave, rock	Rock, Blues	Comedown Machine	2013-03-25	3:24	54	23	98	82	39	3	10	-15 db	110	E Minor	4	2021-04-10	6wwsF6XfsQRFK8fWhifNaC
25	The Adults Are Talking	The Strokes	79	alternative rock, garage rock, modern rock, permanent wave, rock	Rock, Blues	The New Abnormal	2020-04-10	5:09	59	75	1	11	65	5	31	-6 db	165	F Major	4	2021-04-10	5ruzDWcT0vuJIOMW7gMnW
28	Heroin	The Velvet Underground, Nico	58	alternative rock, art rock, classic rock, folk rock, melancholia, permanent wave, protopunk, psychedelic rock, rock, art pop, experimental, post-punk	Rock, Pop, Folk/ Acoustic	The Velvet Underground & Nico 45th Anniversary	1967-03-12	7:13	20	78	30	78	24	7	10	-10 db	146	C#/D♭ Major	4	2021-04-14	5by3w3NXvwDpV9FBSOR35u
30	NEW MAGIC WAND	Tyler, The Creator	71	hip hop, rap	Hip Hop	IGOR	2019-05-17	3:15	62	73	10	0	46	11	67	-5 db	140	F Minor	4	2021-04-14	0fv2KH6hsc06J86hBUTcSf
17	A Lot's Gonna Change	Weyes Blood	55	art pop, experimental folk, freak folk, indie pop	Pop, Folk/ Acoustic, Rock	Titanic Rising	2019-04-05	4:21	22	34	64	0	10	3	12	-10 db	140	C#/D♭ Major	4	2021-03-05	5qspeKX1xBacLJMm2i3Yc0
14	Don't Delete The Kisses	Wolf Alice	60	indie pop, indie rock, modern alternative rock, modern rock	Rock	Visions Of A Life	2017-09-29	4:35	60	79	0	83	35	3	37	-6 db	122	F Major	4	2021-03-05	1LcmCMl6wD3MPVIO6amUHe

Figure 3

For each track in the Spotify database, the service provides it with the following data:

- Song
  - The title of the track
- Artist
  - The artist of the track
- Popularity
  - Number of streams in a small timeframe (out of 100)
- Genres
  - Genres of the track
- Parent Genres
  - Main genres of the track
- Album
  - The album the track belongs to
- Album Date
  - The date the track was released

- Time
  - The length of the track
- Dance
  - How easy it is to dance to the track
- Energy
  - How energetic the track is
- Acoustic
  - How acoustic the track is
- Instrumental
  - How likely the track is to be an instrumental
- Speech
  - How much spoken word the track contains
- Live
  - How likely the track is a live recording
- Loud
  - How loud the song is (in dB)
- Tempo
  - The beats per minute
- Happy
  - How positive the track is (aka valence)
- Key
  - The key the track is mainly in
- Time Signature
  - How many beats per measure
- Added At
  - The earliest date the user added the track to the playlist
- Spotify Track Id
  - Spotify's unique Id for the track to access all the above information



Recommended Songs

#	Song	Artist	Popularity	Genres	Parent Genres	Album	Album Date	Time	Dance	Energy	Acoustic	Instrumental	Happy	Speech	Live	Loud	Tempo	Key	Time Signature	Added At	Spotify Track Id
1	Nude	Radiohead	66	alternative rock, art rock, melancholia, oxford indie, permanent wave, rock	Rock, Folk/ Acoustic	In Rainbows	2007-12-28	4:15	52	34	83	58	17	3	9	-10 db	128	E Major	3	2021-10-21	35YyxFpE0ZTOoqFx5bADW8
2	I Want You To Love Me	Fiona Apple	55	art pop, chamber pop, indie pop, lilith, permanent wave, piano rock, pop rock, singer-songwriter	Pop, Folk/ Acoustic, Rock	Fetch The Bolt Cutters	2020-04-17	3:57	53	50	56	4	71	8	9	-9 db	141	A Minor	3	2021-10-21	73SBAGI4fPFm4VkB3NjXq8
3	7-eleven	black midi	36	art pop, freak folk, modern alternative rock, noise rock, uk noise rock, uk post-punk revival	Pop, Folk/ Acoustic, Rock	7-eleven	2019-11-08	4:53	34	28	18	60	4	4	10	-15 db	111	D Major	4	2021-10-21	29e6BxPIL9QUMubTLmUbLD
4	Maud Gone	Car Seat Headrest	52	alternative rock, indie pop, indie rock, modern rock	Rock	Teens of Style	2015-10-30	5:58	53	36	3	0	21	3	10	-7 db	105	F Major	4	2021-10-21	53eMEHD5vu4kRB458TCuiM
5	After Hours	The Velvet Underground	59	alternative rock, art rock, classic rock, folk rock, melancholia, permanent wave, protopunk, psychedelic rock, rock	Rock, Folk/ Acoustic	The Velvet Underground (45th Anniversary)	1969	2:07	61	15	84	0	47	17	18	-21 db	121	G Minor	4	2021-10-21	6cA1usDL8nTHeSgFewQat3
6	Kyoto	Phoebe Bridgers	71	indie pop, indie rock, la indie, pop	Rock, Pop	Punisher	2020-06-18	3:04	60	75	5	30	49	4	9	-7 db	131	B Major	4	2021-10-21	4vjvx7Zxkb4AltGcZ0BBul

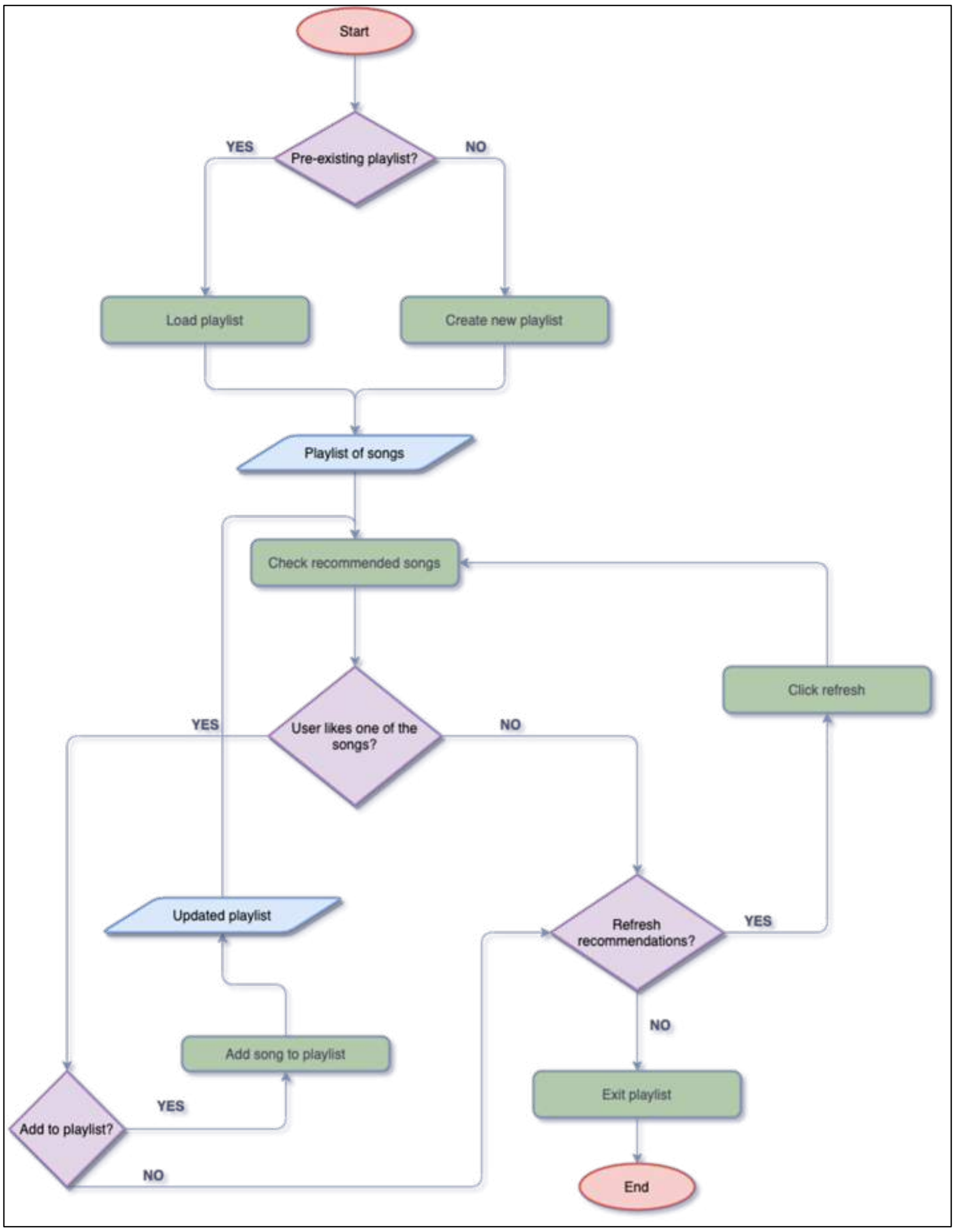
Figure 4

Analysing the data from the tracks within the playlist and the recommended songs, it became evident why those specific tracks were selected. Both songs 5 and 6 [figure 4] have the same artist as songs in the playlist. All songs that were recommended have the parent genre Rock, the most prevalent genre in the playlist. The audio metadata has the same variance as the playlist, and the average tempo of the recommended songs (122) is similar to the average tempo of the playlist (121). Overall, the songs recommended are very similar to the songs in the playlist, through genre but also actual audio data. From already knowing the client's listening tastes, I can assume that her listening history also factored into these decisions.

If the user clicks the refresh button under the songs, a new set of 6 songs will be recommended. Also, if the user leaves the page and then returns, the songs will automatically refresh, without giving the user the chance to listen to or save the songs. Through further testing, it was clear that the recommended songs can end up in a loop if

there is not enough data to recommend songs from, with the same songs being recommended every time the user refreshes. This also goes for covers and different versions of the same song.
The data is stored within the Spotify database, however, using Spotify's API you can easily export the data to a text file. This saves the song name and artist name as well as song information like tempo and factors like 'happiness'.

The following diagram models the process the user follows when using the recommendation system Spotify has for a user-generated playlist. This allows me to view the inefficient sections of the process that I will need to improve - for example, the user may be stuck in a loop where they don't like any of the recommended songs.



‘Discover Weekly’

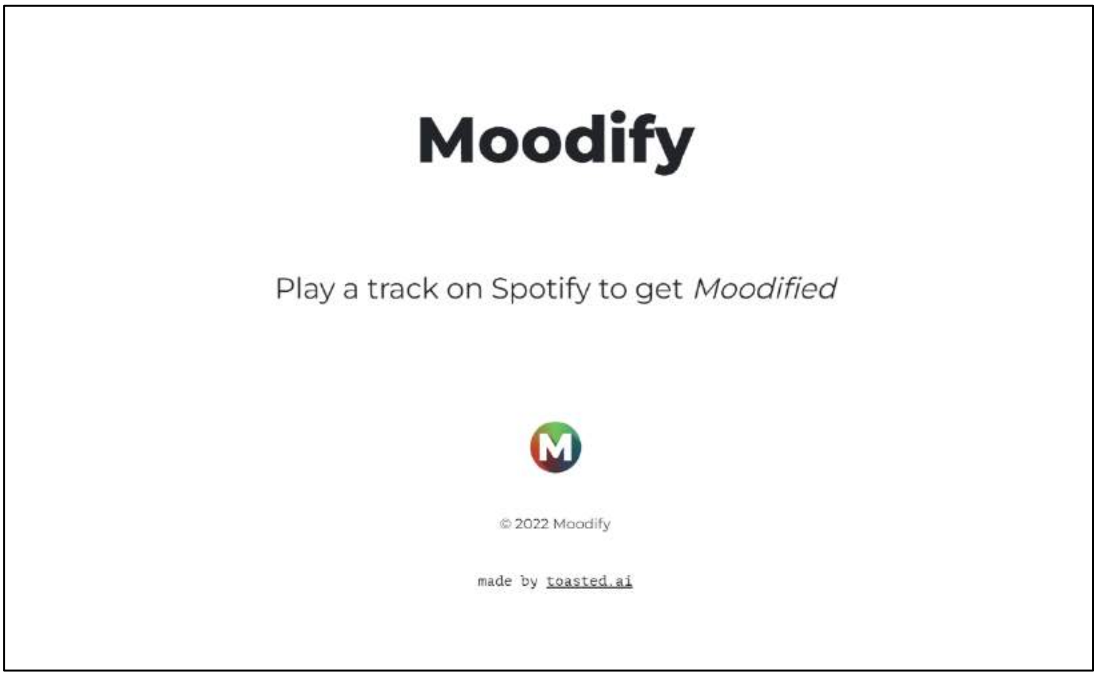
Every Monday morning, Spotify creates a playlist of 30 songs you have never listened to before, based on your listening habits. When it first launched in 2015, within 4 months songs from its playlists had been streamed 1.7 billion times[1]. Discover weekly is based on other people’s playlists and your ‘taste profile’, a mix of micro-genres like southern soul that personalise music recommendations even further. As they have access to users’ listening history, they can ensure the user has never listened to that song before, at least on Spotify. The playlist updates automatically once a week, giving users time to listen and save the recommendations.

Discover Weekly is a feature that is almost impossible to replicate, as it has access to such a large variety of listening and user data that is private. However, the use of micro-genres to tailor music more carefully is something I will take into consideration when developing this recommendation system.

Moodify

An external service that was mentioned by the client is the website Moodify[2], which uses Spotify’s API and the track the user is currently listening to, to recommend a set of 20 songs.

To use the service, a user must log in to their Spotify account and also be playing a track on Spotify. As the user has to be playing a track for the service to work as soon as the track end the recommendations are replaced. While the user could replay the song to get new recommendations, there is no guarantee the recommendations will be the same as before. This pressures the user into saving the recommendations as soon as possible, something my client has expressed she dislikes.



Moodify

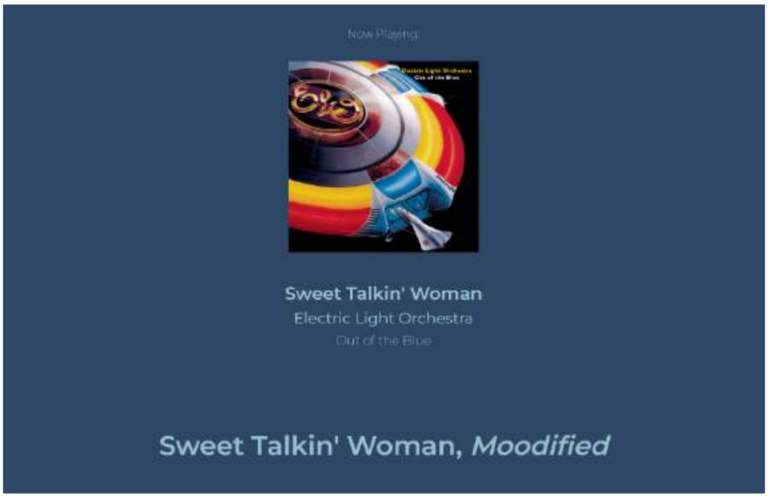
Play a track on Spotify to get *Moodified*

© 2022 Moodify

made by toasted.ai

[1] https://qz.com/571007/the-magic-that-makes-spotifys-discover-weekly-playlists-so-damn-good/

[2] Moodify website - https://moodify.toasted.ai/



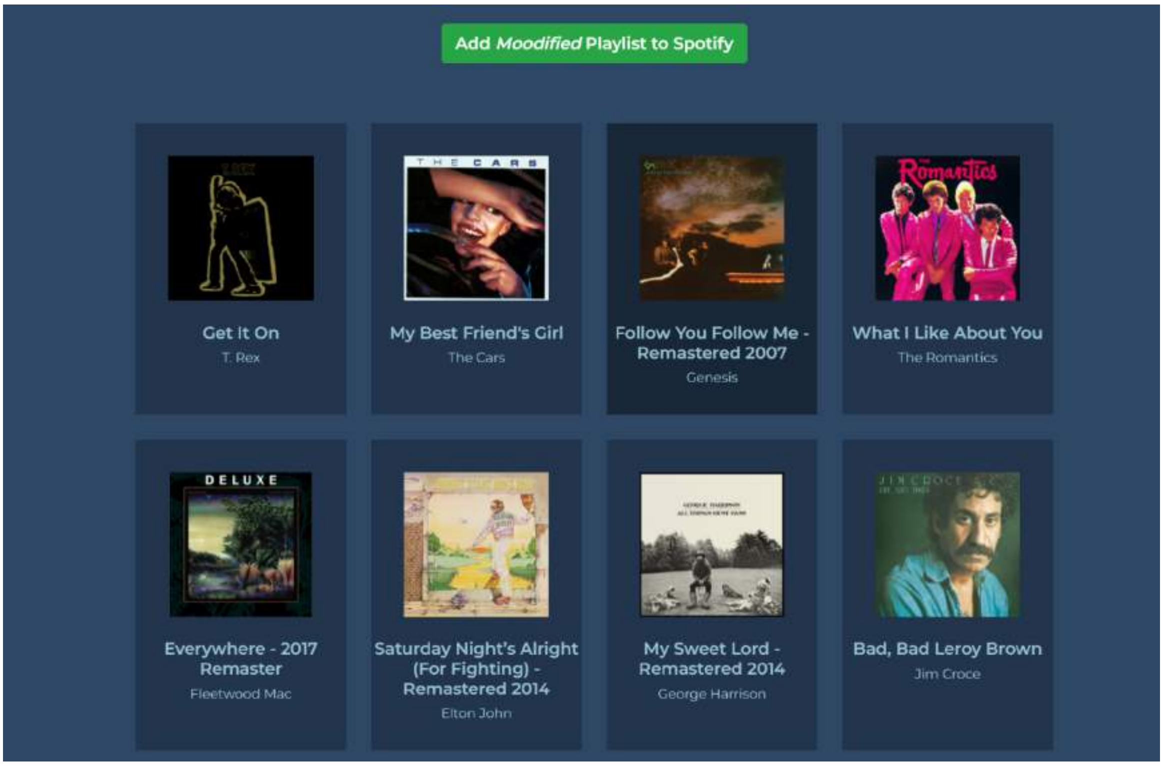
Now Playing:

Sweet Talkin' Woman
Electric Light Orchestra
Out of the Blue

Sweet Talkin' Woman, *Moodified*

Once the user has started playing a song on Spotify, the service recognises it and displays it on the tab. The design of the service is basic, and it is clear to the user what they must do.
The user then can scroll down and view a grid of 24 recommendations, and if you click on any of them it will redirect the user to the track on Spotify and play it for them.

The grid view with the album art makes it easy for the user to see. There is also the option to add the full playlist to the user's Spotify account. From this service, it is clear that the aesthetic of the recommendations is just as important as the actual recommendations. The display should be simply designed so the user does not get overwhelmed.



Add *Moodified* Playlist to Spotify

Get It On
T. Rex

My Best Friend's Girl
The Cars

Follow You Follow Me - Remastered 2007
Genesis

What I Like About You
The Romantics

Everywhere - 2017 Remaster
Fleetwood Mac

Saturday Night's Alright (For Fighting) - Remastered 2014
Elton John

My Sweet Lord - Remastered 2014
George Harrison

Bad, Bad Leroy Brown
Jim Croce

As well, the personal aspect of this service seems to be appealing. The user must start playing a song to get recommendations, and so I will make sure to include some sort of personal aspect within the system, whether that be a specific song, or a playlist made by the user.

Through this research I have learnt the type of data I will need to have access to, to achieve the objectives my client has expressed. I will be using Spotify's API to import data about songs and users into my models. I chose Spotify because their API is easily accessible and easy to integrate into my program, while also being the service most used by my clients, ensuring the listening data will be accurate. Spotify's song database includes specific information on the billions of songs it has on its platform, including tempo, release year, mood, genre, length, etc.

To ensure I achieve a more effective system that is tailored to my client, I have decided that the recommendation factors should be personalised so that the users can experiment with the recommendation.

Track data like the song name and album art should be visible when displaying the recommendations to the user. When saving the recommendations less information needs to be shared, and just the basic information of each track should be saved. However, the data can be fully customised by the user when it comes to recommendations, but data such as the song name and the artist's name will always be used.

Proposed Improvements and their Benefits

The client feels that the music Spotify recommends to them is not as accurate as it could be. As mentioned before, problems within the system include a lack of quality and quantity in terms of the recommendations. As well they are easily lost, and they do not feel as personal. Therefore, the users would like the new system that creates a list of recommendations based on a user-provided playlist. They would like the option to modify how the recommendations are made and to receive new recommendations if they are not happy with a specific one.
I am going to create a recommendation software that considers user-defined factors as well as just a basic recommendation system. It is imperative to create a more straightforward experience, so I will make the UI as simple as possible. The client would prefer the system to be web-based, or application-based. While it will only be designed for the specific client, they would like the ability to create an account and store the recommendations created by the system.

The main features I will add to the system to improve the user experience include:

- Having accounts for each user of the system to store recommendations
- Allowing songs to be discarded by the user if they don't like it
- Allowing the user to save recommendations access them at a later date
- Allowing recommendations to be exported to a text file for easy importation to Spotify
- The ability to customise the recommendation factors

The benefits are as follows:
Qualitative Benefits:

- A more user-friendly interface
- More personalisation in terms of recommendations
- More choice of recommendations

- Recommendations can be stored separately, allowing for exportation whenever the user pleases

Quantitative Benefits:

- A potentially unlimited number of recommendations can be stored in the system
- A potentially unlimited number of users can be registered and use the system
- More recommendations for the user (instead of just 5)

Prospective Users

Daisy Thomas will be the primary user of the system, and she will have access to all the permissions of an administrator. She will be able to decide the properties that the system will use to recommend songs and will be able to modify and delete any user account. Alex Smith will be a user of the system, but she will only have basic permissions. Like Daisy, she will be able to create a user account and modify and delete her own and use the system to get a list of recommended songs from a playlist she has imported.
The users that will be using this system are proficient in technology and therefore a more complex interface can be designed. However, the system's functionality must ensure that the user can easily and quickly navigate the tool so that they can receive recommendations as quickly and as efficiently as possible.

Acceptable Limitations

Currently the system is designed for the two specified users, but there is the possibility of expanding the system to facilitate more users.

The limitations of the system are as follows:

- Skills and knowledge
    - My knowledge in artificial intelligence is not secure enough to tackle more complex techniques, therefore I must create a solution that can be solved with the resources available to me.

- Access to data (both users and general data)
    - The ability to access data about listening habits and general information about songs is limited as I do not have full access to a music streaming service and its information about users. However, I will be using Spotify's Public API to fetch music data as well as access listening data from users who allow access.

- The tool is subject to users who have a Spotify account
    - By using Spotify's API to access this data, limits the users who can use the new system. However, as my clients use Spotify regularly it allows me to have accurate listening data and overall create an improved service to recommend more personalised music suggestions.
    - If the user has a Spotify account but doesn't use it to listen to music, the listening data will not exist, or be inaccurate, meaning the system would not be right for them.

    - While this system will work for Spotify users with a free account, the accuracy will be limited due to Spotify limiting the functionality for users without a subscription. However, this is not a limitation due to the system but rather due to Spotify and the lack of data that will be available.

- Storage of data
    - The storage of user data is very important.
    - Track data used in the recommendations will be locally stored in a database without encryption as there is no personal information, however, the online Spotify database will have to be accessed every time there are new recommendations.
    - The security and integrity of the data are important considerations within my system, and there is further detail in the design section of this report.

- Time constraints
    - The system needs to be completed by Easter, to meet the client's deadline.

- Hardware and Software limits
    - There is a limited amount of hardware and data storage for my project, so I need to ensure the system will be accessible while also functional with a limited amount of data.
    - As I am using Spotify's API, the user will have to have an internet connection to use the program.
    - The clients have limited access to software, so the program has to be desktop compatible.

Modelling

The diagram below identifies both common and format-specific data currently used by the system for playlists. This will allow me to identify the data and data types that will need to be included in the new system.

Data Volumes

Although only one user will be using the system at a time, the system will store thousands of songs' data. Previous recommendations and user profiles will need to be recorded and stored within the system, so a computer that can execute regular programs will be required to run the program.

I aim for my program to store data in a local database during runtime, where it will contain user information as well as song and playlist information. This data volume will vary in size, depending on the number of recommendations made. As well, there will be the option for users to export the playlist recommendations to a text file stored locally on their computer. Each set of recommendations will only have 15 songs, so this data volume shouldn't vary in size drastically.

Data Dictionary

To recommend music, I will be using the Spotify database to access song information. This will be imported into my program with the following data:

Field Name	Field Purpose	Field Type	Field Size	Example data	Validation
#	Index	VARCHAR	4	20	Represents the row number of song
Song	Song name	VARCHAR	100	Like Real People Do	Not blank
Artist	Artist of song	VARCHAR	100	Hozier	Not blank
Popularity	Popularity of song calculated by Spotify from 1-100	VARCHAR	2	66	Within the range 0-100
Genres	All genres song is labelled under	TEXT	255	irish singer-songwriter, pop	Not blank
Parent Genres	Main genre(s) of the song	TEXT	100	Folk/Acoustic, Pop	
Album	Album of song	VARCHAR	100	Hozier	Not blank
Album Date	Date album was released	DATETIME	YYYY-MM-DD (10)	2014-07-20	Date/Time format with minimum of Year only
Time	Length of song	TIME	MM:SS (5)	3:18	Time format (in minutes and seconds)
Dance	Danceability of song (0-100) calculated by Spotify	VARCHAR	2	57	Within the range 0-100
Energy	Energy of song (0-100) calculated by Spotify	VARCHAR	2	18	Within the range 0-100
Acoustic	How acoustic song is (0-100) calculated by Spotify	VARCHAR	2	91	Within the range 0-100

Instrumental	How instrumental song is (0-100) calculated by Spotify	VARCHAR	2	40	Within the range 0-100
Happy	How happy song is (0-100) calculated by Spotify	VARCHAR	2	13	Within the range 0-100
Speech	How much spoken word song is (0-100) calculated by Spotify	VARCHAR	2	3	Within the range 0-100
Live	How likely the song is to be live (0-100) calculated by Spotify	VARCHAR	2	11	Within the range 0-100
Loud	How loud song is	VARCHAR	6	-14 db	
Tempo	Tempo of song	VARCHAR	3	70	More than 0, less than 400
Key	Main key of song	VARCHAR	20	G major	
Time Signature	Time signature of song	CHAR	1	3	Either 2, 3 or 4
Added At	When song was added to playlist	DATETIME	YYYY-MM-DD (10)	2021-03-12	Date/Time format
Spotify Track ID	Unique Spotify Track ID of song	CHAR	22	4LGJ2pLDvTRnul3EcZoYkX	

Furthermore, there will be certain variables that will be imperative for the function of the program:

Field Name	Field Type	Start Value	Required?	Description
Username	String	NULL	Y	User defined username for their account
Password	String	NULL	Y	User set password for their account
Playlist URL	String	NULL	Y	The URL of the playlist the user would like to
Deciding Factors	Checkboxes	All checked	N	Metadata that the user may choose to base the recommendations off

Data Sources and Destinations

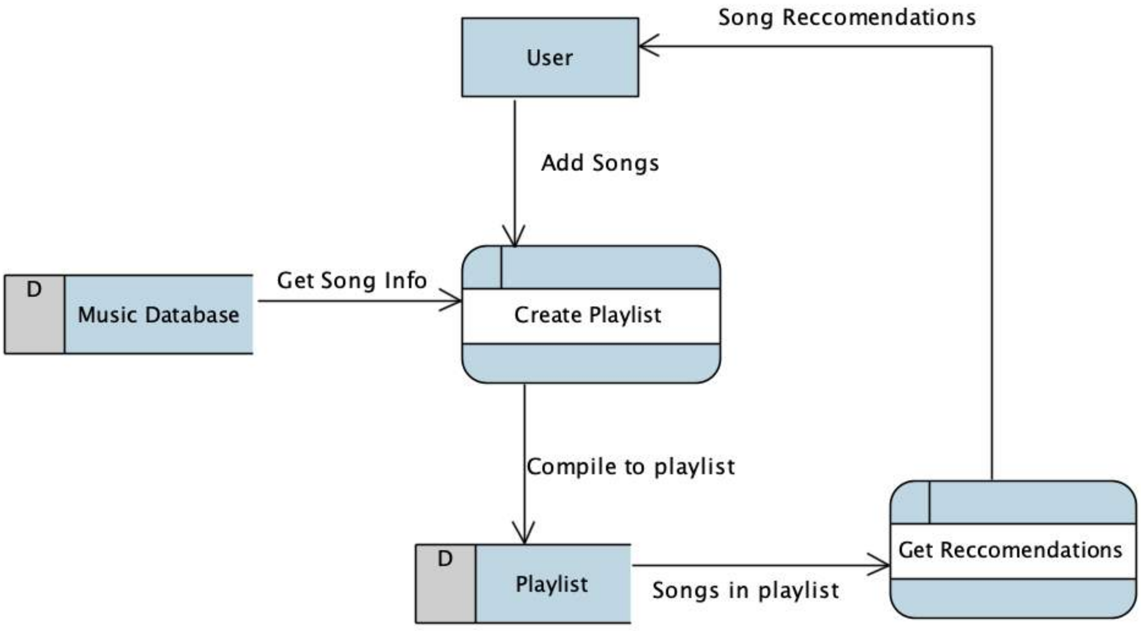
The main data source for my program is the Spotify music database, from which I will extract and search for track data that will benefit the user. The extracted data will be stored locally in the program for modelling and then deleted to save memory once the recommendations have been formatted to a text file. The text file will contain a list of song names, their artists, and their albums. It will be saved locally within the system, behind a protected user account. Storing the data this way will allow the user to access only their data and save it to review later. There will also be the option for the user to convert the recommendation into a Spotify playlist and save it in their Spotify account.

In the Existing System		
What is it?	**Source**	**Destination**
User-Created Playlist	Created by user	Stored in Spotify database
Recommendations	Spotify database	Shown to user within user playlist
Song Information	Spotify database	Can be formatted to .csv files
'Discover Weekly' Playlist	Created by Spotify	Given to user in the 'Made for You' section of Spotify

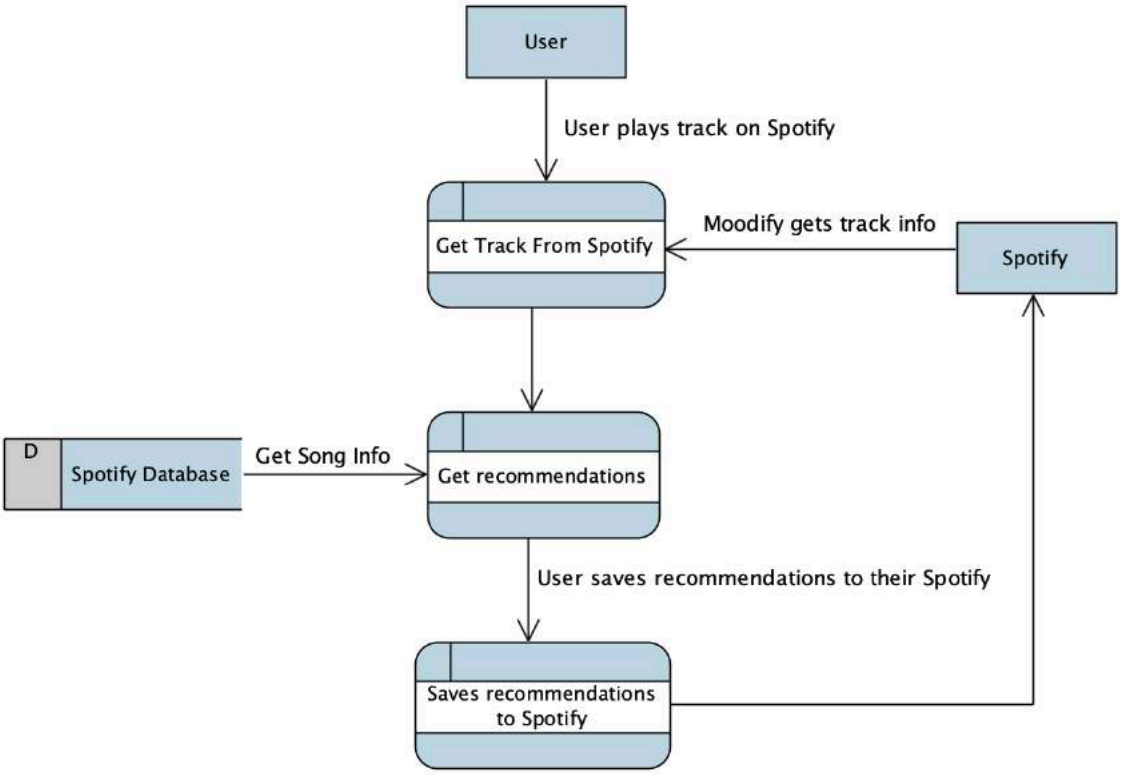
In the Proposed System		
What is it?	**Source**	**Destination**
User-Created playlist	Created by user	Stored in system database to recommend songs
.CSV file of recommendations	Created by model using Spotify database	Converted to a Spotify playlist that the user can save, and saved locally within the system so the user can revisit
List of previous recommendations	System	User can save as text file
Playlist recommendation factors	Chosen by the user but implemented by the system	Stored in database
Song Information	Spotify database	Formatted to objects for use within the system

Data Flow diagram

Spotify data flow diagram:

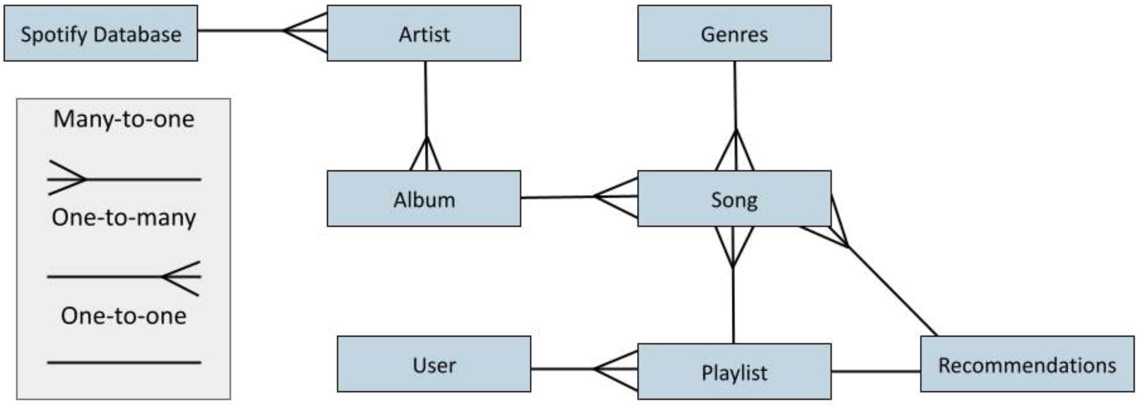


Moodify data flow diagram:



The data flow diagrams along with the data sources and destinations table highlight the processes and flow of data in the current system between the user and the Spotify music database. This further reinforces the observations I made with the current systems and allow me to see which parts of the system use which types of data and to track its path.

Entity Relationship Diagram



The diagram above identifies the relationships between the entities in the system. A song has one artist, one album, and many genres. It can belong to many playlists, and a playlist contains many songs. A list of recommendations has many songs but is related to only one playlist, and a user has many playlists. To accurately recommend songs to users, it will be helpful to do an E-R diagram for the metadata within a song to understand the most effective way to make recommendations.

Objectives for the Proposed System

From the research I have conducted, I have learnt about the existing systems' functions and how they are used. Using this, I have compiled a list of features that work well, as well as any that are unnecessary and features that I need to add to my system to ensure functionality.

1. The system will calculate and display a list of song recommendations based on a user playlist.
2. The system should involve minimum text entry, with the only information being the account details and Spotify playlist URL, to save time and minimise errors.
3. The system should be able to display 15 songs in random order.
4. New users must register to use the system:
    a. The new user must complete a registration form in which they enter their name, a username, and a password.
    b. The username must be unique (on the system).
    c. Passwords must be at least 8 characters long and contain a mix of character types.
    d. Only one user registered on the system must be labelled as 'administrator' and get admin permissions.
5. The user must be able to share a Spotify playlist URL and receive recommendations based on the songs within it.
6. The system should have the option to discard songs that the user doesn't like and recommend a new song instead.
7. The system must be able to store all the relevant details about every song recommended with the following details being essential for each song:
    a. Song Name
    b. Artist Name
    c. Spotify Unique ID
8. The system must store the recommendations in a user profile for the user to access at a later date.
9. The option to save the recommendations in the format of a text file must be available to the user, so it is easily accessible.
10. There should be the option for an administrator to delete/modify user accounts and their details.
11. There should be the option for an administrator to modify the deciding factors of the recommendation system.
12. The recommendations should include a variety of music types – based on genre, artist, album, happiness, etc.
13. The recommendations must not include more than 3 songs per artist.
14. Users must be able to delete their account from the system.

Potential Solutions and Justification

From the research I have performed, as well as considerations for the client and existing systems. I believe that there are 3 potential solutions to my client's problem.

Potential Solution	Positives	Negatives
A manual system would give users complete flexibility with their song choices as they could simply type in the song name and artist of a specified number of songs which the system will then check exists within the track database. This would eliminate the need of having a playlist premade and users would have complete control over the songs they are using for recommendations. The recommendations could then be plainly listed, with the option of saving the list as a text file.	In terms of resources and timeframe, this solution would be ideal for development. Allows for full autonomy on which songs will be used for recommendations. Removes the need for a Spotify account and a pre-made playlist.	It removes the option to recommend songs based on genres and musical data such as tempo. It would be very time-consuming for the user to have to write every song they want to be used in the recommendation, and it removes the functionality the user already has with Spotify playlists and recommendations. The text file can only be used by manually inputting each song name into Spotify and adding it to a playlist. Doesn't meet the user objectives.
A fully computerised system allows the user to input a Spotify playlist link and receive 15 song recommendations relating to the playlist. The user will have to option to discard a song and get a new recommendation, and also save the recommendations when they like all of them. The administrator would be able to control the factors that determine the recommendations, allowing for a more personalised experience. The recommendations will be listed with their album cover, with the option of	Utilises the metadata given by the Spotify database to enhance recommendations. Allows for the users to personalise recommendations and save them.	Ignores user data such as liked songs, artists they are following, and track play counts.

saving them as a text file, and possibly exporting it to Spotify as a new playlist.		
A system involving an artificial intelligence model, which uses user likes and dislikes to personalise the recommendation experience, and ultimately learning which type of songs favour well. The user will have the option to like and dislike a group of 50 songs, with each like and dislike changing the next songs that will be recommended.	Combines both song metadata and user listening data to create more efficient recommendations. Would quickly recommend tracks, can use previous knowledge. Using an AI model would allow the system to learn which recommendations are useful and which the user doesn't like, overall creating a more efficient system.	Too complex for my knowledge. Would require a lot of users listening data that I don't have access to.

From these suggested solutions, it is clear that the second solution is the practical choice to solve my client's problem, a fully computerised system that could potentially meet all of the objectives and remove all of the time-consuming manual processes from the system. I will be using C# and WPF framework to create the program.

Design

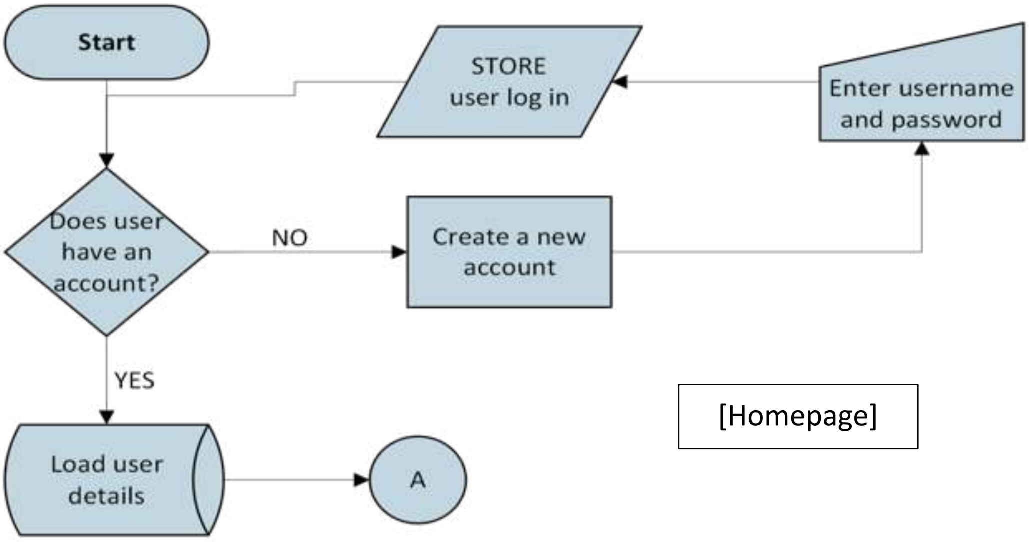
IPSO Chart

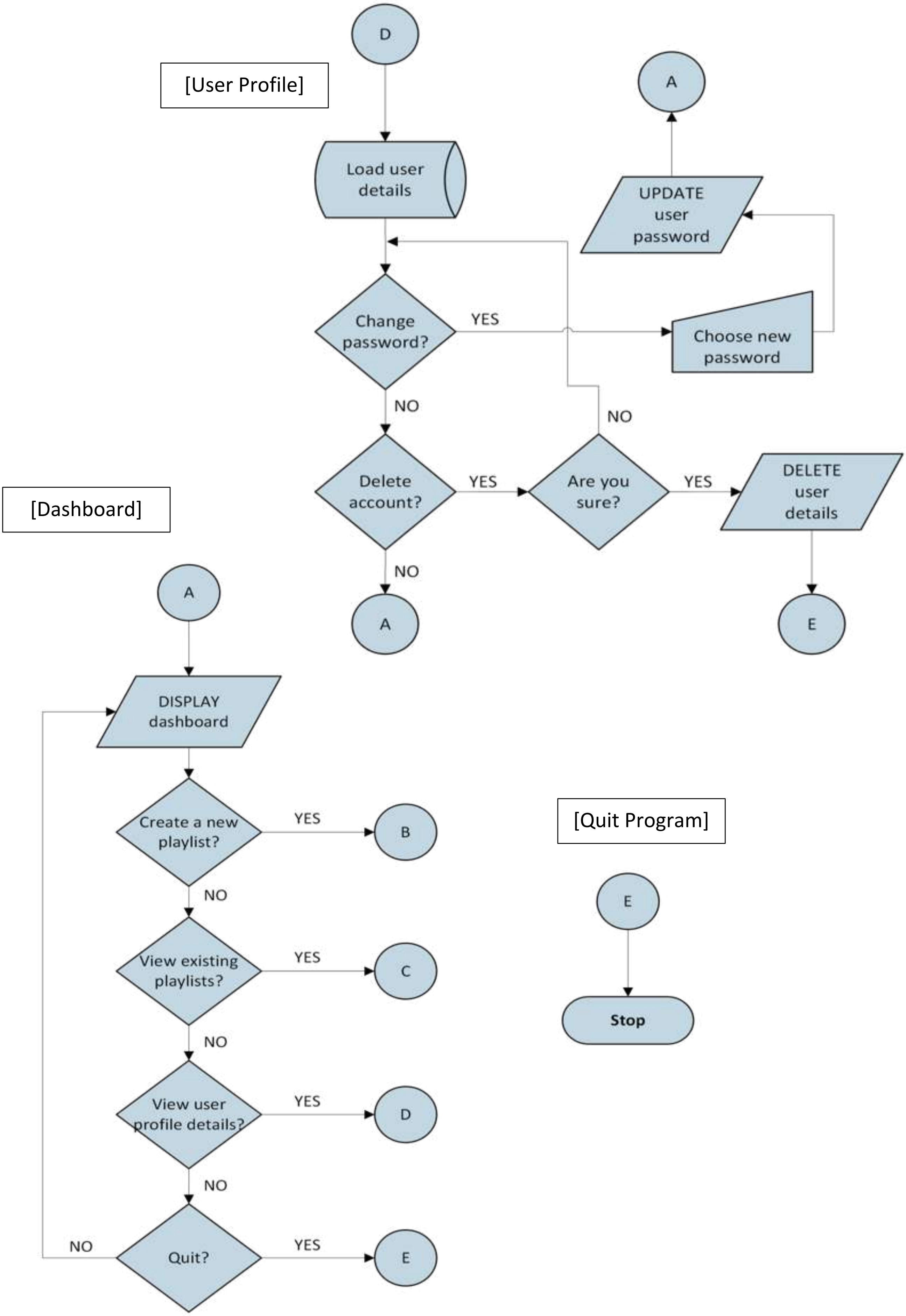
This following chart outlines what happens to the data in the new system at a most basic level, in terms of input/output, processing, and storage:

INPUT • Username • Password • User URL • User likes	PROCESS • Validate input • Fetch songs from URL • Generate recommendations • Convert to text file
STORAGE • Account details (database) • Previous recommendations that link to a certain account	OUTPUT • Message stating registration successful • Display of recommendations • Text file of recommendations

System Flowchart

The following flowchart is an updated and improved version of the current system flowchart, showcasing how the new system should function. It seems more complex than the current system, however, it outlines how the recommendation will work as it will be partly user-defined, whereas that option is not available in the current system. While there will be more processes, the overall results should satisfy the user more. The connectors signify the different windows forms in the system.





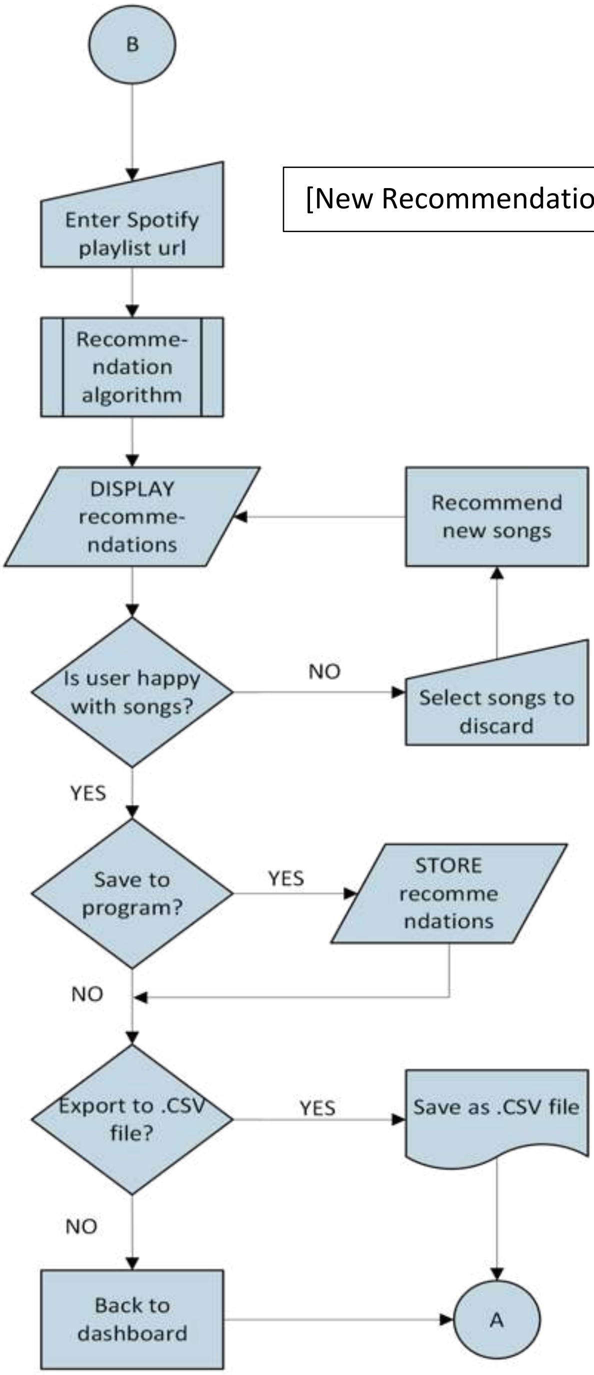
[User Profile]

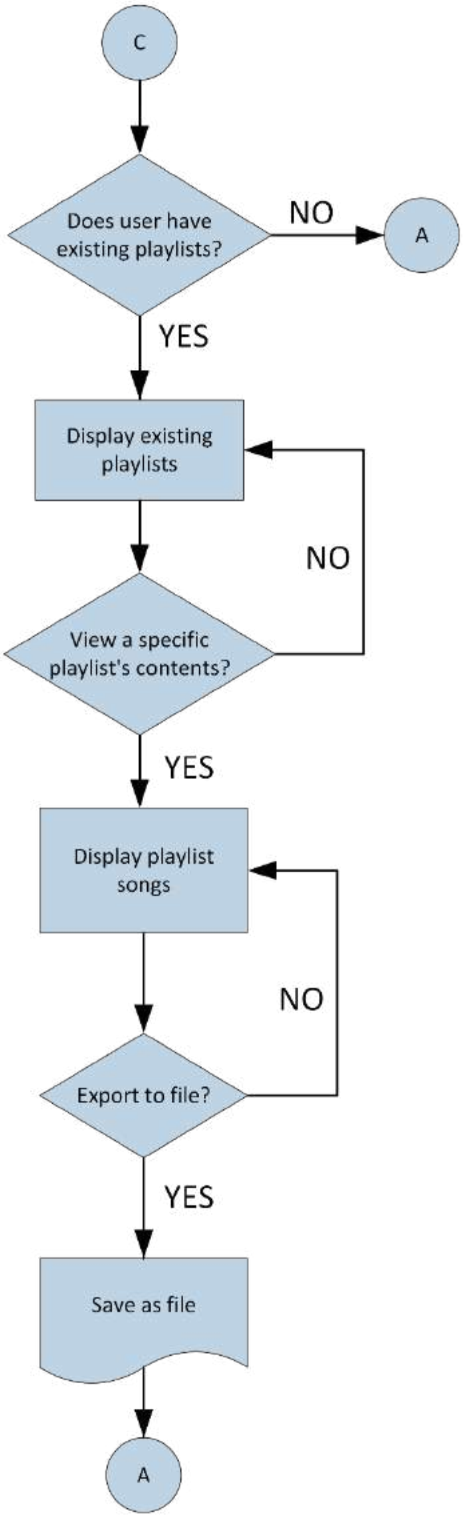
[Dashboard]

[Quit Program]

[Previous Recommendations]

[New Recommendations]

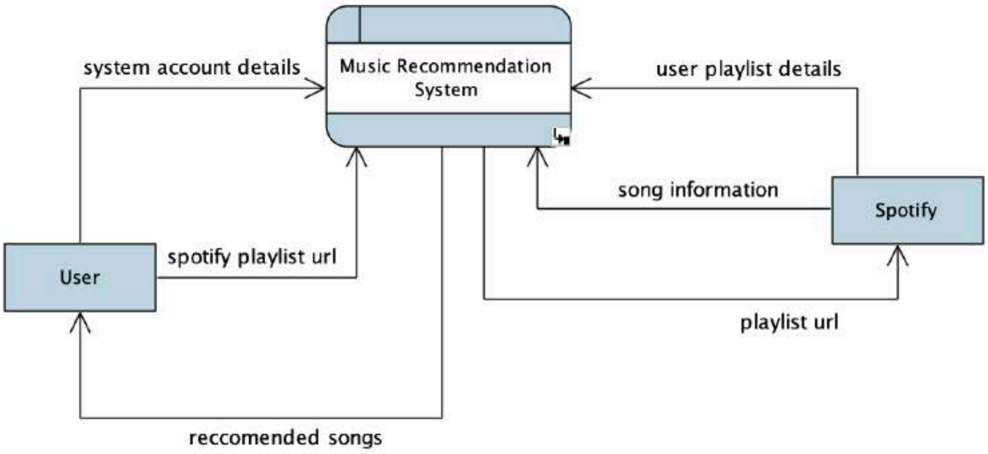



C
Does user have existing playlists?
NO
A
YES
Display existing playlists
NO
View a specific playlist's contents?
YES
Display playlist songs
NO
Export to file?
YES
Save as file
A

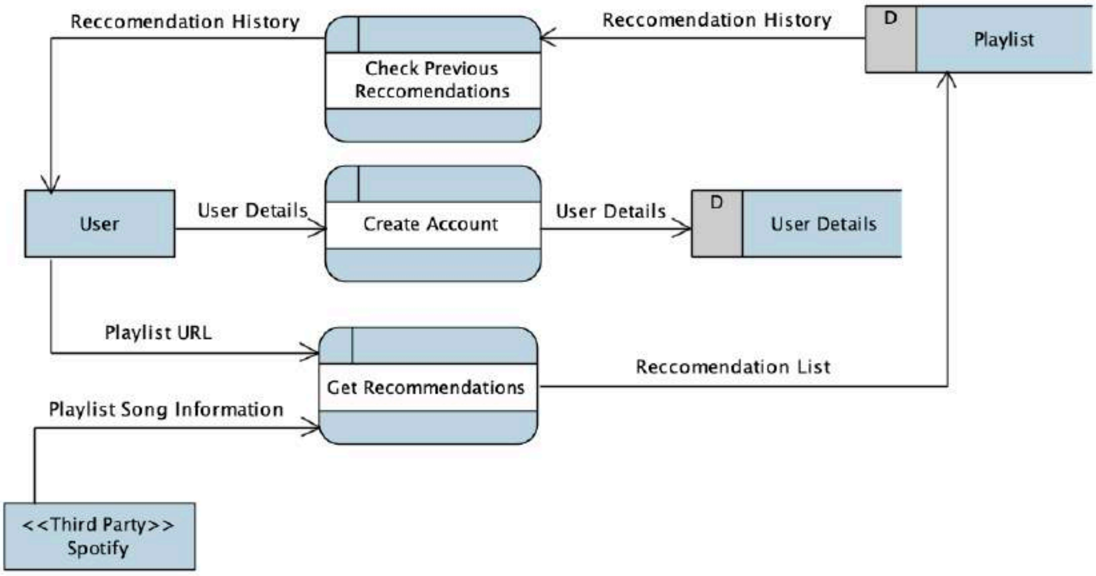
Data Flow Diagrams

I created two data flow diagrams of the new system to understand how I need the system to function, and to visualise the data processing.

Context Data Flow Diagram



Level 1 Data Flow Diagram



The data flow in the current and proposed system differs mainly after the recommendation occurs. While Spotify will refresh the recommendations and the user only has the option to add the songs to the current playlist, the new system will save the recommendations in the format of a basic text file where the user is then able to use the information to add the songs to Spotify. The interactions and flow of data between the Spotify database and the system remain the same.

Stepwise Refinement

While showing the same processes as the system flow chart, this list defines the processes in more detail. This decomposition of each task allows me to see the smaller processes involved and how data will flow and be stored. I have ignored the login system as the flowchart explains it in enough detail.

1. Load user data and dashboard
2. If SELECTING NEW RECOMMENDATIONS
   - 2.1. Input Spotify playlist URL
   - 2.2. Get playlist data and create a new table based on filters checked (admin filter)
   - 2.3. Create a new table of recommendations
   - 2.4. Display recommendations to user
     - 2.4.1. If user does not like a recommendation, then delete that song and recommend a new one
   - 2.5. Save to program
     - 2.5.1. Save recommendations to user profile ID
     - 2.5.2. Export to user storage
3. If SELECTING EXISTING RECOMMENDATIONS
   - 3.1. Check there are existing recommendations
   - 3.2. Show user list of all previous recommendations
     - 3.2.1. Click specific group of recommendations
       - 3.2.1.1. Show each song from recommendations
   - 3.3. Save to text file
     - 3.3.1. Export to user storage
4. If SELECTING USER PROFILE
   - 4.1. Show user details
     - 4.1.1. Change password
       - 4.1.1.1. Enter new password
       - 4.1.1.2. Confirm password
       - 4.1.1.3. Change password in system
     - 4.1.2. Delete account
       - 4.1.2.1. Confirm deletion of account
       - 4.1.2.2. Quit program
     - 4.1.3. Edit Factors
       - 4.1.3.1. Check user is an admin
       - 4.1.3.2. Show list of factors that are selected
       - 4.1.3.3. Check/uncheck factors
         - 4.1.3.3.1. Update recommendation system
5. If SELECTING QUIT
   - 5.1. Quit program

Data Normalisation and Design

The user recommendation playlists will be stored in a database, which will need to be normalised to avoid duplication of data.

A **Bold** column header indicates a primary key (or in the case where more than one attribute in the same table is bold, composite).
An *Italics* column header indicates a foreign key, meaning that the attribute is the primary or partial-composite key in a different table.

Before any attempt at normalisation the following is the original table of a single set of user playlist recommendations:

Username	**Song Id**	Name	Artist	Album	Release Date	Duration in Ms	Popularity

The shaded blue signifies the column headers. Each set of recommendations will have 15 songs, hence where there are 15 rows within it. The table contains non-atomic data, attributes with repeated entries. In this table, the Song Id and relevant information about such song and the User Id can be repeated throughout the table. This not only wastes space, but it makes storing and accessing relevant data inefficient.

These are the tables in first normal form as there are no repeating groups.

UsersId	**PlaylistsId**	*SongsId*

SongsId	Name	Artist	Album	Release Date	Duration in Ms	Popularity

By eliminating redundant data and making a composite table combing the PlaylistsId and SongsId, the tables are in 2NF and hence 3NF:

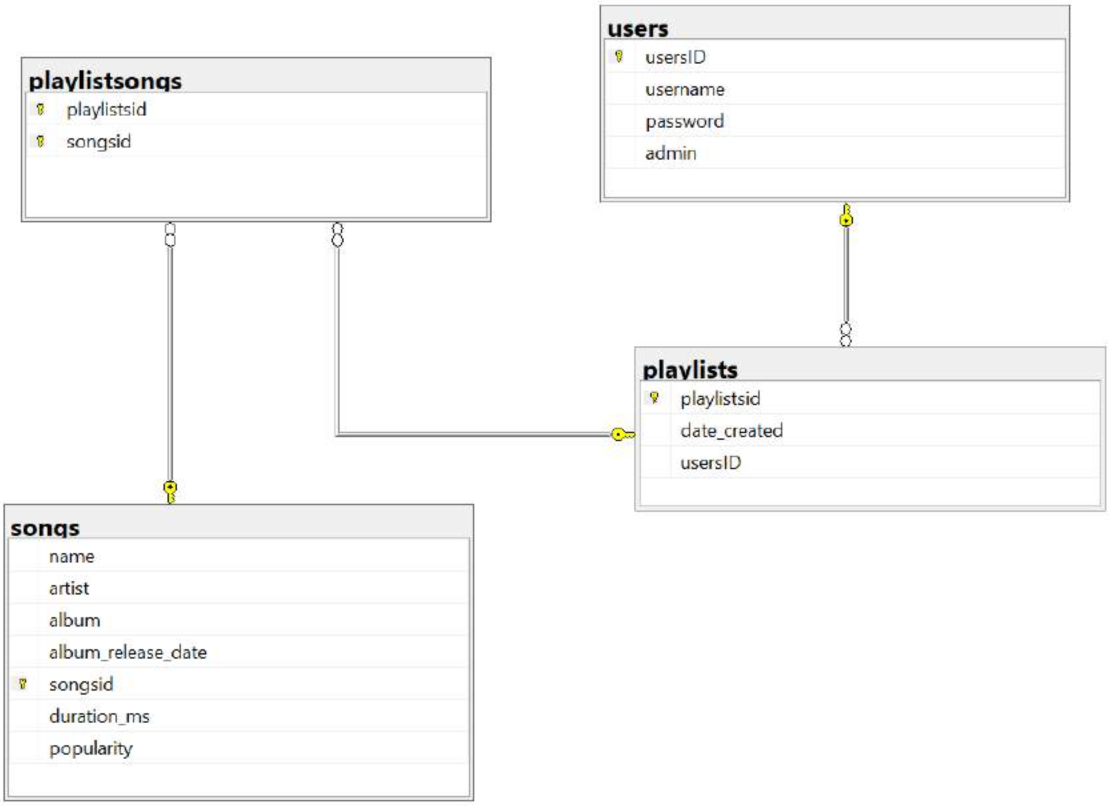
PlaylistsId	Date_Created	*UsersId*

PlaylistsId	***SongsId***

SongsId	Name	Artist	Album	Release Date	Duration in Ms	Popularity

Table Relationship Diagrams

The following diagram highlights the relationships between the normalised tables previously mentioned. It is clear the data has been normalised as there are no many-to-many relationships between the tables.



Data Requirements

I have distinguished common data with the current systems that should be included in terms of user inputs and what's required to run.

Data inputs required in the system:

Data Item	Data Type	Validation/Restrictions
Username	Text	- Required - Must be unique (on system)
Password	Text	- Required - Must be at least 8 characters - Must contain at least one each of the following: uppercase letter, lowercase letter, number, and special character, and no white spaces - Verified by double entry
Administrator	Boolean	- Required - Only one per system
Playlist URL	Text	- Required - Verified using Spotify API

The user and track information will be stored in a database so it can be accessed easily and at a later date. The database is made up of several tables which are linked relationally. It is clear they have been normalised as there are no many-to-many relationships between them, as displayed above. Below is the relational database that will be used in the music recommendation system:

Users					
Field Name	**Description**	**Data Type**	**Data Format**	**Character Length**	**Required?**
UsersID	Auto-incremented id by database	Int	*"4"*	3	Y
Username	Unique username set by user	Char	*"Admin"*	20	Y
Password	Password for account set by user	Char	*"Pas5word!"*	16	Y
Admin	Only one admin per system	Bit	*"1"*	1	Y

Playlists

Field Name	Description	Data Type	Data Format	Character Length	Required?
Playlistsid	Auto-increment index created by database	Int	*“1222”*	4	Y
Date_created	Date playlist was created by user	Varchar	*“16/04/2022”*	50	Y
usersID	Foreign key relating to users table	Int	*“2”*	3	Y

Playlistsongs

Field Name	Description	Data Type	Data Format	Character Length	Required?
Playlistsid	Foreign key relating to playlists table	Int	*“1221”*	4	Y
Songsid	Foreign key relating to songs table	char	*76TpWFiK5YCgw1hy26DWZp*	50	Y

Songs

Field Name	Description	Data Type	Data Format	Character Length	Required?
Name	Name of the song	nchar	*“Melody of Love”*	90	Y
Artist Name	Artist of the song	nchar	*“Hot Chip”*	90	Y
Album Name	Album that the song is from	nchar	*“A Bath Full of Ecstasy”*	90	Y
Album Release Date	The date the album was released	date	*2019-06-21*	YYYY-MM-dd	N
Track Id	Unique Spotify track Id	nchar	*48q7Pc3Zm2nPPVJPsfG30B*	50	Y

Duration in Ms	Length of song in milliseconds	float	*258601*	10	N
Popularity	How popular the song is (0-100)	int	*44*	3	N

Factors is another table within the database, and it does not have any relationships with other tables. Its purpose is to store the Boolean value of each recommendation factor for the system, which can only be changed by the admin of the system. By having this table, I can store the previous choice that the admin has made, hence the factors will not reset every time the program is run.

Factors					
Field Name	**Description**	**Data Type**	**Data Format**	**Character Length**	**Required?**
[Index]	Auto-increment index by database	B	*“4”*	1	Y
Factor	7 Factors provided by the Spotify API that can tailor the recommendation process	Char	*“Instrumentalness””*	30	Y
State	Boolean value as to whether the recommendation includes the factor	Bit	*1*	1	Y

When fetching the track data from the Spotify API, the way in which I store the data to then manipulate is very important. As the data is a group of objects, I decided that formatting them as collections would be the most efficient way. Examples of the collection types are shown under Models in the Technical Solution section of the report.

Security and Integrity of Data

The only personal data stored within the system is the user’s password, but as the user profiles do not contain any sensitive data and the system is only being installed locally on the client’s computer, encryption isn’t required.

To protect the integrity of the stored data, data entry will be limited and protected by strict validation rules. By limiting the free text input from the user, typographic errors will be minimised, and the system will be able to run more efficiently. There will be exception handling put in place so that no incorrect data will be stored within the database, and the duplication of data will be prevented as well. As mentioned previously, the username and password will have validation rules to ensure unique usernames and to encourage strong passwords.

It is important that the system has the functionality to delete data so that it complies with GDPR and ensures referential integrity. A user will have the option to delete their user profile, which includes their username and password and any recommendations that are linked to their profile. Users will have access to their data only once logged in. Modifying the factors used within the recommendation process will only be accessible by the admin, and so there will be a firewall blocking any other user from accessing this function within the program.

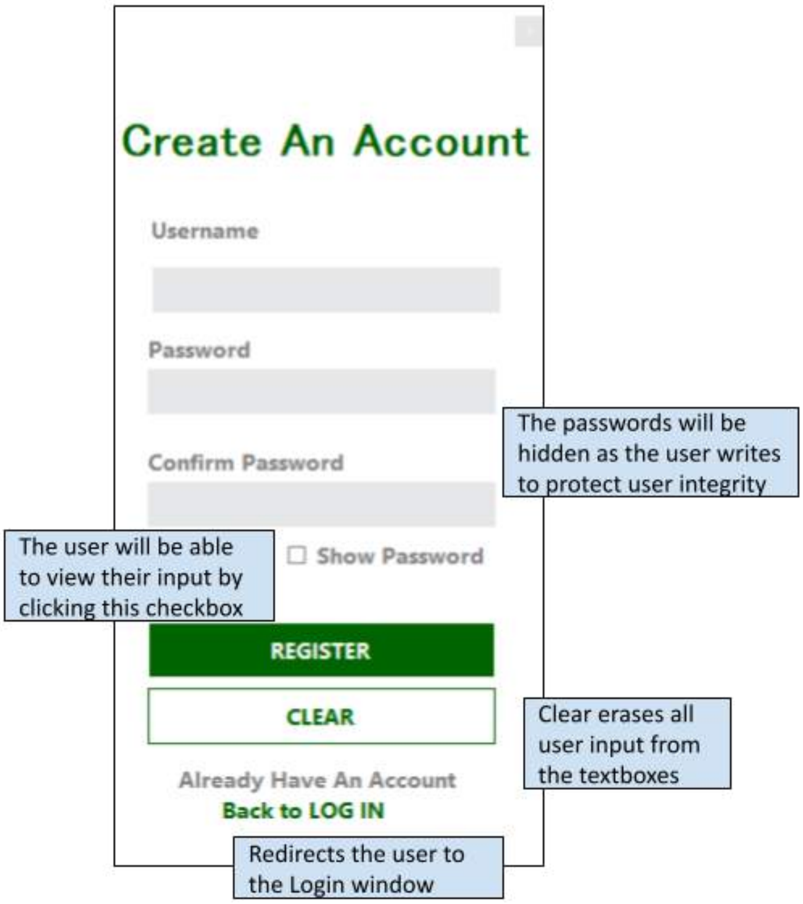
User Interface Design

The following drawings and diagrams are the first outlines of how the system UI should be designed. Whilst they most likely won't be the final designs, they are a starting point for the overall system design.

Visual Studio Form Design

Using an empty version of C# for windows in visual studio I created an initial design of the first 3 windows within the system, to better understand how they would be linked.

The design for the registration page includes 3 textboxes and 2 buttons. When the user types in the password it will be replaced with a sensitive character, and then the user will be able to check that the password matches before they click to register. If the user already has an account, then the user can click on the 'Back to LOG IN' label to go to the login form. It is important to my client that the interface is simple, and I have tried to reflect that in my design. The account system is going to be a very basic user authentication system, as there is a lack of personal data and only a few users will be using the system. When the system adds a new user to the database, the admin bit value will be set to 0 as there is only one admin per system, and that will have been set up before general user usage.

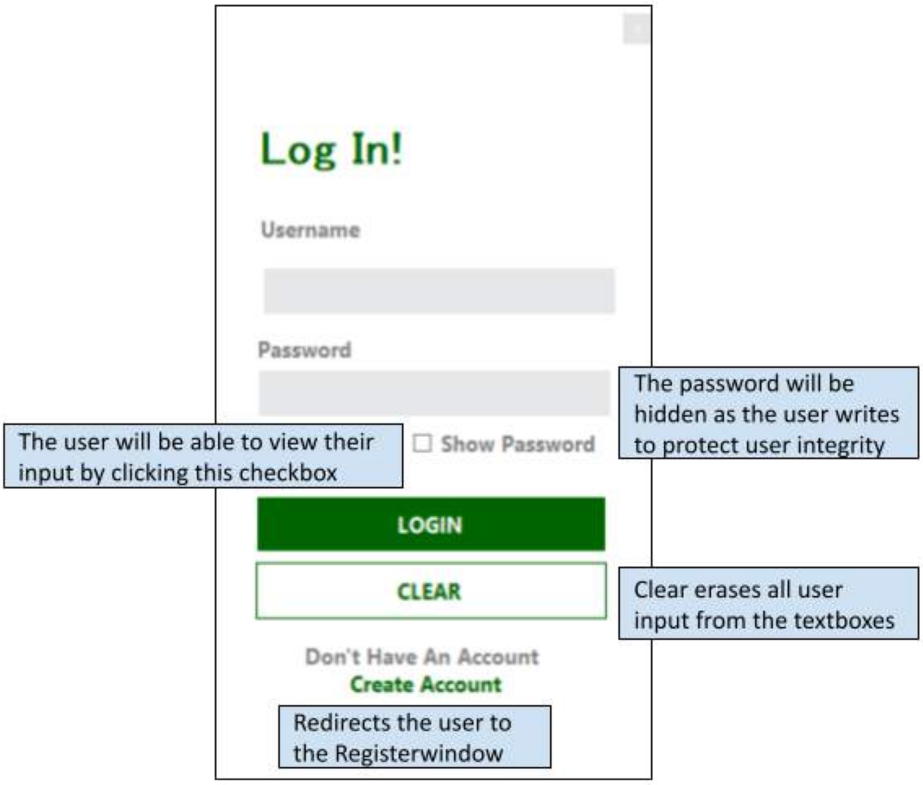


The passwords will be hidden as the user writes to protect user integrity

The user will be able to view their input by clicking this checkbox

Clear erases all user input from the textboxes

Redirects the user to the Login window



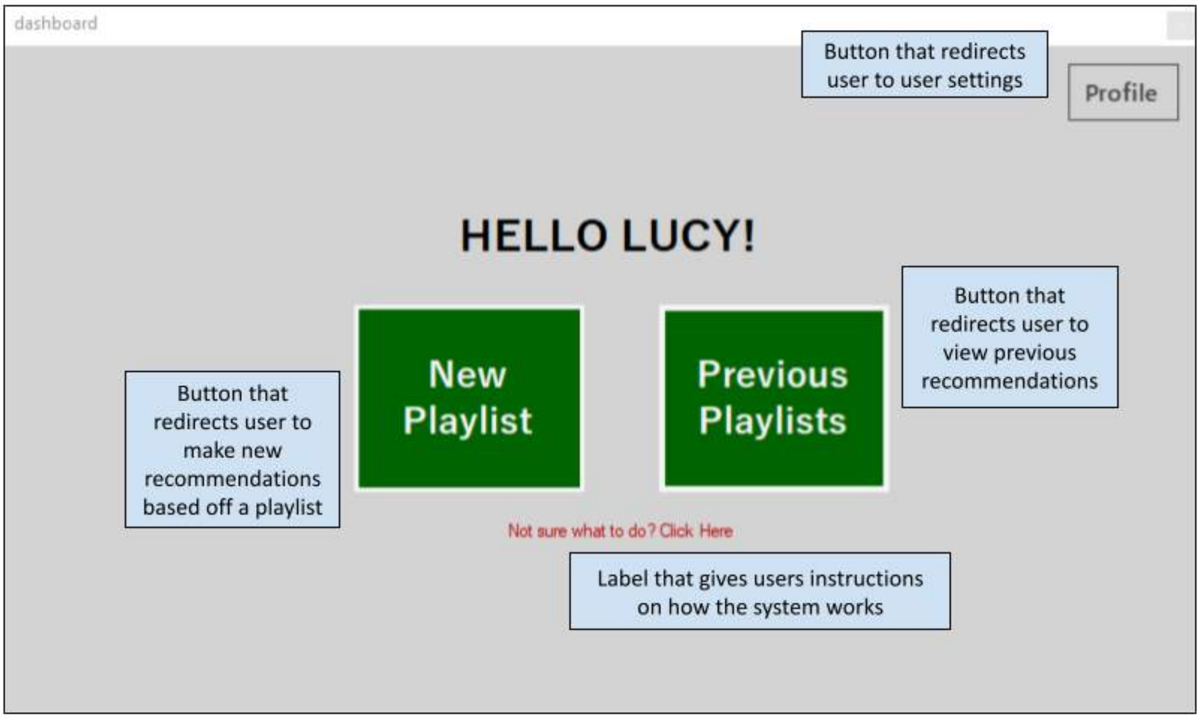
The user will be able to view their input by clicking this checkbox

The password will be hidden as the user writes to protect user integrity

Clear erases all user input from the textboxes

Redirects the user to the Registerwindow

The log-in page contains 2 textboxes and two buttons like the registration page. When the user clicks the login button the system will then check the username and password against the user credentials database. If the credentials match, then the user will be redirected to the dashboard. I have used the colour green throughout this design to keep the system uniform, and also green is the key colour of Spotify.



Button that redirects user to user settings

Button that redirects user to view previous recommendations

Button that redirects user to make new recommendations based off a playlist

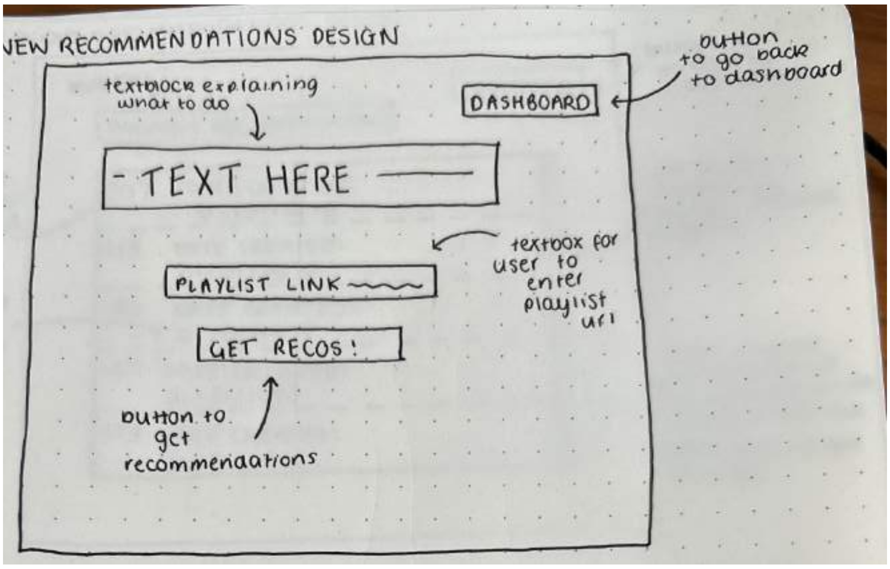
Label that gives users instructions on how the system works

The dashboard will have 3 directions for the user: to make a new playlist, view previous playlists, or go to the user settings. There will also be a label welcoming the user, with the username being displayed. The help label is subject to the client's thoughts on the complexity of the system, if she feels she can use the system with ease then it will be removed. Each of the 3 buttons relates to a specific function of the system.

Initial Drawings

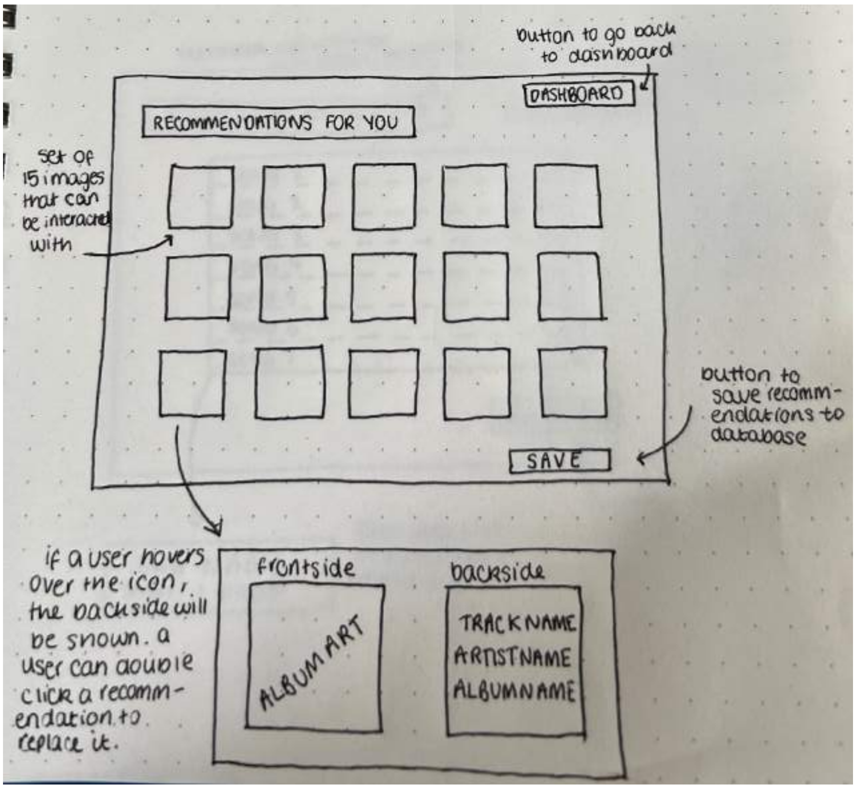
Creating recommendations based on a playlist is the epicentre of this system. The user will be able to enter a Spotify playlist URL to get these recommendations. Once the user clicks the button, the URL will be used to fetch the playlist information using the Spotify Web API, and this data will then be manipulated through various algorithms to get a set of recommendations based on the features of the tracks in the user playlist.

The window content will then be replaced to show the actual recommendations.

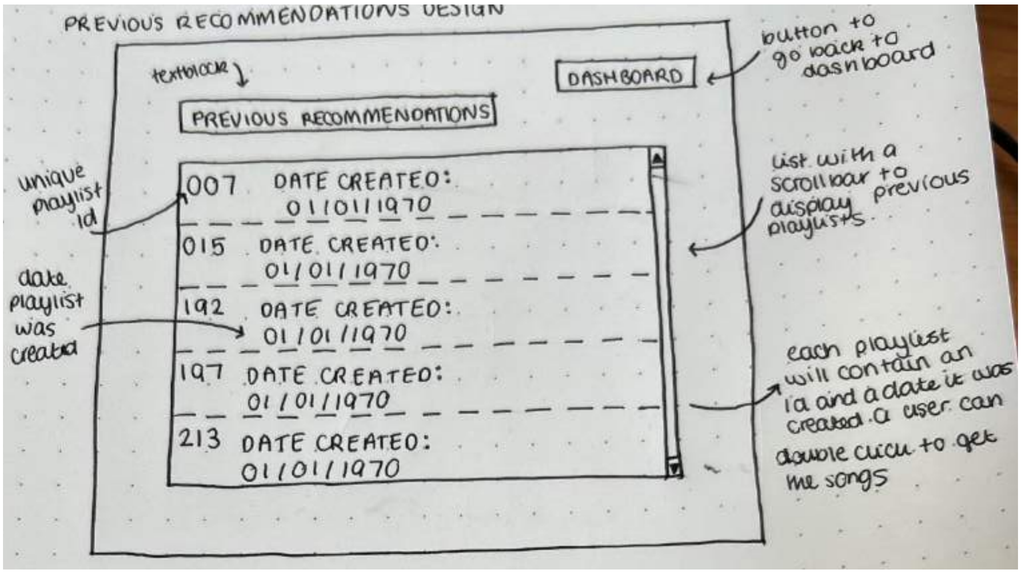

NEW RECOMMENDATIONS DESIGN
textblock explaining what to do
DASHBOARD
button to go back to dashboard
- TEXT HERE ——
textbox for user to enter playlist url
PLAYLIST LINK
GET RECOS!
button to get recommendations

They will be displayed in a grid view of 5 x 3, with the album art being the icon of each track. As explained in the drawing, if the user hovers over a certain recommendation, the album art will be replaced by the track name, artist name, and album name. While only this information will be displayed, it is important to note that key audio features and track features of the song will be stored in the database as well.

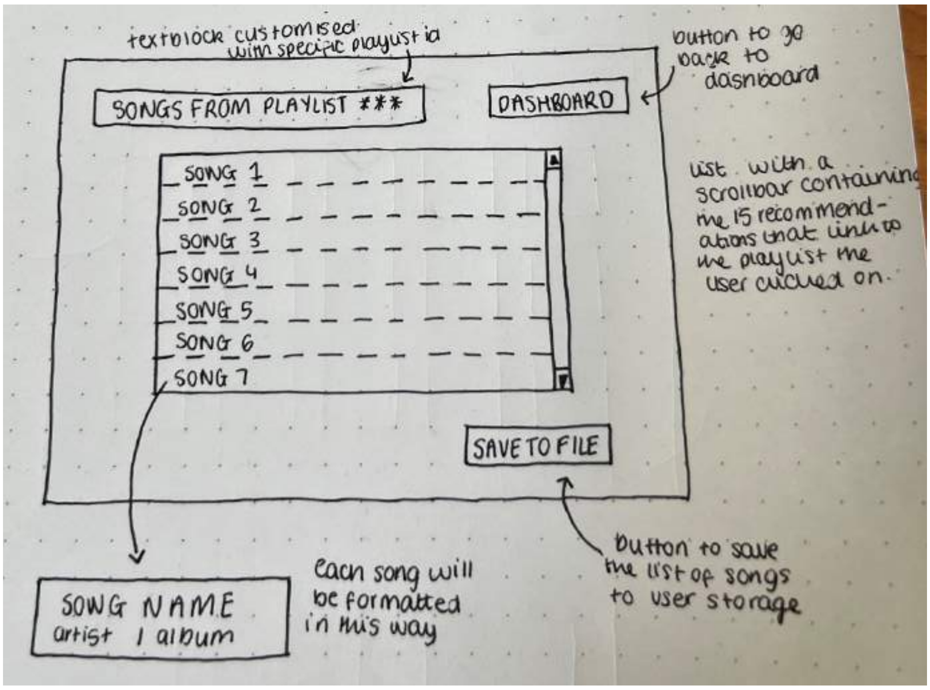
The user can double-click any of the 15 recommendations to remove it from the list and receive a new recommendation. Once the user is happy with the recommendations, they can click the save button to store the recommendations in the database.

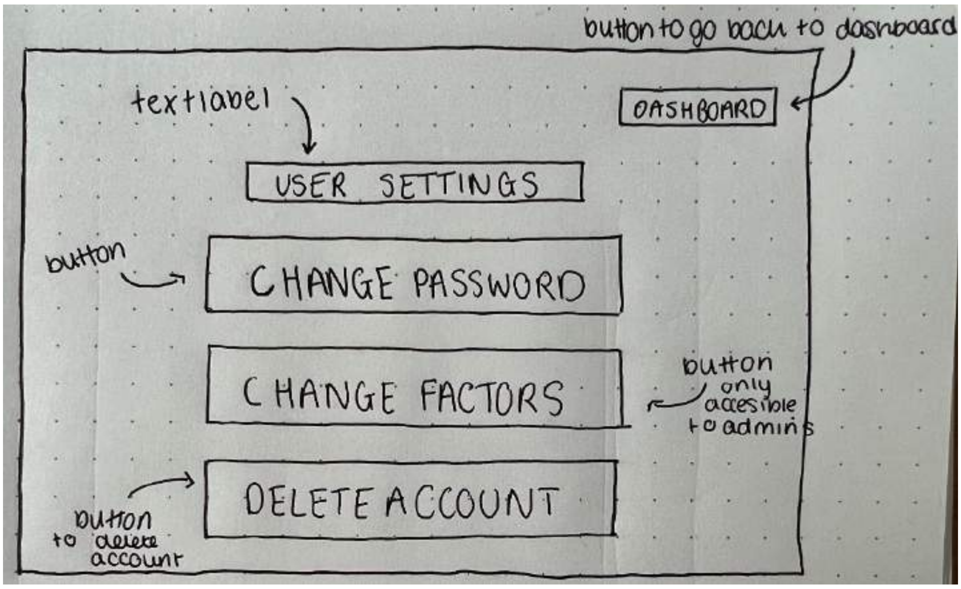

button to go back to dashboard
DASHBOARD
RECOMMENDATIONS FOR YOU
set of 15 images that can be interacted with
button to save recommendations to database
SAVE
if a user hovers over the icon, the backside will be shown. a user can double click a recommendation to replace it.
frontside
ALBUM ART
backside
TRACK NAME
ARTIST NAME
ALBUM NAME

Another feature the client expressed they wanted to have was the ability to view previous recommendations. Previous recommendations will be displayed in a list, with each set of recommendations being displayed in the format of its playlist ID and the date it was created (as seen in the drawing). The user can double-click any of the playlists to then view the songs within that set of recommendations.

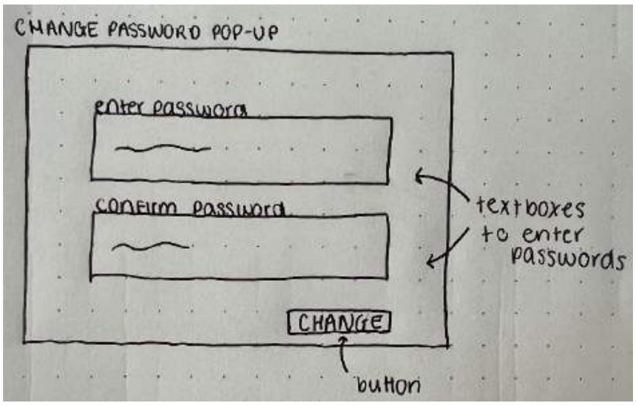

PREVIOUS RECOMMENDATIONS DESIGN
textblock
PREVIOUS RECOMMENDATIONS
DASHBOARD
button to go back to dashboard
unique playlist id
007 DATE CREATED: 01/01/1970
015 DATE CREATED: 01/01/1970
192 DATE CREATED: 01/01/1970
197 DATE CREATED: 01/01/1970
213 DATE CREATED: 01/01/1970
date playlist was created
list with a scrollbar to display previous playlists
each playlist will contain an id and a date it was created. a user can double click to get the songs

There will be a label that contains the playlist ID of the one the user clicked on. Each song will be displayed in a list, as demonstrated in the drawing. If the user clicks on the button to save to file, a text file with the recommendations will be created and stored in the user's local file storage. There is a multitude of data for each song that is stored within the database, but only key information that is beneficial for the user will be written to the text file.


textblock customised with specific playlist id
SONGS FROM PLAYLIST ***
DASHBOARD
button to go back to dashboard
SONG 1
SONG 2
SONG 3
SONG 4
SONG 5
SONG 6
SONG 7
list with a scrollbar containing the 15 recommend-ations that link to the playlist the user clicked on.
SAVE TO FILE
button to save the list of songs to user storage
SONG NAME
artist / album
each song will be formatted in this way

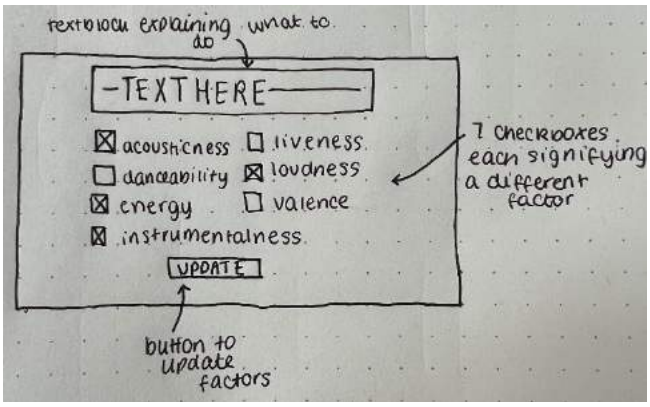


User Settings will compose of 4 buttons, one of which will be the button redirecting the user back to the dashboard. The button Change Factors will only be functional for users who are labelled as an admin. Otherwise, an error message will pop up when the user tries to click it. If the user clicks on the Delete Account button, then they will be asked to confirm that they want to delete their account and if they say yes then their account and all related data will be deleted from the system.



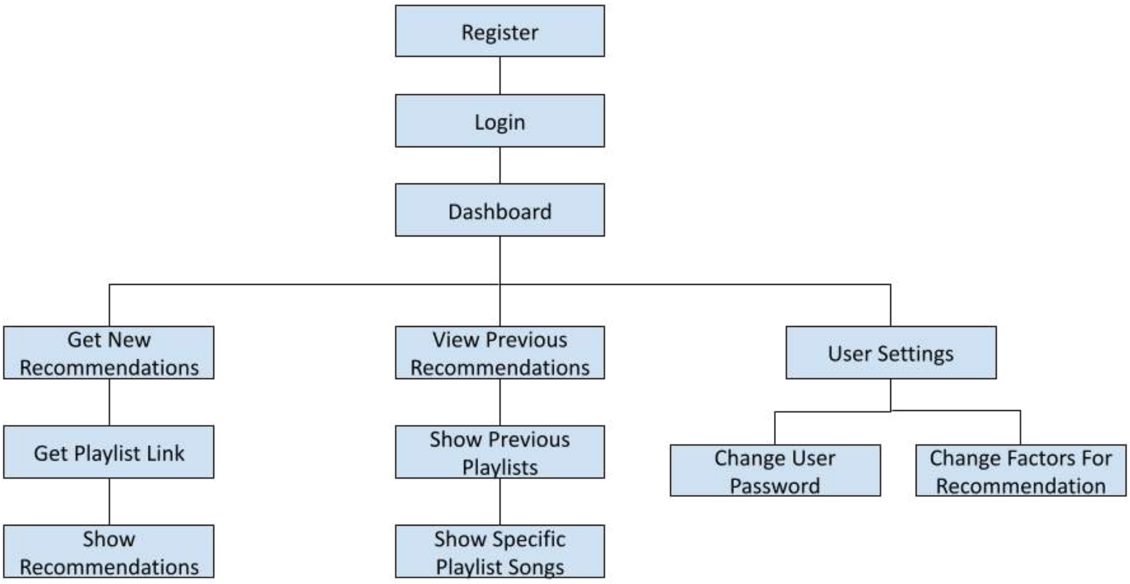
If the user clicks on the Change Password button a new pop-up will be displayed. This will contain 2 textboxes for the user to enter their new password. When the user clicks the button, as long as the password matches the validation criteria, the password will be updated in the database.

Providing the user is an admin, the change factors button will display a pop-up containing 7 checkboxes – each relating to a different factor within the recommendation algorithm. As well, there will be a button that the user can click to confirm their changes. Whether the checkbox is checked/unchecked upon initialization will depend on its state in the database.



Navigation Design

The following chart illustrates how users would navigate between all of the windows in the new system. While Register is the first window of the system, the main window is Dashboard as from here the user can access the full functionality of the program.



System Output Design

The system should export the playlists to text files and also to the main database. The text files output only certain data about the songs for the user, however the database holds more information about the songs.

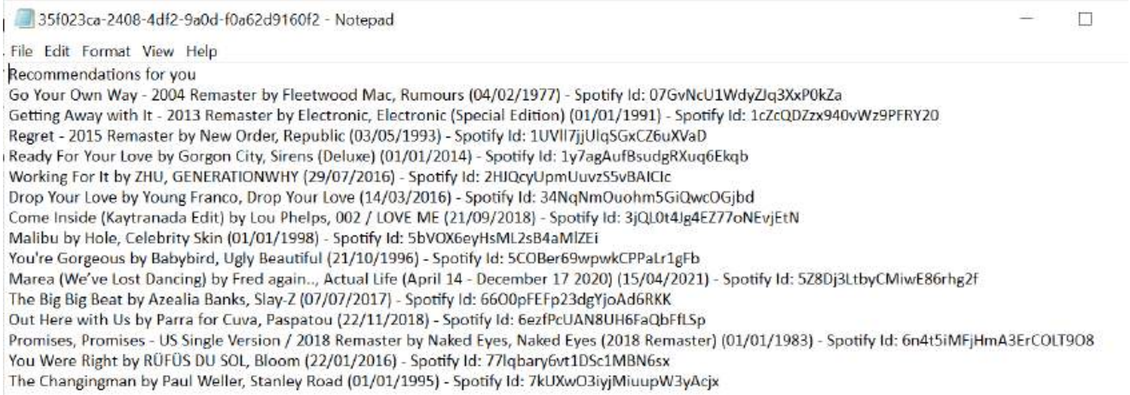
File structure: ***uniquename*.txt**

Song Name	Artist Name	Album Name	Album Release Date	Unique Spotify ID

Each field will be variable length and will be separated using tabs. I opted out of using a .csv file as some of the track names and album names contain commas. As well, the client expressed that they only want a list of the songs, and that they do not care about the compatibility of the file with different programs.

The unique name will be created using a UID and will be stored within a folder on the user's local storage, to minimise confusion.

An example of the output is as follows:


35f023ca-2408-4df2-9a0d-f0a62d9160f2 - Notepad
File Edit Format View Help

```
Recommendations for you
Go Your Own Way - 2004 Remaster by Fleetwood Mac, Rumours (04/02/1977) - Spotify Id: 07GvNcU1WdyZJq3XxP0kZa
Getting Away with It - 2013 Remaster by Electronic, Electronic (Special Edition) (01/01/1991) - Spotify Id: 1cZcQDZzx940vWz9PFRY20
Regret - 2015 Remaster by New Order, Republic (03/05/1993) - Spotify Id: 1UVIl7jjUlqSGxCZ6uXVaD
Ready For Your Love by Gorgon City, Sirens (Deluxe) (01/01/2014) - Spotify Id: 1y7agAufBsudgRXuq6Ekqb
Working For It by ZHU, GENERATIONWHY (29/07/2016) - Spotify Id: 2HJQcyUpmUuvzS5vBAICIc
Drop Your Love by Young Franco, Drop Your Love (14/03/2016) - Spotify Id: 34NqNmOuohm5GiQwcOGjbd
Come Inside (Kaytranada Edit) by Lou Phelps, 002 / LOVE ME (21/09/2018) - Spotify Id: 3jQL0t4Jg4EZ77oNEvjEtN
Malibu by Hole, Celebrity Skin (01/01/1998) - Spotify Id: 5bVOX6eyHsML2sB4aMlZEi
You're Gorgeous by Babybird, Ugly Beautiful (21/10/1996) - Spotify Id: 5COBer69wpwkCPPaLr1gFb
Marea (We've Lost Dancing) by Fred again.., Actual Life (April 14 - December 17 2020) (15/04/2021) - Spotify Id: 5Z8Dj3LtbyCMiwE86rhg2f
The Big Big Beat by Azealia Banks, Slay-Z (07/07/2017) - Spotify Id: 66O0pFEFp23dgYjoAd6RKK
Out Here with Us by Parra for Cuva, Paspatou (22/11/2018) - Spotify Id: 6ezfPcUAN8UH6FaQbFfLSp
Promises, Promises - US Single Version / 2018 Remaster by Naked Eyes, Naked Eyes (2018 Remaster) (01/01/1983) - Spotify Id: 6n4t5iMFjHmA3ErCOLT9O8
You Were Right by RÜFÜS DU SOL, Bloom (22/01/2016) - Spotify Id: 77lqbary6vt1DSc1MBN6sx
The Changingman by Paul Weller, Stanley Road (01/01/1995) - Spotify Id: 7kUXwO3iyjMiuupW3yAcjx
```

Algorithm Design

Throughout the system, there are multiple key processes that will be carried out. The following algorithms are starting points for each of the processes so I can understand how each of them will process and manipulate data.

Algorithm One

Explanation

Creating a new set of recommendations by fetching data using the Spotify Web API

Pseudocode

```
Var playlistLink <= USER INPUT
IF playlistLink = valid
    API call to get playlist info
    Assign data to trackModel collection
    API call to get audio features for each song in playlist
    Assign data to audioFeatures collection
    IdealValues = Calculate average for each factor
    Calculate groups of 5 for song.Ids
    FOR EACH group of song.Ids
        API call to get recommendations using IdealValues
        Assign data to recommendations collection
    ENDFOR
    OUTPUT recommendations
ELSE
OUTPUT "ERROR"
ENDIF
```

Algorithm Two

Explanation

Viewing previous sets of recommendations

Pseudocode

```
currentUser <= User.Id
var[] userPlaylists <= User.Playlists.Count
FOR EACH playlist in database
    IF userId = currentUser
        ADD playlist information TO userPlaylists
    END IF
END FOR
```

Algorithm Three

Explanation
When exporting a set of previous recommendations to a file, the system will need to abstract only the necessary information for each song and format it.

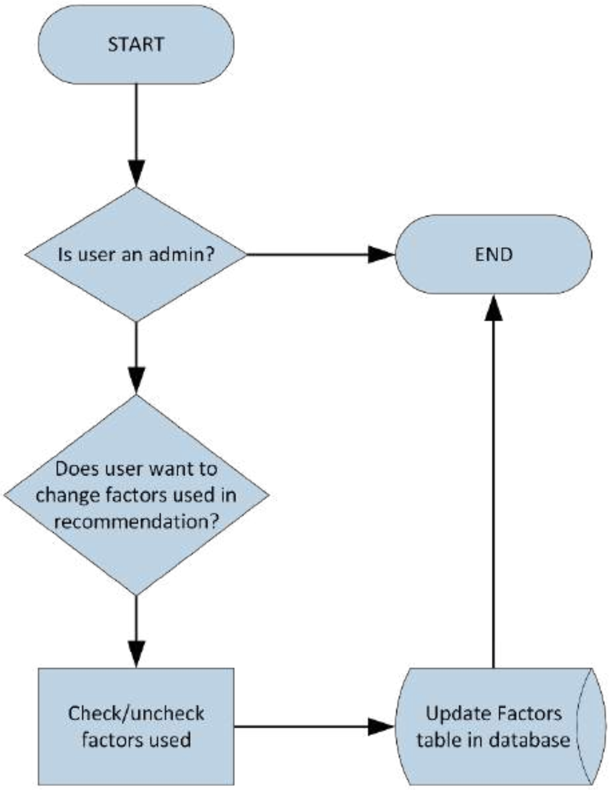
Pseudocode

```
fileName =  “fileName.txt”
OPEN fileName in “WRITE” mode
FOR EACH track in PlayListSongs
        lineFormat = “{track.Name} by {track.Artist}, {track.Album}
        ({track.Album_Release_Date}) – Spotify ID: {track.SongsId}”
        WRITELINE(filename, lineFormat)
ENDFOR
CLOSE fileName
```

Algorithm Four

Explanation
Changing recommendations factors is a feature only available to admins. Admins will be able to decide which recommendation factors should be included in the recommendation algorithm.

Flowchart


START
Is user an admin?
END
Does user want to change factors used in recommendation?
Check/uncheck factors used
Update Factors table in database

Sample of Possible SQL Queries

It is important that all SQL queries I include within the system are parameterized, to prevent SQL injection attacks.

USER LOG - IN

SELECT *
FROM users
WHERE username=? AND password=?

USER REGISTRATION

SELECT * FROM users
WHERE username=? *(check username isn't already present)*
 	
INSERT INTO users
VALUES (?, ?, ?)

CHANGE USER PASSWORD
UPDATE users SET password=? WHERE username=?

DELETE USER ACCOUNT AND ALL DATA ASSOCIATED WITH IT

DELETE FROM playlistsongs WHERE playlistsid =?
DELETE FROM playlists WHERE usersID=?
DELETE FROM users WHERE username=?

DISPLAY ALL PREVIOUS GROUPS OF RECOMMENDATIONS FOR A SPECIFIC USER
SELECT * FROM playlists WHERE usersId=?

INSERT SET OF RECOMMENDATIONS INTO DATABASE

INSERT INTO songs (name, artist, album, album_release_date, songsid, duration_ms, popularity) VALUES (?, ?, ?, ?, ?, ?, ?)
 	
INSERT INTO playlists (date_created, usersid) VALUES (?, ?)
SELECT MAX(playlistsid) FROM playlists
 	
INSERT INTO playlistsongs (playlistsid, songsid) VALUES (?,?)

Testing Plan

To ensure a robust system, I have planned several testing methods (as shown b to ensure that the system can handle all user inputs and that navigation between forms is working as it should. As well, I will test how the algorithms handle different data and ensure that there are sufficient exception handling methods in place.

Input and Output Testing Design

The following table has been designed to test all the features of the program. Each test will have a unique test ID and there will be a description of the test and the expected results. The test data will vary from typical (expected and correct) data to erroneous (that would cause the system to throw an exception, wrong data types) and extreme data (blank fields or boundary data), to test the system properly. The test highlighted in blue was added after implementation.

Test ID	Description	Test data	Expected results
Testing that a user can register on the system			
01	Check that the user can register if all fields have been completed correctly	Username: Testuser Password: Pas5word!	Message confirming user account has been created User details added to database User can now log in
02	Check that all fields are required fields	Username: Random1 Password:	Error message saying one or more fields is blank No user details added to database
03	Check the password matches the validation criteria (in proposed improvements)	Username: Random1 Password: password	Error message saying password isn't secure enough No user details added to database
04	Check that the username doesn't already exist in the database	Username: Lucy Password: password	Error message saying username already exists No user details added to database
Testing that a user can log in on the system			
05	Check that a user can log in if all fields have	Username: Admin Password: Pas5word!	The dashboard will be displayed

	been completed correctly		
06	Check that all fields are required fields	Username: Password: password	Error message saying that one or more fields are blank
07	Check if username is case-sensitive	Username: admin Password: Pas5word!	If the username exists in the database, it should log in and display the dashboard without being concerned about case
Making sure the dashboard works correctly			
08	Navigation to new recommendations	Click BtnNewRecos	Get Playlist Link page should be displayed
09	Navigation to previous recommendations	Click BtnOldRecos with an account that has recommendations and one that doesn’t	Playlist Songs page should be displayed IF user has previous recommendations
10	Navigation to user settings	Click BtnSettings	User Settings should be displayed
Entering a playlist URL to get recommendations			
11	User can enter a valid playlist URL and get a list of 15 recommendations	“5woNRzOWvpWXk4jlM3utL7”	Page display should change to 15 songs in a list view
12	Check that playlists cannot have more than 100 songs	“1UowMoBO8qiU61GIlvnEWr”	Error message saying a playlist mustn’t have more than 100 songs No recommendations are made
13	Check all fields are required fields	No input	Error message saying the field is blank No recommendation is made
14	Check if the playlist URL is valid	“wronginput”	Error message saying that an error occurred and for the user to try again No recommendation is made
Replacing recommendations			
15	Ensure the recommendations display correctly	Click BtnGetRecos	Should be a display of 15 songs with the album cover, song

			name, album name, and artist name Should be able to scroll up and down
16	Recommendations are all unique	No input	Each recommendation should be unique and none of them should be in the playlist given by the user
17	Check that user can delete a recommendation	Double-click any song to try and delete it	User is prompted to confirm they want to delete the recommendation Recommendation is deleted from the list and is replaced by a new recommendation
18	Check the recommendations do not run out	Keep deleting recommendations until something happens	Error message saying that there are no more recommendations that can be used to replace User must save the playlist or go back and try again
19	Check that the program will not crash when user is double clicking	Double-click the screen 20 times quickly	No response from the program
19a	Check user can return to dashboard	Click dashboard button	Message confirming user wants to return then shows dashboard
20	Check user can save recommendations	Click BtnSaveRecos	Success message confirming the recommendations have been saved to the system Recommendations added to the database
Checking the database			
21	Check that if there is a duplicate song, then it isn't added to the database	Check before and after creating a new set of recommendations	Each song should be unique in the database, no track id should be the same

22	Check the songs are successfully linked to a user playlist	Check before and after creating a new set of recommendations	Playlists contains a new unique playlist id that links to the user id Playlistsongs contains 15 fields linking the playlist id with the 15 song ids
Previous Recommendations playlists			
23	Check user can't access previous recommendations if they don't have any recommendations linked to their account	Create a new account and try and access the page	Error message saying the user hasn't made any recommendations The window isn't accessible
24	Check recommendation playlists are displayed correctly	No input	List view of all the playlists linked to user, with the playlist id and the date they were created
25	Check the program will not crash when clicking excessively	Double-click 20 times quickly	No response from program
Previous recommendation songs			
26	Check clicking on a playlist will display the songs within the playlist	Double-click any playlist to access songs	List view of 15 songs that relate to the playlist the user clicked on Should display the song name, artist name, and album name
27	Check the user can export the songs to file in the correct format	Export songs to playlist and check text file	Success message saying file has been exported New unique file name in NEA folder in local storage with the correct format
User Settings			
28	Check user can change password and it will successfully update the database	Check database before and after updating a password	Success message saying the password has successfully been updated User password updated in the database

29	Check user can change factors ONLY if an admin	Try accessing the feature with and without an admin account	Error message saying user cannot change factors IF they are not an admin If they are an admin then the pop-up box will show
30	Check factors successfully update in database if they are modified	Check database before and after changing the factors	Success message saying factors have successfully been updated Factors state updated in the database
31	Check user can delete account and all data will be removed from the database	Check database before and after deleting an account (make sure account has recommendations already to check that feature also)	Success message saying user account has successfully been deleted Information in playlists, playlistsongs, and users table relating to the user will be removed User is redirected to the registration page

When testing I will use the following table format to record the results:

Test Id	Expected results	Actual results	Comments/ Corrections

I have linked each test to a user objective to ensure that the final solution will meet the client's required needs. This is outlined in the following table:

Objectives for proposed system	Applicable Test Id's
1. The system will calculate and display a list of song recommendations based on a user playlist	15, 16
2. The system will calculate and display a list of song recommendations based on a user playlist.	11-14

3. The system should involve minimum text entry, with the only information being the account details and Spotify playlist URL, to save time and minimise errors.	5, 12-14, 19, 25
4. The system should be able to display 15 songs in random order.	16
5. New users must register to use the system: a) The new user must complete a registration form in which they enter their name, a username, and a password. b) The username must be unique (on the system). c) Passwords must be at least 8 characters long and contain a mix of character types. d) Only one user registered on the system must be labelled as 'administrator' and get admin permissions.	1-4
6. The user must be able to share a Spotify playlist URL and receive recommendations based on the songs within it.	11, 15
7. The system should have the option to discard songs that the user doesn't like and recommend a new song instead.	17, 18
8. The system must be able to store all the relevant details about every song recommended with the following details being essential for each song: a) Song Name b) Artist Name c) Spotify Unique ID	21, 22
9. The system must store the recommendations in a user profile for the user to access at a later date.	20
10. The option to save the recommendations in the format of a text file must be available to the user, so it is easily accessible.	27
11. There should be the option for an administrator to delete/modify user accounts and their details.	*Note: objective not achieved in implementation*

12. There should be the option for an administrator to modify the deciding factors of the recommendation system.	29, 30
13. The recommendations should include a variety of music types – based on genre, artist, album, happiness, etc.	15, 16
14. The recommendations must not include more than 3 songs per artist.	15, 16
15. Users must be able to delete their account from the system.	31

Navigation Testing Design

To ensure the program navigation works as I expect it to, I’ve created a table to outline how I want the windows to link. When testing my program, I will simply check the boxes off if the navigation works as intended. From my design, I have decided that 8 windows should be sufficient (A-H), but this will not be confirmed until I have finished implementation.

Navigating from: → **Navigating to: ↓**	A	B	C	D	E	F	G	H
A	▨							
B		▨						
C			▨					
D				▨				
E					▨			
F						▨		
G							▨	
H								▨

Technical Solution

I chose to use WPF .NET when designing my system, as I preferred the graphical design freedom it gave me. I decided to attempt to implement the Model–View–ViewModel (MVVM) architectural pattern when writing my application. There are **6** windows that make up the system, with another **6** pages which are implemented within windows. There are also a number of models which contain the more logical processes, which are used within different windows.

Models

There are seven models within the program, each designed to store information from the API and table.

TrackModel.cs

The TrackModel is used when fetching the playlist song information from the Spotify API. It follows the response information given by the API as seen here: https://developer.spotify.com/documentation/web-api/reference/#/operations/get-playlists-tracks. The implementation of this can be seen in window three: new recommendations.

```
using Newtonsoft.Json;
using System;

namespace GetPlaylistInfo.MVVM.Model
{

    public partial class PlaylistModel
    {
        [JsonProperty("href")]
        public Uri Href { get; set; }

        [JsonProperty("items")]
        public Item[] Items { get; set; }

        [JsonProperty("limit")]
        public long Limit { get; set; }

        [JsonProperty("next")]
        public object Next { get; set; }

        [JsonProperty("offset")]
        public long Offset { get; set; }

        [JsonProperty("previous")]
        public object Previous { get; set; }

        [JsonProperty("total")]
        public long Total { get; set; }
    }

    public partial class Item
    {
        [JsonProperty("added_at")]
        public string AddedAt { get; set; }

        [JsonProperty("added_by")]
        public AddedBy AddedBy { get; set; }

        [JsonProperty("is_local")]
        public bool IsLocal { get; set; }

        [JsonProperty("primary_color")]
        public object PrimaryColor { get; set; }

        [JsonProperty("track")]
        public Track Track { get; set; }

        [JsonProperty("video_thumbnail")]
        public VideoThumbnail VideoThumbnail { get; set; }
    }

    public partial class AddedBy
    {
        [JsonProperty("external_urls")]
        public ExternalUrls ExternalUrls { get; set; }

        [JsonProperty("href")]
        public Uri Href { get; set; }

        [JsonProperty("id")]
        public string Id { get; set; }

        [JsonProperty("type")]
        public string Type { get; set; }

        [JsonProperty("uri")]
        public string Uri { get; set; }

        [JsonProperty("name", NullValueHandling = NullValueHandling.Ignore)]
        public string Name { get; set; }
    }

    public partial class ExternalUrls
    {
        [JsonProperty("spotify")]
        public Uri Spotify { get; set; }
    }

    public partial class Track
    {
        [JsonProperty("album")]
        public Album { get; set; }

        [JsonProperty("artists")]
        public AddedBy[] Artists { get; set; }

        [JsonProperty("available_markets")]
        public string[] AvailableMarkets { get; set; }

        [JsonProperty("disc_number")]
        public long DiscNumber { get; set; }

        [JsonProperty("duration_ms")]
        public long DurationMs { get; set; }

        [JsonProperty("episode")]
        public bool Episode { get; set; }

        [JsonProperty("explicit")]
        public bool Explicit { get; set; }

        [JsonProperty("external_ids")]
        public ExternalIds ExternalIds { get; set; }

        [JsonProperty("external_urls")]
        public ExternalUrls ExternalUrls { get; set; }

        [JsonProperty("href")]
        public Uri Href { get; set; }

        [JsonProperty("id")]
        public string Id { get; set; }

        [JsonProperty("is_local")]
        public bool IsLocal { get; set; }

        [JsonProperty("name")]
        public string Name { get; set; }

        [JsonProperty("popularity")]
        public int Popularity { get; set; }
        [JsonProperty("preview_url")]
        public Uri PreviewUrl { get; set; }

        [JsonProperty("track")]
        public bool TrackTrack { get; set; }

        [JsonProperty("track_number")]
        public long TrackNumber { get; set; }

        [JsonProperty("type")]
        public string Type { get; set; }

        [JsonProperty("uri")]
        public string Uri { get; set; }
    }

    public partial class Album
    {
        [JsonProperty("album_type")]
        public string AlbumType { get; set; }

        [JsonProperty("artists")]
        public AddedBy[] Artists { get; set; }

        [JsonProperty("available_markets")]
        public string[] AvailableMarkets { get; set; }

        [JsonProperty("external_urls")]
        public ExternalUrls ExternalUrls { get; set; }

        [JsonProperty("href")]
        public Uri Href { get; set; }

        [JsonProperty("id")]
        public string Id { get; set; }

        [JsonProperty("images")]
        public Image[] Images { get; set; }

        [JsonProperty("name")]
        public string Name { get; set; }

        [JsonProperty("release_date")]
        public string ReleaseDate { get; set; }

        [JsonProperty("release_date_precision")]
        public string ReleaseDatePrecision { get; set; }
        [JsonProperty("total_tracks")]
        public long TotalTracks { get; set; }

        [JsonProperty("type")]
        public string Type { get; set; }

        [JsonProperty("uri")]
        public string Uri { get; set; }
    }

    public partial class Image
    {
        [JsonProperty("height")]
        public long Height { get; set; }

        [JsonProperty("url")]
        public Uri Url { get; set; }

        [JsonProperty("width")]
        public long Width { get; set; }
    }

    public partial class ExternalIds
    {
        [JsonProperty("isrc")]
        public string Isrc { get; set; }
    }

    public partial class VideoThumbnail
    {
        [JsonProperty("url")]
        public object Url { get; set; }
    }
}
```

AudioFeatures.cs

The AudioFeatures model is used when using the Spotify API to get the audio features of a song. The content of the class is as provided by the API reference: https://developer.spotify.com/documentation/web-api/reference/#/operations/get-several-audio-features.

```
using Newtonsoft.Json;
using System;

namespace GetPlaylistInfo.MVVM.Model
{
    public partial class AudioFeatures
    {
        [JsonProperty("audio_features")]
        public AudioFeature[] Features { get; set; }
    }

    public partial class AudioFeature
    {
        [JsonProperty("danceability")]
        public double Danceability { get; set; }

        [JsonProperty("energy")]
        public double Energy { get; set; }

        [JsonProperty("key")]
        public long Key { get; set; }

        [JsonProperty("loudness")]
        public double Loudness { get; set; }

        [JsonProperty("mode")]
        public long Mode { get; set; }

        [JsonProperty("speechiness")]
        public double Speechiness { get; set; }

        [JsonProperty("acousticness")]
        public double Acousticness { get; set; }

        [JsonProperty("instrumentalness")]
        public double Instrumentalness { get; set; }

        [JsonProperty("liveness")]
        public double Liveness { get; set; }

        [JsonProperty("valence")]
        public double Valence { get; set; }
        [JsonProperty("tempo")]
        public double Tempo { get; set; }

        [JsonProperty("type")]
        public string Type { get; set; }

        [JsonProperty("id")]
        public string Id { get; set; }

        [JsonProperty("uri")]
        public string Uri { get; set; }

        [JsonProperty("track_href")]
        public Uri TrackHref { get; set; }

        [JsonProperty("analysis_url")]
        public Uri AnalysisUrl { get; set; }

        [JsonProperty("duration_ms")]
        public long DurationMs { get; set; }

        [JsonProperty("time_signature")]
        public long TimeSignature { get; set; }
    }
}
```

UserPlaylistInfo.cs

The UserPlaylistInfo model is used to combine the song information from getting the audio features as well as the playlist tracks' information. This removes all of the unnecessary information in the other classes that the recommendation algorithms do not require.

```
namespace GetPlaylistInfo.MVVM.Model
{
    public class TrackInfo
    {
        public int Index { get; set; }

        public string TrackId { get; set; }
        public string SongName { get; set; } //tracks.name
        public string AlbumName { get; set; } //tracks.albums.name
        public string ArtistName { get; set; } //tracks.artists.name
        public long DurationMs { get; set; } //tracks.duration_ms
        public int Popularity { get; set; } //tracks.popularity

        //from audio features
        public double Acousticness { get; set; }
        public double Danceability { get; set; }
        public double Energy { get; set; }
        public double Instrumentalness { get; set; }
        //(integers mapped to pitch class notation)
        public long Key { get; set; }
        public double Liveness { get; set; }
        //(in dB)
        public double Loudness { get; set; }
        //(major is 1 minor is 0)
        public long Mode { get; set; }
        public double Speechiness { get; set; }
        public double Tempo { get; set; }
        public long TimeSignature { get; set; }
        public double Valence { get; set; }
    }
}
```

RecommendationFeatures.cs

The RecommendationFeatures model class contains the ideal values that will be parsed into the get recommendations fetch of the API. They are calculated using the audio features in the GetIdealValues class.

```
namespace GetPlaylistInfo.MVVM.Model
{
    public partial class RecommendationFeatures
    {
        public string Seed_tracks { get; set; }
        public int Limit { get; set; }

        public double Target_acousticness { get; set; }
        public double Target_danceability { get; set; }
        public int Target_duration_ms { get; set; }
        public double Target_energy { get; set; }
        public double Target_instrumentalness { get; set; }
        public int Target_key { get; set; }
        public double Target_liveness { get; set; }
        public double Target_loudness { get; set; }
        public int Target_popularity { get; set; }
        public int Target_tempo { get; set; }
        public int Target_time_signature { get; set; }
        public double Target_valence { get; set; }
    }
}
```

GetRecommendations.cs

The GetRecommendations model stores the recommendation information received from the Spotify API. This is then manipulated so the recommendations only contain the necessary information which can then be stored in the database.

```
using Newtonsoft.Json;
using System;

namespace GetPlaylistInfo.MVVM.Model
{
    public partial class GetRecommendations
    {
        [JsonProperty("tracks")]
        public RecoTrack[] Tracks { get; set; }

        [JsonProperty("seeds")]
        public Seed[] Seeds { get; set; }
    }

    public partial class Seed
    {
        [JsonProperty("initialPoolSize")]
        public long InitialPoolSize { get; set; }

        [JsonProperty("afterFilteringSize")]
        public long AfterFilteringSize { get; set; }

        [JsonProperty("afterRelinkingSize")]
        public long AfterRelinkingSize { get; set; }

        [JsonProperty("id")]
        public string Id { get; set; }

        [JsonProperty("type")]
        public string Type { get; set; }

        [JsonProperty("href")]
        public Uri Href { get; set; }
    }

    public partial class RecoTrack
    {
        [JsonProperty("album")]
        public RecoAlbum Album { get; set; }

        [JsonProperty("artists")]
        public RecoArtist[] Artists { get; set; }

        [JsonProperty("available_markets")]
        public string[] AvailableMarkets { get; set; }

        [JsonProperty("disc_number")]
        public long DiscNumber { get; set; }

        [JsonProperty("duration_ms")]
        public long DurationMs { get; set; }

        [JsonProperty("explicit")]
        public bool Explicit { get; set; }

        [JsonProperty("external_ids")]
        public ExternalIds ExternalIds { get; set; }

        [JsonProperty("external_urls")]
        public ExternalUrls ExternalUrls { get; set; }

        [JsonProperty("href")]
        public Uri Href { get; set; }
        [JsonProperty("id")]
        public string Id { get; set; }

        [JsonProperty("is_local")]
        public bool IsLocal { get; set; }

        [JsonProperty("name")]
        public string Name { get; set; }

        [JsonProperty("popularity")]
        public long Popularity { get; set; }

        [JsonProperty("preview_url")]
        public Uri PreviewUrl { get; set; }

        [JsonProperty("track_number")]
        public long TrackNumber { get; set; }

        [JsonProperty("type")]
        public string Type { get; set; }

        [JsonProperty("uri")]
        public string Uri { get; set; }
    }

    public partial class RecoAlbum
    {
        [JsonProperty("album_type")]
        public string AlbumType { get; set; }

        [JsonProperty("artists")]
        public RecoArtist[] Artists { get; set; }

        [JsonProperty("available_markets")]
        public string[] AvailableMarkets { get; set; }

        [JsonProperty("external_urls")]
        public ExternalUrls ExternalUrls { get; set; }

        [JsonProperty("href")]
        public Uri Href { get; set; }

        [JsonProperty("id")]
        public string Id { get; set; }

        [JsonProperty("images")]
        public Image[] Images { get; set; }
        [JsonProperty("name")]
        public string Name { get; set; }

        [JsonProperty("release_date")]
        public string ReleaseDate { get; set; }

        [JsonProperty("release_date_precision")]
        public string ReleaseDatePrecision { get; set; }

        [JsonProperty("total_tracks")]
        public long TotalTracks { get; set; }

        [JsonProperty("type")]
        public string Type { get; set; }

        [JsonProperty("uri")]
        public string Uri { get; set; }
    }

    public partial class RecoArtist
    {
        [JsonProperty("external_urls")]
        public ExternalUrls ExternalUrls { get; set; }

        [JsonProperty("href")]
        public Uri Href { get; set; }

        [JsonProperty("id")]
        public string Id { get; set; }

        [JsonProperty("name")]
        public string Name { get; set; }

        [JsonProperty("type")]
        public string Type { get; set; }

        [JsonProperty("uri")]
        public string Uri { get; set; }
    }

    public partial class RecoExternalUrls
    {
        [JsonProperty("spotify")]
        public Uri Spotify { get; set; }
    }
```

```
    public partial class RecoImage
    {
        [JsonProperty("height")]
        public long Height { get; set; }

        [JsonProperty("url")]
        public Uri Url { get; set; }

        [JsonProperty("width")]
        public long Width { get; set; }
    }

    public partial class RecoExternalIds
    {
        [JsonProperty("isrc")]
        public string Isrc { get; set; }
    }
}
```

SaveToPlaylist.cs

SaveToPlaylist contains the song information that will be stored in the database.

```
namespace GetPlaylistInfo.MVVM.Model
{
    public class SaveToPlaylist
    {
        public int Index { get; set; }
        public string Name { get; set; }
        public string Artist { get; set; }
        public string Album { get; set; }
        public string Album_Release_Date { get; set; }
        public long DurationMs { get; set; }
        public string SongsId { get; set; }
        public int Popularity { get; set; }
    }
}
```

TblPlaylists.cs

TblPlaylists is a simple class that stores the playlist information from the database to display in the list on the ShowPlaylists page.

```
namespace GetPlaylistInfo.MVVM.Model
{
    public class TblPlaylists
    {
        public int PlaylistId { get; set; }
        public string Date_Created { get; set; }
    }
}
```

Theme

I created a custom theme called Crabtree.ListView.Theme.xaml where I designed how I wanted each list view to look. Each element has content binding which corresponds with the data context in each window where such list formats are used.

```
<ResourceDictionary
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">

    <Style TargetType="ListView" x:Key="NewRecoListStyle">
        <Setter Property="Foreground" Value="White"/>
        <Setter Property="BorderThickness" Value="0"/>
        <Setter Property="ItemContainerStyle">
            <Setter.Value>
                <Style TargetType="ListViewItem">
                    <Setter Property="Template">
                        <Setter.Value>
                            <ControlTemplate TargetType="ListViewItem">
                                <ContentPresenter/>
                            </ControlTemplate>
                        </Setter.Value>
                    </Setter>
                </Style>
            </Setter.Value>
        </Setter>

        <Setter Property="ItemTemplate">
            <Setter.Value>
                <DataTemplate>
                    <DockPanel Margin="4">
                        <DockPanel.Style>
                            <Style TargetType="DockPanel">
                                <Style.Triggers>
                                    <Trigger Property="IsMouseOver"
                                             Value="True">
                                        <Setter Property="Background"
                                                Value="#303030"/>
                                    </Trigger>
                                    <Trigger Property="IsMouseOver"
                                             Value="False">
                                        <Setter Property="Background"
                                                Value="Transparent"/>
                                    </Trigger>
                                    <DataTrigger Binding="{Binding RelativeSource={RelativeSource
Mode=FindAncestor,
                                                    AncestorType={x:Type ListBoxItem}},
                                                    Path=IsSelected}"
                                                 Value="True">
                    <Setter Property="Background"
                        Value="#303030"/>
                  </DataTrigger>
                </Style.Triggers>
              </Style>
            </DockPanel.Style>
            <Image Source="{Binding Album.Images[2].Url}" Width="50" Height="50"
DockPanel.Dock="Left"/>
            <StackPanel Margin="4,0,0,0">
              <TextBlock Text="{Binding Name }"
            Foreground="White"
            FontSize="14"
            FontWeight="Medium"/>
              <TextBlock Text="{Binding Artists[0].Name }"
            Foreground="Gray"
            FontSize="14"/>
            </StackPanel>
          </DockPanel>
        </DataTemplate>
      </Setter.Value>
    </Setter>
  </Style>

  <Style TargetType="ListView" x:Key="OldRecoListStyle">
    <Setter Property="Foreground" Value="White"/>
    <Setter Property="BorderThickness" Value="0"/>
    <Setter Property="ItemContainerStyle">
      <Setter.Value>
        <Style TargetType="ListViewItem">
          <Setter Property="Template">
            <Setter.Value>
              <ControlTemplate TargetType="ListViewItem">
                <ContentPresenter/>
              </ControlTemplate>
            </Setter.Value>
          </Setter>
        </Style>
      </Setter.Value>
    </Setter>
    <Setter Property="ItemTemplate">
      <Setter.Value>
        <DataTemplate>
          <DockPanel Margin="4">
            <DockPanel.Style>
              <Style TargetType="DockPanel">
                <Style.Triggers>
                  <Trigger Property="IsMouseOver"
```

```
                      Value="True">
                    <Setter Property="Background"
                        Value="#303030"/>
                  </Trigger>
                  <Trigger Property="IsMouseOver"
                      Value="False">
                    <Setter Property="Background"
                        Value="Transparent"/>
                  </Trigger>
                  <DataTrigger Binding="{Binding RelativeSource={RelativeSource
Mode=FindAncestor,
                          AncestorType={x:Type ListBoxItem}},
                          Path=IsSelected}"
                        Value="True">
                    <Setter Property="Background"
                        Value="#303030"/>
                  </DataTrigger>
                </Style.Triggers>
              </Style>
            </DockPanel.Style>
            <TextBlock Text="{Binding PlaylistId }"
            Foreground="White"
            FontSize="14"
            FontWeight="Medium" DockPanel.Dock="Left" />
            <StackPanel Margin="4,0,0,0">
              <TextBlock Text="Date Created:"
            Foreground="Gray"
            FontSize="14"/>
              <TextBlock Text="{Binding Date_Created}"
            Foreground="Gray"
            FontSize="14"/>
            </StackPanel>
          </DockPanel>
        </DataTemplate>
      </Setter.Value>
    </Setter>
  </Style>

  <Style TargetType="ListView" x:Key="ShowSongsListStyle">
    <Setter Property="Foreground" Value="White"/>
    <Setter Property="BorderThickness" Value="0"/>
    <Setter Property="ItemContainerStyle">
      <Setter.Value>
        <Style TargetType="ListViewItem">
          <Setter Property="Template">
            <Setter.Value>
              <ControlTemplate TargetType="ListViewItem">
```

```
                                <ContentPresenter/>
                            </ControlTemplate>
                        </Setter.Value>
                    </Setter>
                </Style>
            </Setter.Value>
        </Setter>
        <Setter Property="ItemTemplate">
            <Setter.Value>
                <DataTemplate>
                    <DockPanel Margin="4">
                        <DockPanel.Style>
                            <Style TargetType="DockPanel">
                                <Style.Triggers>
                                    <Trigger Property="IsMouseOver"

                                            Value="True">
                                        <Setter Property="Background"
                                                Value="#303030"/>
                                    </Trigger>
                                    <Trigger Property="IsMouseOver"
                                            Value="False">
                                        <Setter Property="Background"
                                                Value="Transparent"/>
                                    </Trigger>
                                    <DataTrigger Binding="{Binding RelativeSource={RelativeSource
Mode=FindAncestor,
                                                    AncestorType={x:Type ListBoxItem}},
                                                    Path=IsSelected}"
                                                Value="True">
                                        <Setter Property="Background"
                                                Value="#303030"/>
                                    </DataTrigger>
                                </Style.Triggers>
                            </Style>
                        </DockPanel.Style>
                        <StackPanel Margin="4,0,0,0">
                            <TextBlock Text="{Binding Name}"
                        Foreground="White"
                        FontSize="14"
                        FontWeight="Medium"/>
                            <TextBlock Text="{Binding Artist}"
                        Foreground="Gray"
                        FontSize="12"/>
                            <TextBlock Text="{Binding Album}"
                        Foreground="Gray"
                        FontSize="12"/>
                        </StackPanel>
                        <TextBlock/>
                    </DockPanel>
                </DataTemplate>
            </Setter.Value>
        </Setter>
    </Style>
</ResourceDictionary>
```

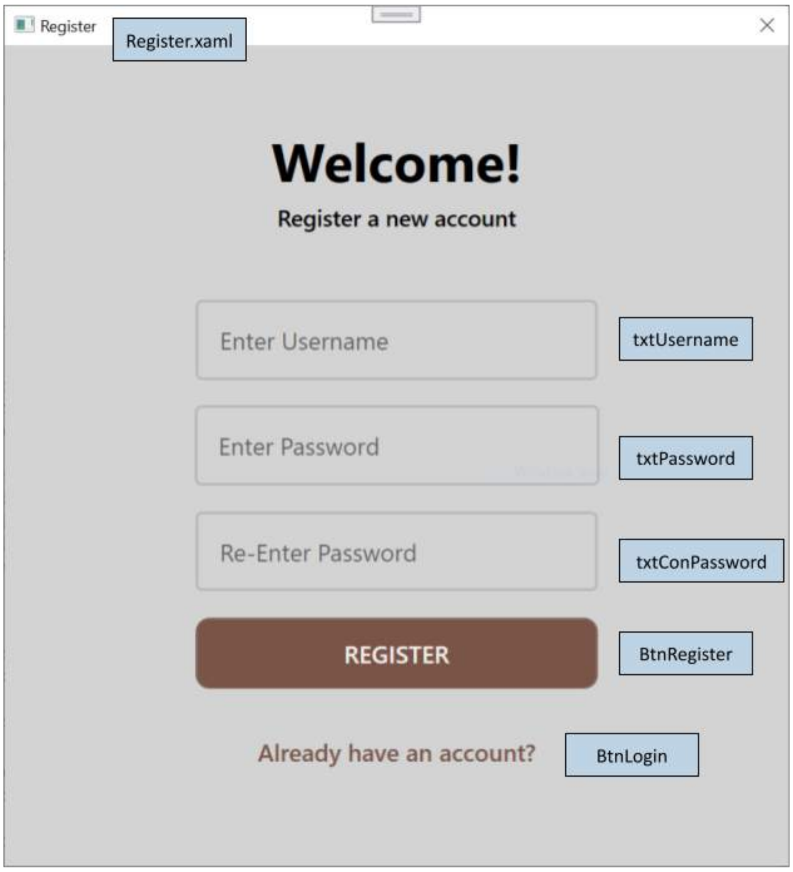
Authorisation

In order to access Spotify data and features I have to obtain an access token through Spotify's implantation of the OAuth 2.0 authorization framework. Using the client credentials Spotify granted me using their developer's dashboard, I am able to create a request for access. I used the client credentials flow as I am not accessing any personal data, and to simplify the process I used the library SpotifyAPI (https://johnnycrazy.github.io/SpotifyAPI-NET/) to initialize the authentication.

```
using SpotifyAPI.Web;
using System.Threading.Tasks;

namespace GetPlaylistInfo.MVVM.Model
{

    public class Authorize
    {
        public static string AccessToken { get; set; } //accessible by the other models to fetch
data from the API
        public static async Task GetAccessToken() //aysnchronous task to run as the system
boots up
        {
            var config = SpotifyClientConfig.CreateDefault();

            var request = new
ClientCredentialsRequest("d2734de58b6047219c75ccba18a7fab",
"4b6aacec430a4828900e7cf6d784d417"); //secret client credentials created in admin
dashboard
            var response = await new OAuthClient(config).RequestToken(request);
            AccessToken = response.AccessToken; //gets access token
        }
    }
}
```

Window One: Register

The Register window is the opening window upon starting the program. It contains 3 textboxes, 2 buttons, and 2 labels. Both password textboxes hide the password as the user types their password. Once the user clicks BtnRegister the password and username will be checked and if they pass the validation criteria then the user details will be added to the database. The user must then click BtnLogin to be redirected to the Login window.



Register.xaml

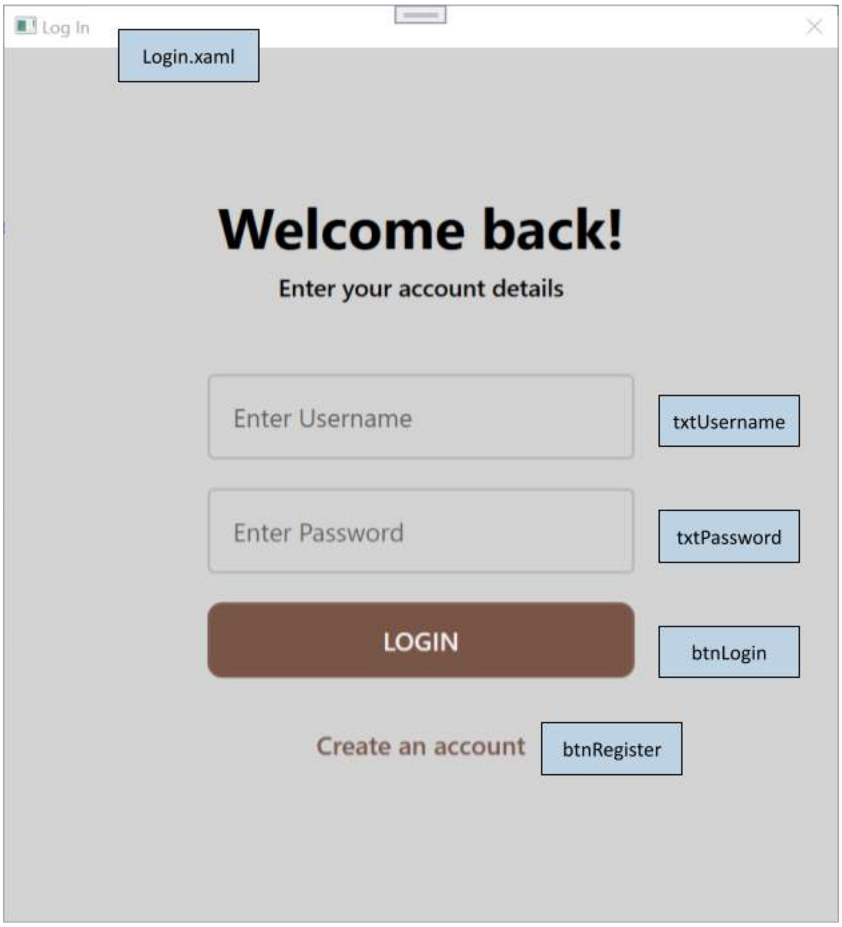
```
<Window x:Class="GetPlaylistInfo.MVVM.View.Register"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:GetPlaylistInfo.MVVM.View"
        xmlns:materialDesign="http://materialdesigninxaml.net/winfx/xaml/themes"
        mc:Ignorable="d"
        Background="LightGray"
        WindowStyle="SingleBorderWindow"
        ResizeMode="NoResize"
        WindowStartupLocation="CenterScreen"
        Title="Register" Height="650" Width="600">
    <StackPanel VerticalAlignment="Center">

        <TextBlock Margin="0 00 0 5" HorizontalAlignment="Center"
                   FontSize="40" FontWeight="Bold"
                   Text="Welcome!"/>
        <TextBlock FontSize="17" FontWeight="SemiBold"
                   HorizontalAlignment="Center"
                   Text="Register a new account"/>

        <TextBox Margin="0 50 0 0" x:Name="txtUsername" Width="300" FontSize="18"
                   BorderThickness="2" BorderBrush="{DynamicResource
MaterialDesignDivider}"
                 Style="{StaticResource MaterialDesignOutlinedTextBox}"
                   materialDesign:HintAssist.Hint="Enter Username"/>

        <PasswordBox Margin="0 20 0 0 " x:Name="txtPassword" Width="300"
FontSize="18"
                     BorderThickness="2" BorderBrush="{DynamicResource
MaterialDesignDivider}"
                     Style="{StaticResource MaterialDesignOutlinedPasswordBox}"
                     materialDesign:HintAssist.Hint="Enter Password"/>

        <PasswordBox Margin="0 20 0 0 " x:Name="txtConPassword" Width="300"
FontSize="18"
                     BorderThickness="2" BorderBrush="{DynamicResource
MaterialDesignDivider}"
                     Style="{StaticResource MaterialDesignOutlinedPasswordBox}"
                     materialDesign:HintAssist.Hint="Re-Enter Password"/>

        <Button Margin="0 20 0 0" x:Name="BtnRegister" Style="{StaticResource
MaterialDesignFlatMidBgButton}"
                materialDesign:ShadowAssist.ShadowDepth="Depth0" Height="53"
Width="300"
                materialDesign:ButtonAssist.CornerRadius="10" FontSize="18"
Content="REGISTER"
                Click="BtnRegister_Click"/>

        <Button Margin="0 20 0 0" x:Name="BtnLogin" Style="{StaticResource
MaterialDesignFlatButton}"
                materialDesign:ShadowAssist.ShadowDepth="Depth0" Height="53"
Width="300"
                materialDesign:ButtonAssist.CornerRadius="10" FontSize="18"
                Content="Already have an account?" Click="BtnLogin_Click"/>

    </StackPanel>

</Window>
```

Register.xaml.cs

```
using System.Data.SqlClient;
using System.Linq;
using System.Windows;


namespace GetPlaylistInfo.MVVM.View
{
    public partial class Register : Window
    {
        public Register()
        {
            InitializeComponent();
        }

        static readonly SqlConnection conn = new(@"Data Source=DESKTOP-6HB3967;Initial Catalog=recommendationSystem;Integrated Security=True"); //creates a new connection to the database

        //when the register button is clicked the following validation checks are carried out
        private void BtnRegister_Click(object sender, RoutedEventArgs e)
        {
            if (txtUsername.Text == "" | txtPassword.Password == "" || txtConPassword.Password == "") //checks that none of the fields are blank
            {
                MessageBox.Show("One or more fields are empty", "Registration Failed", MessageBoxButton.OK, MessageBoxImage.Error); //error message
            }
            else if (txtPassword.Password == txtConPassword.Password) //if password matches then the validation checks are carried out
            {
                bool u = UniqueCheck(txtUsername.Text); //calls to check the username is unique
                bool v = ValidationCheck(txtConPassword.Password); //calls to check the password strength
                if (!(u && v)) //if either of them are false
                {
                    if (!u)
                    {
                        MessageBox.Show("This username already exists, please choose a new one.", "Registration Failed", MessageBoxButton.OK, MessageBoxImage.Error); //error message
                        txtUsername.Clear();
                        txtUsername.Focus();
                    }

                    if (!v)
                    {
                        MessageBox.Show("Your password must be between 8 and 14 characters, include at least one number or special character, and a mixture of upper and lowercase
letters.", "Registration Failed", MessageBoxButton.OK, MessageBoxImage.Error); //error
message
                        txtPassword.Clear();
                        txtConPassword.Clear();

                    }

                }
                else //if both username and password are okay then they are added to the users
table in the database
                {
                    conn.Open();
                    SqlCommand register = new()
                    {
                        Connection = conn,
                        CommandText = "INSERT INTO users VALUES (@username, @password,
@admin)"
                    };
                    //parameterized sql
                    register.Parameters.AddWithValue("@username", txtUsername.Text);
                    register.Parameters.AddWithValue("@password", txtPassword.Password);
                    register.Parameters.AddWithValue("@admin", "0");
                    register.ExecuteNonQuery();
                    conn.Close();

                    txtUsername.Clear();
                    txtPassword.Clear();
                    txtConPassword.Clear();

                    MessageBox.Show("Your account has been successfuly created", "Registration
Success", MessageBoxButton.OK, MessageBoxImage.Information); //tells the user the
account has been created
                    LoginPage(); //sends the user to the login page
                }
            }

            else
            {
                MessageBox.Show("Passwords do not match, please re-enter", "Registration
Failed", MessageBoxButton.OK, MessageBoxImage.Error); //error message
                txtPassword.Clear();
                txtConPassword.Clear();
                txtPassword.Focus();
            }
        }

        private void BtnLogin_Click(object sender, RoutedEventArgs e)
        {
            LoginPage(); //if login button is clicked then call loginpage subroutine
        }

        static bool UniqueCheck(string username) //checks the username doesn't already
exist in the system
        {
            conn.Open();
            SqlCommand unique = new("SELECT * FROM users WHERE username=@username
", conn);
            unique.Parameters.AddWithValue("@username", username); //parameterized sql
            SqlDataReader reader = unique.ExecuteReader();

            if (reader.Read() == true) //if the username is present in the table then true
            {
                conn.Close();
                return false; //therefore not unique and fails
            }
            else
            {
                conn.Close();
                return true; //if it doesn't exist then the username is unique
            }
        }
        static bool ValidationCheck(string password) //checks the password follows the
validation criteria
        {
            int error = 0;
            if (password.Length < 8 || password.Length > 14) error++;
            if (!password.Any(char.IsLower)) error++;
            if (!password.Any(char.IsUpper)) error++;
            if (!password.Any(char.IsPunctuation)) error++;
            if (password.Any(char.IsWhiteSpace)) error++;
            if (!password.Any(char.IsDigit)) error++;

            if (error == 0) return true; //if any error is present then error > 0
            else return false;
        }

        void LoginPage()
        {
            Login login = new();
            login.Show();
            this.Close(); //makes a new login page and then closes this window
        }
    }
}
```

Window Two – Login

The second window is the login form. Here the user can log in to their already made account to create or access recommendations. There are 2 textboxes, 2 buttons, and 2 labels. Once the user clicks BtnLogin the system will check that the username and password match a user in the database, and if so, it will redirect them to the dashboard. If the user would like to create a new account, they can click BtnRegister to be redirected to the registration window.



Login.xaml

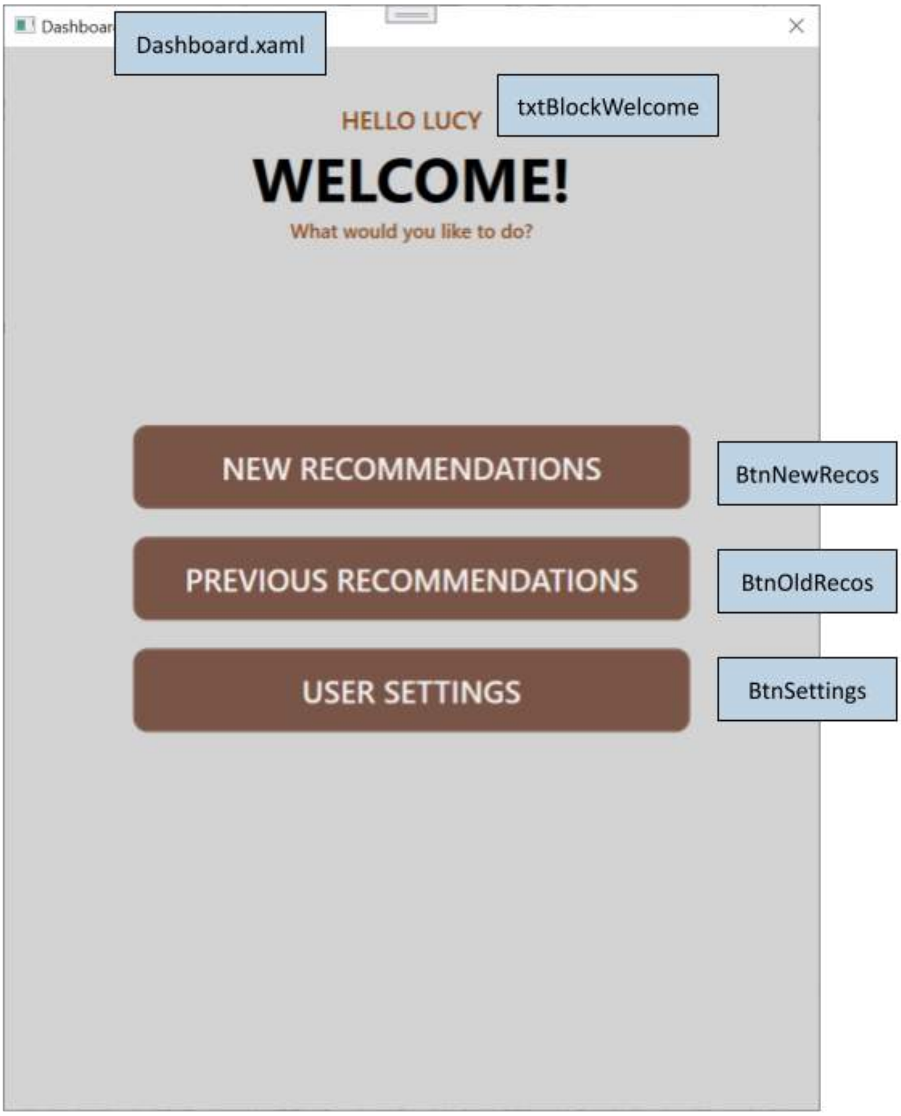
```
<Window x:Class="GetPlaylistInfo.MVVM.View.Login"
         xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:materialDesign="http://materialdesigninxaml.net/winfx/xaml/themes"
        mc:Ignorable="d"
        Background="LightGray"
        WindowStyle="SingleBorderWindow"
        ResizeMode="NoResize"
        WindowStartupLocation="CenterScreen"
        Title="Log In" Height="650" Width="600">
    <StackPanel VerticalAlignment="Center">

        <TextBlock Margin="0 00 0 5" HorizontalAlignment="Center"
                   FontSize="40" FontWeight="Bold"
                   Text="Welcome back!"/>
        <TextBlock FontSize="17" FontWeight="SemiBold"
                   HorizontalAlignment="Center"
                   Text="Enter your account details"/>

        <TextBox Margin="0 50 0 0" x:Name="txtUsername" Width="300" FontSize="18"
                   BorderThickness="2" BorderBrush="{DynamicResource
MaterialDesignDivider}"
                 Style="{StaticResource MaterialDesignOutlinedTextBox}"
                   materialDesign:HintAssist.Hint="Enter Username"/>

        <PasswordBox Margin="0 20 0 0 " x:Name="txtPassword" Width="300"
FontSize="18"
                     BorderThickness="2" BorderBrush="{DynamicResource
MaterialDesignDivider}"
                     Style="{StaticResource MaterialDesignOutlinedPasswordBox}"
                     materialDesign:HintAssist.Hint="Enter Password"/>

        <Button Margin="0 20 0 0" x:Name="btnLogin" Style="{StaticResource
MaterialDesignFlatMidBgButton}"
                materialDesign:ShadowAssist.ShadowDepth="Depth0" Height="53"
Width="300"
                materialDesign:ButtonAssist.CornerRadius="10" FontSize="18"
Content="LOGIN"
                Click="BtnLogin_Click"/>

        <Button Margin="0 20 0 0" x:Name="btnRegister" Style="{StaticResource
MaterialDesignFlatButton}"
                materialDesign:ShadowAssist.ShadowDepth="Depth0" Height="53"
Width="300"
                materialDesign:ButtonAssist.CornerRadius="10" FontSize="18"
                Content="Create an account" Click="BtnRegister_Click"/>

    </StackPanel>
</Window>
```

Login.xaml.cs

```
using System;
using System.Data.SqlClient;
using System.Windows;

namespace GetPlaylistInfo.MVVM.View
{
    public partial class Login : Window
    {
        public static string Username { get; set; } //username variable to be used elsewhere
        public static int UserId { get; set; } //user id from table to be used throughout program
        public Login()
        {
            InitializeComponent();
        }

        static readonly SqlConnection conn = new(@"Data Source=DESKTOP-6HB3967;Initial Catalog=recommendationSystem;Integrated Security=True"); //creates a new connection to the database

        private void BtnLogin_Click(object sender, RoutedEventArgs e)
        {
            Username = txtUsername.Text;
            bool present = CheckUsername(); //username existence
            if (present)
            {
                GetUserId(); //gets the user id for later use
                Dashboard dashboard = new();
                dashboard.Show();
                this.Close(); //opens the dashboard window and closes this window

            }
            else
            {
                MessageBox.Show("Invalid Username or Password, Please Try Again", "Login Failed", MessageBoxButton.OK, MessageBoxImage.Error); //error message
                txtUsername.Clear();
                txtPassword.Clear();
                txtUsername.Focus();
            }
        }

        private void BtnRegister_Click(object sender, RoutedEventArgs e) //if the register button is clicked then it goes back to the registration page
        {
            Register register = new();
            register.Show();
            this.Close();
        }

        private bool CheckUsername()
        {
            try
            {
                bool present = false;
                conn.Open();
                SqlCommand login = new("SELECT * FROM users WHERE 
username=@username and password=@password ", conn); //checks that the username 
and password exist within the table
                login.Parameters.AddWithValue("@password", txtPassword.Password);
                login.Parameters.AddWithValue("@username", txtUsername.Text); 
//parameterized sql
                SqlDataReader reader = login.ExecuteReader();

                if (reader.Read() == true) present = true; //if they exist then present is true
                conn.Close();
                return present;

            }
            catch (Exception ex) //if there are any errors within the connection then this will 
catch it
            {
                MessageBox.Show(ex.Message); //error message
                conn.Close();
                return false; //present is false is there is an error
            }
        }

        private static void GetUserId() //get the user id from the database
        {
            string query = "SELECT * FROM users WHERE username=@username";
            try
            {
                conn.Open();
                using SqlCommand getUserId = new(query, conn);
                getUserId.Parameters.AddWithValue("@username", Login.Username); 
//parameterized sql (Login.Username) is what the user typed in the username box
                UserId = Convert.ToInt32(getUserId.ExecuteScalar()); //gets the user id
                conn.Close();
            }
            catch (Exception ex) //if there is an error this will catch it
            {
                MessageBox.Show(ex.Message); //error message
                conn.Close();
            }
        }
    }
}
```

Window Three – Dashboard

The dashboard is the main hub of the system once a user has logged in. There are 3 buttons and 1 text block. The text block is personalised to greet the user with their username once they log in. BtnNewRecos redirects the user to create a new set of recommendations in the New Recommendations window. BtnOldRecos redirects the user to view previous recommendations in the Old Recommendations window if they have already had recommendations saved. If not, the button does not redirect the user and prompts them to create a new set. BtnSettings redirects the user to the User Settings window.



Dashboard.xaml

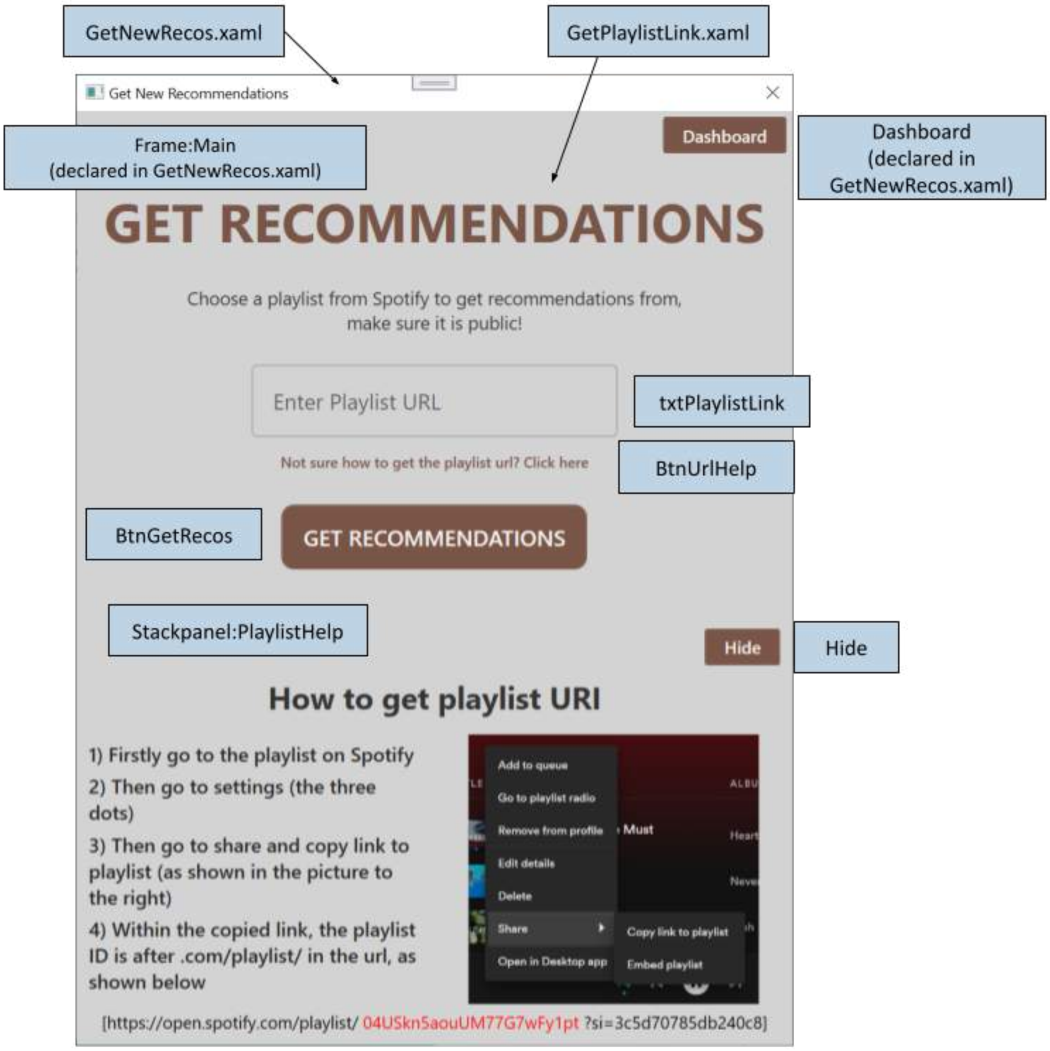
```
<Window x:Class="GetPlaylistInfo.MVVM.View.Dashboard"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:materialDesign="http://materialdesigninxaml.net/winfx/xaml/themes"
        mc:Ignorable="d"
        Background="LightGray"
        WindowStyle="SingleBorderWindow"
        ResizeMode="NoResize"
        WindowStartupLocation="CenterScreen"
        Title="Dashboard" Height="800" Width="600">
    <Grid>
        <StackPanel VerticalAlignment="Top" Margin="20">

            <TextBlock Margin="0 20 0 0" Text="Hello"
                       x:Name="txtBlockWelcome"
                       HorizontalAlignment="Center"
                       Foreground="SaddleBrown"
                       FontSize="18"
                       FontWeight="Medium"/>

            <TextBlock Text="WELCOME!"
                       HorizontalAlignment="Center"
                       FontSize="44"
                       FontWeight="Bold"/>

            <TextBlock Text="What would you like to do?"
                       HorizontalAlignment="Center"
                       Foreground="SaddleBrown"
                       FontSize="14"
                       FontWeight="Medium" />

        </StackPanel>
        <StackPanel Margin="0 0 0 0" Orientation="Vertical" VerticalAlignment="Center">
            <Button Margin="0 0 0 0" x:Name="BtnNewRecos" Style="{StaticResource
MaterialDesignFlatMidBgButton}"
                materialDesign:ShadowAssist.ShadowDepth="Depth0" Height="60"
Width="400"
                materialDesign:ButtonAssist.CornerRadius="10" FontSize="22" Content="NEW
RECOMMENDATIONS"
                Click="BtnNewRecos_Click"/>

            <Button Margin="0 20 0 0" x:Name="BtnOldRecos" Style="{StaticResource
MaterialDesignFlatMidBgButton}"
                materialDesign:ShadowAssist.ShadowDepth="Depth0" Height="60"
Width="400"
                materialDesign:ButtonAssist.CornerRadius="10" FontSize="22"
Content="PREVIOUS RECOMMENDATIONS"
                Click="BtnOldRecos_Click"/>

            <Button Margin="0 20 0 0" x:Name="BtnSettings" Style="{StaticResource
MaterialDesignFlatMidBgButton}"
```

```
                materialDesign:ShadowAssist.ShadowDepth="Depth0" Height="60" Width="400"
                materialDesign:ButtonAssist.CornerRadius="10" FontSize="22" Content="USER SETTINGS"
                Click="BtnSettings_Click"/>
        </StackPanel>




    </Grid>
</Window>
```

Dashboard.xaml.cs

```
using GetPlaylistInfo.MVVM.View.Recommendations;
using GetPlaylistInfo.MVVM.ViewModel;
using System;
using System.Windows;

namespace GetPlaylistInfo.MVVM.View
{
    public partial class Dashboard : Window
    {
        public Dashboard()
        {
            _ = Authorize.GetAccessToken(); //runs the authorisation as the system starts up
            InitializeComponent();
            txtBlockWelcome.Text = ("Hello " + Login.Username).ToUpper(); //uses the username to welcome the user
        }

        private void BtnNewRecos_Click(object sender, RoutedEventArgs e)
        {
            GetNewRecos newRecos = new();
            newRecos.Show();
            this.Close(); //if the new recos button is pressed then a new recos window will open and this window will close
        }

        private void BtnOldRecos_Click(object sender, RoutedEventArgs e)
        {
            bool present = false;
           try
            {
                present = FetchFromTable.CheckRecos(); //goes to the fetchfromtable class and uses the check recos method to see if the user already has recommendations (class code available in window five)

            }
            catch(Exception ex) //exception handling
            {
                MessageBox.Show(ex.Message);
            }

            if (present) //if there is recommendations for the user then the window will open otherwise an error message will show
            {
                PreviousRecommendations previous = new();
                previous.Show();
                this.Close(); //an old recos window will open and this window will close
            }
            else
            {
                MessageBox.Show("You have no recommendations saved! Please create a new set of recommendations.", "No Recommendations", MessageBoxButton.OK, MessageBoxImage.Error);
            }
        }

        private void BtnSettings_Click(object sender, RoutedEventArgs e)
        {
            UserSettings userSettings = new();
            userSettings.Show();
            this.Close(); //if the user settings button is pressed then a user settings window will open and this window will close
        }
    }
}
```

Window Four – New Recommendations



Choose a playlist from Spotify to get recommendations from,
make sure it is public!

1) Firstly go to the playlist on Spotify
2) Then go to settings (the three dots)
3) Then go to share and copy link to playlist (as shown in the picture to the right)
4) Within the copied link, the playlist ID is after .com/playlist/ in the url, as shown below

[https://open.spotify.com/playlist/ 04USkn5aouUM77G7wFy1pt ?si=3c5d70785db240c8]

GetNewRecos.xaml is the main window. It contains the Dashboard button and a frame called Main. If a user clicks on the Dashboard button, it will redirect them to the dashboard. This button is visible throughout all the pages in the window. The frame will contain the other pages that relate to the window. Upon opening the window, the frame will contain the **GetPlaylistLink.xaml** page.

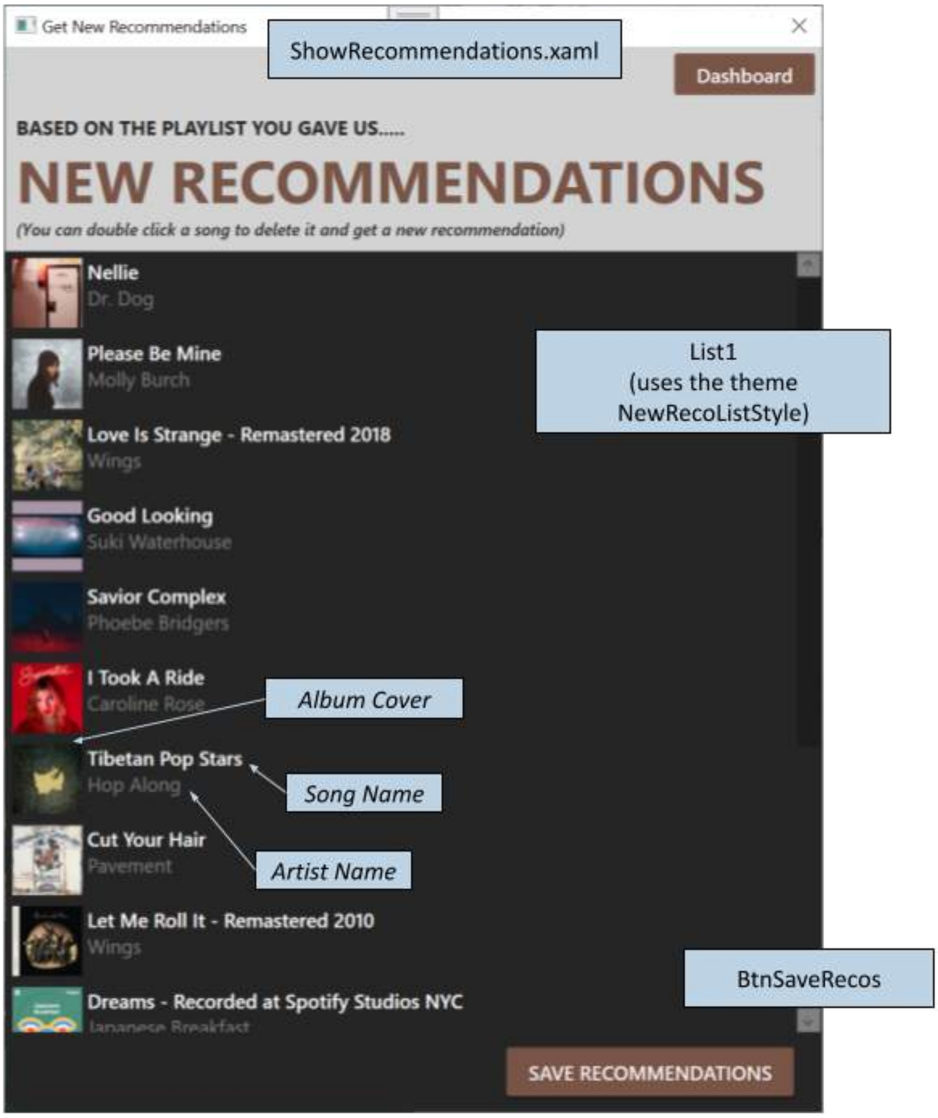
The GetPlaylistLink page contains a textbox, 2 buttons, and a stack panel called PlaylistHelp which contains other elements, including the Hide button. If the user is unsure what the playlist URL should be, if they click BtnUrlHelp, the stack panel will become visible, and it will explain to the user where to find it. If the user clicks the Hide button, then the information disappears. Once the user presses BtnGetRecos, the system will use the input from txtPlaylistLink to fetch information about the playlist from the Spotify API.

Using this information and the factors that have been previously defined (which can be modified in user settings if the user is an admin), ideal values for the recommendations are then calculated. The system then fetches the recommendations from the Spotify API using this data. Providing the recommendation process went smoothly, the frame in

GetNewRecos.xaml then changes its content to the Show Recommendations page, where it will display the first 15 recommendations.

ShowRecommendations.xaml displays the recommendations in a list view called List1, which uses the theme NewRecoListStyle that has designed the elements of each record of the list. Each record shows the album cover, the song name, and the artist's name of the track. The list has a scroll view. If the user would like to replace a recommendation, they can double click the track and they will be prompted if they want to delete the recommendation. If they confirm then the recommendation will be deleted and a new one will be added to the list.

Once the user is happy with the recommendations (or there are no more to replace), the user can click BtnSaveRecos to save the set of recommendations to their user profile. The list of recommendations will then be saved to the database and linked to their unique user ID. The user can then click the Dashboard button to return to the dashboard.



BASED ON THE PLAYLIST YOU GAVE US.....

NEW RECOMMENDATIONS

(You can double click a song to delete it and get a new recommendation)

Nellie
Dr. Dog
Please Be Mine
Molly Burch
Love Is Strange - Remastered 2018
Wings
Good Looking
Suki Waterhouse
Savior Complex
Phoebe Bridgers
I Took A Ride
Caroline Rose
Tibetan Pop Stars
Hop Along
Cut Your Hair
Pavement
Let Me Roll It - Remastered 2010
Wings
Dreams - Recorded at Spotify Studios NYC
Japanese Breakfast

The code for the aforementioned algorithms and displays can be found below:

GetNewRecos.xaml
<Window x:Class="GetPlaylistInfo.MVVM.View.Recommendations.GetNewRecos" xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation" xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" xmlns:d="http://schemas.microsoft.com/expression/blend/2008" xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

```
        xmlns:materialDesign="http://materialdesigninxaml.net/winfx/xaml/themes"
        xmlns:local="clr-namespace:GetPlaylistInfo.MVVM.View.Recommendations"
        mc:Ignorable="d"
         Background="LightGray"
        WindowStyle="SingleBorderWindow"
        ResizeMode="NoResize"
        WindowStartupLocation="CenterScreen"
        Title="Get New Recommendations" Height="800" Width="600">
    <Grid>
        <StackPanel Margin=" 0 5 0 0" Orientation="Horizontal" Height="30"
VerticalAlignment="Top" HorizontalAlignment="Right">

            <Button x:Name="Dashboard" Content="Dashboard" MinWidth="100"
Height="30" Margin="0 0 5 0" Click="Dashboard_Click"
                    Style="{StaticResource MaterialDesignFlatMidBgButton}"
                materialDesign:ShadowAssist.ShadowDepth="Depth0" />

        </StackPanel>
        <Frame  x:Name="Main" Margin="0 35 0 0" NavigationUIVisibility="Hidden"/>
    </Grid>
</Window>
```

GetNewRecos.xaml.cs

```
using System.Windows;

namespace GetPlaylistInfo.MVVM.View.Recommendations
{
    public partial class GetNewRecos : Window
    {
        public GetNewRecos()
        {
            InitializeComponent();
            GetPlaylistLink p1 = new();
            Main.NavigationService.Navigate(p1); //when the new recommendations window
opens then change the main content to the get playlist link page

        }

        private void Dashboard_Click(object sender, RoutedEventArgs e) //return to
dashboard
        {
            var result = MessageBox.Show("Are you sure? This will lose any recommendations
already made.", "Return to Dashboard", MessageBoxButton.YesNo,
MessageBoxImage.Question);
            if (result == MessageBoxResult.Yes)
            {
                Dashboard dashboard = new();
                dashboard.Show();
                this.Close(); //if user confirms they want to return to dashboard then close this
window and open dashboard
            }
        }
    }
}
```

Page One – Get Playlist Link

GetPlaylistLink.xaml

```
<Page x:Class="GetPlaylistInfo.MVVM.View.Recommendations.GetPlaylistLink"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:materialDesign="http://materialdesigninxaml.net/winfx/xaml/themes"
mc:Ignorable="d"
d:DesignHeight="765" d:DesignWidth="600"
Title="GetPlaylistLink">

<DockPanel>
<Grid>
    <StackPanel Margin="0 5 0 5" VerticalAlignment="Top">
        <TextBlock Margin="0 20 0 0" Text="GET RECOMMENDATIONS" FontSize="44"
                   FontWeight="Bold" HorizontalAlignment="Center" Foreground="#795548"/>

        <Label Margin="0 20 0 0" Content="Choose a playlist from Spotify to get
recommendations from," FontSize="15" HorizontalAlignment="Center"/>
        <Label Margin="0 -8 0 0" Content="make sure it is public!" FontSize="15"
HorizontalAlignment="Center"/>


        <TextBox Margin="0 20 0 0" x:Name="txtPlaylistLink" Width="300" FontSize="18"
            BorderThickness="2" BorderBrush="{DynamicResource MaterialDesignDivider}"
            Style="{StaticResource MaterialDesignOutlinedTextBox}"
            materialDesign:HintAssist.Hint="Enter Playlist URL"/>

        <Button Margin="0 5 0 0" x:Name="BtnUrlHelp" Style="{StaticResource
MaterialDesignFlatButton}"
        materialDesign:ShadowAssist.ShadowDepth="Depth0" Height="30" Width="300"
        materialDesign:ButtonAssist.CornerRadius="10" FontSize="12"
        Content="Not sure how to get the playlist url? Click here" Click="BtnUrlHelp_Click"/>
        <Button Margin="0 20 0 20" x:Name="BtnGetRecos" Style="{StaticResource
MaterialDesignFlatMidBgButton}"
        materialDesign:ShadowAssist.ShadowDepth="Depth0" Height="53" Width="250"
        materialDesign:ButtonAssist.CornerRadius="10" FontSize="18" Content="GET 
RECOMMENDATIONS"
        Click="BtnGetRecos_Click"/>

    </StackPanel>

    <StackPanel Visibility="Hidden" x:Name="PlaylistHelp" Margin="0 10 0 0" 
VerticalAlignment="Bottom">

        <Button x:Name="Hide" Content="Hide" MinWidth="50" Height="30" Margin="0 10 
10 5" Click="Hide_Click"
            Style="{StaticResource MaterialDesignFlatMidBgButton}" 
HorizontalAlignment="Right"
        materialDesign:ShadowAssist.ShadowDepth="Depth0" />

        <Label Margin="0 0 0 10" Content="How to get playlist URI" FontSize="25" 
FontWeight="Bold" HorizontalAlignment="Center"/>
        <Grid>
            <Grid.ColumnDefinitions>
                <ColumnDefinition/>
                <ColumnDefinition/>
            </Grid.ColumnDefinitions>
            <Grid.RowDefinitions>
                <RowDefinition Height="220"/>
            </Grid.RowDefinitions>

            <TextBlock Margin="10 0 10 5"

                    TextWrapping="WrapWithOverflow"
                    FontSize="16" FontWeight="Medium" Foreground="#252525">

                <TextBlock Margin="0 5 0 0" TextWrapping="WrapWithOverflow" Text="1) 
Firstly go to the playlist on Spotify"/>
                <TextBlock Margin="0 5 0 0" TextWrapping="WrapWithOverflow" Text="2) 
Then go to settings (the three dots)"/>
                <TextBlock Margin="0 5 0 0" TextWrapping="WrapWithOverflow" Text="3) 
Then go to share and copy link to playlist (as shown in the picture to the right)"/>
                <TextBlock TextWrapping="WrapWithOverflow" Text="4) Within the copied link, 
the playlist ID is after .com/playlist/ in the url, as shown below"
                            Margin="0 5 0 0"/>
            </TextBlock>
            <Image Grid.Column="1"

Source="C:\Users\lcrab\source\repos\GetPlaylistInfo\GetPlaylistInfo\copyLink.PNG"
                        Stretch="Uniform" VerticalAlignment="Center" 
HorizontalAlignment="Center"
                        Margin="0 0 0 0"/>
        </Grid>


        <TextBlock Margin="5 5 5 10" HorizontalAlignment="Center" FontSize="14"
FontWeight="Regular">
            <TextBlock Text="[https://open.spotify.com/playlist/"/>
            <TextBlock Text="04USkn5aouUM77G7wFy1pt" Foreground="Red"/>
            <TextBlock Text="?si=3c5d70785db240c8]"/>
        </TextBlock>
    </StackPanel>
</Grid>
</DockPanel>
</Page>
```

GetPlaylistLink.xaml.cs

```
using System.Windows;
using System.Windows.Controls;

namespace GetPlaylistInfo.MVVM.View.Recommendations
{
    public partial class GetPlaylistLink : Page
    {
        public static string PlaylistUrl { get; set; } //playlist url to be accessed in the other page
        public GetPlaylistLink()
        {
            InitializeComponent();
        }

        private void BtnGetRecos_Click(object sender, RoutedEventArgs e)
        {
            if (txtPlaylistLink.Text == "")
            {
                MessageBox.Show("Please enter a playlist url", "Enter a URL",
MessageBoxButton.OK, MessageBoxImage.None); //error message if nothing is entered in
the box
            }
            else
            {
                PlaylistUrl = txtPlaylistLink.Text; //set playlisturl to user input

                ShowRecommendations p2 = new();
                this.NavigationService.Navigate(p2); //change the content of the main page to
the show recommendations page

            }
        }
        private void Hide_Click(object sender, RoutedEventArgs e)
        {
            PlaylistHelp.Visibility = Visibility.Hidden; //hides the help if the button is clicked
        }

        private void BtnUrlHelp_Click(object sender, RoutedEventArgs e)
        {
            PlaylistHelp.Visibility = Visibility.Visible; //if the user needs help then the help will
be visible

        }
    }
}
```

Page Two – Show Recommendations

ShowRecommendations.xaml

```
<Page x:Class="GetPlaylistInfo.MVVM.View.Recommendations.ShowRecommendations"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
      xmlns:viewmodel="clr-namespace:GetPlaylistInfo.MVVM.ViewModel"
      mc:Ignorable="d"
      d:DesignHeight="765" d:DesignWidth="600"
      Title="ShowRecommendations">
    <Page.DataContext>
        <viewmodel:ModifyRecommendations/> //as the data content is the model
ModifyRecommendations, when the page is loaded, this model will be executed.
    </Page.DataContext>
    <DockPanel>
        <Grid Margin="0 5 0 0" DockPanel.Dock="Top">


            <StackPanel VerticalAlignment="Bottom" Margin="8">
                <TextBlock Text="BASED ON THE PLAYLIST YOU GAVE US....."
                       Foreground="#252525"
                       FontSize="14"
                       FontWeight="Bold"/>

                <TextBlock Text="NEW RECOMMENDATIONS"
                       Foreground="#795548"
                       FontSize="42"
                       FontWeight="Bold"/>

            <TextBlock Text="(You can double click a song to delete it and get a new
recommendation)"
                 Foreground="#252525"
                 FontSize="12"
                 FontWeight="SemiBold"
                   FontStyle="Italic"/>
          </StackPanel>
        </Grid>

        <StackPanel Background="#252525">
          <ListView  Name="List1" Background="#252525" VerticalAlignment="Top"
              MaxHeight="560"
             ScrollViewer.CanContentScroll="False" SelectionMode="Single"
             MouseDoubleClick="ListView_MouseDoubleClick"
             ItemsSource="{Binding UserRecommendations}"
             Style="{StaticResource NewRecoListStyle}" Margin="0 0 0 10"/>


          <Button Margin="20 0 20 10" x:Name="BtnSaveRecos" Content="SAVE
RECOMMENDATIONS"
              Height="35" HorizontalAlignment="Right" Click="BtnSaveRecos_Click" />
        </StackPanel>
    </DockPanel>

</Page>
```

ShowRecommendations.xaml.cs

```
using GetPlaylistInfo.MVVM.ViewModel;
using System;
using System.ComponentModel;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Input;

namespace GetPlaylistInfo.MVVM.View.Recommendations
{
public partial class ShowRecommendations : Page
{
    public ShowRecommendations()
    {
    InitializeComponent();
    }

    private void ListView_MouseDoubleClick(object sender, MouseButtonEventArgs e) //
once the recommendations have loaded, the user can click on one to replace it
```

```
    {
        var song = List1.SelectedItem;
        if (song != null) //checks the user has clicked an actual song and not the side
        {
            string name = ((GetPlaylistInfo.MVVM.Model.RecoTrack)song).Name;
            string artist = ((GetPlaylistInfo.MVVM.Model.RecoTrack)song).Artists[0].Name;
            string songToDelete = ((GetPlaylistInfo.MVVM.Model.RecoTrack)song).Id;



            MessageBoxResult result = MessageBox.Show("Delete '" + name + "' by " + artist +
"?", "Delete Recommendation?", MessageBoxButton.YesNo, MessageBoxImage.Question);
//asks the user to confirm

            if (result == MessageBoxResult.Yes)
            {

                ModifyRecommendations.DeleteTrack(songToDelete); //calls the delete track
method from modify recommendations
                ICollectionView view =
CollectionViewSource.GetDefaultView(ModifyRecommendations.UserRecommendations);
//gets the updated list of the recommendations
                view.Refresh(); //refreshes the view of recommendations for the user so they get
a new recommendation
            }

        }
    }

    private void BtnSaveRecos_Click(object sender, RoutedEventArgs e) //saves the
recommendations to the databsae
    {

        var result = MessageBox.Show("Save to acoount?", "Save Recommendations",
MessageBoxButton.YesNo, MessageBoxImage.Question);
        if (result == MessageBoxResult.Yes)
        {
            try
            {
                _ = new FormatRecoData();
                FormatRecoData.FormatData(); //formats the data ready for the database
                string[] uniqueIds = GetSongsIds(); // calls the get songs ids function
                FormatRecoData.InsertSongs(FormatRecoData.ToPlaylist); //inserts the songs to
the playlist using the formatrecodata class

                int uniquePlaylistId = FormatRecoData.CreateUnqiuePlaylist(); //gets a unique
playlist id
```

```
                FormatRecoData.AddPlaylistSongs(uniqueIds, uniquePlaylistId); //updates the
playlistsongs table for the unique playlist Id
                MessageBox.Show("Successfully saved to account.", "Saved.",
MessageBoxButton.OK, MessageBoxImage.None);

                }
                catch (Exception ex) //exception handling
            {
                throw new ApplicationException("Help: ", ex);
            }

        }

    }

    static string[] GetSongsIds()
    {
        string[] songsIds = new string[FormatRecoData.ToPlaylist.Count];
        int count = 0;
        foreach (var song in FormatRecoData.ToPlaylist) //for each song in the playlist get the
songs ids which is the primary key for the songs table in the database
        {
            songsIds[count] += song.SongsId;
            count++;
        }
        return songsIds;
    }
}
}
```

ModifyRecommendations.cs

ModifyRecommendations.cs

```
using GetPlaylistInfo.MVVM.Model;
using System.Collections.ObjectModel;
using System.Linq;
using System.Windows.Navigation;
using System.Windows;
using GetPlaylistInfo.MVVM.View.Recommendations;

namespace GetPlaylistInfo.MVVM.ViewModel
{
public class ModifyRecommendations
{
    public static Collection<RecoTrack> OriginalRecommendations { get; set; } //the original
list of recommendations
```

```
    public static Collection<RecoTrack> UserRecommendations { get; set; } //the list of 15
recommendations for the user
    public static int Val { get; set; } //position in the recommendations

    public ModifyRecommendations()
    {

        UserRecommendations = new Collection<RecoTrack>(); //new collection with the
reco track format

        _ = new GetPlaylistSongInfo(); //get the current playlist info using the
getplaylistsonginfo class
        if (GetPlaylistSongInfo.Total <= 100) //can only fetch info for playlists under 100
songs
        {
            if (GetPlaylistSongInfo.Total != 0) //if the total = 0 then there is an error
            {
                Initialize(); //if the total is between 0 and 100 then can call the subroutine
            }
            else
            {
                MessageBox.Show("There has been an error, ensure your playlist link is correct.
Please return to dashboard and try again");

            }
        }
        else MessageBox.Show("Please return to dashboard and try again"); //error message
    }

    static void Initialize()
    {
        GetSpotifyRecommendations GetRecos = new(); //then uses the
getspotifyrecommendations class to get recommendations based off the original playlist
        OriginalRecommendations = GetRecos.GetTheSpotifyRecommendations(); //gets the
recommendation info and stores it in the original reccomendations collection

        for (int i = 0; i < 15; i++)
        {
            UserRecommendations.Add(OriginalRecommendations[i]); //add the first 15 songs
in the recommendations to user recommendations
        }
        Val = 15; //position 15 in the original recommendations now
    }

    public static void DeleteTrack(string songId)
    {
        var dupes = UserRecommendations.ToList().Where(x => x.Id == songId).ToList(); //gets the item within the user recommendations

        if (Val < OriginalRecommendations.Count) //as long as there are still more recommendations to suggest then do the following
        {
            foreach (var item in dupes) //formats the item to a recotrack
            {
                UserRecommendations.Remove(item); //remove such time from the user recommendations
            }
            UpdateList(); //calls the update list subroutine

        }
        else
        {
            MessageBox.Show("Sorry, you have run out of recommendations! Please restart the process to get new recommendations.", "No more", MessageBoxButton.OK, MessageBoxImage.Error); //error message
        }

    }

    static void UpdateList()
    {
        UserRecommendations.Add(OriginalRecommendations[Val]); //add the next song to the user recommendations
        Val++;
    }
}
}
```

GetPlaylistInfo.cs

GetPlaylistInfo.cs

```
using GetPlaylistInfo.MVVM.Model;
using GetPlaylistInfo.MVVM.View.Recommendations;
using Newtonsoft.Json;
using RestSharp;
using RestSharp.Authenticators.OAuth2;
using System;
using System.Collections;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Windows;
using System.Windows.Navigation;
```

```
namespace GetPlaylistInfo.MVVM.ViewModel
{
public class GetPlaylistSongInfo
{
    public ObservableCollection<Item> Songs { get; set; } //basic info
    public ObservableCollection<AudioFeature> SongAudio { get; set; } //the audio features
of the songs
    public static ObservableCollection<TrackInfo> SongInformation { get; set; } // the info
used to make recommendatioms
    public static ArrayList TrackIds { get; set; } //track ids
    public static int Total { get; set; } //total number of songs in playlist

    public GetPlaylistSongInfo() //when it is first called this is what is executed
    {
        string accessToken = Authorize.AccessToken; //gets the authorisation token

        Songs = new ObservableCollection<Item>(); //new instance of songs
        SongAudio = new ObservableCollection<AudioFeature>(); //new instance of
songaudio
        SongInformation = new ObservableCollection<TrackInfo>(); //new isntance of
songinformation
        TrackIds = new ArrayList(); //new instance of track ids



        GetPlaylistInformation(accessToken, GetPlaylistLink.PlaylistUrl); //get the playlist info
of the playlisturl

    }

    void GetPlaylistInformation(string accessToken, string playlistId)
    {
        var client = new RestClient
        {
            Authenticator = new
OAuth2AuthorizationRequestHeaderAuthenticator(accessToken, "Bearer")
        };



        string requestLink = "https://api.spotify.com/v1/playlists/" + playlistId + "/tracks";
//creates the link for the playlistid
        var request = new RestRequest(requestLink, Method.Get); //get request for the api
        request.AddHeader("Accept", "application/json");
        request.AddHeader("Content-Type", "application/json");
        bool success = true;

        var check = client.GetAsync(request).GetAwaiter().GetResult();
        if (Convert.ToInt16(check.StatusCode) != 200) //200 is the success code so if not
success then the program doesn't continue
        {
            success = false;
        }

        if (success) //if successful then get the data
        {

            var response = client.GetAsync(request).GetAwaiter().GetResult();
            var data = JsonConvert.DeserializeObject<PlaylistModel>(response.Content!);
//deserialize the object to the format of playlistmodel
            Total = Convert.ToInt16(data.Total); //get the total

            if (Total > 100)
            {
                MessageBox.Show("Playlist can not be longer than 100 songs, please try again.",
"Error", MessageBoxButton.OK, MessageBoxImage.Error); //error message
            }
            else
            {

                for (int i = 0; i < Total; i++) //for each song within the playlist
                {
                    var song = data.Items[i];
                    Songs.Add(song); //add it to the songs collection
                    TrackIds.Add(song.Track.Id); //and add the track ids to the array list

                }
                GetAudioFeaures(accessToken); //calls the subroutine
                CombineData(); //combine the data together
            }

        }
    }

    void GetAudioFeaures(string accessToken) //get the audio features of the songs within
the playlist
    {
        var client = new RestClient("https://api.spotify.com/v1/audio-features")
        {
            Authenticator = new
OAuth2AuthorizationRequestHeaderAuthenticator(accessToken, "Bearer")
        };

        string[] Id = (string[])TrackIds.ToArray(typeof(string));
        string Ids = String.Join(",", Id); //create a string of the ids
        var request = new RestRequest($"?ids={Ids}", Method.Get); //get audio information of
the ids
        request.AddHeader("Content-Type", "application/json");
        var response = client.GetAsync(request).GetAwaiter().GetResult();
        var audio = JsonConvert.DeserializeObject<AudioFeatures>(response.Content!);
//deserialize response content

        for (int i = 0; i < Total; i++)
        {
            var song = audio.Features[i];
            SongAudio.Add(song); //add the audio features to the songaudio
        }
    }

    void CombineData() //create one collection with the information combined
    {
        var infoSongs = CreateSongIndex(Total); //call the createsongindex to get an index
for the songs
        int index = 0;
        foreach (var trackinfo in infoSongs)
        {
            var data = Songs[index];
            var audioData = SongAudio[index];
            trackinfo.TrackId = data.Track.Id;
            trackinfo.SongName = data.Track.Name;
            trackinfo.ArtistName = data.Track.Artists[0].Name;
            trackinfo.AlbumName = data.Track.Album.Name;
            trackinfo.DurationMs = data.Track.DurationMs;
            trackinfo.Popularity = data.Track.Popularity;
            trackinfo.DurationMs = data.Track.DurationMs;

            trackinfo.Acousticness = audioData.Acousticness;
            trackinfo.Danceability = audioData.Danceability;
            trackinfo.Energy = audioData.Energy;
            trackinfo.Instrumentalness = audioData.Instrumentalness;
            trackinfo.Key = audioData.Key;
            trackinfo.Liveness = audioData.Liveness;
            trackinfo.Loudness = audioData.Loudness;
            trackinfo.Mode = audioData.Mode;
            trackinfo.Speechiness = audioData.Speechiness;
            trackinfo.Tempo = audioData.Tempo;
            trackinfo.TimeSignature = audioData.TimeSignature;
            trackinfo.Valence = audioData.Valence;
            SongInformation.Add(trackinfo); //get the information for each song and add it to the song information
            index++;

        }
    }
    public static IEnumerable<TrackInfo> CreateSongIndex(long count)
    {
        for (int i = 0; i < count; i++)
        {
            yield return new TrackInfo { Index = i }; //creates an index of the songs to then use it to make a new collection
        }
    }
}
}
```

GetSpotifyRecommendations.cs

GetSpotifyRecommendations.cs

```
using GetPlaylistInfo.MVVM.Model;
using Newtonsoft.Json;
using RestSharp;
using RestSharp.Authenticators.OAuth2;
using System;
using System.Collections.ObjectModel;
using System.Linq;

namespace GetPlaylistInfo.MVVM.ViewModel
{
public class GetSpotifyRecommendations
{
    RecommendationFeatures Features = new(); //create a new instance of features (the ideal features for recommendations)
    public ObservableCollection<RecoTrack> SpotifyRecommendations { get; set; } //raw collection of spotify recommendation tracks of type recotrack
    public Collection<RecoTrack> Recommendations { get; set; } //modified colelction of recommendations

    public Collection<RecoTrack> GetTheSpotifyRecommendations() //when called it will return a collection of type reco track
    {
        string token = Authorize.AccessToken;
        SpotifyRecommendations = new ObservableCollection<RecoTrack>(); //new instance

        GetIdealValues getIdeal = new(); //new getidealvalues class
        GetIdealValues.CheckValues(); //call check values within the ideal values
        getIdeal.GetValues(); //get the values
        Features = getIdeal.RecommendationFeature; //
        Features.Limit = 10; //set limit to 10

        string[] seedTracks = SplitIds(); //split the track ids to groups of 5 for the api as the api only takes track ids in groups of 5
        foreach (var seeds in seedTracks)
        {
            FetchRecos(token, seeds); //get recommendations

        }
        FormatRecos(); // format the recommendations and then return them
        return Recommendations;
    }

    static string[] SplitIds()
    {
        string[] ids = (string[])GetPlaylistSongInfo.TrackIds.ToArray(typeof(string));
        bool full = true;
        int remainder = 0;
        int groups = ids.Length / 5;
        if (ids.Length % 5 != 0)
        {
            groups++;
            remainder = ids.Length % 5;
            full = false;
        } //get the amount of times recommednations will need to be made

        Random rnd = new();
        string[] randomIds = ids.OrderBy(x => rnd.Next()).ToArray(); //randomize the ids

        string[] seedTracks = new string[groups]; //array with length of groups
        int inc = 0;
        for (int i = 0; i < groups - 1; i++)
        {
            for (int j = 0; j < 5; j++)
            {
                seedTracks[i] += randomIds[j + inc];
                if (j != 4) seedTracks[i] += ',';
            }
            inc += 5;
        }
        if (full) //if the playlist total is divisble by 5 then just add 5 to each group
        {
            for (int j = 0; j < 5; j++)
            {
                seedTracks[groups - 1] += randomIds[j + inc];
                if (j != 4) seedTracks[groups - 1] += ',';
            }
        }
        else
        {
            for (int j = 0; j < remainder; j++) //if not divisble by 5 then add them into groups of
5 with the remainding being added into the last group
            {
                seedTracks[groups - 1] += randomIds[j + inc];
                if (j != remainder - 1) seedTracks[groups - 1] += ',';
            }
        }

        return seedTracks; //return the grouped track ids
    }

    void FetchRecos(string accessToken, string seeds) //how to get the recommendations
    {
        Features.Seed_tracks = seeds;


        var client = new RestClient
        {
            Authenticator = new
OAuth2AuthorizationRequestHeaderAuthenticator(accessToken, "Bearer")
        };

        string requestLink = "https://api.spotify.com/v1/recommendations"; //link for the api
request
        var request = new RestRequest(requestLink, Method.Get);
        request.AddQueryParameter("seed_tracks", Features.Seed_tracks); //parameters for
the request
        request.AddQueryParameter("limit", Features.Limit);


        //if the factors have been included by the user for recommendation then add the
parameter to the request
        if (GetIdealValues.Values[0]) request.AddQueryParameter("target_acousticness",
Features.Target_acousticness);
        if (GetIdealValues.Values[1]) request.AddQueryParameter("target_danceability",
Features.Target_danceability);
        request.AddQueryParameter("target_duration_ms", Features.Target_duration_ms);
        if (GetIdealValues.Values[2]) request.AddQueryParameter("target_energy",
Features.Target_energy);
        if (GetIdealValues.Values[3]) request.AddQueryParameter("target_instrumentalness",
Features.Target_instrumentalness);
        request.AddQueryParameter("target_key", Features.Target_key);
        if (GetIdealValues.Values[4]) request.AddQueryParameter("target_liveness", Features.Target_liveness);
        if (GetIdealValues.Values[5]) request.AddQueryParameter("target_loudness", Features.Target_loudness);
        request.AddQueryParameter("target_popularity", Features.Target_popularity);
        request.AddQueryParameter("target_tempo", Features.Target_tempo);
        request.AddQueryParameter("target_time_signature", Features.Target_time_signature);
        if (GetIdealValues.Values[6]) request.AddQueryParameter("target_valence", Features.Target_valence);



        request.AddHeader("Accept", "application/json");
        request.AddHeader("Content-Type", "application/json");

        var response = client.GetAsync(request).GetAwaiter().GetResult();
        var data = JsonConvert.DeserializeObject<GetRecommendations>(response.Content!); //deserialize the response data

        for (int i = 0; i < Features.Limit; i++)
        {
            var song = data.Tracks[i];
            SpotifyRecommendations.Add(song); //add each recommendation to the collection
        }

    }

    void FormatRecos()
    {
        string[] ids = (string[])GetPlaylistSongInfo.TrackIds.ToArray(typeof(string));

        Recommendations = new Collection<RecoTrack>(SpotifyRecommendations
        .DistinctBy(x => x.Id)
        .ToList()); //remove duplicates from the recommendation tracks

        for (int i = 0; i < GetPlaylistSongInfo.Total; i++)
        {
            string playlistId = ids[i];
            FindDupes(playlistId); //call the find dupes subroutine for each distinct track id
        }
    }

    void FindDupes(string playlistId) //removes any recommendations already in the og playlist
    {
        var dupes = Recommendations.ToList().Where(x => x.Id == playlistId).ToList();
//check if the track id exists within the original playlist given by user

        foreach (var item in dupes)
        {
            Recommendations.Remove(item); //remove item from recommendations
        }
    }
}
}
```

GetIdealValues.cs

GetIdealValues.cs

```
using GetPlaylistInfo.MVVM.Model;
using System;
using System.Data.SqlClient;

namespace GetPlaylistInfo.MVVM.ViewModel
{
public class GetIdealValues
{
    public readonly RecommendationFeatures RecommendationFeature = new(); //class of
the recommendation features that are optional to the process

    public static bool[] Values { get; set; }

    static readonly SqlConnection conn = new(@"Data Source=DESKTOP-6HB3967;Initial
Catalog=recommendationSystem;Integrated Security=True"); //new connection to
database

    public static void CheckValues() //get the current factor boolean values in the table
    {
        Values = new bool[7]; //each value represents whether the factor is turned on for
recommendation
        string[] Factors = { "acousticness", "danceability", "energy", "instrumentalness",
"liveness", "loudness", "valence" };

        string query = "SELECT state FROM Factors WHERE factor = @fvalue";

        conn.Open();

        for (int i = 0; i < Values.Length; i++) //because table indexing starts at one
        {

            using SqlCommand getValues = new(query, conn);
            getValues.Parameters.AddWithValue("@fvalue", Factors[i]); //parameterized sql
            using SqlDataReader reader = getValues.ExecuteReader();
            if (reader.HasRows)
            {
                while (reader.Read())
                {
                    Values[i] = Convert.ToBoolean(reader[0]); //for each factor get the boolean
value in the table
                }
            }
        }
        conn.Close();
    }

    public void GetValues() //get the ideal values based on the playlist information for the
recommendations
    {
        CalculateKey();
        CalculateDurationMs();
        CalculatePopularity();
        CalculateTempo();
        CalculateTimeSignature();

        //the following factors are optional only if the boolean value is on
        if (Values[0]) CalculateAcousticness();
        if (Values[1]) CalculateDanceability();
        if (Values[2]) CalculateEnergy();
        if (Values[3]) CalculateInstrumentalness();
        if (Values[4]) CalculateLiveness();
        if (Values[5]) CalculateLoudness();
        if (Values[6]) CalculateValence();
    }

//for each of the following subroutines it just finds the average value based on the playlist
songs
    void CalculateAcousticness()
    {
        double totalVal = 0;
        int index = 0;
        foreach (var track in GetPlaylistSongInfo.SongInformation)
        {
            totalVal += track.Acousticness;
            index++;
        }
        double average = totalVal / GetPlaylistSongInfo.Total;
        RecommendationFeature.Target_acousticness = average;
    }
    void CalculateDanceability()
```

```
{
    double totalVal = 0;
    int index = 0;
    foreach (var track in GetPlaylistSongInfo.SongInformation)
    {
        totalVal += track.Danceability;
        index++;
    }
    double average = totalVal / GetPlaylistSongInfo.Total;
    RecommendationFeature.Target_danceability = average;
}
void CalculateDurationMs()
{
    int totalVal = 0;
    int index = 0;
    foreach (var track in GetPlaylistSongInfo.SongInformation)
    {
        totalVal += Convert.ToInt32(track.DurationMs);
        index++;
    }
    int average = totalVal / GetPlaylistSongInfo.Total;
    RecommendationFeature.Target_duration_ms = average;
}
void CalculateEnergy()
{
    double totalVal = 0;
    int index = 0;
    foreach (var track in GetPlaylistSongInfo.SongInformation)
    {
        totalVal += track.Energy;
        index++;
    }
    double average = totalVal / GetPlaylistSongInfo.Total;
    RecommendationFeature.Target_energy = average;
}
void CalculateInstrumentalness()
{
    double totalVal = 0;
    int index = 0;
    foreach (var track in GetPlaylistSongInfo.SongInformation)
    {
        totalVal += track.Instrumentalness;
        index++;
    }
    double average = totalVal / GetPlaylistSongInfo.Total;
    RecommendationFeature.Target_instrumentalness = average;
}
```

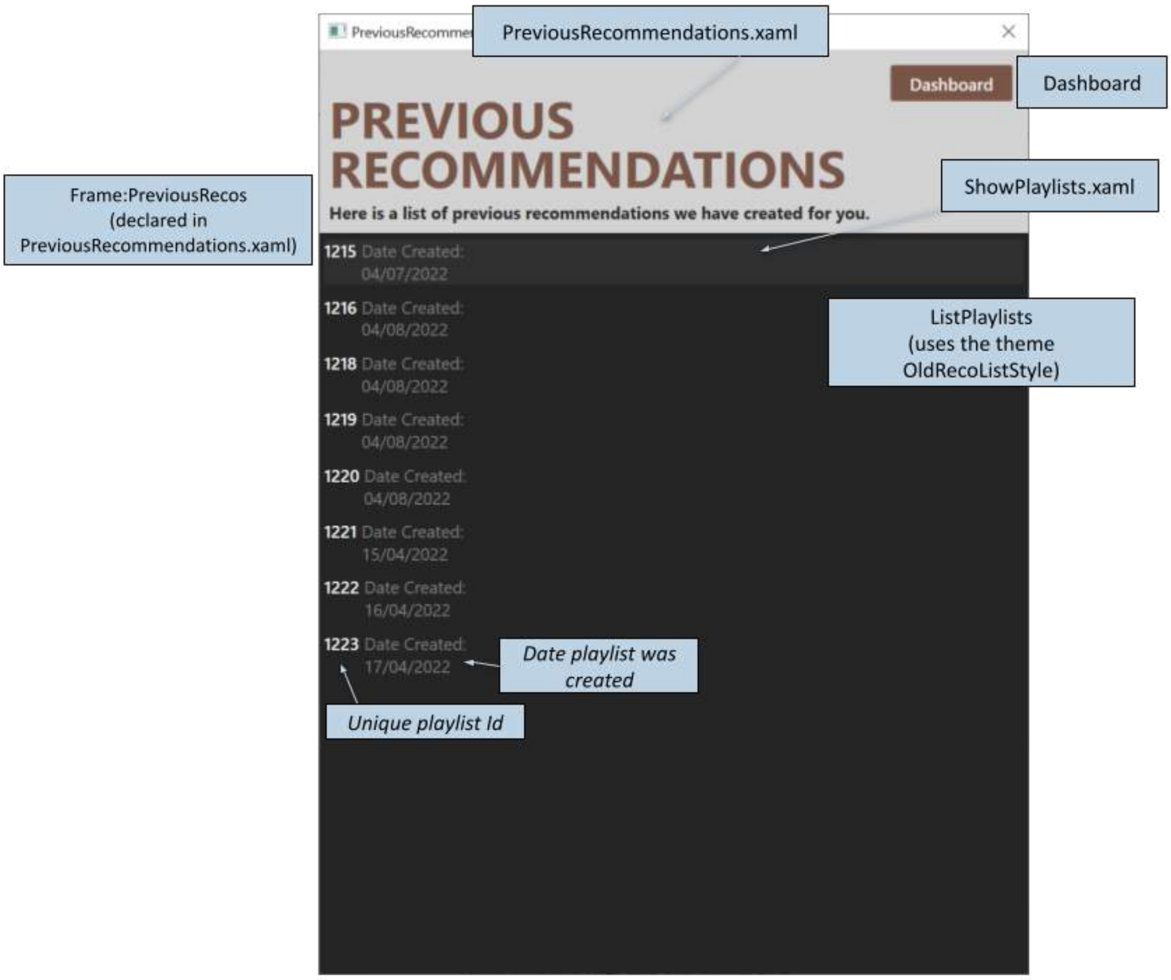
```
void CalculateKey()
{
    int totalVal = 0;
    int index = 0;
    foreach (var track in GetPlaylistSongInfo.SongInformation)
    {
        totalVal += Convert.ToInt32(track.Key);
        index++;
    }
    int average = totalVal / GetPlaylistSongInfo.Total;
    RecommendationFeature.Target_key = average;
}
void CalculateLiveness()
{
    double totalVal = 0;
    int index = 0;
    foreach (var track in GetPlaylistSongInfo.SongInformation)
    {
        totalVal += track.Liveness;
        index++;
    }
    double average = totalVal / GetPlaylistSongInfo.Total;
    RecommendationFeature.Target_liveness = average;
}
void CalculateLoudness()
{
    double totalVal = 0;
    int index = 0;
    foreach (var track in GetPlaylistSongInfo.SongInformation)
    {
        totalVal += track.Loudness;
        index++;
    }
    double average = totalVal / GetPlaylistSongInfo.Total;
    RecommendationFeature.Target_loudness = average;
}
void CalculatePopularity()
{
    int totalVal = 0;
    int index = 0;
    foreach (var track in GetPlaylistSongInfo.SongInformation)
    {
        totalVal += Convert.ToInt32(track.Popularity);
        index++;
    }
    int average = totalVal / GetPlaylistSongInfo.Total;
    RecommendationFeature.Target_popularity = average;
    }
    void CalculateTempo()
    {
        int totalVal = 0;
        int index = 0;
        foreach (var track in GetPlaylistSongInfo.SongInformation)
        {
            totalVal += Convert.ToInt32(track.Tempo);
            index++;
        }
        int average = totalVal / GetPlaylistSongInfo.Total;
        RecommendationFeature.Target_tempo = average;
    }
    void CalculateTimeSignature()
    {
        int totalVal = 0;
        int index = 0;
        foreach (var track in GetPlaylistSongInfo.SongInformation)
        {
            totalVal += Convert.ToInt32(track.TimeSignature);
            index++;
        }
        int average = totalVal / GetPlaylistSongInfo.Total;
        RecommendationFeature.Target_time_signature = average;
    }
    void CalculateValence()
    {
        double totalVal = 0;
        int index = 0;
        foreach (var track in GetPlaylistSongInfo.SongInformation)
        {
            totalVal += track.Valence;
            index++;
        }
        double average = totalVal / GetPlaylistSongInfo.Total;
        RecommendationFeature.Target_valence = average;
    }
}
}
```

FormatRecoData.cs

FormatRecoData.cs

```
using GetPlaylistInfo.MVVM.Model;
using GetPlaylistInfo.MVVM.View;
using System;
using System.Collections.ObjectModel;
using System.Data.SqlClient;
using System.Windows;

namespace GetPlaylistInfo.MVVM.ViewModel
{

public class FormatRecoData
{
    public static SaveToPlaylist FormattedTrack { get; set; } //version of the savetoplaylist
class
    public static Collection<SaveToPlaylist> ToPlaylist { get; set; } //new collection of type
savetoplaylist
    static readonly SqlConnection conn = new(@"Data Source=DESKTOP-6HB3967;Initial
Catalog=recommendationSystem;Integrated Security=True"); //new connection to
database
    /// </summary>


    public static void FormatData()
    {

        ToPlaylist = new Collection<SaveToPlaylist>(); //new instance of collection
        Collection<RecoTrack> ToSave = ModifyRecommendations.UserRecommendations;
//gets the recommendations from the modify recommendations class
        for (int i = 0; i < ToSave.Count; i++) //for each of the recommendations add only the
specific information
        {
            FormattedTrack = new(); //new instance of the class

            //adds the information to the class
            FormattedTrack.Index = i;
            FormattedTrack.Name = ToSave[i].Name;
            FormattedTrack.Artist = ToSave[i].Artists[0].Name;
            FormattedTrack.Album = ToSave[i].Album.Name;
            FormattedTrack.Album_Release_Date = ToSave[i].Album.ReleaseDate;
            FormattedTrack.SongsId = ToSave[i].Id;
            FormattedTrack.DurationMs = ToSave[i].DurationMs;
            FormattedTrack.Popularity = Convert.ToInt32(ToSave[i].Popularity);
            ToPlaylist.Add(FormattedTrack); //adds the object to the collection



        }
    }

    public static void InsertSongs(Collection<SaveToPlaylist> songList) //when the user
wants to save the playlist to the database this is the subroutine that will execute
    {
        string query = "INSERT INTO songs (name, artist, album, album_release_date, songsid,
duration_ms, popularity) VALUES (@name, @artist, @album, @album_release_date,
@songsid, @duration_ms, @popularity)";

        try
        {

            foreach (var song in songList)
            {
                bool present = CheckPresence(song.SongsId); //check that the song isn't already
in the table (stops duplicate data)
                if (!present)
                {
                    conn.Open();
                    using SqlCommand addSong = new(query, conn);
                    //parameterized sql
                    addSong.Parameters.AddWithValue("@name", song.Name);
                    addSong.Parameters.AddWithValue("@artist", song.Artist);
                    addSong.Parameters.AddWithValue("@album", song.Album);
                    addSong.Parameters.AddWithValue("@album_release_date",
song.Album_Release_Date);
                    addSong.Parameters.AddWithValue("@songsid", song.SongsId);
                    addSong.Parameters.AddWithValue("@duration_ms", song.DurationMs);
                    addSong.Parameters.AddWithValue("@popularity", song.Popularity);


                    addSong.ExecuteNonQuery(); //add song to table
                    addSong.Parameters.Clear();
                    conn.Close();
                }
            }
            MessageBox.Show("Successfully saved to account.", "Saved.",
MessageBoxButton.OK, MessageBoxImage.None);

        }
        catch (Exception ex) //exception handling
        {

            MessageBox.Show(ex.Message);
            conn.Close();
        }
    }

    static bool CheckPresence(string id)
    {
        conn.Open();
        SqlCommand check = new("SELECT * FROM songs where songsid=@id", conn);
        check.Parameters.AddWithValue("@id", id); //parameterized sql
        SqlDataReader reader = check.ExecuteReader();

        if (reader.Read() == true) //checks the song is in the table
        {
            conn.Close();
            return true;
        }
        else
        {
            conn.Close();
            return false;
        }
    }

    public static int CreateUnqiuePlaylist()
    {
        string query1 = "INSERT INTO playlists (date_created, usersid) VALUES 
(@date_created, @usersId)"; //first creates a new playlist
        string query2 = "SELECT MAX(playlistsid) FROM playlists"; //then gets the new playlist 
id which is the unique playlist id

        int uniquePlaylistId = 0;


        try
        {
            conn.Open();
            using SqlCommand newPlaylist = new(query1, conn);
            //parameterized sql
            newPlaylist.Parameters.AddWithValue("@date_created", 
DateTime.Now.ToShortDateString());
            newPlaylist.Parameters.AddWithValue("@usersId", Login.UserId);
            newPlaylist.ExecuteNonQuery(); //creates a new playlist
            conn.Close();


        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
            conn.Close();
        }
        try
        {
            conn.Open();
            using SqlCommand getPlaylistId = new(query2, conn);
            uniquePlaylistId = Convert.ToInt32(getPlaylistId.ExecuteScalar()); //gets the value of
the most recent playlist just made
            conn.Close();
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
            conn.Close();
        }

        return uniquePlaylistId; //returns value to show recommendations
    }

    public static void AddPlaylistSongs(string[] uniqueIds, int uniquePlaylistId) //updates the
playlistsongs table
    {
        string query = "INSERT INTO playlistsongs (playlistsid, songsid) VALUES (@playlistsId,
@songsId)"; //for each song create a link between the songid and playlistid
        try
        {
            for (int i = 0; i < uniqueIds.Length; i++)
            {
                conn.Open();
                using SqlCommand insert = new(query, conn);
                //parameterized sql
                insert.Parameters.AddWithValue("@playlistsId", uniquePlaylistId);
                insert.Parameters.AddWithValue("@songsId", uniqueIds[i]);

                insert.ExecuteNonQuery(); //add them to the table
                insert.Parameters.Clear();
                conn.Close();
            }
        }
        catch (Exception ex) //error handling
        {
            MessageBox.Show(ex.Message);
            conn.Close();
        }
    }
}
}
```

Window Five – Previous Recommendations



PreviousRecomme

Dashboard

PREVIOUS RECOMMENDATIONS

Here is a list of previous recommendations we have created for you.

1215	Date Created: 04/07/2022
1216	Date Created: 04/08/2022
1218	Date Created: 04/08/2022
1219	Date Created: 04/08/2022
1220	Date Created: 04/08/2022
1221	Date Created: 15/04/2022
1222	Date Created: 16/04/2022
1223	Date Created: 17/04/2022

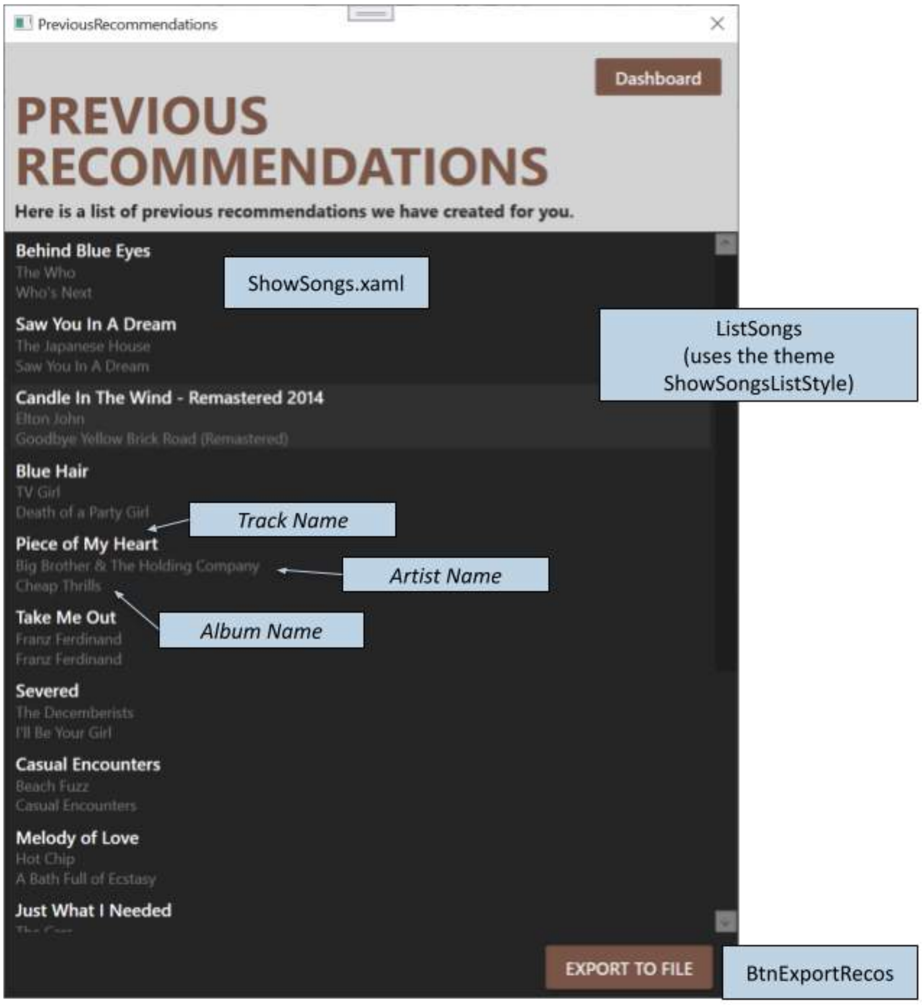
PreviousRecommendations.xaml
Dashboard
ShowPlaylists.xaml
Frame:PreviousRecos (declared in PreviousRecommendations.xaml)
ListPlaylists (uses the theme OldRecoListStyle)
Date playlist was created
Unique playlist Id

PreviousRecommendations.xaml is the main window for this feature of the system. It contains a button called Dashboard that will redirect the user to the dashboard at any point, and a frame called PreviousRecos that contains the content for the window. As the window opens, the content of the frame will be the page **ShowPlaylists.xaml**. Show Playlists contains a list called ListPlaylists. ListPlaylists uses the theme OldRecoListStyle that has been previously defined, and this formats each element of the list. Each element of the list displays the unique playlist id and the date the playlist was created.

When the button in the dashboard is clicked, the system acceses the database and fetches all playlists that are linked with the user's unique user ID, and then stores this information in a temporary collection to display it in the list. While the playlist ID's shown above are consecutive, they are auto-incremented by the database every time any user creates a new playlist.

If the user double-clicks on a playlist, then the list of 15 recommendations will be shown. The system fetches the unique playlist ID of the selected list item and then gets the song information from the database for each song in the playlist. The content of PreviousRecos will be replaced with the **ShowSongs.xaml** page.

ShowSongs.xaml contains a list called ListSongs and 1 button. The list uses a previously defined theme called ShowSongsListStyle and for each element in the list, the track name, artist name, and album name are displayed. If the user would like to export the list of recommendations to a text file, they can click BtnExportRecos which will then create a unique identifier to name the file and store it in a folder stored in the user's local storage.



PREVIOUS RECOMMENDATIONS

Here is a list of previous recommendations we have created for you.

Behind Blue Eyes
The Who
Who's Next

Saw You In A Dream
The Japanese House
Saw You In A Dream

Candle In The Wind - Remastered 2014
Elton John
Goodbye Yellow Brick Road (Remastered)

Blue Hair
TV Girl
Death of a Party Girl

Piece of My Heart
Big Brother & The Holding Company
Cheap Thrills

Take Me Out
Franz Ferdinand
Franz Ferdinand

Severed
The Decemberists
I'll Be Your Girl

Casual Encounters
Beach Fuzz
Casual Encounters

Melody of Love
Hot Chip
A Bath Full of Ecstasy

Just What I Needed

The code for the aforementioned displays can be found below:

PreviousRecommendations.xaml

```
<Window x:Class="GetPlaylistInfo.MVVM.View.PreviousRecommendations"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
       xmlns:materialDesign="http://materialdesigninxaml.net/winfx/xaml/themes"
        mc:Ignorable="d"
        Background="LightGray"
        WindowStyle="SingleBorderWindow"
        ResizeMode="NoResize"
        WindowStartupLocation="CenterScreen"
        Title="PreviousRecommendations" Height="800" Width="600">
    <DockPanel>
        <Grid Margin="0 5 0 0" DockPanel.Dock="Top">

            <StackPanel VerticalAlignment="Bottom" Margin="8">
            <Button x:Name="Dashboard" Content="Dashboard" MinWidth="100"
Height="30" Margin="0 0 5 0" Click="Dashboard_Click"
               Style="{StaticResource MaterialDesignFlatMidBgButton}"
            materialDesign:ShadowAssist.ShadowDepth="Depth0"
HorizontalAlignment="Right" />

            <TextBlock Margin="0 -15 0 0" Text="PREVIOUS"
                 Foreground="#795548"
                 FontSize="42"
                 FontWeight="Bold"/>
            <TextBlock Margin="0 -15 0 0" Text="RECOMMENDATIONS"
                 Foreground="#795548"
                 FontSize="42"
                 FontWeight="Bold"/>

            <TextBlock Text="Here is a list of previous recommendations we have created
for you."
                 Foreground="#252525"
                 FontSize="14"
                 FontWeight="Bold"/>
         </StackPanel>
      </Grid>
      <Frame x:Name="PreviousRecos" Background="#252525"
NavigationUIVisibility="Hidden"/>

   </DockPanel>
</Window>
```

PreviousRecommendations.xaml.cs

```
using GetPlaylistInfo.MVVM.View.PreviousRecos;
using GetPlaylistInfo.MVVM.ViewModel;
using System.Windows;

namespace GetPlaylistInfo.MVVM.View
{
public partial class PreviousRecommendations : Window
{
   public PreviousRecommendations()
   {
      InitializeComponent();
      ShowPlaylists p1 = new(); //creates a new instance of the show playlists page
      PreviousRecos.NavigationService.Navigate(p1); //shows the playlists of the user first

   }

   private void Dashboard_Click(object sender, RoutedEventArgs e)
   {
        Dashboard dashboard = new();
        dashboard.Show();
        this.Hide(); //go back to dashboard if user clicks the button
    }
}
}
```

Page One – Show Playlists

ShowPlaylists.xaml

```
<Page x:Class="GetPlaylistInfo.MVVM.View.PreviousRecos.ShowPlaylists"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
      xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
      xmlns:local="clr-namespace:GetPlaylistInfo.MVVM.View.PreviousRecos"
xmlns:viewmodel="clr-namespace:GetPlaylistInfo.MVVM.ViewModel"
      mc:Ignorable="d"
      d:DesignHeight="630" d:DesignWidth="600"
      Title="ShowPlaylists">
    <Page.DataContext>
        <viewmodel:FetchFromTable/> //when the page is called then fetchfromtable is also
called
    </Page.DataContext>
    <Grid>
        <StackPanel>

            <ListView  Name="ListPlaylists" Background="#252525" VerticalAlignment="Top"
                    MaxHeight="560" MouseDoubleClick="ListPlaylists_MouseDoubleClick"
                  ScrollViewer.CanContentScroll="False" SelectionMode="Single"
                  ItemsSource="{Binding Playlists}" Style="{StaticResource OldRecoListStyle}"
                       Margin="0 0 0 10" d:ItemsSource="{d:SampleData ItemCount=5}"
IsSynchronizedWithCurrentItem="True">
                <ListView.View>
                    <GridView/>
                </ListView.View>
            </ListView>

        </StackPanel>
    </Grid>
</Page>
```

ShowPlaylists.xaml.cs

```
using GetPlaylistInfo.MVVM.ViewModel;
using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Input;

namespace GetPlaylistInfo.MVVM.View.PreviousRecos
{
public partial class ShowPlaylists : Page
{
public bool Success { get; set; } //checks that the page is running as it should

public ShowPlaylists()
{
    Success = false;
    try
    {
        FetchFromTable.GetPlaylistsFromTable(); //get playlists from database table
        Success = true;
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }

    //it gets the playlists before intialization otherwise they won't show on opening
    InitializeComponent();
}

private void ListPlaylists_MouseDoubleClick(object sender, MouseButtonEventArgs e)
{
    if (Success)
    {
        var playlist = ListPlaylists.SelectedItem;
        if (playlist != null) //checks they have actually clicked on a playlist
        {
            int id = ((GetPlaylistInfo.MVVM.Model.TblPlaylists)playlist).PlaylistId;
            try
            {
                bool success1 = FetchFromTable.GetSongsFromPlaylist(id); //gets songs from
the table
                if (success1)
                {
                    try
                    {
                        FetchFromTable.GetSongsInfo(FetchFromTable.SongsIds); //get the
information of the songs within the playlist
                        ShowSongs p2 = new(); //opens the show songs page
                        this.NavigationService.Navigate(p2);

                    }
                    catch (Exception ex) //error handling
                    {
                        MessageBox.Show(ex.Message);
                    }
                }
                else MessageBox.Show("ERROR"); //error handling
            }
            catch (Exception ex) //error handling
            {
                MessageBox.Show(ex.Message);
            }
        }
    }
}
}
}
```

Page Two – Show Songs

ShowSongs.xaml

```
<Page x:Class="GetPlaylistInfo.MVVM.View.PreviousRecos.ShowSongs"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
      xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
      xmlns:local="clr-namespace:GetPlaylistInfo.MVVM.View.PreviousRecos"
xmlns:viewmodel="clr-namespace:GetPlaylistInfo.MVVM.ViewModel"
      mc:Ignorable="d"
      d:DesignHeight="630" d:DesignWidth="600"
      Title="ShowSongs">
    <Page.DataContext>
        <viewmodel:FetchFromTable/> //when the page is called then fetchfromtable is also
called
    </Page.DataContext>
    <Grid>
        <StackPanel>
            <ListView  Name="ListSongs" Background="#252525" VerticalAlignment="Top"
                    MaxHeight="560" ItemsSource="{Binding PlaylistSongs}"
Style="{StaticResource ShowSongsListStyle}"
                  ScrollViewer.CanContentScroll="False" SelectionMode="Single"
```

```
                Margin="0 0 0 10"/>


              <Button Margin="20 0 20 10" x:Name="BtnExportRecos" Content="EXPORT TO
FILE"
                 Height="35" HorizontalAlignment="Right" Click="BtnExportRecos_Click"/>



        </StackPanel>

    </Grid>
</Page>
```

ShowSongs.xaml.cs

```
using GetPlaylistInfo.MVVM.ViewModel;
using System;
using System.IO;
using System.Windows;
using System.Windows.Controls;

namespace GetPlaylistInfo.MVVM.View.PreviousRecos
{
    public partial class ShowSongs : Page
    {
        public ShowSongs()
        {
            InitializeComponent();
        }

        private void BtnExportRecos_Click(object sender, RoutedEventArgs e)
        {
            string guidName = Guid.NewGuid().ToString(); //creates a random identifier for the
file name
            string extension = ".csv";
            string link = (@"C:\Users\lcrab\Downloads\NEA\" + guidName + extension); //file
extension
            string toFile;
            bool success = false;
            try
            {
                using StreamWriter writeToFile = new(link);
                writeToFile.WriteLine("Recommendations for you");
                foreach (var track in FetchFromTable.PlaylistSongs) //for each track add it to the
file
                {
                    toFile = $"{track.Name.Trim()} by {track.Artist.Trim()}, {track.Album.Trim()}
({track.Album_Release_Date}) - Spotify Id: {track.SongsId}"; //writes the information to one
line for each song
                    writeToFile.WriteLine(toFile); //save to file
                    success = true;
                }
                if(success) MessageBox.Show("Success!, Saved in file."); //tells the user the file
has been saved
            }
            catch (Exception ex) //error handling
            {
                MessageBox.Show(ex.Message);
            }
        }
    }
}
```

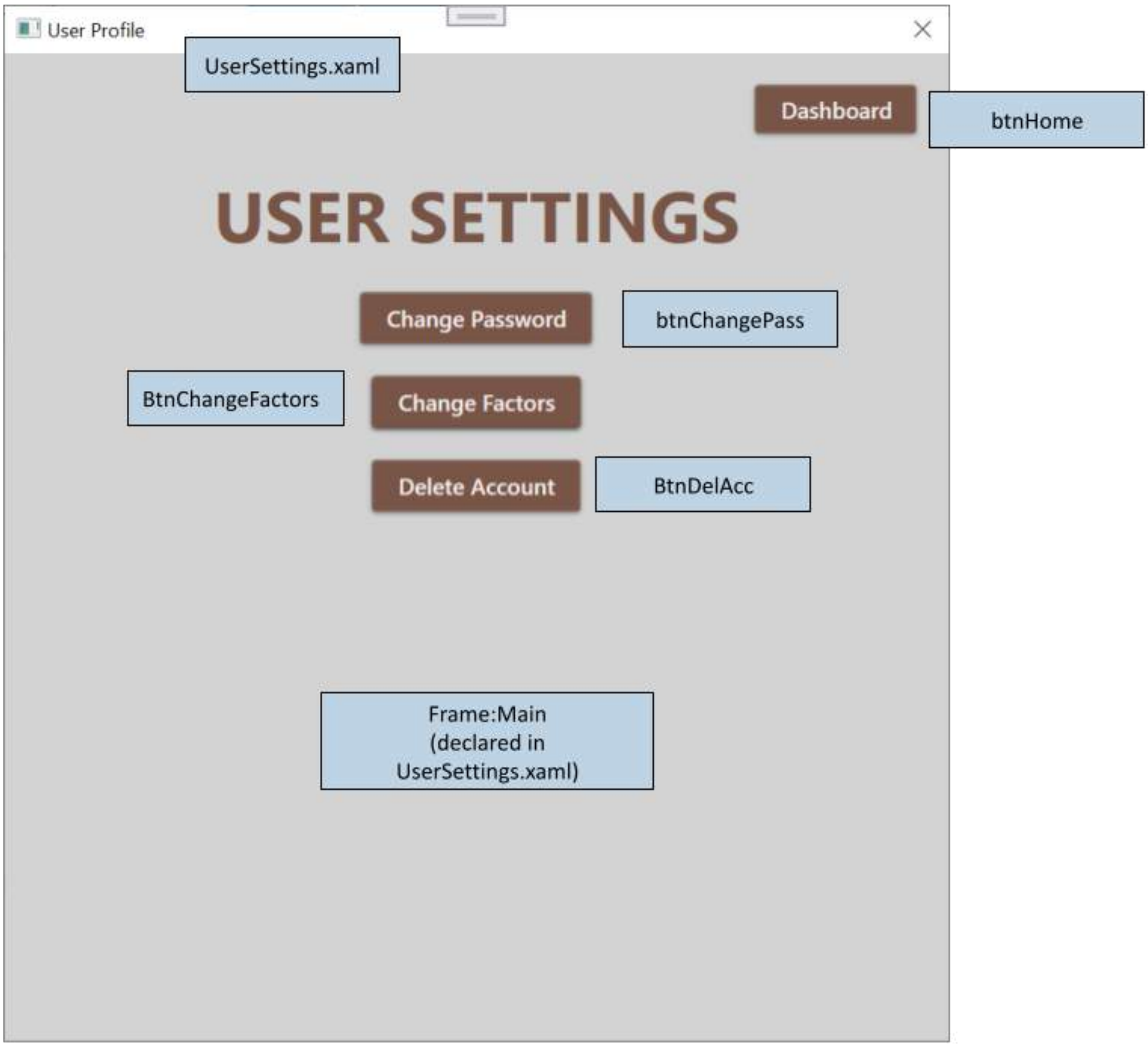
FetchFromTable.cs

FetchFromTable.cs

```
using GetPlaylistInfo.MVVM.Model;
using GetPlaylistInfo.MVVM.View;
using System;
using System.Collections.ObjectModel;
using System.Data.SqlClient;
using System.Windows;

namespace GetPlaylistInfo.MVVM.ViewModel
{
    public class FetchFromTable
{
public static Collection<SaveToPlaylist> PlaylistSongs { get; set; }
public static Collection<TblPlaylists> Playlists { get; set; }
public static SaveToPlaylist[] TablePlaylistSongs { get; set; }
public static TblPlaylists[] TableData { get; set; }
public static string[] SongsIds { get; set; }

static readonly SqlConnection conn = new(@"Data Source=DESKTOP-6HB3967;Initial
Catalog=recommendationSystem;Integrated Security=True"); //new connecction

public static bool CheckRecos()
{
    bool present = false;
    try
    {

        conn.Open();
        SqlCommand getRecos = new("SELECT * FROM playlists WHERE usersid =@usersid ",
conn);
        getRecos.Parameters.AddWithValue("@usersid", Login.UserId); //parameterized sql
        SqlDataReader reader = getRecos.ExecuteReader();
        if (reader.Read() == true) present = true; //user does have recommendations
        conn.Close();
        return present;
    }
    catch (Exception ex) //exception handling
    {
        MessageBox.Show(ex.Message);
        conn.Close();
        return present;
    }

}
public static bool GetSongsFromPlaylist(int playlistid) //get songs where playlistid is what
the user clicked on
{
    SongsIds = new string[15];

    string query = "SELECT songsid from playlistsongs WHERE playlistsid= @playlistID";
    try
    {
        int index = 0;
        conn.Open();
        SqlCommand getSongs = new(query, conn);
        getSongs.Parameters.AddWithValue("@playlistID", playlistid);
        using SqlDataReader reader = getSongs.ExecuteReader();

        while (reader.Read())
        {
            SongsIds[index] = reader["songsid"].ToString()!; //for each song get the songsid
            index++;

        }
        reader.NextResult();
        conn.Close();
        return true;
    }
    catch (Exception ex) //exception handling
    {
        MessageBox.Show(ex.Message);
        conn.Close();
        return false;
    }
```

```
}
public static void GetPlaylistsFromTable()
{

    int size = 0;

    string query1 = "SELECT COUNT(playlistsid) FROM playlists WHERE usersID =@usersid";
//COUNT THE PLAYLISTSIDS
    string query2 = "SELECT * FROM playlists WHERE usersID =@usersid";
    try
    {
        conn.Open();
        SqlCommand countPlaylist = new(query1, conn);
        countPlaylist.Parameters.AddWithValue("@usersid", Login.UserId); //parameterized
sql
        size = Convert.ToInt32(countPlaylist.ExecuteScalar()); //get how many
playlists/recommendations the user has
        conn.Close();
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
        conn.Close();
    }

    TableData = new TblPlaylists[size]; //for each playlist add it to the table (this is what is
shown on the list)
    Playlists = new Collection<TblPlaylists>();

    try
    {
        conn.Open();
        int index = 0;
        SqlCommand getPlaylists = new(query2, conn);
        getPlaylists.Parameters.AddWithValue("@usersid", Login.UserId); //parameterized sql
        using (SqlDataReader reader = getPlaylists.ExecuteReader())
        {
            while (reader.Read())
            {
                TableData[index] = new(); //create a new playlist
                TableData[index].PlaylistId = Convert.ToInt32(reader["playlistsid"]); //playlistsid
                TableData[index].Date_Created =
Convert.ToDateTime(reader["date_created"]).ToShortDateString(); //date to string to
display
                index++;

            }
            reader.NextResult();
        }
        conn.Close();

        foreach (var song in TableData)
        {
            Playlists.Add(song); //add each song to the playlist
        }



    }
    catch (Exception ex) //exception handling
    {
        MessageBox.Show(ex.Message);
        conn.Close();

    }
}

public static void GetSongsInfo(string[] SongsIds) //get the song information for the
specific set of recommendations
{
    TablePlaylistSongs = new SaveToPlaylist[15];
    PlaylistSongs = new Collection<SaveToPlaylist>();

    string query = "SELECT * FROM songs WHERE songsid= @songsid";
    try
    {
        int count = 0;
        foreach (var songid in SongsIds) //for each song get the info from the table
        {
            conn.Open();
            SqlCommand getSongInfo = new(query, conn);
            getSongInfo.Parameters.AddWithValue("@songsid", songid); //parameterized sql
            using (SqlDataReader reader = getSongInfo.ExecuteReader())
            {
                while (reader.Read())
                {
                    //converts to correct data type
                    TablePlaylistSongs[count] = new();
                    TablePlaylistSongs[count].Index = count;
                    TablePlaylistSongs[count].Name = reader["name"].ToString()!;
                    TablePlaylistSongs[count].Artist = reader["artist"].ToString()!;
                    TablePlaylistSongs[count].Album = reader["album"].ToString()!;
                    TablePlaylistSongs[count].Album_Release_Date =
Convert.ToDateTime(reader["album_release_date"]).ToShortDateString();
                    TablePlaylistSongs[count].SongsId = songid;
                    TablePlaylistSongs[count].DurationMs = Convert.ToInt32(reader["duration_ms"]);
                    TablePlaylistSongs[count].Popularity = Convert.ToInt16(reader["popularity"]);
                }
                count++;
                getSongInfo.Parameters.Clear();
            }
            conn.Close();
        }

        foreach (var song in TablePlaylistSongs)
        {
            PlaylistSongs.Add(song); //finalize the collection that will be displayed in show songs
        }
    }
    catch (Exception ex) //error handling
    {
        MessageBox.Show(ex.Message);
        conn.Close();
    }
}
}
}
```

Window Six – User Settings



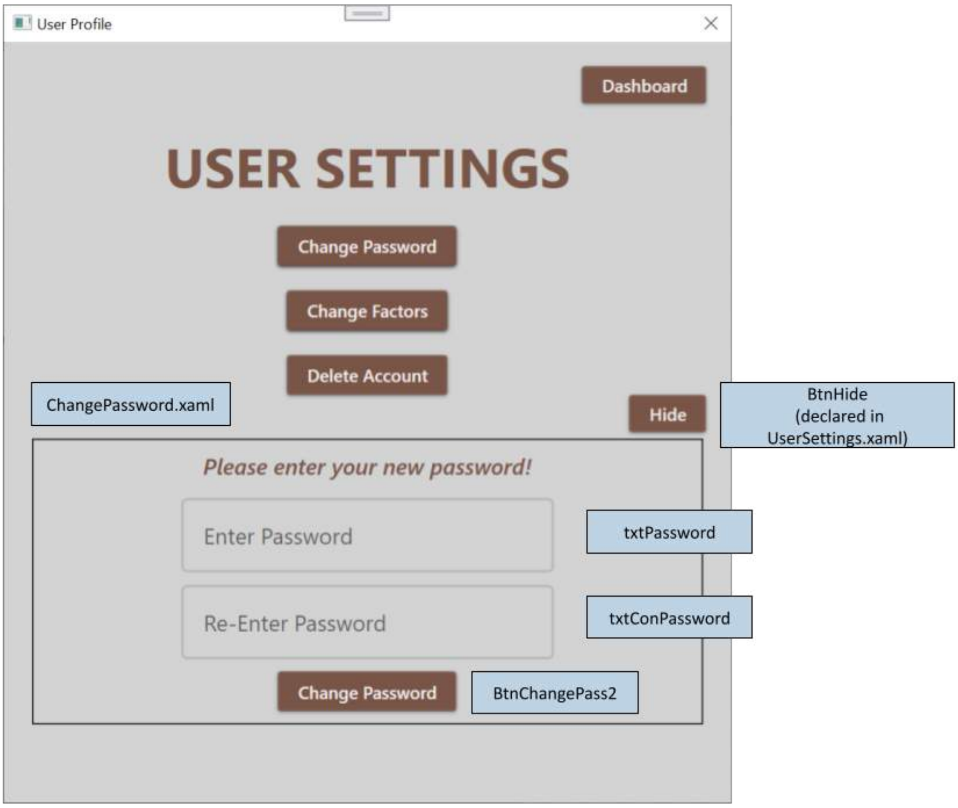
UserSettings.xaml is the main window for the user settings section of the system. It contains 5 buttons and a frame called Main.

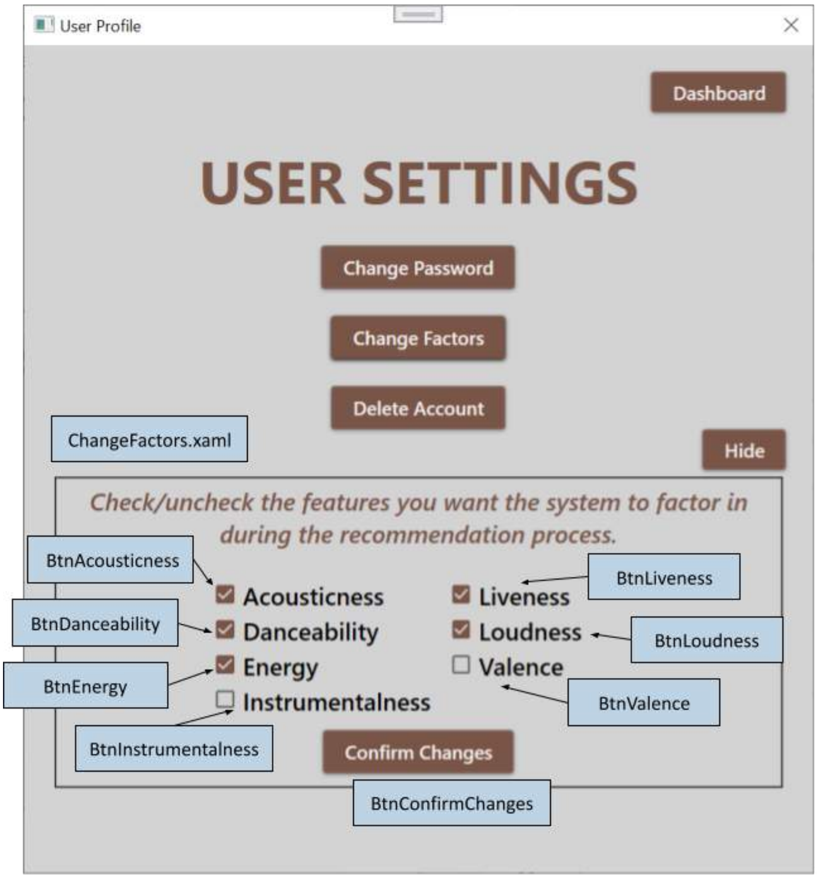
There is a button called BtnHide which is declared in this window but is only visible when the user either clicks the "Change Password" or "Change Factors" button. When a user presses it, it clears the content of the frame. There is an example of the button later.

The dashboard button called btnHome redirects the user back to the dashboard, whereas btnChangePass changes the content of the frame to the page called **ChangePassword.xaml.** If the user clicks BtnChangeFactors, the system will check if the current user is an admin. If the user is then the content of Main will change to the page **ChangeFactors.xaml**, otherwise there will be an error message telling the user that they do not have the permissions to access that certain area of the program.

If BtnDelAcc is pressed, the user will be asked to confirm that they do indeed want to delete their account and all the corresponding data. If they say yes, then the system will delete all user and recommendation data relating to the current user in the database. The program will then redirect the user back to the registration window.

The page **ChangePassword.xaml** contains 2 textboxes and a button. The user is prompted to type in a new password and then confirm the password, to check the user knows what password they are typing in. Once the user clicks BtnChangePass2, the system will use the same validation criteria as the registration page to check whether the password is strong enough. If it isn't an error message will pop up, prompting the user to retry. If it is, the system will update the user's password in the database.





The page **ChangeFactors.xaml** is only accessible to users who are classed as an admin. The page contains 7 checkboxes and 1 button. Each checkbox corresponds to a certain recommendation factor, and each is labelled accordingly. When the page is first shown, the checkboxes will be checked/unchecked according to their status in the database. If they are checked then they are being used in the recommendation process, and vice versa. The user can then check/uncheck each factor and then press BtnConfirmChanges to update the database. If the value of the checkbox is different from its original value, the system will update the database.

UserSettings.xaml

```
<Window x:Class="GetPlaylistInfo.MVVM.View.UserSettings"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        mc:Ignorable="d"
        Background="LightGray"
        WindowStyle="SingleBorderWindow"
        ResizeMode="NoResize"
        WindowStartupLocation="CenterScreen"
        Title="User Profile" Height="650" Width="600">
    <Grid>

        <StackPanel VerticalAlignment="Top" Margin="0">
            <Button Margin="0 20 20 0" VerticalAlignment="Top"
HorizontalAlignment="Right"
                    Height="30" Width="100" x:Name="btnHome" Click="BtnHome_Click"
                    Content="Dashboard"/>

            <StackPanel Margin="0 10 0 0">

                <TextBlock Margin="0 10 0 0" Text="USER SETTINGS" FontSize="44"
                           FontWeight="Bold" HorizontalAlignment="Center"
Foreground="#795548"/>

                <Button Margin ="0 20 0 0" HorizontalAlignment="Center"
VerticalAlignment="Bottom"
                    Content="Change Password" x:Name="btnChangePass"
Click="BtnChangePass_Click"/>

                <Button Margin ="0 20 0 0" HorizontalAlignment="Center"
VerticalAlignment="Bottom"
                    Content="Forgot Password?" x:Name="btnForgotPass"
Click="BtnForgotPass_Click"/>

                <Button Margin ="0 20 0 0" HorizontalAlignment="Center"
VerticalAlignment="Bottom"
                    Content="Change Factors" x:Name="BtnChangeFactors"
Click="BtnChangeFactors_Click"/>

                <Button Margin="0 20 0 00" HorizontalAlignment="Center"
VerticalAlignment="Bottom"
                    Content="Delete Account" x:Name="BtnDelAcc" Click="BtnDelAcc_Click"/>
            </StackPanel>
            <StackPanel>
            <Button Visibility="Hidden" Margin="0 0 20 0" x:Name="BtnHide"
Content="Hide"
                  HorizontalAlignment="Right" Height="30" Click="BtnHide_Click"/>
            <Frame x:Name="Main" Margin="5 0 5 0" NavigationUIVisibility="Hidden"/>


        </StackPanel>
        </StackPanel>
    </Grid>
</Window>
```

UserSettings.xaml.cs

```
using GetPlaylistInfo.MVVM.View.UserSetting;
using GetPlaylistInfo.MVVM.ViewModel;
using System;
using System.Data.SqlClient;
using System.Windows;


namespace GetPlaylistInfo.MVVM.View
{

    public partial class UserSettings : Window
    {
        public static readonly SqlConnection conn = new(@"Data Source=DESKTOP-
6HB3967;Initial Catalog=recommendationSystem;Integrated Security=True"); //new
connection

        public UserSettings()
        {
            InitializeComponent();
        }

        private void BtnDelAcc_Click(object sender, RoutedEventArgs e)
        {
            //check user wants to delete account
            var result = MessageBox.Show("Are you sure you want to delete your account? This
will delete all your account data as well.", "DELETE ACCOUNT", MessageBoxButton.YesNo,
MessageBoxImage.None);
            if (result == MessageBoxResult.Yes) DeleteAccount(); //delete account and all data


        }

        private void BtnChangePass_Click(object sender, RoutedEventArgs e)
        {
            Main.Content = new ChangePassword(); //change the main frame to change password
            Main.Visibility = Visibility.Visible;
            BtnHide.Visibility = Visibility.Visible;
        }

        private void BtnHome_Click(object sender, RoutedEventArgs e)
        {
            Dashboard dashboard = new();
            dashboard.Show();
            this.Close(); //go back to dashboard and close this page
        }

        private void BtnChangeFactors_Click(object sender, RoutedEventArgs e)
        {
            conn.Open();
            SqlCommand checkAdmin = new("SELECT admin FROM users WHERE username=@username", conn); //can only change the factors if they are an admin so this checks that
            checkAdmin.Parameters.AddWithValue("@username", Login.Username); //parameterized sql
            bool admin = (bool)checkAdmin.ExecuteScalar();
            conn.Close();

            if (!admin)
            {
                //error message if not admin
                MessageBox.Show("You are not authorised to do this, contact your administrator", "Not authorised!", MessageBoxButton.OK, MessageBoxImage.Error);

            }
            else
            {
                //otherwise show the factors page
                Main.Content = new ChangeFactors();
                Main.Visibility = Visibility.Visible;
                BtnHide.Visibility = Visibility.Visible;

            }
        }

        private void BtnHide_Click(object sender, RoutedEventArgs e)
        {
            BtnHide.Visibility = Visibility.Hidden;
            Main.Visibility = Visibility.Hidden;
        }
```

```
    void DeleteAccount()
    {
        bool success = false;

        try
        {
            FetchFromTable.GetPlaylistsFromTable(); //have to get all playlists where userid
matches the user to delete them
            success = true;
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message); //exception handling
        }

        //delete from playlistsongs table where the playlistid is one of the users
        string query1 = "DELETE FROM playlistsongs WHERE playlistsid= @playlistID";
        try
        {
            foreach (var playlist in FetchFromTable.Playlists)
            {
                conn.Open();

                SqlCommand deleteSongs = new(query1, conn);
                deleteSongs.Parameters.AddWithValue("@playlistID", playlist.PlaylistId);
//parameterized sql
                deleteSongs.ExecuteNonQuery();
                deleteSongs.Parameters.Clear();
                conn.Close();
            }
        }
        catch (Exception ex) //excception handling
        {
            MessageBox.Show(ex.Message);
            conn.Close();
            success = false;
        }

        //delete playlists belogning to user in playliststable
        string query2 = "DELETE FROM playlists WHERE usersID= @usersid";
        try
        {
            foreach (var playlist in FetchFromTable.Playlists)
            {
                conn.Open();
```

```
                    SqlCommand deleteSongs = new(query2, conn);
                    deleteSongs.Parameters.AddWithValue("@usersid", Login.UserId);
                    deleteSongs.ExecuteNonQuery();
                    deleteSongs.Parameters.Clear();
                    conn.Close();
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
                conn.Close();
                success = false;
            }

            //delete user id from users table
            try
            {
                conn.Open();
                SqlCommand delete = new("DELETE FROM users WHERE 
username=@username", conn);
                delete.Parameters.AddWithValue("@username", Login.Username);
                delete.ExecuteNonQuery();
                conn.Close();
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
                success = false;
            }

            if (success)
            {
                MessageBox.Show("Account successfully deleted.", "Account deletion 
successful", MessageBoxButton.OK, MessageBoxImage.None); //tells user account is 
deleted
                Register register = new();
                register.Show(); //redirects user back to registration page and closes this
                this.Close();
            }
            else
            {
                MessageBox.Show("Error");
            }
        }
    }
}
```

Page One – Change Password

ChangePassword.xaml

```
<Page x:Class="GetPlaylistInfo.MVVM.View.UserSetting.ChangePassword"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:materialDesign="http://materialdesigninxaml.net/winfx/xaml/themes"
      xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
      mc:Ignorable="d" Width="550"
      Title="ChangePassword" Height="240">

    <Grid VerticalAlignment="Center">
        <Border x:Name="blckChangePass" Margin="0 0 0 0" BorderBrush="Black"
BorderThickness="1" Height="230" Width="540">
            <StackPanel HorizontalAlignment="Center">
                <Label Margin="0 5 0 0" Content="Please enter your new password!"
HorizontalAlignment="Center" FontSize="18" FontWeight="SemiBold" FontStyle="Italic"
Foreground="#FF795548"/>
                <PasswordBox Margin="0 10 0 0 " x:Name="txtPassword" Width="300"
FontSize="18"
                     BorderThickness="2" BorderBrush="{DynamicResource
MaterialDesignDivider}"
                     HorizontalAlignment="Center" Style="{StaticResource
MaterialDesignOutlinedPasswordBox}"
                     materialDesign:HintAssist.Hint="Enter Password"/>

                <PasswordBox Margin="0 10 0 0 " x:Name="txtConPassword" Width="300"
FontSize="18"
                     BorderThickness="2" BorderBrush="{DynamicResource
MaterialDesignDivider}"
                     HorizontalAlignment="Center" Style="{StaticResource
MaterialDesignOutlinedPasswordBox}"
                     materialDesign:HintAssist.Hint="Re-Enter Password"/>

                <Button Margin ="0 10 0 10" HorizontalAlignment="Center"
VerticalAlignment="Bottom"
                    Content="Change Password" x:Name="BtnChangePass2"
Click="BtnChangePass2_Click"/>
            </StackPanel>
        </Border>
    </Grid>
</Page>
```

ChangePassword.xaml.cs

```
using System.Data.SqlClient;
using System.Linq;
using System.Windows;
using System.Windows.Controls;

namespace GetPlaylistInfo.MVVM.View.UserSetting
{
/// <summary>
/// Interaction logic for ChangePassword.xaml
/// </summary>
public partial class ChangePassword : Page
{
    public ChangePassword()
    {
        InitializeComponent();
    }

    private void BtnChangePass2_Click(object sender, RoutedEventArgs e)
    {
        if (txtPassword.Password == "" | txtConPassword.Password == "") //checks fields aren't empty
        {
            MessageBox.Show("One or more fields are empty", "Change Password Failed",
MessageBoxButton.OK, MessageBoxImage.Error);

        }
        else if (txtConPassword.Password != txtPassword.Password) //checks passwords match
        {
            MessageBox.Show("Your passwords do not match, try again.", "Passwords don't match",
MessageBoxButton.OK, MessageBoxImage.Error);
            txtPassword.Clear();
            txtConPassword.Clear();
        }
        else if (txtPassword.Password == txtConPassword.Password)
        {
            bool v = ValidationCheck(txtConPassword.Password); //checks password is strong enough
            if (!v)
            {
                //if not error message and user can retry
                MessageBox.Show("Your password must be between 8 and 14 characters, include at least one
number or special character, and a mixture of upper and lowercase letters.", "Registration Failed",
MessageBoxButton.OK, MessageBoxImage.Error);
                txtPassword.Clear();
                txtConPassword.Clear();
                txtPassword.Focus();

            }
            else
            {
                UserSettings.conn.Open();
                SqlCommand updatePass = new("UPDATE users SET password =@password WHERE username
=@username", UserSettings.conn); //else update the users table with the new password
                updatePass.Parameters.AddWithValue("@password", txtConPassword.Password);
                updatePass.Parameters.AddWithValue("@username", Login.Username); //parameterized sql
                updatePass.ExecuteNonQuery();
                UserSettings.conn.Close();

                txtPassword.Clear();
                txtConPassword.Clear();

                MessageBox.Show("Your password has been successfully updated.", "Changed Password",
MessageBoxButton.OK, MessageBoxImage.Information); //tells user the password has been updated
                blckChangePass.Visibility = Visibility.Hidden;

            }

        }
        else
        {
            MessageBox.Show("Unexpected error please try again", "Unexpected error",
MessageBoxButton.OK, MessageBoxImage.None); //error message
            txtPassword.Clear();
            txtConPassword.Clear();
            txtPassword.Focus();

        }

    }

    static bool ValidationCheck(string password)
    {
        //validation criteria
        int error = 0;
        if (password.Length < 8 || password.Length > 14) error++;
        if (!password.Any(char.IsLower)) error++;
        if (!password.Any(char.IsUpper)) error++;
        if (!password.Any(char.IsPunctuation)) error++;
        if (password.Any(char.IsWhiteSpace)) error++;
        if (!password.Any(char.IsDigit)) error++;

        if (error == 0) return true; //if error is more than 0 then it is not strong enough or there is invalid
characters
        else return false;
    }
}
}
```

Page Two – Change Factors

ChangeFactors.xaml

```
<Page x:Class="GetPlaylistInfo.MVVM.View.UserSetting.ChangeFactors"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
   mc:Ignorable="d" Width="550"
   Title="ChangeFactors" Height="240">

  <Grid>
    <Border x:Name="blckChangeFactors" Margin="0 0 0 0" BorderBrush="Black"
BorderThickness="1" Height="230" Width="540">

      <StackPanel Margin="0 0 0 0" HorizontalAlignment="Center">
        <TextBlock Margin="0 5 0 0" HorizontalAlignment="Center" FontSize="18"
FontWeight="SemiBold" FontStyle="Italic" TextWrapping="WrapWithOverflow"
TextAlignment="Center" Text="Check/uncheck the features you want the system to factor
in during the recommendation process." Foreground="#FF795548"/>
        <StackPanel Margin="0 20 0 0" HorizontalAlignment="Center">
          <Grid HorizontalAlignment="Center">
            <Grid.ColumnDefinitions>
              <ColumnDefinition Width="175"/>
              <ColumnDefinition Width="130"/>
            </Grid.ColumnDefinitions>
            <Grid.RowDefinitions>
              <RowDefinition/>
              <RowDefinition/>
              <RowDefinition/>
              <RowDefinition/>
            </Grid.RowDefinitions>
            <CheckBox IsChecked="False" x:Name="BtnAcousticness" FontSize="18"
Content="Acousticness" FontWeight="SemiBold"  FontFamily="Yu Gothic UI"/>
            <CheckBox  Grid.Row="1" x:Name="BtnDanceability" FontSize="18"
Content="Danceability" FontWeight="SemiBold" FontFamily="Yu Gothic UI"/>
            <CheckBox  Grid.Row="2" x:Name="BtnEnergy" FontSize="18"
Content="Energy" FontWeight="SemiBold" FontFamily="Yu Gothic UI"/>
            <CheckBox Grid.Row ="3" x:Name="BtnInstrumentalness" FontSize="18"
Content="Instrumentalness" FontWeight="SemiBold" FontFamily="Yu Gothic UI"/>
            <CheckBox Grid.Column="1" Grid.Row="0" x:Name="BtnLiveness"
FontSize="18" Content="Liveness" FontWeight="SemiBold" FontFamily="Yu Gothic UI"/>
            <CheckBox  Grid.Column="1" Grid.Row="1" x:Name="BtnLoudness"
FontSize="18" Content="Loudness" FontWeight="SemiBold" FontFamily="Yu Gothic UI"/>
            <CheckBox Grid.Column="1" Grid.Row="2" x:Name="BtnValence"
FontSize="18" Content="Valence" FontWeight="SemiBold" FontFamily="Yu Gothic UI"/>
          </Grid>
```

```
            </StackPanel>
            <Button Margin ="0 10 0 0" HorizontalAlignment="Center"
VerticalAlignment="Bottom"
               Content="Confirm Changes" x:Name="BtnConfirmChanges"
Click="BtnConfirmChanges_Click"/>
         </StackPanel>
      </Border>

   </Grid>
</Page>
```

ChangeFactors.xaml.cs

```
using GetPlaylistInfo.MVVM.ViewModel;
using System.Data.SqlClient;
using System.Windows;
using System.Windows.Controls;

namespace GetPlaylistInfo.MVVM.View.UserSetting
{
   // 1 - Acousticness
   // 2 - Danceability
   // 3 - Energy
   // 4 - Instrumentalness
   // 5 - Liveness
   // 6 - Loudness
   // 7 - Valence

   public partial class ChangeFactors : Page
   {
      static readonly SqlConnection conn = new(@"Data Source=DESKTOP-6HB3967;Initial
Catalog=recommendationSystem;Integrated Security=True");

      public ChangeFactors()
      {

         InitializeComponent();
         GetCurrentValues();

      }

      void GetCurrentValues()
      {
         //checks if the values are selected in the table and if so they will be checked when
the user opens the factors
         GetIdealValues.CheckValues();
         if (GetIdealValues.Values[0]) BtnAcousticness.IsChecked = true;
```

```
            if (GetIdealValues.Values[1]) BtnDanceability.IsChecked = true;
            if (GetIdealValues.Values[2]) BtnEnergy.IsChecked = true;
            if (GetIdealValues.Values[3]) BtnInstrumentalness.IsChecked = true;
            if (GetIdealValues.Values[4]) BtnLiveness.IsChecked = true;
            if (GetIdealValues.Values[5]) BtnLoudness.IsChecked = true;
            if (GetIdealValues.Values[6]) BtnValence.IsChecked = true;

        }
        private void BtnConfirmChanges_Click(object sender, RoutedEventArgs e)
        {
            //once the user confirms changes, if the value in the table doesn't match the state
of the checkbox then it will update the table to match the checkbox
            if (BtnAcousticness.IsChecked != GetIdealValues.Values[0])
            {
                conn.Open();
                SqlCommand updateState = new("UPDATE Factors SET state = @state WHERE
factor = @factor", conn);
                updateState.Parameters.AddWithValue("@state", BtnAcousticness.IsChecked);
                updateState.Parameters.AddWithValue("@factor", "acousticness");
                updateState.ExecuteNonQuery();
                conn.Close();
            }
            if (BtnDanceability.IsChecked != GetIdealValues.Values[1])
            {
                conn.Open();
                SqlCommand updateState = new("UPDATE Factors SET state = @state WHERE
factor = @factor", conn);
                updateState.Parameters.AddWithValue("@state", BtnDanceability.IsChecked);
                updateState.Parameters.AddWithValue("@factor", "danceability");
                updateState.ExecuteNonQuery();
                conn.Close();
            }
            if (BtnEnergy.IsChecked != GetIdealValues.Values[2])
            {
                conn.Open();
                SqlCommand updateState = new("UPDATE Factors SET state = @state WHERE
factor = @factor", conn);
                updateState.Parameters.AddWithValue("@state", BtnEnergy.IsChecked);
                updateState.Parameters.AddWithValue("@factor", "energy");
                updateState.ExecuteNonQuery();
                conn.Close();
            }
            if (BtnInstrumentalness.IsChecked != GetIdealValues.Values[3])
            {
                conn.Open();
                SqlCommand updateState = new("UPDATE Factors SET state = @state WHERE
factor = @factor", conn);
```

```
                updateState.Parameters.AddWithValue("@state",
BtnInstrumentalness.IsChecked);
                updateState.Parameters.AddWithValue("@factor", "instrumentalness");
                updateState.ExecuteNonQuery();
                conn.Close();
            }
            if (BtnLiveness.IsChecked != GetIdealValues.Values[4])
            {
                conn.Open();
                SqlCommand updateState = new("UPDATE Factors SET state = @state WHERE
factor = @factor", conn);
                updateState.Parameters.AddWithValue("@state", BtnLiveness.IsChecked);
                updateState.Parameters.AddWithValue("@factor", "liveness");
                updateState.ExecuteNonQuery();
                conn.Close();
            }
            if (BtnLoudness.IsChecked != GetIdealValues.Values[5])
            {
                conn.Open();
                SqlCommand updateState = new("UPDATE Factors SET state = @state WHERE
factor = @factor", conn);
                updateState.Parameters.AddWithValue("@state", BtnLoudness.IsChecked);
                updateState.Parameters.AddWithValue("@factor", "loudness");
                updateState.ExecuteNonQuery();
                conn.Close();
            }
            if (BtnValence.IsChecked != GetIdealValues.Values[6])
            {
                conn.Open();
                SqlCommand updateState = new("UPDATE Factors SET state = @state WHERE
factor = @factor", conn);
                updateState.Parameters.AddWithValue("@state", BtnValence.IsChecked);
                updateState.Parameters.AddWithValue("@factor", "valence");
                updateState.ExecuteNonQuery();
                conn.Close();
            }
            MessageBox.Show("The recommendation factors have successfully been updated.",
"Updated Factors", MessageBoxButton.OK, MessageBoxImage.Information); //success
        }
    }
}
```

Testing

Navigation Testing

The following table tests the unidirectional navigation between windows which link to each other. Grey cells mean that navigation is impossible, and blue cells mean that the windows/pages aren't linked.

The key for the windows is as follows:
A – Register.xaml
B – Login.xaml
C – Dashboard.xaml
D – GetNewRecos.xaml / GetPlaylistLink.xaml
E – ShowRecommendations.xaml
F – PreviousRecommendations.xaml / ShowPlaylists.xaml
G – ShowSongs.xaml
H – UserSettings.xaml
I – ChangePassword.xaml
J – ChangeFactors.xaml

[D and F contain two different windows/pages because even though GetNewRecos and PreviousRecommendations are windows, they do not exist on their own or hold content that serves a significant purpose to the user. The first instance of them contains the pages GetPlaylistLink or ShowPlaylists, so for the purposes of testing they will be classed together.]

✓ - Navigation successful
* - Navigation dependent on user deleting account
! - Within the window so accessible at all times

Navigating from: → **Navigating to: ↓**	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**
A	▨	✓	🟦	🟦	🟦	🟦	🟦	*	🟦	🟦
B	✓	▨	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦
C	🟦		▨	✓	✓	✓	✓	✓	✓	✓
D	🟦	🟦	✓	▨	🟦	🟦	🟦	🟦	🟦	🟦
E	🟦	🟦	🟦	✓	▨	🟦	🟦	🟦	🟦	🟦
F	🟦	🟦	✓	🟦	🟦	▨	🟦	🟦	🟦	🟦
G	🟦	🟦	🟦	🟦	🟦	✓	▨	🟦	🟦	🟦
H	🟦	🟦	✓	🟦	🟦	🟦	🟦	▨	✓	✓
I	🟦	🟦	🟦	🟦	🟦	🟦	🟦	✓	▨	!
J	🟦	🟦	🟦	🟦	🟦	🟦	🟦	✓	!	▨

Input and Output Testing

Test Id	Expected results	Actual results	Comments/ Corrections
01	Message confirming user account has been created User details added to database User can now log in	As expected.[01]	
02	Error message saying one or more fields is blank No user details added to database	As expected.[02]	
03	Error message saying password isn't secure enough No user details added to database	As expected.[03]	
04	Error message saying username already exists No user details added to database	The correct error message was displayed as well as the password strength error message afterwards.[04]	
05	The dashboard will be displayed	As expected.[05]	
06	Error message saying that one or more fields are blank	Error message saying invalid username or password[06]	Turns out I did not check for blank fields in log in, however the function still works.
07	If the username exists in the database, it should log in and display the dashboard without being concerned about case	As expected.[07]	Usernames are not case-sensitive
08	Get Playlist Link page should be displayed	As expected.	*No proof required*

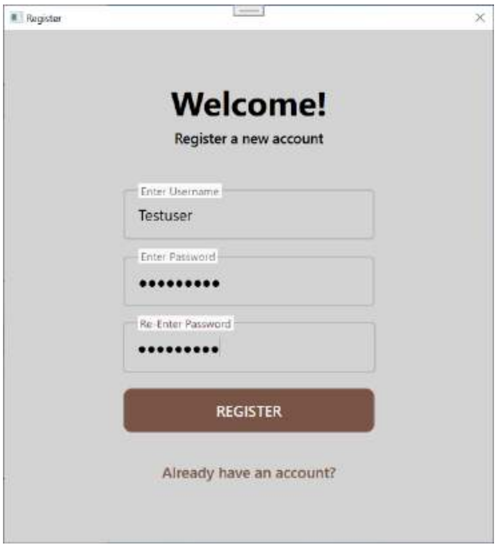
09	Playlist Songs page should be displayed IF user has previous recommendations	As expected.	*No proof required*
10	User Settings should be displayed	As expected.	*No proof required*
11	Page display should change to 15 songs in a list view	As expected.[11]	
12	Error message saying a playlist mustn't have more than 100 songs No recommendations are made	As expected.[12]	
13	Error message saying the field is blank No recommendation is made	As expected.[13]	
14	Error message saying that an error occurred and for the user to try again No recommendation is made	As expected.[14]	
15	Should be a display of 15 songs with the album cover, song name, album name, and artist name Should be able to scroll up and down	As expected.[15]	
16	Each recommendation should be unique and none of them should be in the playlist given by the user	As expected.	Clear in other screenshots there are no duplications
17	User is prompted to confirm they want to delete the recommendation Recommendation is deleted from the list	As expected.[17]	

	and is replaced by a new recommendation		
18	Error message saying that there are no more recommendations that can be used to replace User must save the playlist or go back and try again	User can first try to delete the song, but then the error message comes up and the song cannot be deleted.[18]	Still works as expected
19	No response from the program	As expected.	*No proof required*
19a	Message confirming user wants to return then shows dashboard	As expected.[19a]	
20	Success message confirming the recommendations have been saved to the system Recommendations added to the database	As expected.[20]	
21	Each song should be unique in the database, no track id should be the same	As expected.[21]	Screenshot is a section of the table, there are no duplications.
22	Playlists contains a new unique playlist id that links to the user id Playlistsongs contains 15 fields linking the playlist id with the 15 song ids	As expected.[22]	
23	Error message saying the user hasn't made any recommendations The window isn't accessible	As expected.[23]	
24	List view of all the playlists linked to user, with the playlist id and the date they were created	As expected.[24]	
25	No response from program	As expected. Did not crash.	*No proof required.*

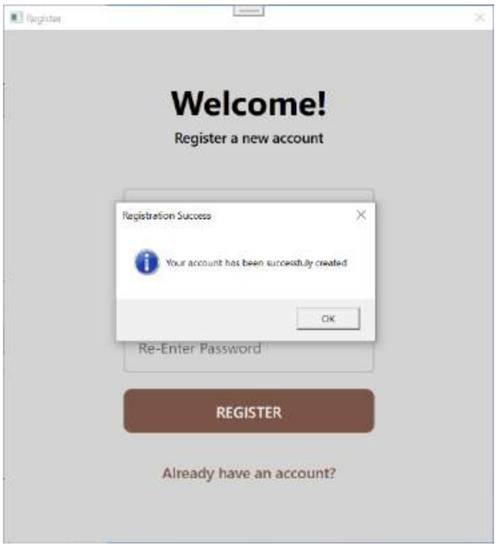
26	List view of 15 songs that relate to the playlist the user clicked on Should display the song name, artist name, and album name	As expected.[26]	
27	Success message saying file has been exported New unique file name in NEA folder in local storage with the correct format	As expected.[27]	
28	Success message saying the password has successfully been updated User password updated in the database	As expected.[28]	
29	Error message saying user cannot change factors IF they are not an admin If they are an admin then the pop-up box will show	As expected.[29]	
30	Success message saying factors have successfully been updated Factors state updated in the database	As expected.[30]	
31	Success message saying user account has successfully been deleted Information in playlists, playlistsongs, and users table relating to the user will be removed User is redirected to the registration page	As expected.[31]	Program sends success message and redirects user to registration page as expected (not shown in screenshots).

Screenshots

01



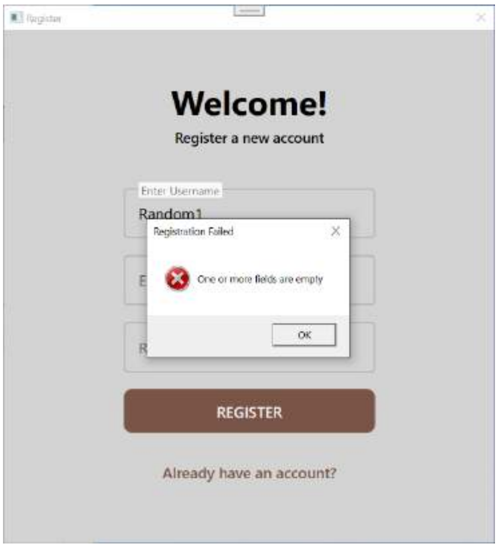
Enter Username
Testuser
Enter Password
•••••••••
Re-Enter Password
•••••••••



Registration Success

Your account has been successfully created

02

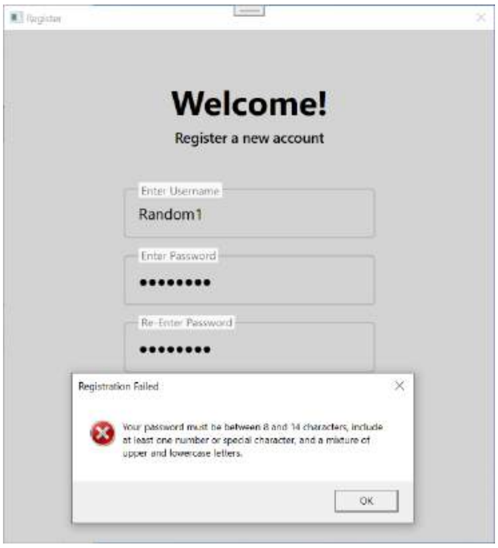


Enter Username
Random1

Registration Failed

One or more fields are empty

03

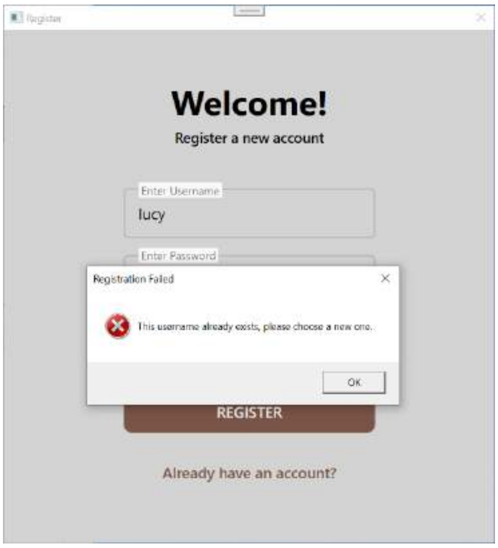


Enter Username
Random1
Enter Password
••••••••
Re-Enter Password
••••••••

Registration Failed

Your password must be between 8 and 14 characters, include at least one number or special character, and a mixture of upper and lowercase letters.

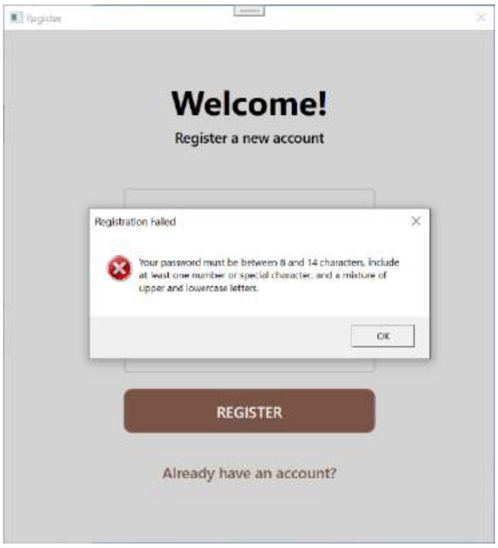
04



Enter Username: lucy

Registration Failed

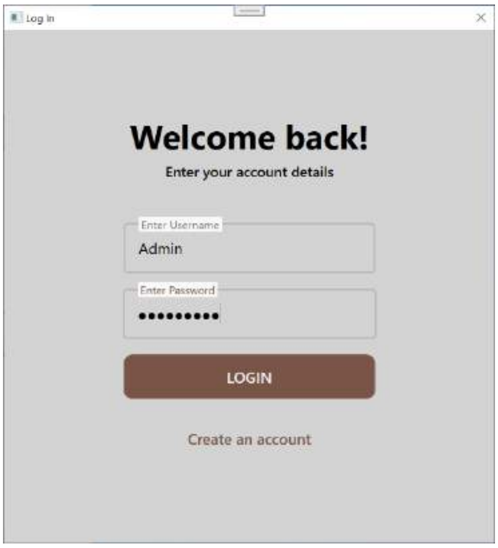
This username already exists, please choose a new one.



Registration Failed

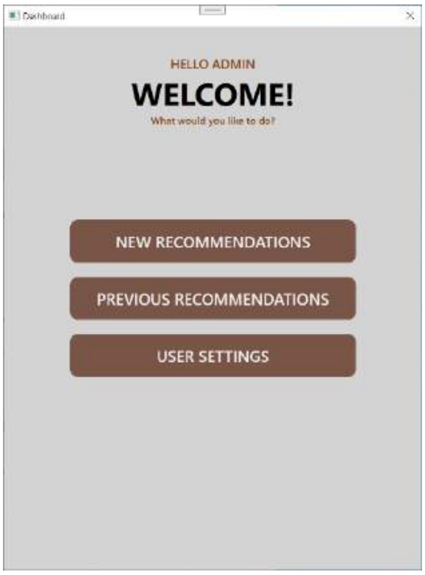
Your password must be between 8 and 14 characters, include at least one number or special character, and a mixture of upper and lowercase letters.

05

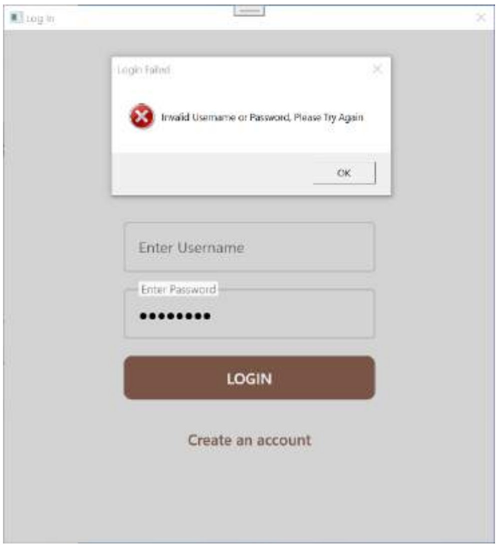


Enter Username: Admin

Enter Password: •••••••••



06

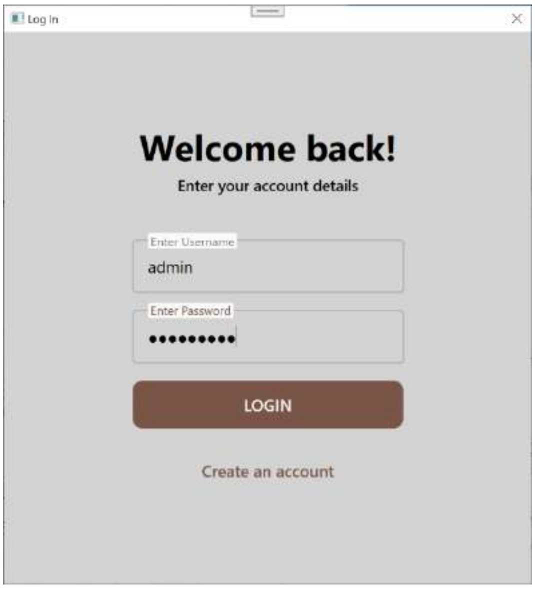


Login Failed

Invalid Username or Password, Please Try Again

Enter Password: ••••••••

07


Log In

Welcome back!

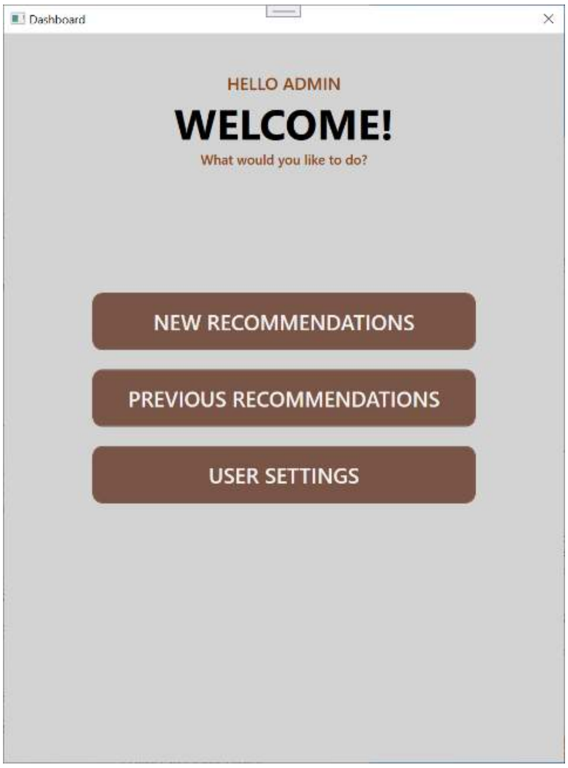
Enter your account details

Enter Username: admin

Enter Password: ••••••••••

LOGIN

Create an account


Dashboard

HELLO ADMIN

WELCOME!

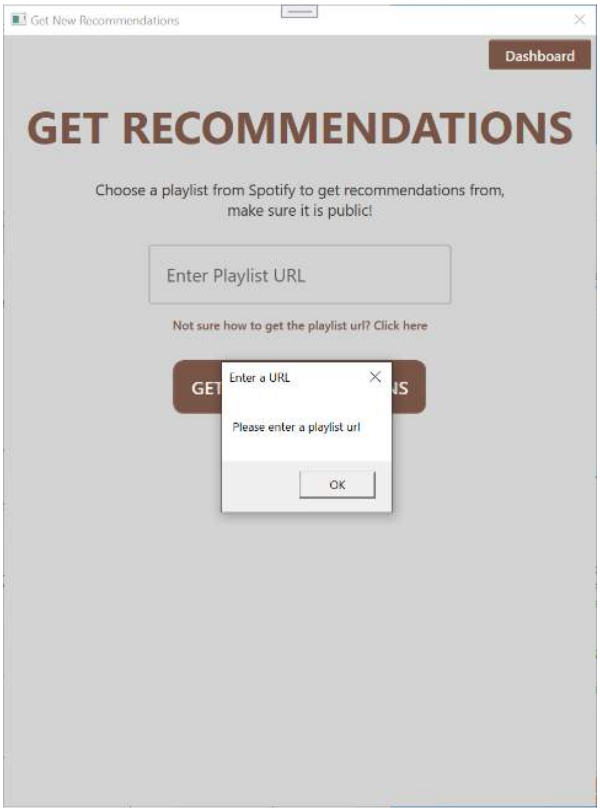
What would you like to do?

NEW RECOMMENDATIONS

PREVIOUS RECOMMENDATIONS

USER SETTINGS

11


Get New Recommendations

Dashboard

GET RECOMMENDATIONS

Choose a playlist from Spotify to get recommendations from, make sure it is public!

Enter Playlist URL

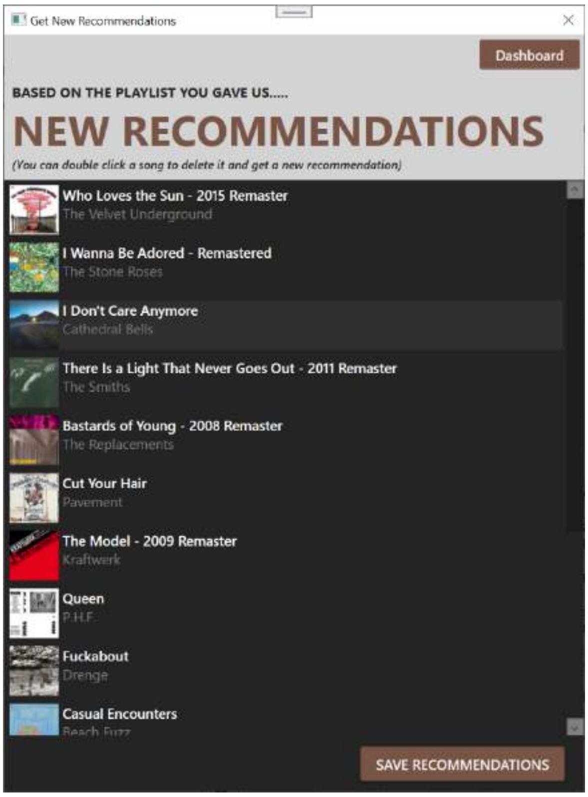
Not sure how to get the playlist url? Click here

GET RECOMMENDATIONS

Enter a URL

Please enter a playlist url

OK


Get New Recommendations

Dashboard

BASED ON THE PLAYLIST YOU GAVE US.....

NEW RECOMMENDATIONS

(You can double click a song to delete it and get a new recommendation)

Who Loves the Sun - 2015 Remaster
The Velvet Underground

I Wanna Be Adored - Remastered
The Stone Roses

I Don't Care Anymore
Cathedral Bells

There Is a Light That Never Goes Out - 2011 Remaster
The Smiths

Bastards of Young - 2008 Remaster
The Replacements

Cut Your Hair
Pavement

The Model - 2009 Remaster
Kraftwerk

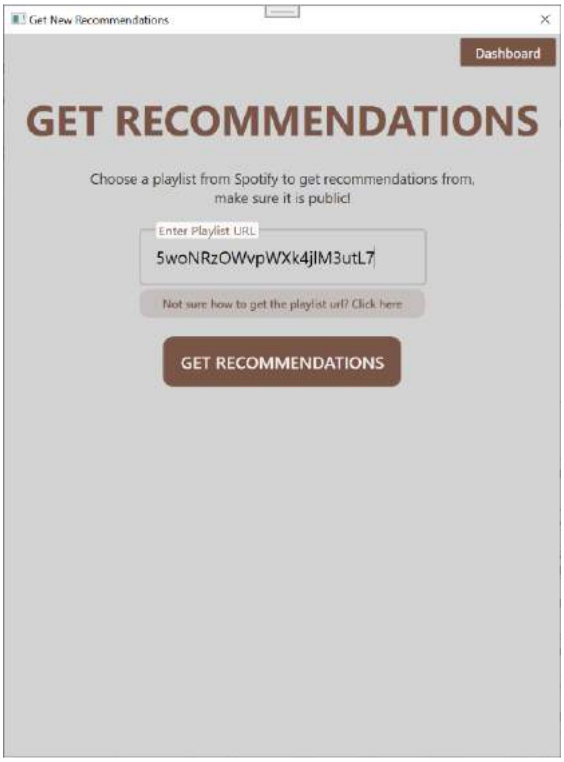
Queen
P.H.F.

Fuckabout
Drenge

Casual Encounters
Beach Fuzz

SAVE RECOMMENDATIONS

12

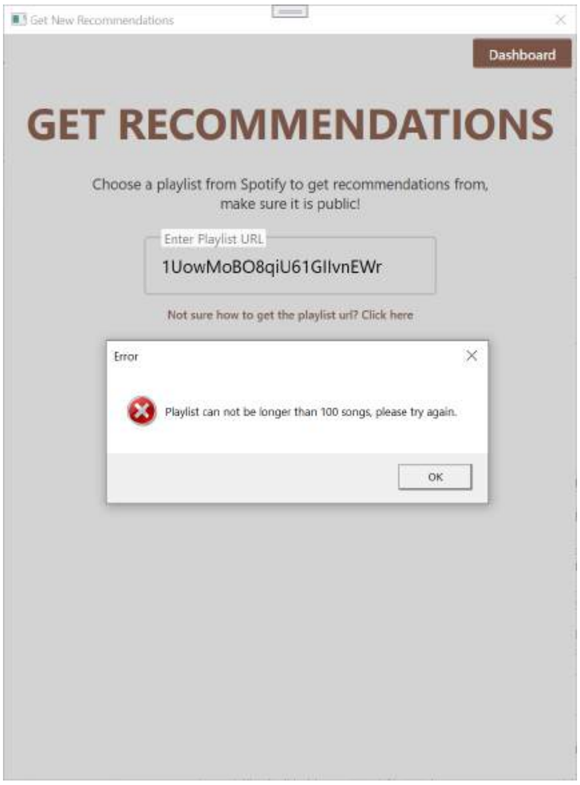


GET RECOMMENDATIONS

Choose a playlist from Spotify to get recommendations from, make sure it is public!

Enter Playlist URL: 5woNRzOWvpWXk4jlM3utL7

13



GET RECOMMENDATIONS

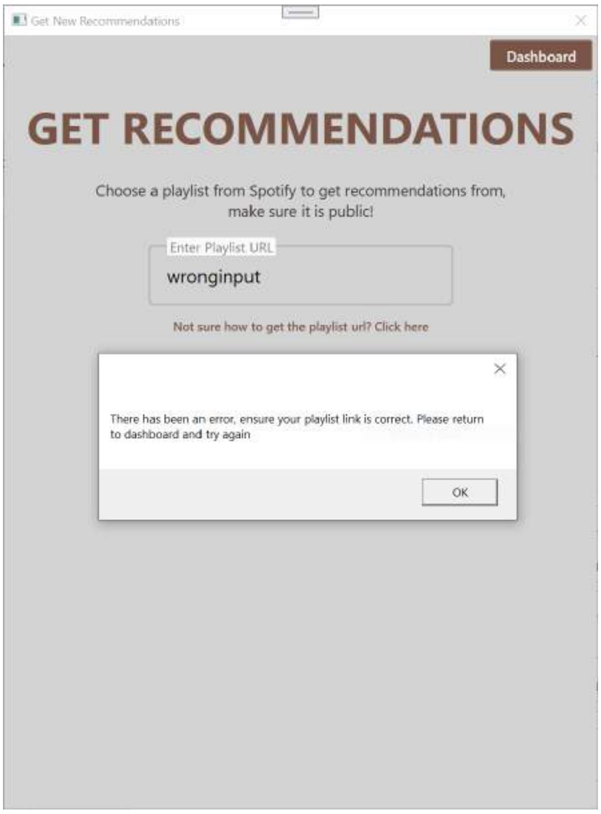
Choose a playlist from Spotify to get recommendations from, make sure it is public!

Enter Playlist URL: 1UowMoBO8qiU61GIlvnEWr

Error

Playlist can not be longer than 100 songs, please try again.

14



GET RECOMMENDATIONS

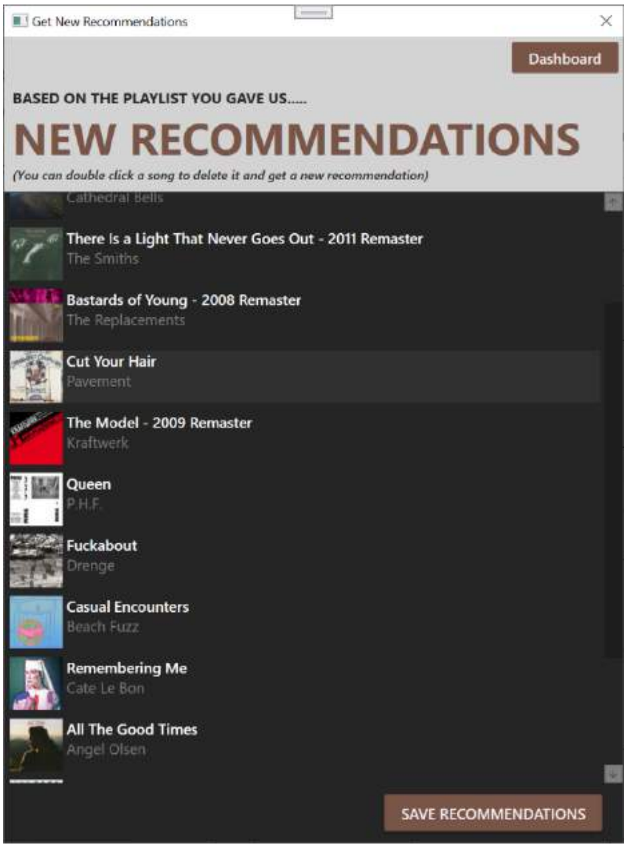
Choose a playlist from Spotify to get recommendations from, make sure it is public!

Enter Playlist URL: wronginput

Not sure how to get the playlist url? Click here

There has been an error, ensure your playlist link is correct. Please return to dashboard and try again

15



BASED ON THE PLAYLIST YOU GAVE US.....

NEW RECOMMENDATIONS

(You can double click a song to delete it and get a new recommendation)

Cathedral Bells

There Is a Light That Never Goes Out - 2011 Remaster
The Smiths

Bastards of Young - 2008 Remaster
The Replacements

Cut Your Hair
Pavement

The Model - 2009 Remaster
Kraftwerk

Queen
P.H.F.

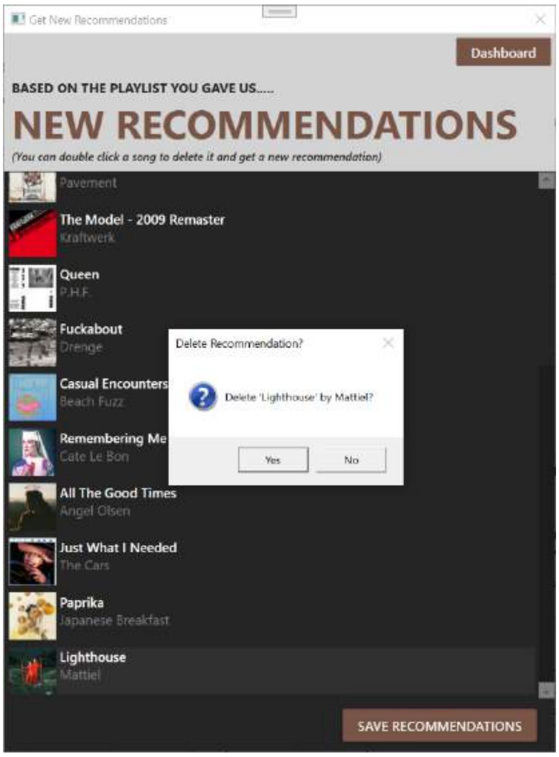
Fuckabout
Drenge

Casual Encounters
Beach Fuzz

Remembering Me
Cate Le Bon

All The Good Times
Angel Olsen

17



BASED ON THE PLAYLIST YOU GAVE US.....

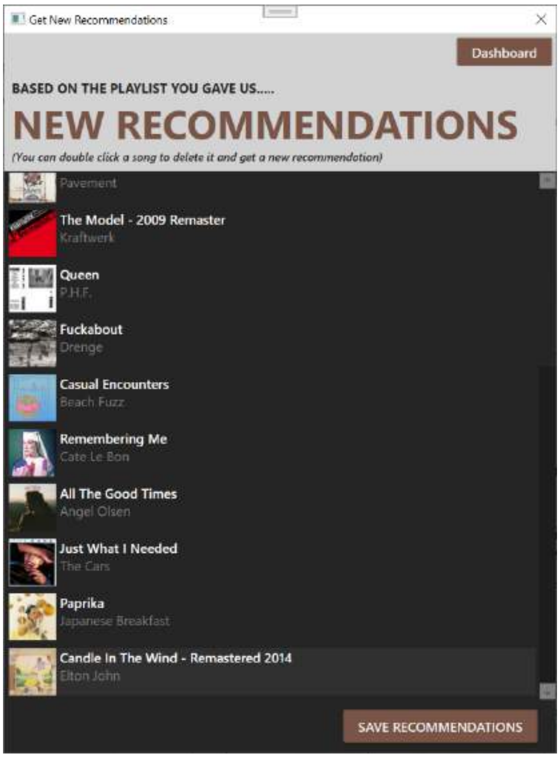
NEW RECOMMENDATIONS

(You can double click a song to delete it and get a new recommendation)

Pavement
The Model - 2009 Remaster
Kraftwerk
Queen
P.H.F.
Fuckabout
Drenge
Casual Encounters
Beach Fuzz
Remembering Me
Cate Le Bon
All The Good Times
Angel Olsen
Just What I Needed
The Cars
Paprika
Japanese Breakfast
Lighthouse
Mattiel

Delete Recommendation?

Delete 'Lighthouse' by Mattiel?



BASED ON THE PLAYLIST YOU GAVE US.....

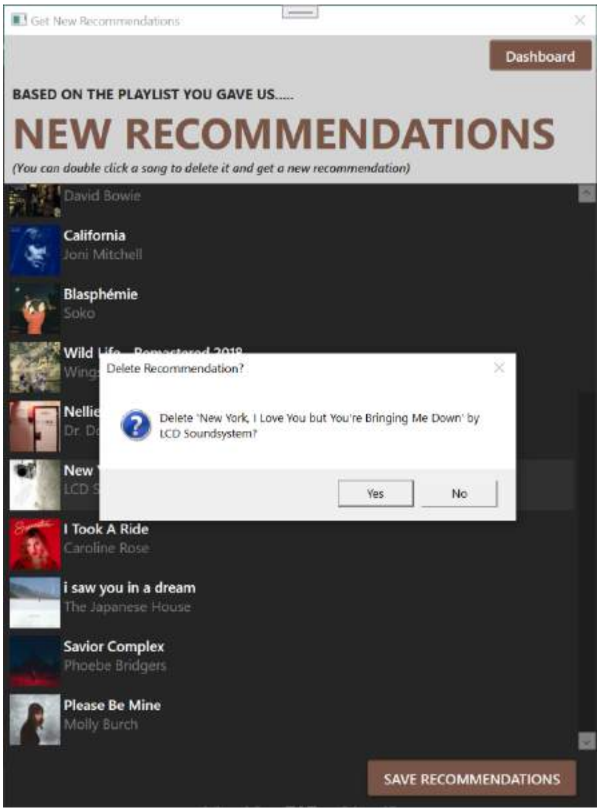
NEW RECOMMENDATIONS

(You can double click a song to delete it and get a new recommendation)

Pavement
The Model - 2009 Remaster
Kraftwerk
Queen
P.H.F.
Fuckabout
Drenge
Casual Encounters
Beach Fuzz
Remembering Me
Cate Le Bon
All The Good Times
Angel Olsen
Just What I Needed
The Cars
Paprika
Japanese Breakfast
Candle In The Wind - Remastered 2014
Elton John

The track Lighthouse is deleted and replaced with 'Candle In The Wind – Remastered 2014' by Elton John.

18



BASED ON THE PLAYLIST YOU GAVE US.....

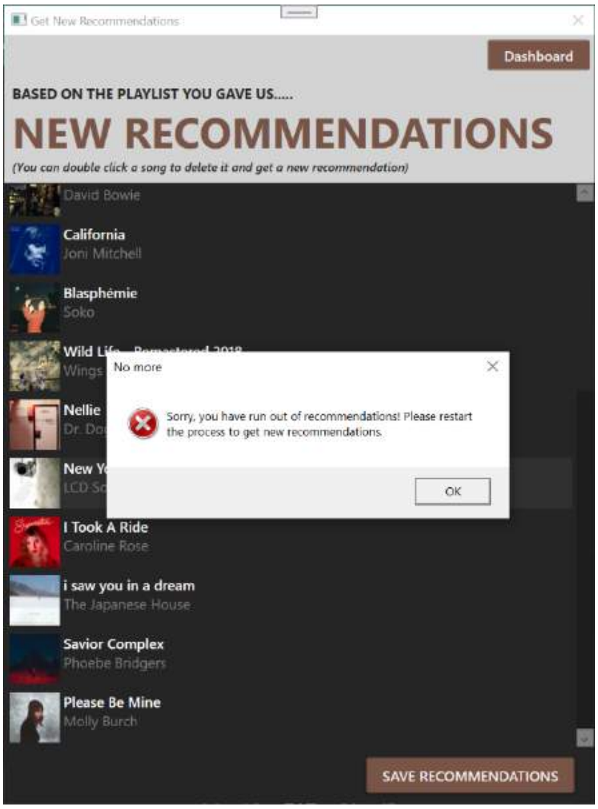
NEW RECOMMENDATIONS

(You can double click a song to delete it and get a new recommendation)

David Bowie
California
Joni Mitchell
Blasphémie
Soko
Wild Life - Remastered 2018
Wing
Nellie
Dr. D
New
LCD S
I Took A Ride
Caroline Rose
i saw you in a dream
The Japanese House
Savior Complex
Phoebe Bridgers
Please Be Mine
Molly Burch

Delete Recommendation?

Delete 'New York, I Love You but You're Bringing Me Down' by LCD Soundsystem?



BASED ON THE PLAYLIST YOU GAVE US.....

NEW RECOMMENDATIONS

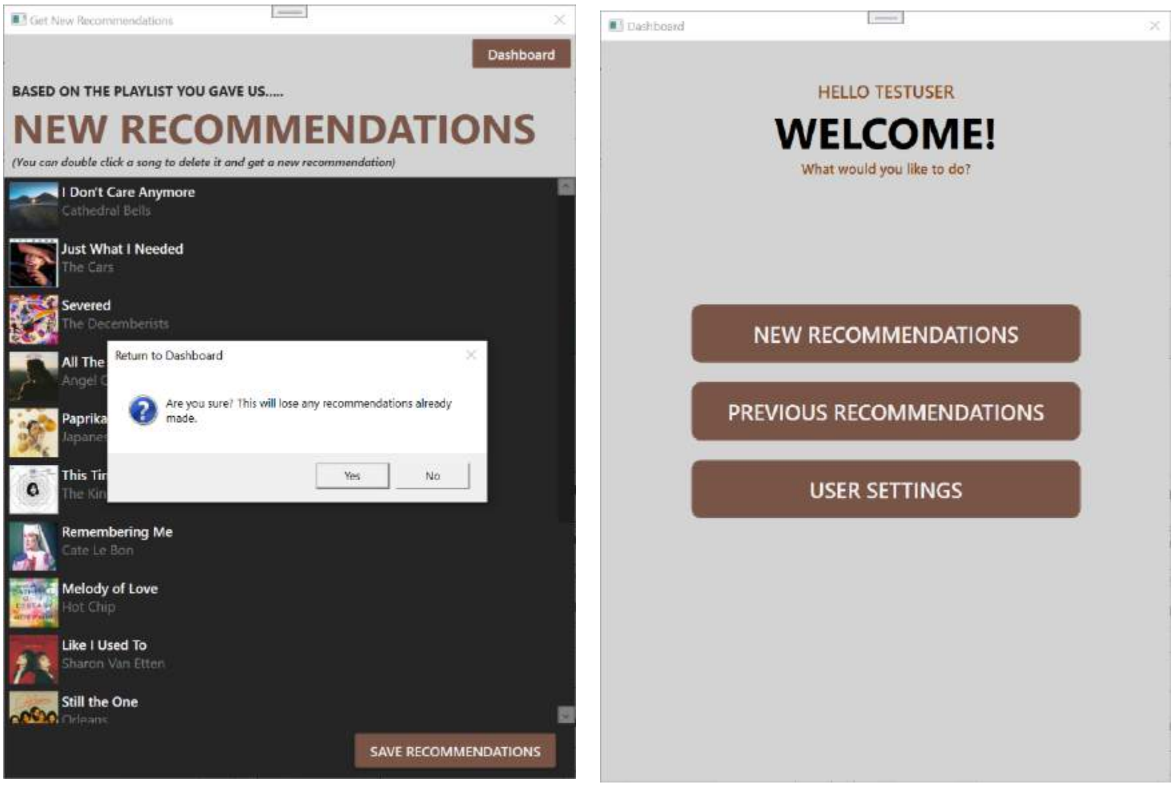
(You can double click a song to delete it and get a new recommendation)

David Bowie
California
Joni Mitchell
Blasphémie
Soko
Wild Life - Remastered 2018
Wings
Nellie
Dr. Do
New Y
LCD So
I Took A Ride
Caroline Rose
i saw you in a dream
The Japanese House
Savior Complex
Phoebe Bridgers
Please Be Mine
Molly Burch

No more

Sorry, you have run out of recommendations! Please restart the process to get new recommendations.

19a



Get New Recommendations

Dashboard

BASED ON THE PLAYLIST YOU GAVE US.....

NEW RECOMMENDATIONS

(You can double click a song to delete it and get a new recommendation)

I Don't Care Anymore
Cathedral Bells

Just What I Needed
The Cars

Severed
The Decemberists

All The
Angel C

Paprika
Japanes

This Tir
The Kin

Remembering Me
Cate Le Bon

Melody of Love
Hot Chip

Like I Used To
Sharon Van Etten

Still the One
Orleans

SAVE RECOMMENDATIONS

Return to Dashboard

Are you sure? This will lose any recommendations already made.

Yes No

Dashboard

HELLO TESTUSER

WELCOME!

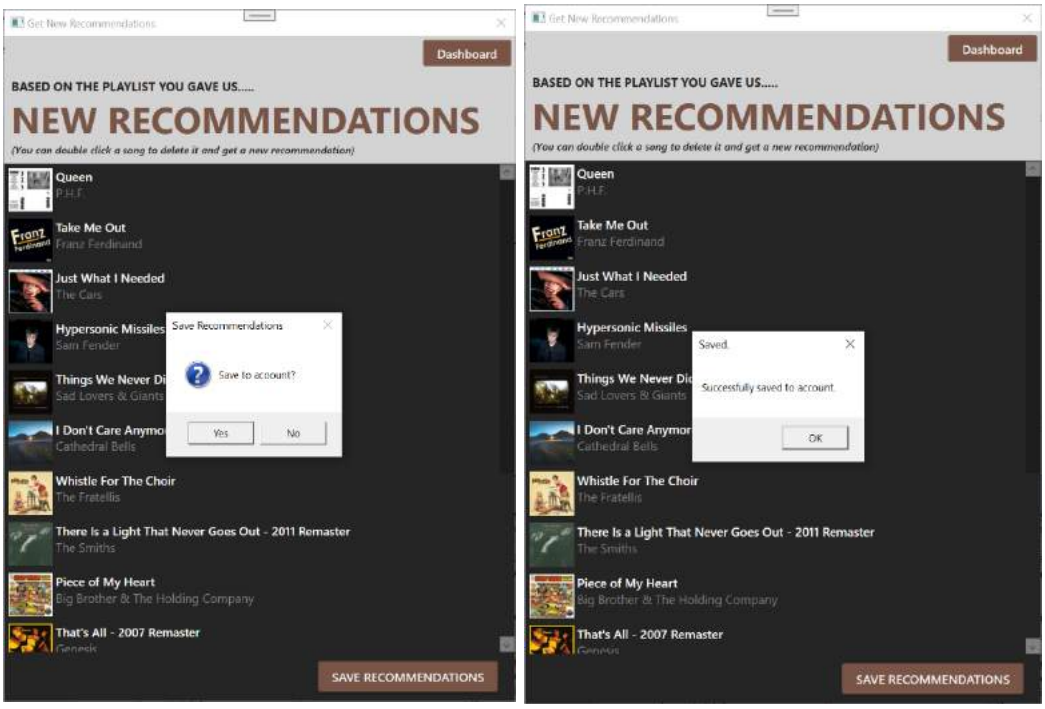
What would you like to do?

NEW RECOMMENDATIONS

PREVIOUS RECOMMENDATIONS

USER SETTINGS

20



Get New Recommendations

Dashboard

BASED ON THE PLAYLIST YOU GAVE US.....

NEW RECOMMENDATIONS

(You can double click a song to delete it and get a new recommendation)

Queen
P.H.F.

Take Me Out
Franz Ferdinand

Just What I Needed
The Cars

Hypersonic Missiles
Sam Fender

Things We Never Di
Sad Lovers & Giants

I Don't Care Anymo
Cathedral Bells

Whistle For The Choir
The Fratellis

There Is a Light That Never Goes Out - 2011 Remaster
The Smiths

Piece of My Heart
Big Brother & The Holding Company

That's All - 2007 Remaster
Genesis

SAVE RECOMMENDATIONS

Save Recommendations

Save to account?

Yes No

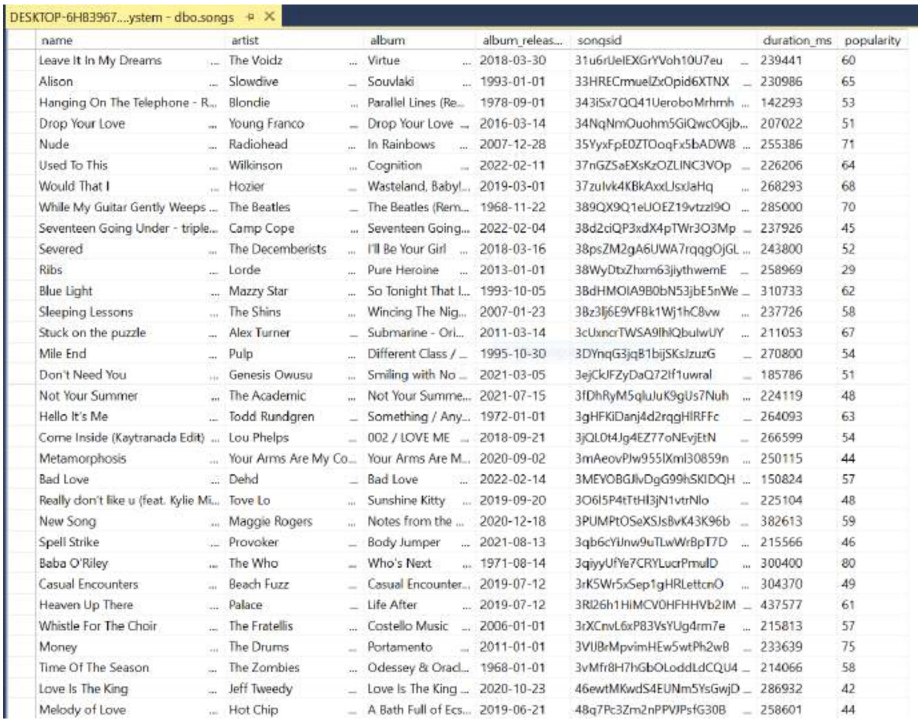
Saved.

Successfully saved to account.

OK

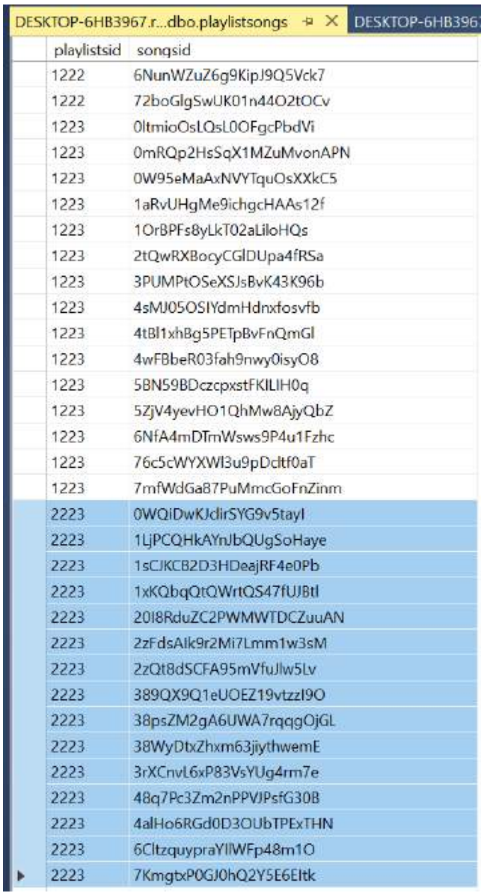
The playlist is added to the database and a success message is displayed to the user.

21

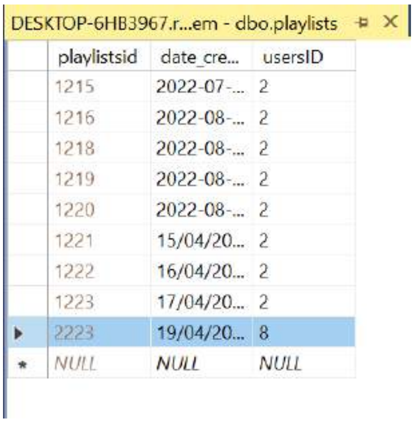

DESKTOP-6HB3967....ystem - dbo.songs

name	artist	album	album_releas...	songsid	duration_ms	popularity
Leave It In My Dreams	The Voidz	Virtue	2018-03-30	31u6rUeIEXGrYVoh10U7eu	239441	60
Alison	Slowdive	Souvlaki	1993-01-01	33HRECrmuelZxOpid6XTNX	230986	65
Hanging On The Telephone - R...	Blondie	Parallel Lines (Re...	1978-09-01	343iSx7QQ41UeroboMrhmh	142293	53
Drop Your Love	Young Franco	Drop Your Love	2016-03-14	34NqNmOuohm5GiQwcOGjb...	207022	51
Nude	Radiohead	In Rainbows	2007-12-28	35YyxFpE0ZTOoqFx5bADW8	255386	71
Used To This	Wilkinson	Cognition	2022-02-11	37nGZSaEXsKzOZLINC3VOp	226206	64
Would That I	Hozier	Wasteland, Babyl...	2019-03-01	37zulvk4KBkAxxLJsxJaHq	268293	68
While My Guitar Gently Weeps ...	The Beatles	The Beatles (Rem...	1968-11-22	389QX9Q1eUOEZ19vtzzI9O	285000	70
Seventeen Going Under - triple...	Camp Cope	Seventeen Going...	2022-02-04	38d2ciQP3xdX4pTWr3O3Mp	237926	45
Severed	The Decemberists	I'll Be Your Girl	2018-03-16	38psZM2gA6UWA7rqqgOjGL	243800	52
Ribs	Lorde	Pure Heroine	2013-01-01	38WyDtxZhxm63jiythwemE	258969	29
Blue Light	Mazzy Star	So Tonight That I...	1993-10-05	3BdHMOIA9B0bN53jbE5nWe	310733	62
Sleeping Lessons	The Shins	Wincing The Nig...	2007-01-23	3Bz3Ij6E9VFBk1Wj1hC8vw	237726	58
Stuck on the puzzle	Alex Turner	Submarine - Ori...	2011-03-14	3cUxncrTWSA9lhlQbuIwUY	211053	67
Mile End	Pulp	Different Class / ...	1995-10-30	3DYnqG3jqB1bijSKsJzuzG	270800	54
Don't Need You	Genesis Owusu	Smiling with No ...	2021-03-05	3ejCkJFZyDaQ72If1uwral	185786	51
Not Your Summer	The Academic	Not Your Summe...	2021-07-15	3fDhRyM5qluJuK9gUs7Nuh	224119	48
Hello It's Me	Todd Rundgren	Something / Any...	1972-01-01	3gHFKiDanj4d2rqgHIRFFc	264093	63
Come Inside (Kaytranada Edit)	Lou Phelps	002 / LOVE ME	2018-09-21	3jQL0t4Jg4EZ77oNEvjEtN	266599	54
Metamorphosis	Your Arms Are My Co...	Your Arms Are M...	2020-09-02	3mAeovPJw955IXml30859n	250115	44
Bad Love	Dehd	Bad Love	2022-02-14	3MEYOBGJlvDgG99hSKIDQH	150824	57
Really don't like u (feat. Kylie Mi...	Tove Lo	Sunshine Kitty	2019-09-20	3O6I5P4tTtHI3jN1vtrNlo	225104	48
New Song	Maggie Rogers	Notes from the ...	2020-12-18	3PUMPtOSeXSJsBvK43K96b	382613	59
Spell Strike	Provoker	Body Jumper	2021-08-13	3qb6cYiJnw9uTLwWrBpT7D	215566	46
Baba O'Riley	The Who	Who's Next	1971-08-14	3qiyyUfYe7CRYLucrPmulD	300400	80
Casual Encounters	Beach Fuzz	Casual Encounter...	2019-07-12	3rK5Wr5xSep1gHRLettcnO	304370	49
Heaven Up There	Palace	Life After	2019-07-12	3RI26h1HIMCV0HFHHVb2IM	437577	61
Whistle For The Choir	The Fratellis	Costello Music	2006-01-01	3rXCnvL6xP83VsYUg4rm7e	215813	57
Money	The Drums	Portamento	2011-01-01	3VIJBrMpvimHEw5wtPh2wB	233639	75
Time Of The Season	The Zombies	Odessey & Oracl...	1968-01-01	3vMfr8H7hGbOLoddLdCQU4	214066	58
Love Is The King	Jeff Tweedy	Love Is The King ...	2020-10-23	46ewtMKwdS4EUNm5YsGwjD	286932	42
Melody of Love	Hot Chip	A Bath Full of Ecs...	2019-06-21	48q7Pc3Zm2nPPVJPsfG30B	258601	44

22

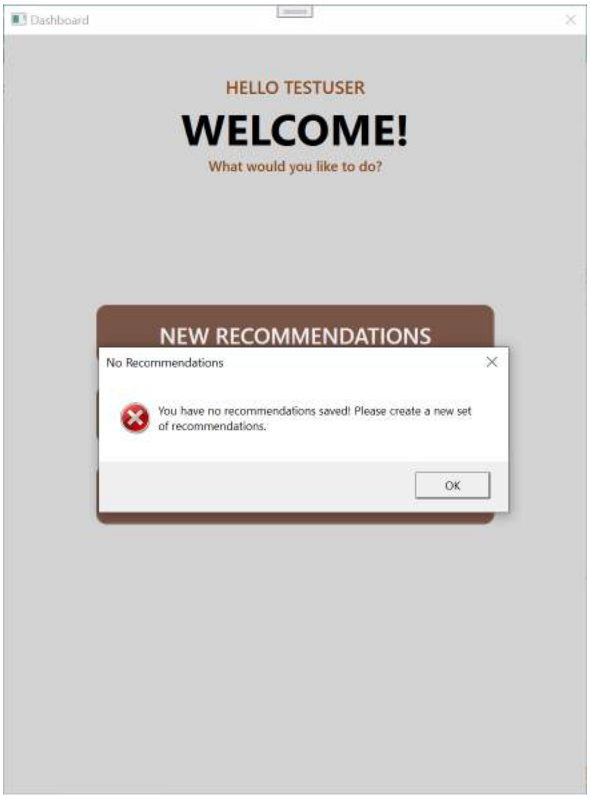

DESKTOP-6HB3967.r...dbo.playlistsongs DESKTOP-6HB396

playlistsid	songsid
1222	6NunWZuZ6g9KipJ9Q5Vck7
1222	72boGlgSwUK01n44O2tOCv
1223	0ltmioOsLQsL0OFgcPbdVi
1223	0mRQp2HsSqX1MZuMvonAPN
1223	0W95eMaAxNVYTquOsXXkC5
1223	1aRvUHgMe9ichgcHAAs12f
1223	1OrBPFs8yLkT02aLiloHQs
1223	2tQwRXBocyCGlDUpa4fRSa
1223	3PUMPtOSeXSJsBvK43K96b
1223	4sMJ05OSIYdmHdnxfosvfb
1223	4tBl1xhBg5PETpBvFnQmGl
1223	4wFBbeR03fah9nwy0isyO8
1223	5BN59BDczcpxstFKILIH0q
1223	5ZjV4yevHO1QhMw8AjyQbZ
1223	6NfA4mDTmWsws9P4u1Fzhc
1223	76c5cWYXWl3u9pDcltf0aT
1223	7mfWdGa87PuMmcGoFnZinm
2223	0WQIDwKJdirSYG9v5tayl
2223	1LjPCQHkAYnJbQUgSoHaye
2223	1sCJKCB2D3HDeajRF4e0Pb
2223	1xKQbqQtQWrtQS47fUJBtl
2223	2018RduZC2PWMWTDCZuuAN
2223	2zFdsAlk9r2Mi7Lmm1w3sM
2223	2zQt8dSCFA95mVfuJw5Lv
2223	389QX9Q1eUOEZ19vtzzI9O
2223	38psZM2gA6UWA7rqqgOjGL
2223	38WyDtxZhxm63jiythwemE
2223	3rXCnvL6xP83VsYUg4rm7e
2223	48q7Pc3Zm2nPPVJPsfG30B
2223	4alHo6RGd0D3OUbTPExTHN
2223	6CltzquypraYllWFp48m1O
2223	7KmgtxP0GJ0hQ2Y5E6Eltk


DESKTOP-6HB3967.r...em - dbo.playlists

playlistsid	date_cre...	usersID
1215	2022-07-...	2
1216	2022-08-...	2
1218	2022-08-...	2
1219	2022-08-...	2
1220	2022-08-...	2
1221	15/04/20...	2
1222	16/04/20...	2
1223	17/04/20...	2
2223	19/04/20...	8
NULL	NULL	NULL

23



Dashboard

HELLO TESTUSER

WELCOME!

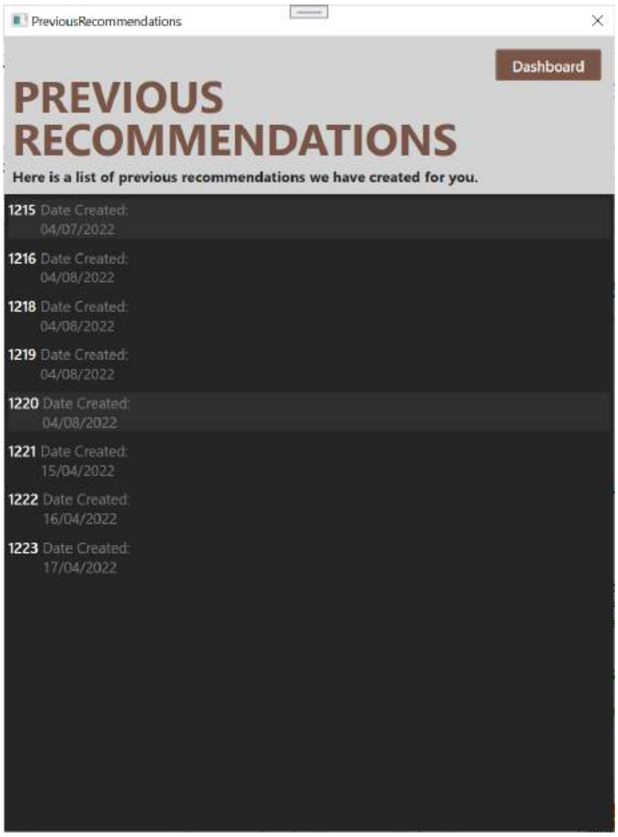
What would you like to do?

NEW RECOMMENDATIONS

No Recommendations

You have no recommendations saved! Please create a new set of recommendations.

24



PreviousRecommendations

PREVIOUS RECOMMENDATIONS

Here is a list of previous recommendations we have created for you.

1215 Date Created: 04/07/2022

1216 Date Created: 04/08/2022

1218 Date Created: 04/08/2022

1219 Date Created: 04/08/2022

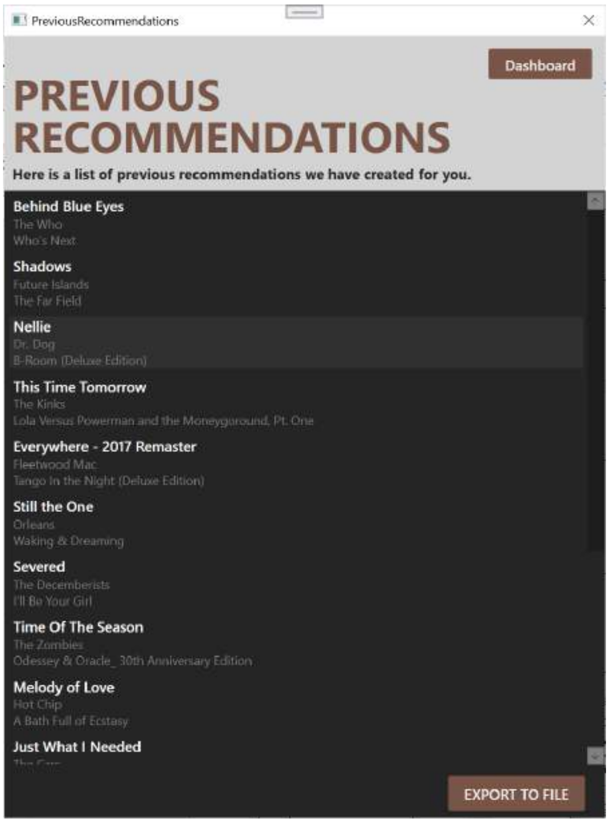
1220 Date Created: 04/08/2022

1221 Date Created: 15/04/2022

1222 Date Created: 16/04/2022

1223 Date Created: 17/04/2022

26


PreviousRecommendations

Dashboard

PREVIOUS RECOMMENDATIONS

Here is a list of previous recommendations we have created for you.

Behind Blue Eyes
The Who
Who's Next

Shadows
Future Islands
The Far Field

Nellie
Dr. Dog
B-Room (Deluxe Edition)

This Time Tomorrow
The Kinks
Lola Versus Powerman and the Moneygoround, Pt. One

Everywhere - 2017 Remaster
Fleetwood Mac
Tango In the Night (Deluxe Edition)

Still the One
Orleans
Waking & Dreaming

Severed
The Decemberists
I'll Be Your Girl

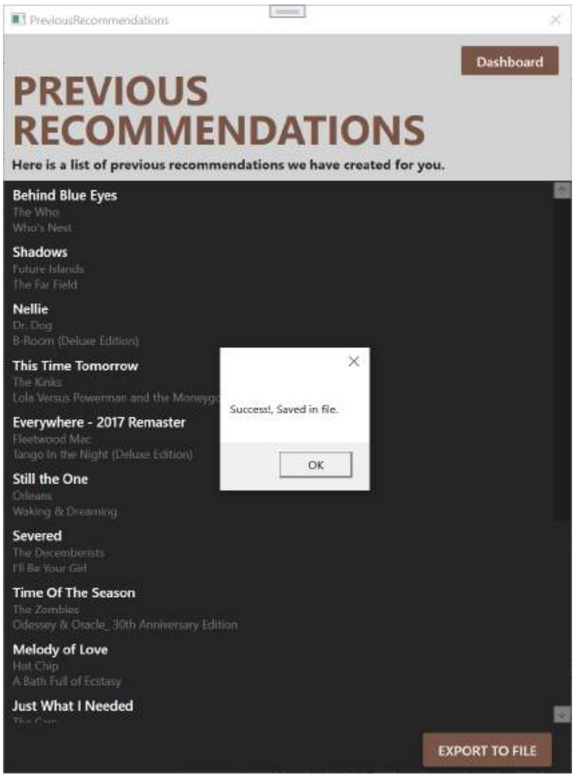
Time Of The Season
The Zombies
Odessey & Oracle_ 30th Anniversary Edition

Melody of Love
Hot Chip
A Bath Full of Ecstasy

Just What I Needed

EXPORT TO FILE

27
The file output from the test is shown below as well.


PreviousRecommendations

Dashboard

PREVIOUS RECOMMENDATIONS

Here is a list of previous recommendations we have created for you.

Behind Blue Eyes
The Who
Who's Next

Shadows
Future Islands
The Far Field

Nellie
Dr. Dog
B-Room (Deluxe Edition)

This Time Tomorrow
The Kinks
Lola Versus Powerman and the Moneyg

Everywhere - 2017 Remaster
Fleetwood Mac
Tango In the Night (Deluxe Edition)

Still the One
Orleans
Waking & Dreaming

Severed
The Decemberists
I'll Be Your Girl

Time Of The Season
The Zombies
Odessey & Oracle_ 30th Anniversary Edition

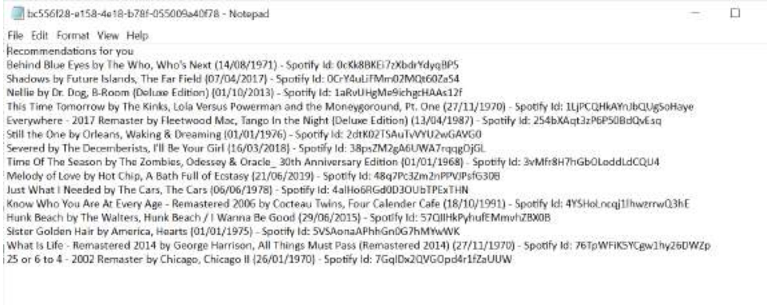
Melody of Love
Hot Chip
A Bath Full of Ecstasy

Just What I Needed

Success!, Saved in file.

OK

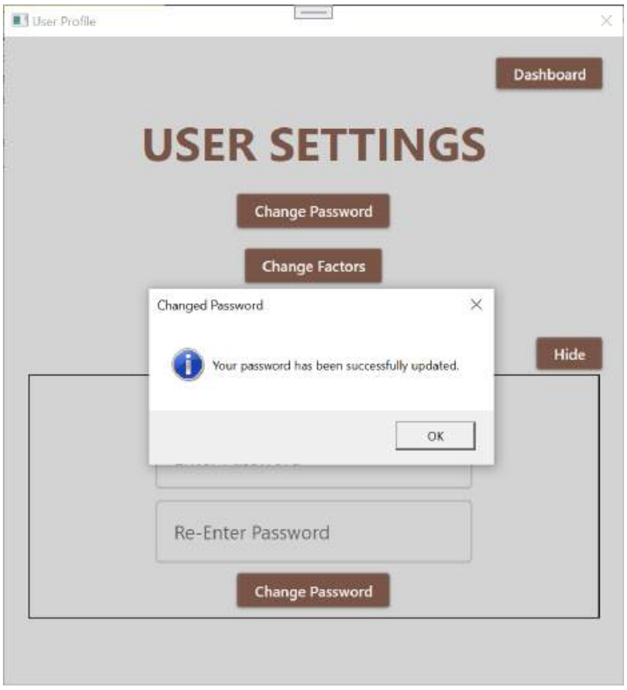
EXPORT TO FILE


bc556f28-e158-4e18-b78f-055009a40f78 - Notepad

File Edit Format View Help

```
Recommendations for you
Behind Blue Eyes by The Who, Who's Next (14/08/1971) - Spotify Id: 0cKk8BKEi7zXbdrYdyqBP5
Shadows by Future Islands, The Far Field (07/04/2017) - Spotify Id: 0CrY4uLiFMm02MQt60Za54
Nellie by Dr. Dog, B-Room (Deluxe Edition) (01/10/2013) - Spotify Id: 1aRvUHgMe9ichgcHAAs12f
This Time Tomorrow by The Kinks, Lola Versus Powerman and the Moneygoround, Pt. One (27/11/1970) - Spotify Id: 1LjPCQHkAYnJbQUgSoHaye
Everywhere - 2017 Remaster by Fleetwood Mac, Tango In the Night (Deluxe Edition) (13/04/1987) - Spotify Id: 254bXAqt3zP6P50BdQvEsq
Still the One by Orleans, Waking & Dreaming (01/01/1976) - Spotify Id: 2dtK02TSAuTvVYU2wGAVG0
Severed by The Decemberists, I'll Be Your Girl (16/03/2018) - Spotify Id: 38psZM2gA6UWA7rqqgOjGL
Time Of The Season by The Zombies, Odessey & Oracle_ 30th Anniversary Edition (01/01/1968) - Spotify Id: 3vMfr8H7hGbOLoddLdCQU4
Melody of Love by Hot Chip, A Bath Full of Ecstasy (21/06/2019) - Spotify Id: 48q7Pc3Zm2nPPVJPsfG30B
Just What I Needed by The Cars, The Cars (06/06/1978) - Spotify Id: 4alHo6RGd0D3OUbTPExTHN
Know Who You Are At Every Age - Remastered 2006 by Cocteau Twins, Four Calender Cafe (18/10/1991) - Spotify Id: 4YSHoLncqj1lhwzrrwQ3hE
Hunk Beach by The Walters, Hunk Beach / I Wanna Be Good (29/06/2015) - Spotify Id: 57QIIHkPyhufEMmvhZBX0B
Sister Golden Hair by America, Hearts (01/01/1975) - Spotify Id: 5VSAonaAPhhGn0G7hMYwWK
What Is Life - Remastered 2014 by George Harrison, All Things Must Pass (Remastered 2014) (27/11/1970) - Spotify Id: 76TpWFiK5YCgw1hy26DWZp
25 or 6 to 4 - 2002 Remaster by Chicago, Chicago II (26/01/1970) - Spotify Id: 7GqlDx2QVGOpd4r1fZaUUW
```

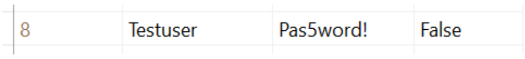
28



Changed Password

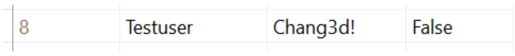
Your password has been successfully updated.

The user data in the users table before the password was changed:



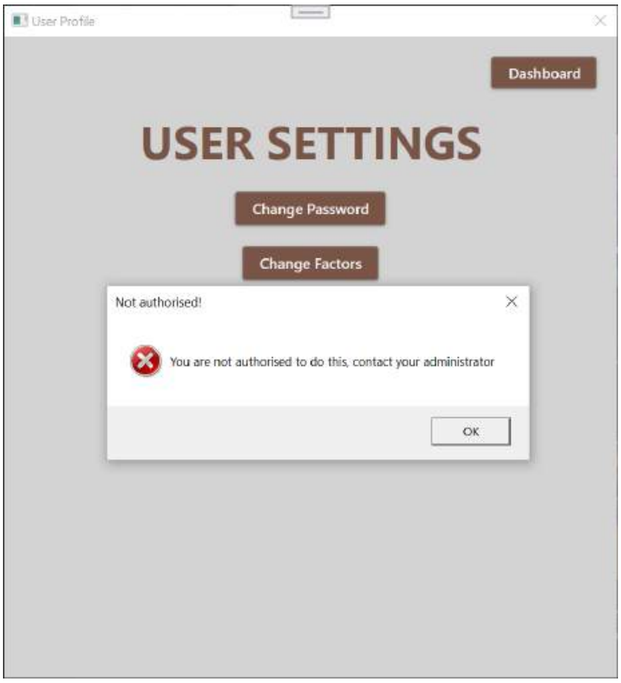
8	Testuser	Pas5word!	False

The user data in the users table after the password was changed:



8	Testuser	Chang3d!	False

29

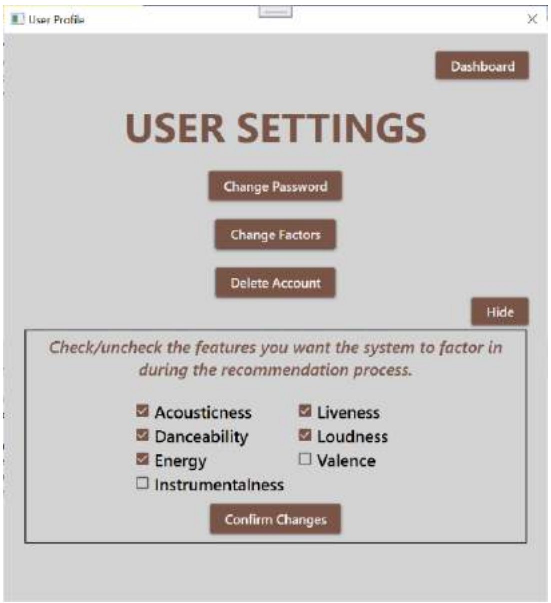


Not authorised!

You are not authorised to do this, contact your administrator

30

State of factors upon clicking the button

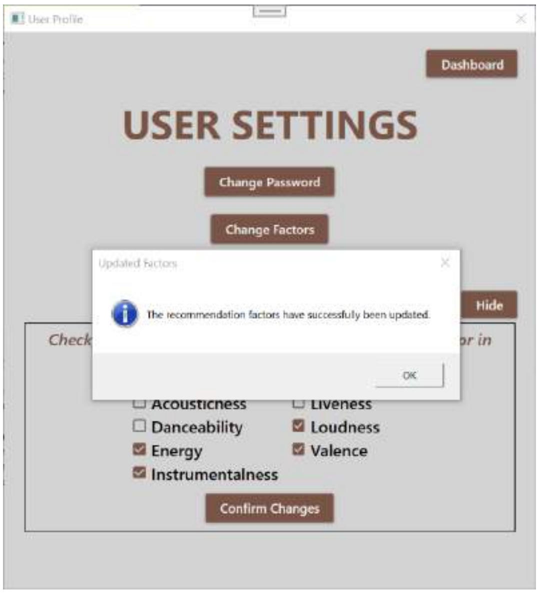


USER SETTINGS

Check/uncheck the features you want the system to factor in during the recommendation process.

☑ Acousticness ☑ Liveness
☑ Danceability ☑ Loudness
☑ Energy □ Valence
□ Instrumentalness

Success message

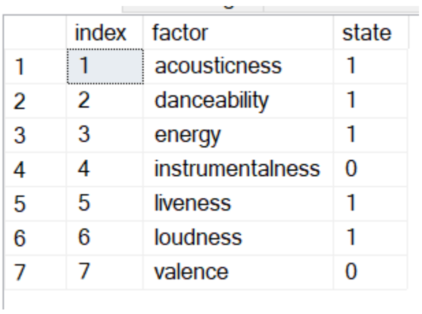


USER SETTINGS

The recommendation factors have successfuly been updated.

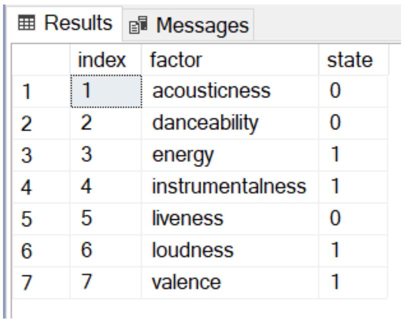
□ Acousticness □ Liveness
□ Danceability ☑ Loudness
☑ Energy ☑ Valence
☑ Instrumentalness

Factors table before user modifies them



	index	factor	state
1	1	acousticness	1
2	2	danceability	1
3	3	energy	1
4	4	instrumentalness	0
5	5	liveness	1
6	6	loudness	1
7	7	valence	0

Factors after user has modified them

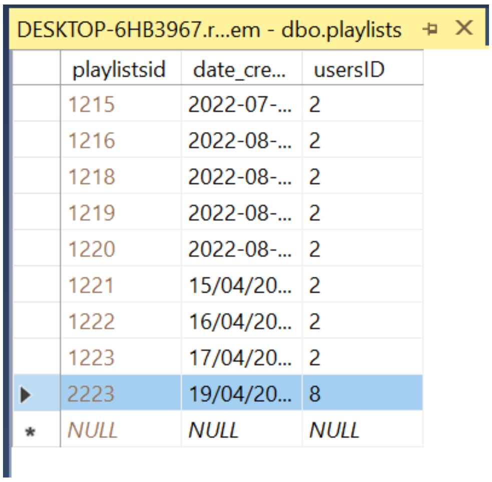

Results
Messages

	index	factor	state
1	1	acousticness	0
2	2	danceability	0
3	3	energy	1
4	4	instrumentalness	1
5	5	liveness	0
6	6	loudness	1
7	7	valence	1

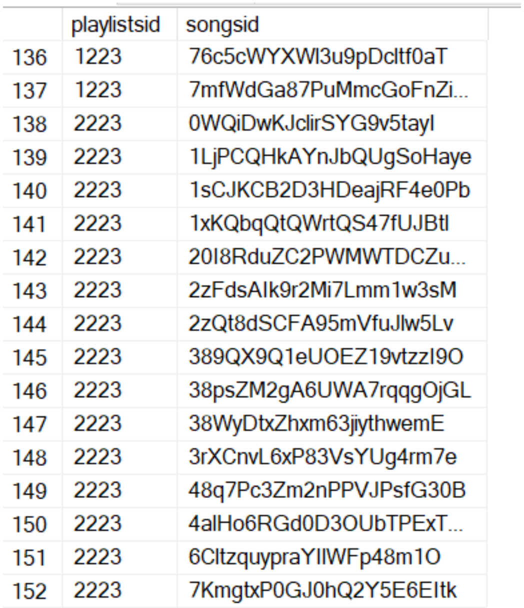
31

[The usersID for the account being deleted is 8]

The database before the account was deleted:

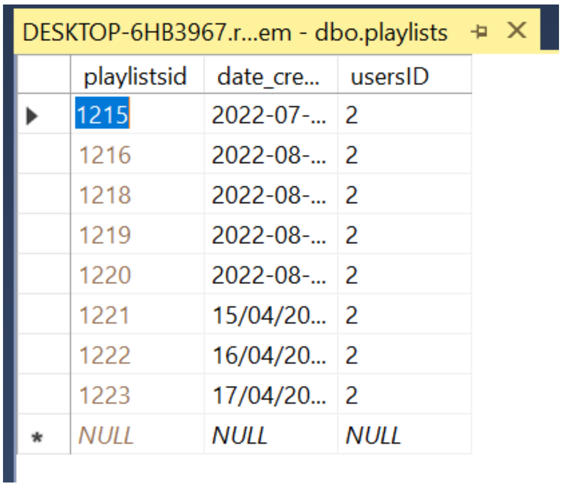


	playlistsid	date_cre...	usersID
	1215	2022-07-...	2
	1216	2022-08-...	2
	1218	2022-08-...	2
	1219	2022-08-...	2
	1220	2022-08-...	2
	1221	15/04/20...	2
	1222	16/04/20...	2
	1223	17/04/20...	2
▶	2223	19/04/20...	8
*	NULL	NULL	NULL

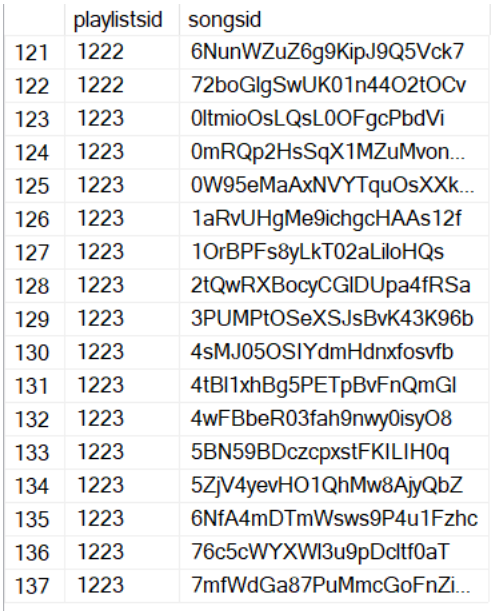


	playlistsid	songsid
136	1223	76c5cWYXWl3u9pDcltf0aT
137	1223	7mfWdGa87PuMmcGoFnZi...
138	2223	0WQiDwKJclirSYG9v5tayl
139	2223	1LjPCQHkAYnJbQUgSoHaye
140	2223	1sCJKCB2D3HDeajRF4e0Pb
141	2223	1xKQbqQtQWrtQS47fUJBtl
142	2223	20l8RduZC2PWMWTDCZu...
143	2223	2zFdsAlk9r2Mi7Lmm1w3sM
144	2223	2zQt8dSCFA95mVfuJlw5Lv
145	2223	389QX9Q1eUOEZ19vtzzI9O
146	2223	38psZM2gA6UWA7rqqgOjGL
147	2223	38WyDtxZhxm63jiythwemE
148	2223	3rXCnvL6xP83VsYUg4rm7e
149	2223	48q7Pc3Zm2nPPVJPsfG30B
150	2223	4alHo6RGd0D3OUbTPExT...
151	2223	6CltzquypraYlIWFp48m1O
152	2223	7KmgtxP0GJ0hQ2Y5E6EItk

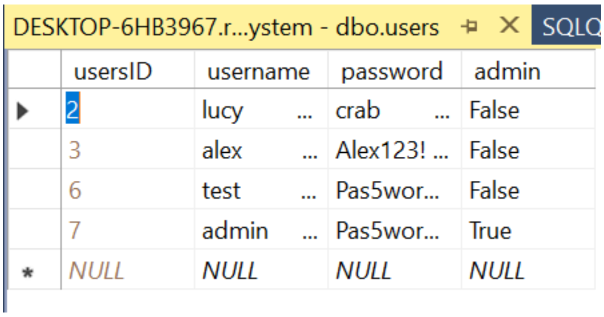
The database after the account was deleted:



	playlistsid	date_cre...	usersID
▶	1215	2022-07-...	2
	1216	2022-08-...	2
	1218	2022-08-...	2
	1219	2022-08-...	2
	1220	2022-08-...	2
	1221	15/04/20...	2
	1222	16/04/20...	2
	1223	17/04/20...	2
*	NULL	NULL	NULL



	playlistsid	songsid
121	1222	6NunWZuZ6g9KipJ9Q5Vck7
122	1222	72boGlgSwUK01n44O2tOCv
123	1223	0ltmioOsLQsL0OFgcPbdVi
124	1223	0mRQp2HsSqX1MZuMvon...
125	1223	0W95eMaAxNVYTquOsXXk...
126	1223	1aRvUHgMe9ichgcHAAs12f
127	1223	1OrBPFs8yLkT02aLiloHQs
128	1223	2tQwRXBocyCGlDUpa4fRSa
129	1223	3PUMPtOSeXSJsBvK43K96b
130	1223	4sMJ05OSIYdmHdnxfosvfb
131	1223	4tBl1xhBg5PETpBvFnQmGl
132	1223	4wFBbeR03fah9nwy0isyO8
133	1223	5BN59BDczcpxstFKILIH0q
134	1223	5ZjV4yevHO1QhMw8AjyQbZ
135	1223	6NfA4mDTmWsws9P4u1Fzhc
136	1223	76c5cWYXWl3u9pDcltf0aT
137	1223	7mfWdGa87PuMmcGoFnZi...



	usersID	username	password	admin
▶	2	lucy ...	crab ...	False
	3	alex ...	Alex123! ...	False
	6	test ...	Pas5wor...	False
	7	admin ...	Pas5wor...	True
*	NULL	NULL	NULL	NULL

As shown, there is no presence of the user in the database after deletion.

Evaluation

The completed system successfully recommends a set of 15 songs based on a playlist the user provides. The recommendations are varied, and it is easy to receive new recommendations if the user is unhappy with any. The process is very simple to use, as is the process of saving the playlists and exporting them to files for personal use. There is room for personalisation within the recommendations, with the admin being able to choose which factors are used within the algorithm.

There are notable differences between the design of the navigation and algorithms in the system and the actual implementation. The system flowcharts suggest that the user will be able to export the recommendations to a .CSV file, but also once they are first created. Since implementation, the ability to save a set of recommendations is only available in the previous recommendations section of the system, with the user only being able to save the set of recommendations to the database once they are first created.
Furthermore, using frames to display pages of content within a window has meant that certain navigation directions no longer exist. However, this has meant I do not have to duplicate code for certain features that would be present in several windows, saving memory.

Assessment Of Overall Outcome Against Original Objectives

The following table compares the execution of the completed system to the objectives originally outlined on page 22, to ensure all client's needs have been met.

Key:

OBJECTIVE NOT MET	**OBJECTIVE PARTIALLY MET**	**OBJECTIVE MET**	**OBJECTIVE EXCEEDED**

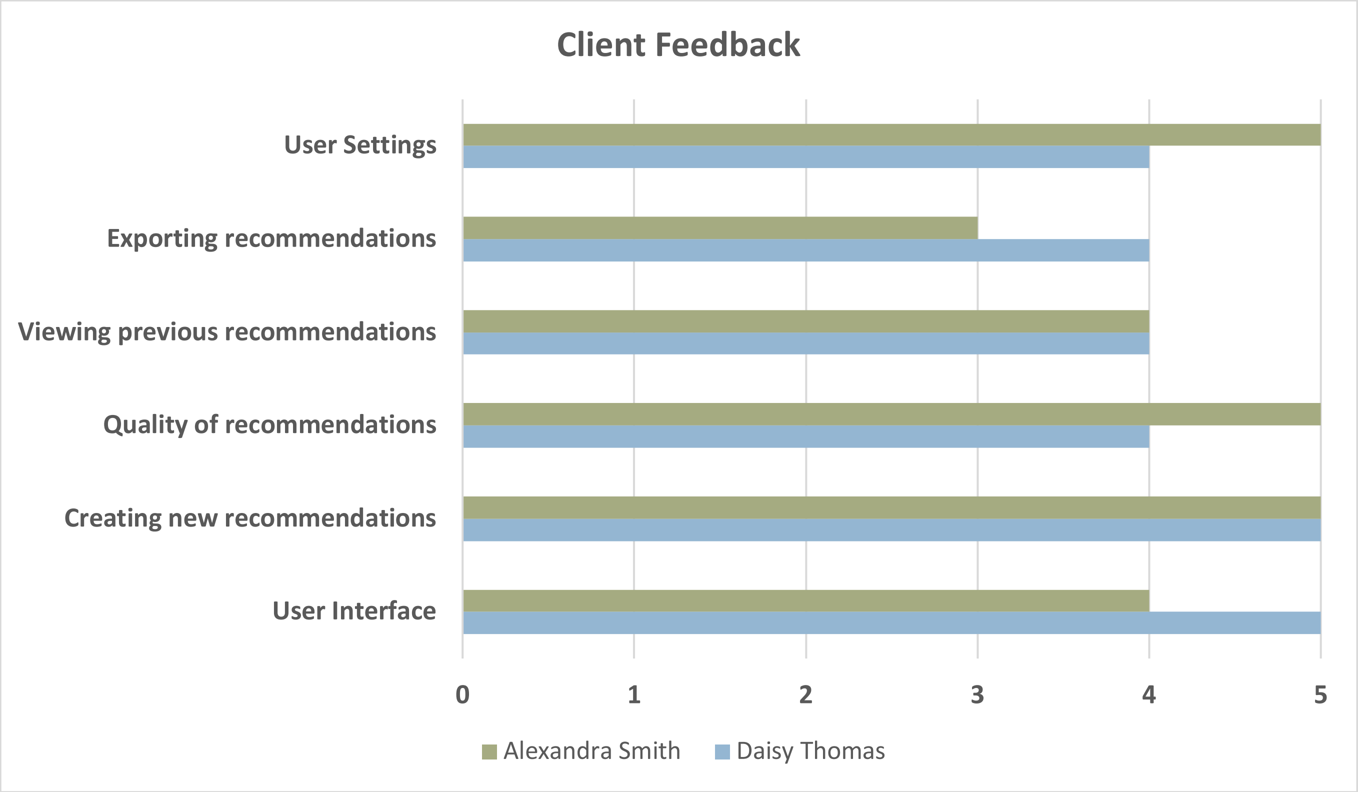
Original Objectives	Completed System
The system will calculate and display a list of song recommendations based of a user playlist.	Not only does the completed system display the song recommendations, but it aesthetically displays them, and each song is displayed using four different pieces of information.
The system should involve minimum text entry, with the only information being the account details and Spotify playlist URL, to save time and minimise errors.	There is minimum text entry within the system, with the only text input from the user being the user account details and the Spotify playlist URL. All other user input is through the form of buttons and checkboxes.
The system should be able to display 15 songs in a random order.	The original playlist order is randomised when generating recommendations, meaning the recommendations are more likely to be songs that the user hasn't considered before.

The new user must complete a registration form in which they enter their name, a username, and a password.	While the system does require a username and a password, there is no name input. This is because the intended use of the system is only for 2 users, therefore the usernames can be more personal.
The username must be unique (on the system).	There are multiple checks throughout the system to ensure that each username is unique. Usernames are not case-sensitive meaning there can't be multiple variations of one word.
Passwords must be at least 8 characters long and contain a mix of character types.	There are multiple validation criteria checks when creating or updating a user password. The password must be between 8 and 16 characters and contain a mix of character types. The password must contain at least one lowercase letter, one uppercase letter, one punctuation character, and one number. There must also be no whitespaces.
Only one user registered on system must be labelled as 'administrator' and get admin permissions.	The users are automatically marked as '0' when their account is registered, meaning that there cannot be multiple admins on the system.
The user must be able to share a Spotify playlist URL and receive recommendations based on the songs within it.	Users can receive a multitude of recommendations and depending on the size of the playlist (as long as it is less than 100), there will be more recommendations available.
The system should have the option to discard songs that the user doesn't like and recommend a new song instead.	Users are prompted to confirm when they want to replace a recommendation just in case they accidentally double-click on a song.
The system must be able to store all the relevant details about every song recommended with the following details being essential for each song: • Song Name • Artist Name • Spotify Unique Id	Not only does the system store the mentioned details in the database, but it also stores the audio features and duration of the songs. While this data isn't exactly useful in the current stage of the system, it could prove useful in future developments of the system.
The system must store the recommendations in a user profile for the user to access at a later date.	The user can access all previous recommendations by clicking the previous recommendations button on the dashboard.
The option to save the recommendations in the format of a text file must be available to the user, so it is easily accessible.	Once the user has chosen which playlist they want to view, they have the option to save the set of recommendations to a file,

	which is formatted to only contain relevant information for each song.
There should be the option for an administrator to delete/modify user accounts and their details.	This option does not exist in the current state of the system. As there are so few users, it seemed a waste of a feature.
There should be the option for an administrator to modify the deciding factors of the recommendation system.	Not only can admins modify the factors, but they are also able to see which recommendation factors are currently being used in the system.
The recommendations should include a variety of music types – based on genre, artist, album, happiness, etc.	While the recommendations aren't explicitly made by factoring in these music types, it seems that through the way I designed the recommendation algorithm that there is a range of music recommended anyway.
The recommendations must not include more than 3 songs per artist.	As mentioned above.
Users must be able to delete their account from the system.	If the user double confirms they want to delete their account from the system, the system also deletes all relating playlist data within the database – ensuring that no trace of the user is left. While the song data is kept, it has no relation to the user apart from when they are linked with a playlist.

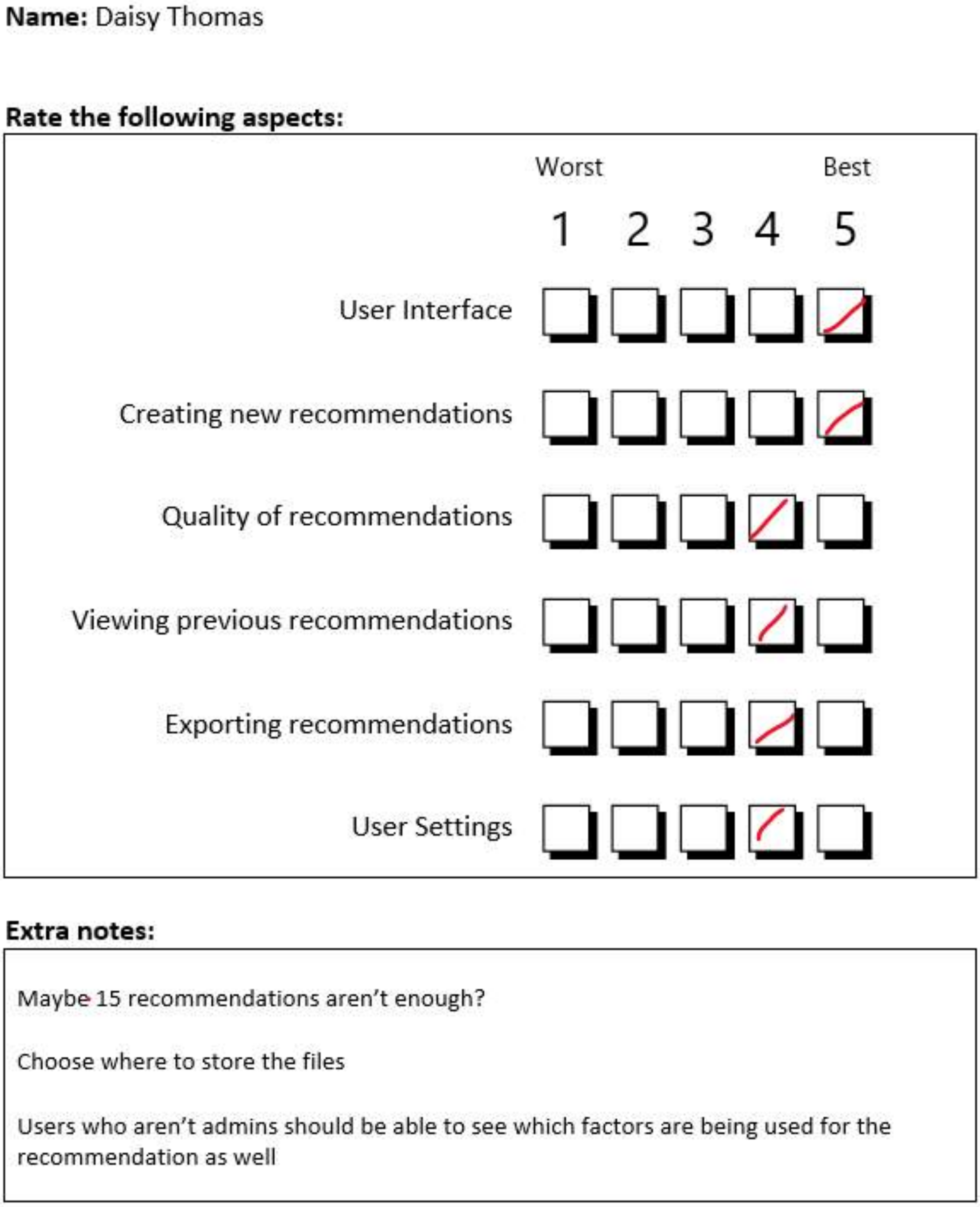
Client Feedback

Once I completed testing, I gave the program to the two clients of the system. I created a survey for them to complete while using the system, which can be seen below. I also left a space for any additional notes and though, which have been summarised below.



Improvements	
Daisy Thomas	**Alexandra Smith**
Factors visible while making new recommendations even if user isn't an admin File storage location	Colour scheme of system File storage location

Surveys



Name: Daisy Thomas

Rate the following aspects:

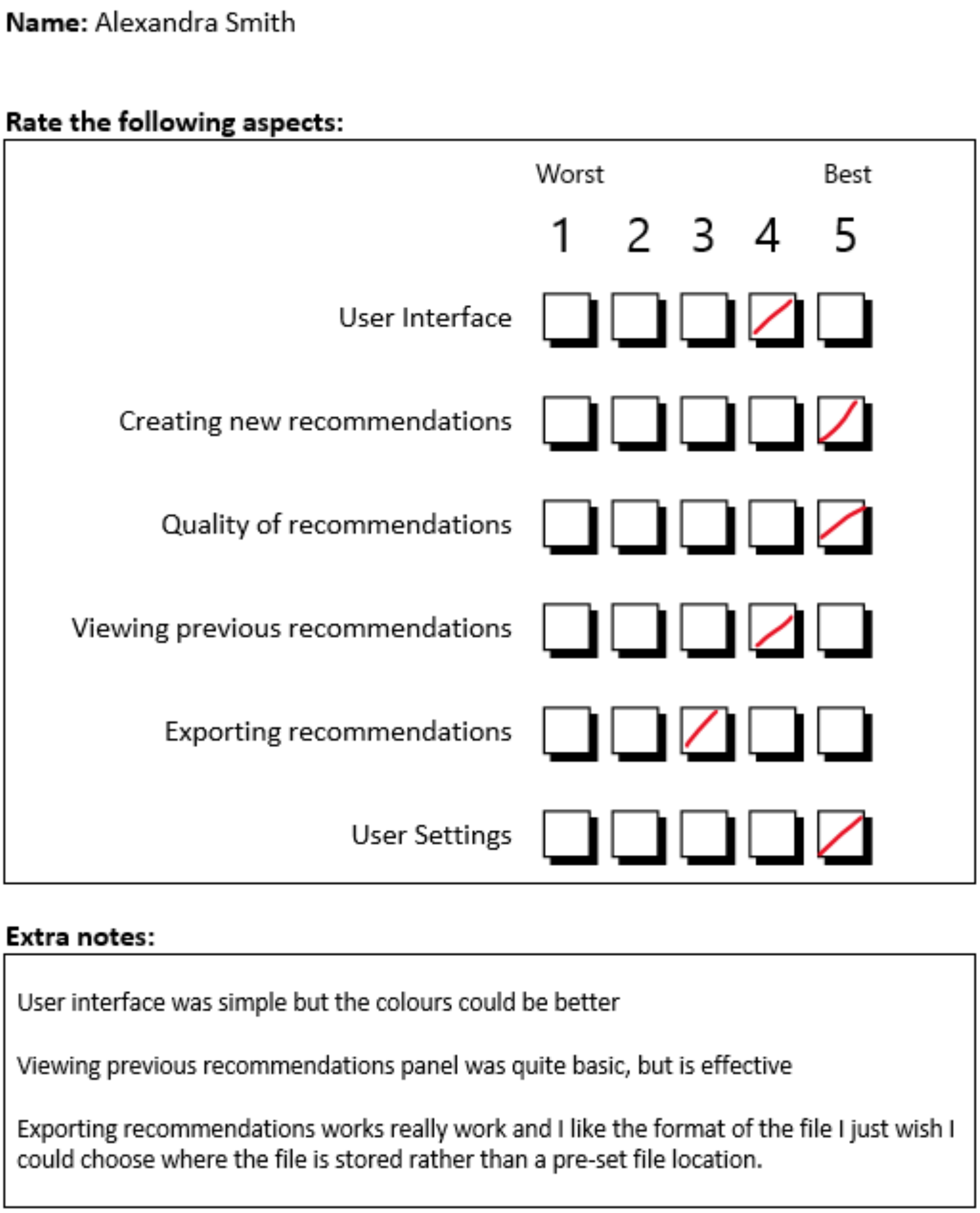
	Worst				Best
	1	2	3	4	5
User Interface	□	□	□	□	☑
Creating new recommendations	□	□	□	□	☑
Quality of recommendations	□	□	□	☑	□
Viewing previous recommendations	□	□	□	☑	□
Exporting recommendations	□	□	□	☑	□
User Settings	□	□	□	☑	□

Extra notes:

Maybe 15 recommendations aren't enough?

Choose where to store the files

Users who aren't admins should be able to see which factors are being used for the recommendation as well



Name: Alexandra Smith

Rate the following aspects:

	Worst				Best
	1	2	3	4	5
User Interface	☐	☐	☐	☒	☐
Creating new recommendations	☐	☐	☐	☐	☒
Quality of recommendations	☐	☐	☐	☐	☒
Viewing previous recommendations	☐	☐	☐	☒	☐
Exporting recommendations	☐	☐	☒	☐	☐
User Settings	☐	☐	☐	☐	☒

Extra notes:

User interface was simple but the colours could be better

Viewing previous recommendations panel was quite basic, but is effective

Exporting recommendations works really work and I like the format of the file I just wish I could choose where the file is stored rather than a pre-set file location.

Conclusion from client feedback

From the feedback, it is clear that while the system functions as it should, there is room for improvement with regard to the file export. Both users expressed their desire to be able to choose where the file is exported to. As well, the primary client shared that they would like to be able to see which factors are being incorporated into the recommendation process, even if they aren't on an admin account. This is something I would develop in later versions of the system.

Future Improvements

Whilst the program is functional, I feel that the memory usage could be optimised. Although I experimented with the MVMM architectural pattern, I would redesign the models to have similar functions and methods within one class, instead of being grouped by windows. As well, I would alter the algorithms, focusing more on object-orientated paradigms rather than functional programming. A lot of algorithms share objects so this would optimise memory.

To improve, admins should also be able to access user profiles (not playlist data) within the system itself rather than just in the database. This should have been in the original system as it was one of the original objectives however, I felt it was not relevant to the current scale of the project.

To enhance the user experience, I would incorporate the ability to name the set of recommendations. When saving the recommendations to the database I would have a pop-up dialog box containing a textbox prompting the user to name the playlist. This would then be stored in the database along with the other playlist information. When viewing previous recommendations, the playlist names would be visible. When exporting a specific set of recommendations, the playlist name would be written at the top. Also, when the user goes to export the recommendations, I would add the ability to choose the file source rather than a predefined folder.

Whilst the following development is not in the scope of the project outline, if I were to develop the system even further, I would include the ability to receive recommendations based on two songs given by the user. Using the audio features of each song, the system would compare and calculate the ideal metadata of a playlist relating to the songs, which would then be displayed to the user.