

Music Player Application

Milestone: PROJECT REPORT

Group25

ETIKALA SAI MAHITHA

HEMA VENKATA SAI TEJA GINNEGOLLA

+1 857 395 9958 (Tel of Etikala Sai Mahitha)

+1 339 237 7638 (Tel of Hema Venkata Sai Teja)

EMAIL:

etikala.s@northeastern.edu

ginnegolla.h@northeastern.edu

Percentage of Effort Contributed by ETIKALA SAI MAHITHA : 50%

Percentage of Effort Contributed by HEMA VENKATA SAITEJA : 50%

Signature of Sai Mahitha : SAI MAHITHA

Signature of Hema Venkata Sai Teja : SAITEJA

SUBMISSION DATE : 10th December 2023

Report

Introduction

Introduction Digital innovation has significantly evolved the way we experience music during this period. The development of music streaming services has totally transformed the music industry by providing a huge collection of songs at our fingertips. The enormous number of music choices, meanwhile, can be intimidating. To deal with the issue, our project sets forth to develop an innovative music streaming and recommendation platform, an online soundscape where users may discover, appreciate, and personalize their musical experience.

This project aims to build a complete and user-centric experience similar to that of market leaders like Spotify as a response to the evolving landscape of music discovery. With the help of our platform, music buffs will be able to explore new songs, revisit old favorites, and connect to a large community of other listeners across the globe. We aim to reinvent how people interact with music in the age of technology with an innovative recommendation engine, social features for sharing playlists, and strong data protection. Welcome to the age of musical exploration and enjoyment, where each note creates a unique rhythm.

The project is designed to transform the way people engage with music by using data analysis and visualization. Unlike traditional music platforms, our focus goes beyond merely offering songs. With features like the ability to like songs, follow favorite artists, and create personalized playlists, we're shaping a more interactive and user-driven music experience.

Through data analysis, we aim to understand each user's unique music preferences. By allowing users to express their likes, follow artists they enjoy, and curate playlists that suit their moods, our platform becomes a personalized soundscape. These features not only enhance individual enjoyment but also foster a sense of community by connecting music enthusiasts around the world.

In essence, our project brings together the analytical power of data with the social aspects of music appreciation. Welcome to a platform where your musical journey is not only personalized but also shared with a global community of like-minded listeners.

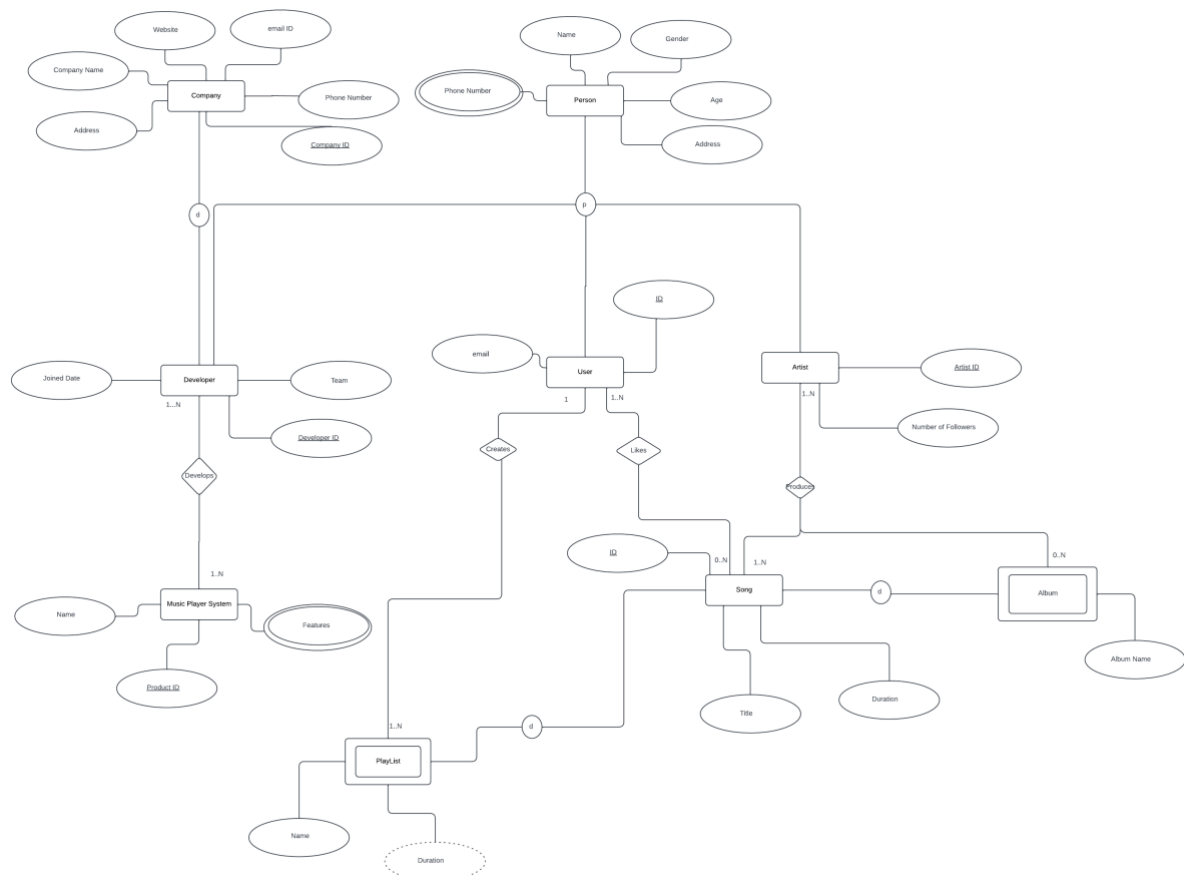
Framework

The significance of this theoretical framework lies in its structured representation of entities and relationships, which mirrors real-world interactions within music streaming platforms. It fosters a sense of community among users, empowering them to create, personalize, and engage with their musical journey. The framework's emphasis on user preferences, likes, and interactions enhances the platform's ability to deliver personalized music experiences. At our music streaming platform's core, users are the foundation of our platform. Users play a pivotal role in creating playlists, forming connections with others, and expressing their musical preferences through likes. Each user can create multiple playlists, establishing a one-to-many relationship. The social network aspect allows users to follow and be followed by others, resulting in a many-to-many relationship known as "Follows." Additionally, users can convey their music preferences by liking songs, albums, and artists, resulting in a one-to-many relationship termed "Liked." Playlists, as user-generated collections of songs, are inherently linked to their creators, forming a many-to-one relationship ("Created_By"). They also share a many-to-many relationship with songs, while songs themselves can be liked by multiple users. Albums represent compilations of songs, creating a one-to-many

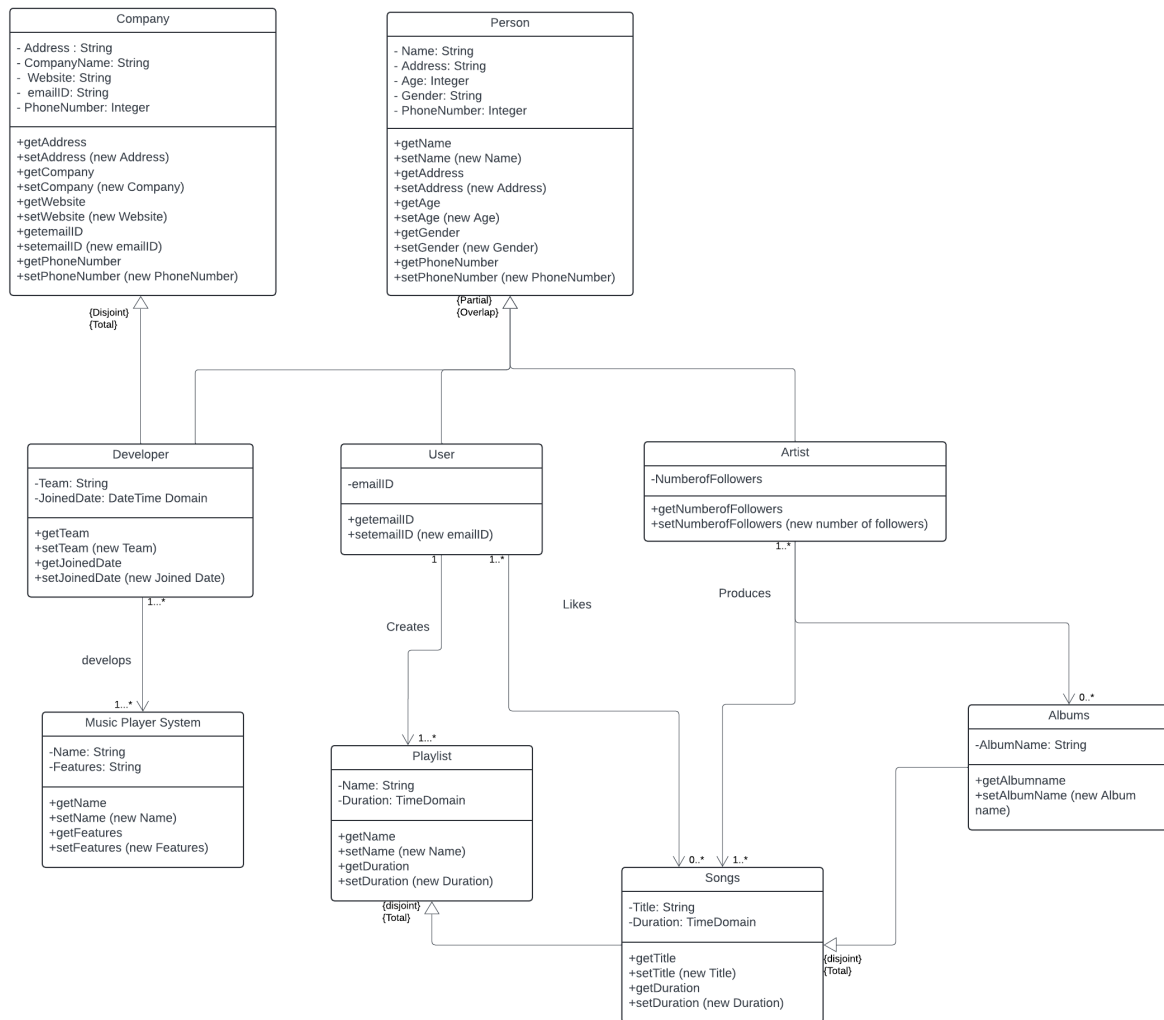
relationship with songs, and artists, in turn, can have multiple songs and albums associated with them. Genres serve as a means to categorize music entities, facilitating exploration and discovery. Genres are interconnected with songs, albums, and artists through a many-to-many relationship, simplifying music exploration. This comprehensive framework underpins the intricate web of interactions and relationships within our music streaming service, offering a multifaceted and interconnected music experience for our users while making music discovery and sharing more entertaining and sociable.

Conceptual Data Modelling

1. EER Diagram



2. UML Diagram



Mapping Conceptual Model to Relational Model

Entities = Bold Text

Primary Key = Underlined Text

Foreign key = Italic Text

Company(Company ID, Company_Name, Website, Address)

Person(PID, Email ID, Name, Phone_number, Age, Gender, Address)

Developer(Developer ID, *PID*, Joined_date)

* Foreign key here is the PID referencing Person Entity also Not Null

Team(Team ID, *DeveloperID*)

*Foreign key Developer ID referencing Developer Entity.

User(ID, *PID*)

*Foreign key PID referencing Person Entity , ID also Not Null

Artist(Artist_ID, Number_of_followers, *PID*)

*Foreign key PID referencing Person Entity , ID also Not Null

Playlist (ID, *UID*, Name)

*Foreign key UID referencing USER Entity also Not Null

Songs (ID, Title, Duration, *ArtistID*)

*Foreign key Artist ID referencing Artist Entity also Not Null

Albums (Album_name, *ArtistID*)

*Foreign key Artist ID referencing Artist Entity also Not Null

Music Player System (Product_ID, Name, Features)

User Likes Songs (*UID*, *SongID*)

*Foreign key UID referencing User Entity also Not Null *Foreign

key Song ID referencing Song Entity also Not Null

Songs In Playlist (*SongID*, *PlaylistID*)

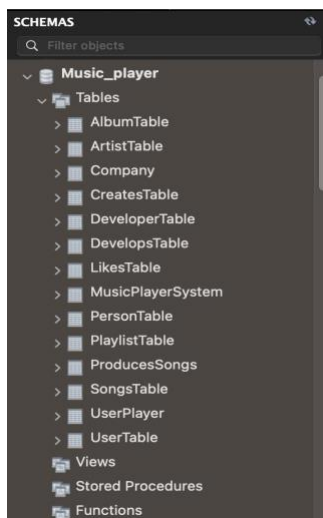
*Foreign key Song ID referencing Song Entity also Not Null

*Foreign key Playlist ID referencing Playlist Entity also Not Null

Implementation of Relation Model via MySQL and NoSQL

Schema (Music_player)

There are total of 15 tables in this schema



Album Table: (PK: AlbumId , columns : Albumid, AlbumName, SongID, Rows : 50)

SongID refers to ID column from SongsTable

1 • select * from AlbumTable

100% 25:1

Result Grid Filter Rows: Search

	AlbumID	AlbumName	SongID
1	1	Greatest Hits	1
2	2	Love Songs	5
3	3	Best of 90s	12
4	4	Classic Rock Anthems	8
5	5	Acoustic Favorites	15
6	6	Pop Extravaganza	21
7	7	Summer Vibes	28
8	8	Indie Showcase	33
9	9	Hip Hop Classics	40
10	10	Jazz Standards	45
11	11	EDM Party Mix	50
12	12	Country Road Trip	2
13	13	Rock Legends	7
14	14	R&B Soulful Sounds	14
15	15	Reggae Roots	20
16	16	Metal Mayhem	26
17	17	Folk Fables	32
18	18	Blues Chronicles	39
19	19	Latin Beats Fiesta	44
20	20	Classical Masterpieces	49
21	21	Electronic Odyssey	3
22	22	Alternative Universe	9
23	23	Piano Serenity	16

ArtistTable: (PK : Artist_ID , COLUMNS : Id, Artist_Id< Followers, PId, Rows : 500)
 In ArtistTable PID refers to PID from PersonTable

1 • select * from ArtistTable

100% 26:1

Result Grid Filter Rows: Search

	id	Artist_ID	Followers	PID
1	1	64	451	
2	2	356	452	
3	3	182	453	
4	4	163	454	
5	5	187	455	
6	6	69	456	
7	7	67	457	
8	8	283	458	
9	9	87	459	
10	10	214	460	
11	11	235	461	
12	12	267	462	
13	13	100	463	
14	14	362	464	
15	15	141	465	
16	16	187	466	
17	17	95	467	
18	18	317	468	
19	19	144	469	
20	20	287	470	
21	21	130	471	
22	22	230	472	
23	23	188	473	

CompanyTable : (PK : CompanyID , Columns: CompanyID, CompanyName, Email , Address, PhoneNumber, Website , Rows: 1)

13 • select * from company

100% 3:8

Result Grid Filter Rows: Search Edit Export/Import

	CompanyID	CompanyName	Email	Address	PhoneNumber	Website
1	1	Sample Music Company	info@samplemusic.com	123 Main Street, Cityville, Country	+1234567890	http://www.samplemusic.com
2	2	NULL	NULL	NULL	NULL	NULL

CreatesTable : (Columns: PlaylistID, UserID, Rows: 500)

In CreatesTable there are 2 foreign keys. PlaylistID refers to PlaylistID from PlaylistTable UserID refers to UID from UserTable

```

1 select * from CreateTable;
2 SELECT
3 *
4 FROM
5 CreateTable
6 ORDER BY
7 playlistid;

```

	PlaylistID	UserID
1	47	
2	205	
3	123	
4	294	
5	312	
6	178	
7	58	
8	259	
9	113	
10	162	
11	291	
12	44	
13	190	
14	320	
15	99	
16	248	
17	321	

DeveloperTable: (PK:Developer_ID , Columns: id, Developer_ID, Joined Date, Role, PID , Rows:87)

PID refers to PID from personTable

```

1 select * from DeveloperTable;
2

```

	id	Developer_ID	Joined Date	Role	PID
1	1		Jan 21, 2021	System Architect/Designer	1
2	2		Sep 13, 2022	System Architect/Designer	2
3	3		Jul 29, 2023	DevOps Engineer	3
4	4		Aug 11, 2022	Business Analyst	4
5	5		Aug 2, 2022	Quality Assurance (QA) Engineer/Tester	5
6	6		Feb 7, 2020	Support and Maintenance	6
7	7		Nov 5, 2021	Developers/Programmers	7
8	8		Apr 2, 2020	Developers/Programmers	8
9	9		Mar 5, 2022	Quality Assurance (QA) Engineer/Tester	9
10	10		Jun 26, 2022	Business Analyst	10
11	11		Apr 12, 2020	Business Analyst	11
12	12		Apr 5, 2020	Quality Assurance (QA) Engineer/Tester	12
13	13		Apr 25, 2023	Database Administrator (DBA)	13
14	14		Jun 10, 2022	System Architect/Designer	14
15	15		Nov 25, 2019	System Architect/Designer	15
16	16		Jul 9, 2021	Database Administrator (DBA)	16
17	17		Sep 24, 2020	User Experience (UX) Designer	17
18	18		Sep 8, 2020	Technical Writer	18
19	19		Jan 28, 2023	Quality Assurance (QA) Engineer/Tester	19
20	20		Mar 12, 2022	Database Administrator (DBA)	20
21	21		Jul 10, 2023	Developers/Programmers	21
22	22		Jan 30, 2022	Support and Maintenance	22
23	23		Dec 15, 2019	System Architect/Designer	23

DevelopsTable : (Columns: DeveloperID, MusicPlayerSystemID , Rows:87)

DeveloperID refers to Developer_ID from DeveloperTable

MusicPlayerSystemID refers to ID from MusicPlayerSystem

```
1 select * from DevelopsTable;
2 SELECT
3 *
4 FROM
5 DevelopsTable
6 ORDER BY
7 DeveloperID;
```

Result Grid

	DeveloperID	MusicPlayerSystemID
1	1	1
2	2	2
3	3	3
4	4	4
5	5	5
6	6	6
7	1	1
8	2	2
9	3	3
10	4	4
11	5	5
12	6	6
13	1	1
14	2	2
15	3	3
16	4	4
17	5	5
18	6	6
19	1	1

LikesTable: (PK: LikeID,Columns: LikeID, UserID, SongId , Rows: 84)

UserID refers to UID from UserTable

SongID refers to ID from SongsTable

```
1 select * from LikesTable;
```

Result Grid

	LikeID	UserID	SongID
1	28	221	
2	32	215	
3	34	204	
4	37	245	
5	45	201	
6	49	195	
7	50	171	
8	53	32	
9	56	56	
10	57	89	
11	58	179	
12	62	7	
13	67	231	
14	70	238	
15	74	56	
16	74	134	
17	78	139	
18	82	198	
19	86	154	
20	89	109	
21	89	127	
22	92	159	
23	92	166	

MusicPlayerSystem:(PK: ID , Columns:ID, ProductName, Features , Rows:6)

MusicPlayerSystem -- ID is foreign key for other tables

```
1 select * from MusicPlayerSystem;
```

Result Grid

	ID	ProductName	Features
1	1	Mobile Music Player	On-the-go music experience, Portable design, S...
2	2	TV Audio Console	Home entertainment hub, High-quality audio out...
3	3	Computer Sound System	Desktop music powerhouse, Customizable equ...
4	4	Tablet Audio Device	Large screen display, Touch interface, Multi-app...
5	5	Web Browser Player	Online music streaming, Cross-platform access,...
6	6	Gaming Console Audio Hub	Immersive gaming audio, Entertainment center i...
	NULL	NULL	NULL

PersonTable:(PK:id , Columns: id,PID,Name,Gender,Age,Phone,Addresss, Rows:500)
 PID is foreign key to other tables

id	PID	Name	Gender	Age	phone	address
1	1	Marsden Levine	Female	19	1-336-250-9427	Ap #492-6962 Curae Road
2	2	Shelly Flores	Male	50	1-884-637-2487	Ap #623-4803 Ligula. Rd.
3	3	Blossom Evans	Male	23	1-995-746-8709	Ap #263-9754 Dui. Ave
4	4	Colt Blackburn	Female	70	1-411-343-7518	Ap #995-9999 Interdum. Road
5	5	Dorian Schwartz	Male	56	1-648-824-1731	Ap #627-1497 Et, Road
6	6	Leo Poole	Male	59	1-188-980-6441	8750 Et, Rd.
7	7	Lois Sparks	Male	51	1-756-772-1377	3202 Id, Avenue
8	8	Lester Berger	Female	40	1-776-679-3394	233-9513 Euismod St.
9	9	Echo Hill	Female	27	1-535-868-4721	864-6283 Integer St.
10	10	Wesley Gallagher	Male	45	1-607-135-4688	Ap #846-7668 Tristique Rd.
11	11	Myra Gregory	Female	68	1-405-728-2566	P.O. Box 779, 9358 Morbi R...
12	12	Scarlett Mccray	Female	31	1-761-708-1731	P.O. Box 945, 715 Magna Rd.
13	13	Aiko Norman	Male	26	1-141-670-3418	Ap #134-463 Sem, Road
14	14	Cedric Powell	Male	30	1-642-783-0153	Ap #190-6396 Nonummy Ave
15	15	Cruz Davidson	Male	39	1-844-613-6816	Ap #823-5290 Placerat, Street
16	16	Branden Pearson	Female	64	1-818-956-0830	2780 Metus. Av.
17	17	Georgia Palmer	Male	20	1-961-279-2645	P.O. Box 548, 3756 Purus Ave
18	18	Blythe Greer	Female	36	1-358-792-4087	5302 Sapient. Av.
19	19	Ruth Mcguire	Male	72	1-210-888-6579	125-2407 Sollicitudin Ave
20	20	Samuel Cervantes	Male	42	1-167-423-9615	9526 Nam Avenue
21	21	Brianna Dunn	Male	47	1-665-927-2673	P.O. Box 618, 8827 Dui. St.
22	22	Uta Eaton	Female	46	1-242-385-2595	P.O. Box 952, 5908 Suspend...
23	23	Thomas Porter	Female	65	1-633-561-5576	An #110-8235 Nunc Rd.

PlaylistTABLE:(Columns:DEVELOPERID,MUSICPLAYERSYSTEMID , Rows:87)
 DeveloperID refers to Developer_ID from DeveloperTable
 MusicPlayerSystemID refers to ID from MusicPlayerSystem

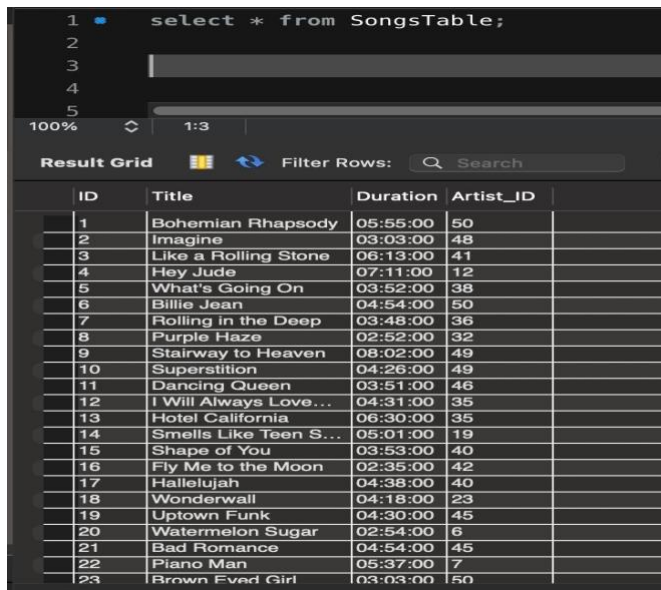
DeveloperID	MusicPlayerSystemID
1	1
2	2
3	3
4	4
5	5
6	1
7	2
8	3
9	4
10	5
11	6
12	1
13	2
14	3
15	4
16	5
17	6
18	1
19	2
20	3
21	4
22	5

ProducesSongs: (Columns: ProducesID,SongID,ArtistID , Rows: 250)
 SongID refers to ID from SongsTable
 ArtistID refers to Artist_ID from ArtistTable

ProducesID	SongID	ArtistID
1	1	5
2	2	23
3	3	17
4	4	42
5	5	1
6	6	8
7	7	13
8	8	29
9	9	19
10	10	36
11	11	2
12	12	44
13	13	11
14	14	49
15	15	7
16	16	15
17	17	26
18	18	50
19	19	33
20	20	10
21	21	14
22	22	35
23	23	21

SongsTable:(PK: ID , Columns: ID, Title, duration,Artist_ID , Rows:250)

ArtistID refers to Artis_ID from ArtistTable

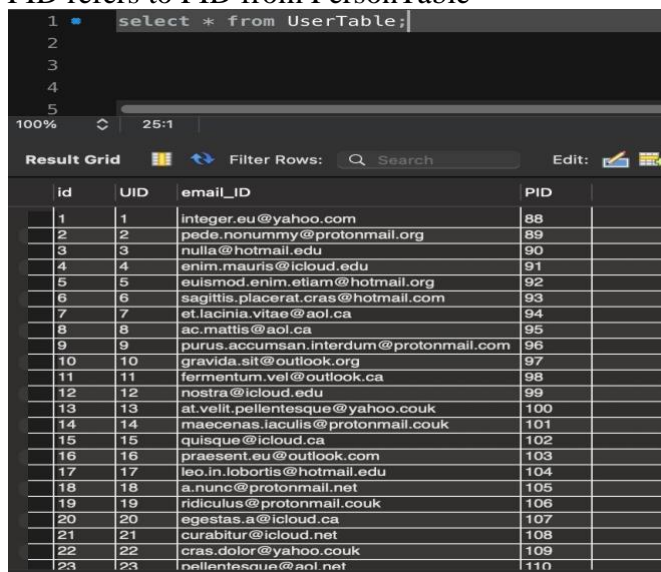


```
1 select * from SongsTable;
```

ID	Title	Duration	Artist_ID
1	Bohemian Rhapsody	05:55:00	50
2	Imagine	03:03:00	48
3	Like a Rolling Stone	06:13:00	41
4	Hey Jude	07:11:00	12
5	What's Going On	03:52:00	38
6	Billie Jean	04:54:00	50
7	Rolling in the Deep	03:48:00	36
8	Purple Haze	02:52:00	32
9	Stairway to Heaven	08:02:00	49
10	Superstition	04:26:00	49
11	Dancing Queen	03:51:00	46
12	I Will Always Love...	04:31:00	35
13	Hotel California	06:30:00	35
14	Smells Like Teen S...	05:01:00	19
15	Shape of You	03:53:00	40
16	Fly Me to the Moon	02:35:00	42
17	Hallelujah	04:38:00	40
18	Wonderwall	04:18:00	23
19	Uptown Funk	04:30:00	45
20	Watermelon Sugar	02:54:00	6
21	Bad Romance	04:54:00	45
22	Piano Man	05:37:00	7
23	Brown Eyed Girl	03:03:00	50

User:(PK:id , Columns: id,UID,emailID,PID , Rows:363)

PID refers to PID from PersonTable



```
1 select * from UserTable;
```

id	UID	email_ID	PID
1	1	integer.eu@yahoo.com	88
2	2	pede.nonummy@protonmail.org	89
3	3	nulla@hotmail.edu	90
4	4	enim.mauris@icloud.edu	91
5	5	euismod.enim.etiam@hotmail.org	92
6	6	sagittis.placerat.cras@hotmail.com	93
7	7	et.iacina.vitae@aol.ca	94
8	8	ac.mattis@aol.ca	95
9	9	purus.accumsan.interdum@protonmail.com	96
10	10	gravida.sit@outlook.org	97
11	11	fermentum.vel@outlook.ca	98
12	12	nostra@icloud.edu	99
13	13	at.velit.pellentesque@yahoo.couk	100
14	14	maecenas.iaculis@protonmail.couk	101
15	15	quisque@icloud.ca	102
16	16	praesent.eu@outlook.com	103
17	17	leo.in.lobortis@hotmail.edu	104
18	18	a.nunc@protonmail.net	105
19	19	ridiculus@protonmail.couk	106
20	20	egestas.a@icloud.ca	107
21	21	curabitur@icloud.net	108
22	22	cras.dolor@yahoo.couk	109
23	23	pellentesque@aol.net	110

Implementation in NoSQL

1) This query filtered the Album details with the AlbumName "Indie Showcase"

```
> db.AlbumTable.find({
  AlbumName: 'Indie Showcase'
})
< {
  _id: ObjectId("656d4755a5ce939e3d699401"),
  AlbumID: 8,
  AlbumName: 'Indie Showcase',
  SongID: 33
}
MusicPlayerSystem> |
```

2) This query filters out all the males whose age is between 23 and 35

```
> _MONGOOSH
> db.PersonTable.find({
  Gender: 'Male',
  Age: {
    $gte: 23,
    $lte: 35
  }
});
< [
  {
    _id: ObjectId("656d4c24a5ce939e3d699732"),
    id: 3,
    PID: 3,
    Name: 'Blossom Evans',
    Gender: 'Male',
    Age: 23,
    phone: '1-995-746-8709',
    address: 'Ap #263-9754 Dui. Ave'
  },
  {
    _id: ObjectId("656d4c24a5ce939e3d69973c"),
    id: 13,
    PID: 13,
    Name: 'Aiko Norman',
    Gender: 'Male',
    Age: 26,
    phone: '1-141-670-3418',
    address: 'Ap #134-463 Sem, Road'
  },
  {
    _id: ObjectId("656d4c24a5ce939e3d69973d"),
    id: 14,
    PID: 14,
    Name: 'Cedric Powell',
    Gender: 'Male',
    Age: 30,
    phone: '1-642-783-0153',
  }
]
```

3) This query gives the song Title and number of likes it got

```
>_MONGOSH
> db.LikesTable.aggregate([
  {
    $lookup: {
      from: "SongsTable",
      localField: "SongID",
      foreignField: "ID",
      as: "songData"
    }
  },
  {
    $group: {
      _id: "$SongID",
      count: { $sum: 1 },
      songDetails: { $first: "$songData" }
    }
  },
  {
    $project: {
      _id: 0,
      SongName: "$songDetails.Title", // Replace with the actual field name in SongsTable
      Count: "$count"
    }
  }
]);
< {
  SongName: [
    'Unchained Melody'
  ],
  Count: 1
}
{
  SongName: [
    'My Way'
  ],
  Count: 1
}
```

4) This query combines CreatesTable with PlaylistTable and UserTable

```
>_MONGOSH
> db.CreatesTable.aggregate([
  {
    $lookup: {
      from: "PlaylistTable",
      localField: "PlaylistID",
      foreignField: "PlaylistID",
      as: "playlistData"
    }
  },
  {
    $lookup: {
      from: "UserTable",
      localField: "UserID",
      foreignField: "UID",
      as: "userData"
    }
  }
]);
< {
  _id: ObjectId("656d48eca5ce939e3d699466"),
  PlaylistID: 39,
  UserID: 11,
  playlistData: [
    {
      _id: ObjectId("656d4c3fa5ce939e3d69994b"),
      PlaylistID: 39,
      Name: 'Pop Divas Anthems',
      Duration: '03:10:00',
      CreatorID: 237
    }
  ],
  userData: [
    {
      _id: ObjectId("656d4caca5ce939e3d699e86"),
      id: 11,
      UID: 11,
      email_ID: 'fermentum.vel@outlook.ca',
      PID: 98
    }
  ]
}
```

```
>_MONGOSH
},
  userData: [
    {
      _id: ObjectId("656d4caca5ce939e3d699e86"),
      id: 11,
      UID: 11,
      email_ID: 'fermentum.vel@outlook.ca',
      PID: 98
    }
  ]
}
{
  _id: ObjectId("656d48eca5ce939e3d699467"),
  PlaylistID: 89,
  UserID: 11,
  playlistData: [
    {
      _id: ObjectId("656d4c3fa5ce939e3d69997d"),
      PlaylistID: 89,
      Name: 'Funky Disco Grooves',
      Duration: '02:40:00',
      CreatorID: 166
    }
  ],
  userData: [
    {
      _id: ObjectId("656d4caca5ce939e3d699e86"),
      id: 11,
      UID: 11,
      email_ID: 'fermentum.vel@outlook.ca',
      PID: 98
    }
  ]
}
{
  _id: ObjectId("656d48eca5ce939e3d699468"),
  PlaylistID: 89,
  UserID: 11,
  playlistData: [
    {
      _id: ObjectId("656d4c3fa5ce939e3d69997d"),
      PlaylistID: 89,
      Name: 'Funky Disco Grooves',
      Duration: '02:40:00',
      CreatorID: 166
    }
  ],
  userData: [
    {
      _id: ObjectId("656d4caca5ce939e3d699e86"),
      id: 11,
      UID: 11,
      email_ID: 'fermentum.vel@outlook.ca',
      PID: 98
    }
  ]
}
```

5) This query gives us Name, Gender, Age, Phone, Address of users from age 21 and 35 in ascending order and groups people of same age

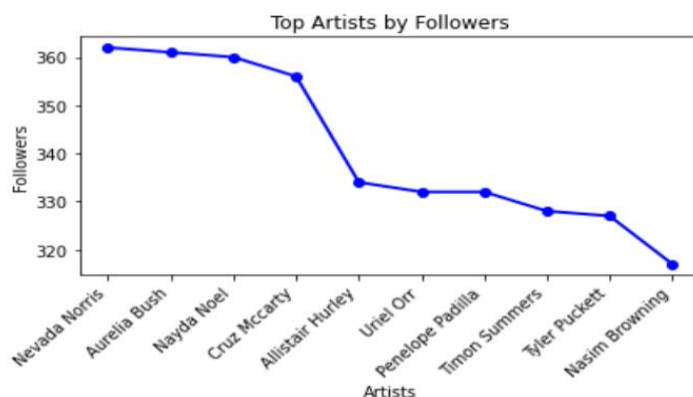
```
> db.PersonTable.aggregate([
  {
    $match: {
      Age: { $gte: 21, $lte: 35 }
    }
  },
  {
    $lookup: {
      from: "UserTable",
      localField: "PID",
      foreignField: "PID",
      as: "userData"
    }
  },
  {
    $group: {
      _id: "$Age",
      persons: { $push: "$$ROOT" },
      userData: { $first: "$userData" }
    }
  },
  {
    $project: {
      _id: 0,
      "persons.Name": 1,
      "persons.Gender": 1,
      "persons.Age": 1,
      "persons.phone": 1,
      "persons.address": 1
    }
  },
  {
    $sort: {
      "_id": 1
    }
  }
]);
```

```
Age: 23,
phone: '1-995-746-8709',
address: 'Ap #263-9754 Du1. Ave'
},
{
  Name: 'Valentine Henderson',
  Gender: 'Female',
  Age: 23,
  phone: '1-316-695-5500',
  address: '5702 In Road'
},
{
  Name: 'Tamara Holland',
  Gender: 'Male',
  Age: 23,
  phone: '1-206-184-4431',
  address: 'P.O. Box 620, 1990 Orci Avenue'
},
{
  Name: 'Astra Murray',
  Gender: 'Male',
  Age: 23,
  phone: '1-561-635-6589',
  address: '290-2880 Aliquam St.'
},
{
  Name: 'Kyle Simpson',
  Gender: 'Female',
  Age: 23,
  phone: '1-412-931-2767',
  address: '214-2298 Quisque Avenue'
},
{
  Name: 'Brent Cruz',
  Gender: 'Female',
  Age: 23,
  phone: '1-627-338-3513',
  address: '5850 Non St.'
}
]
persons: [
```

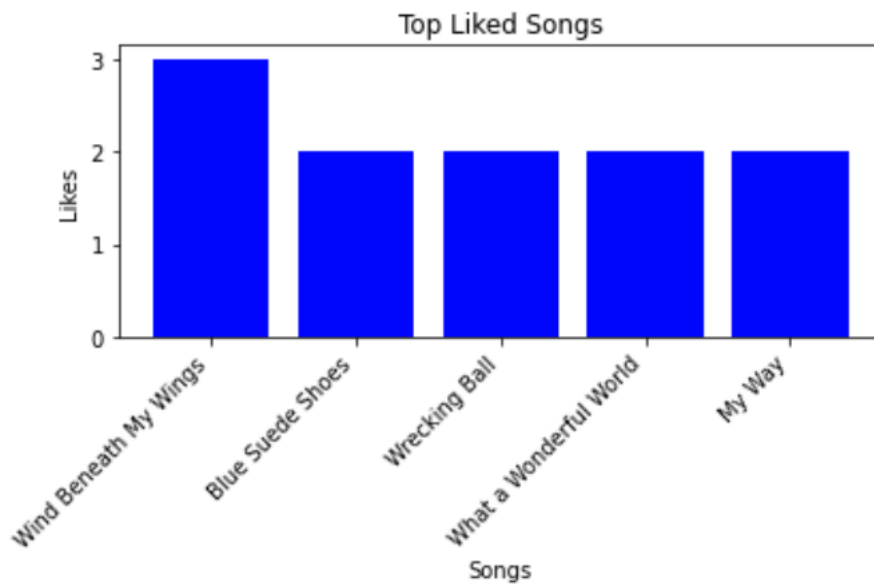
Database Access via Python IDE.

The database is accessed to jupyter notebook by means of mysql connector. Various SQL queries were executed through cursor.connection() and the corresponding query outputs were converted into dataframes for further analysis. Libraries like matplotlib and seaborn were used to generate visualizations to give meaningful insights to the data.

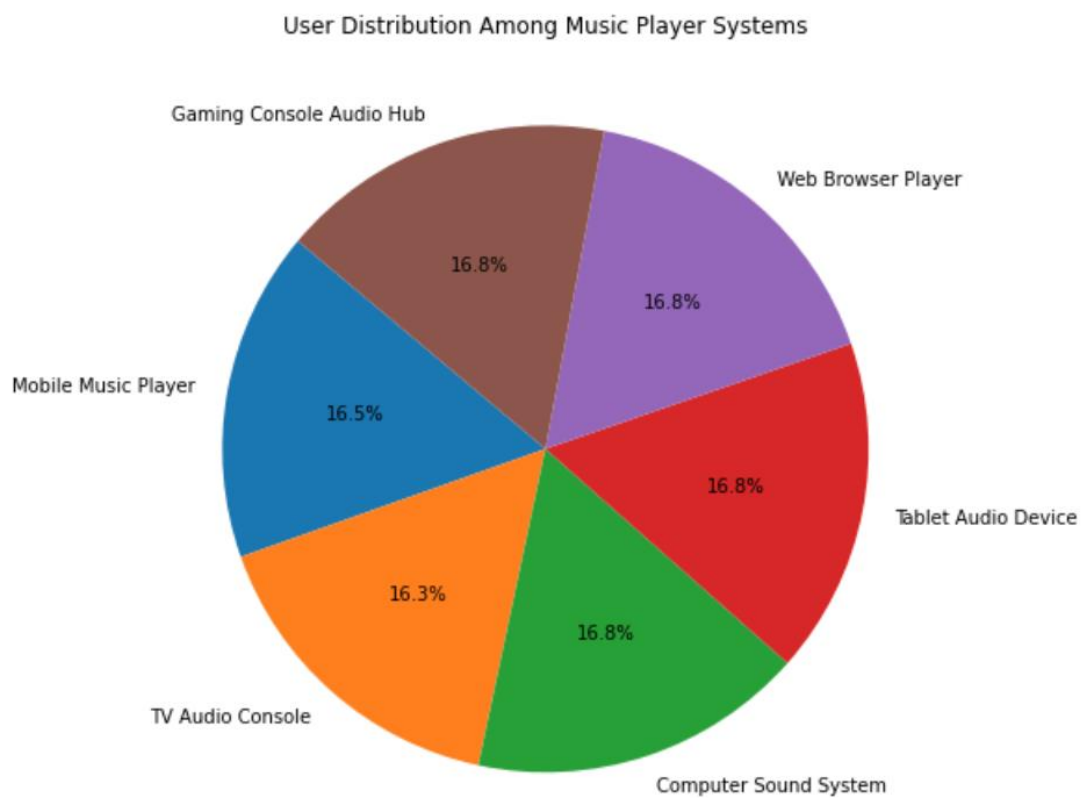
Graph – 1: Top 10 Artists based on number of followers



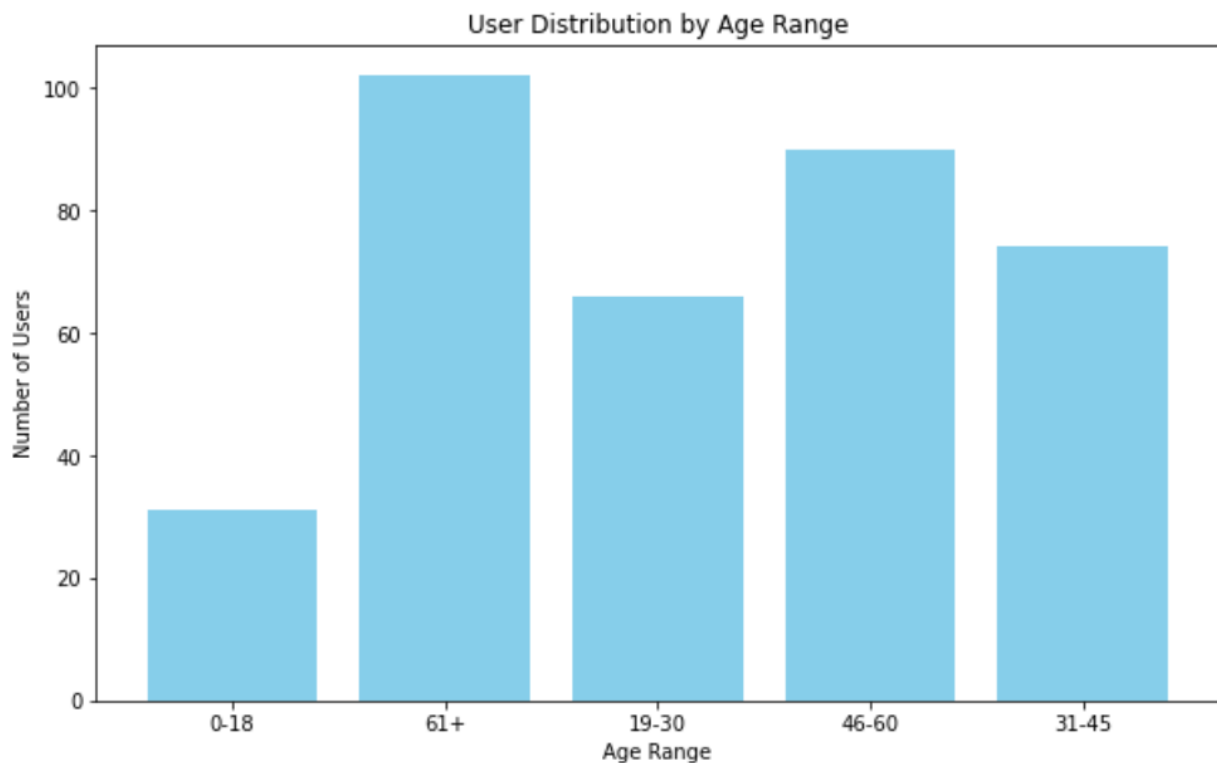
Graph – 2: Top 5 songs based on number of likes of the song



Graph – 3: User distribution among music player systems



Graph – 4: User distribution by age range



Conclusion:

In conclusion, our project endeavors to redefine the music streaming experience by merging digital innovation, user-centric design, and data analysis. The development of our innovative music streaming and recommendation platform seeks to address the overwhelming choices in the vast music landscape. This platform is envisioned as an online soundscape where users can not only discover and appreciate but also personalize their musical journey.

The framework underlying our project emphasizes the significance of user interactions, preferences, and community-building. By incorporating features like song liking, artist following, and playlist creation, we aim to provide a more engaging and user-driven music experience. The marriage of data analysis with the social aspects of music appreciation creates a unique platform where individual enjoyment is enhanced, and a global community of music enthusiasts is connected.

The conceptual data model, represented through both EER and UML diagrams, provides a structured overview of entities and relationships within our music streaming service. The mapping of the conceptual model to the relational model illustrates how data is organized in tables, fostering a clear understanding of the database structure.

The implementation of the relational model in MySQL and NoSQL (MongoDB) further solidifies our commitment to providing a robust and scalable platform. The MySQL schema, consisting of 15 tables, demonstrates the intricate relationships between entities. Meanwhile, NoSQL queries showcase the flexibility and adaptability of our database design.

Database access through Python IDE, specifically Jupyter Notebook, facilitates efficient querying and analysis. Visualizations generated from the database data, such as top artists, popular songs, user distribution among music player systems, and age range distribution, offer valuable insights into user behavior and preferences.

In summary, our project not only aspires to reinvent the music streaming experience but has laid a comprehensive foundation for a platform where users can truly connect with music on a personal and communal level. The integration of data analysis, user-centric design, and a robust database structure positions our project at the forefront of the evolving landscape of music discovery and enjoyment.