# Features and Functionality to Test:

1. Registration
2. Login
3. Homepage
4. Password Change

CEN 4010 Principles of Software Engineering

Milestone 3

**CoreShare**

Core5

Team Number: 25

October 19, 2021

**Team Members**

Mac Consolabe

mconsolabe2020@fau.edu

-

Victor Marinho Correa

jmarinhocorrea2021@fau.edu

-

Greydi Lora

Glora2019@fau.edu

-

Dominic Robbins

Drobbins2019@fau.edu

-

Michael Cevallos

Mcevallos2020@fau.edu

**History Revision Table**

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Revision | Actions | Performed by |
| 2021-10-19 | 1 | Made changes according to TA feedback | Group 25 |
| 2021-10-23 | 2 | Added diagrams | Group 25 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Table of Contents**

**Table of Contents**

[***1.***](#_gjdgxs) ***Executive Summary 4***

[***2.***](#_30j0zll) ***Competitive analysis 5***

[***3.***](#_1fob9te) ***Data definition 6***

[***4.***](#_3znysh7) ***Overview, scenarios and use cases 10***

[***5.***](#_2et92p0) ***Initial list of high-level functional requirements 16***

[***6.***](#_tyjcwt) ***List of non-functional requirements 17***

[***7.***](#_3dy6vkm) ***High-level system architecture (UML) 18***

[***8.***](#_1t3h5sf) ***Identify risk and actions 19***

[***9.***](#_4d34og8) ***Vertical demo 19***

# Executive Summary

CoreShare is a social media web application that will be geared towards connecting people across the world that are far away from each other, to remind them, and help them reminisce on old memories, with the ability to create new ones. Throughout a time like we’re in right now, with COVID, there are a lot of different platforms that are aimed to connect people, mind you people probably won't even need a social media platform, they can just call each other. but with CoreShare the aim is to give them the opportunity to connect on a level that's deeper than just a phone call, or a simple text message, or even a tweet.

One of the biggest ways to connect with people that exists in this world is music.

Emotions and feelings sometimes cannot be described simply with words, being far away from your loved ones and not being able to talk & connect can sometimes make you lost for words for when you do get in touch with them, but sometimes just one song shared can completely transform this melancholy into happiness by reminiscing on old memories or even by creating a bridge of sympathy between yourself & another.

CoreShare’s goal is to develop that bridge, a bridge with many lanes, not simply to share sad feelings with others, but also to transfer more upbeat feelings with others as we’re stuck at home most of the day, yearning for something to do, experience, & feel, by connecting Music and People on a level that has never been done before. The purpose of this application is to allow you, me, & everyone else to have the ability to connect with our loved ones by just sharing a song, and it doesn’t stop there because the app is made for CoreConnection, not only by sympathy, but by curiosity, simply wanting to try something new, perhaps feel something new, and by new we mean new music.

We will give you the ability to not only connect directly with your friends and family but also to connect indirectly by providing you with the ability to learn more about your family and friends taste in music, even if it doesn’t involve a deeper connection. Everyone will be able to connect in such a way where, so long as someone is in your CoreCircle on the platform, they can tap into your experience & feelings by seeing what you are currently listening or have listened to in the past and in the process develop their own Unique experience & feelings toward new music.

All the experience, connection and sharing will be made inside the platform, without having to leave and grab a song somewhere else and come back. Everything will be within your grasp, your CoreCircle and music, all in the same place. Connection, love, harmony is the fuel for the application, but giving you a good user experience is also in our top priority.

# Competitive analysis

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Features/Apps** | **CoreShare** | **Twitter** | **Facebook** | **Spotify** | **YouTube Red** | **Tidal** | **Apple Music** |
| **Send Direct Messages** | **X** | **X** | **X** |  |  |  |  |
| **Music** | **X** |  |  | **X** | **X** | **X** | **X** |
| **In-app Music sharing** | **X** |  |  |  |  |  |  |
| **Free to use** | **X** | **X** | **X** | **X** |  |  |  |
| **Ad-Free** | **X** |  |  |  | **X** | **X** | **X** |
| **Send/Receive Friend Requests** | **X** | **X** | **X** |  |  |  |  |
| **Create Posts** | **X** | **X** | **X** |  |  |  |  |

The ability to communicate already exists within one of our competitors’ products, such as Twitter and the ability to give people music as well, such as with Spotify, but having the ability to do both within a single platform is what CoreShare is about. CoreShare brings the ability of sending direct messages, sharing music, creating posts, and sending/receiving friend requests together.

With the In-App music sharing feature, CoreShare’s advantages on these two platforms is its biggest strength. CoreShare requires no subscription to use unlike YouTube Red, Tidal, and Apple Music. Twitter, Facebook, and Spotify all display ads on their platform, but CoreShare does not. Bringing something that is already available is easy but bringing something new to the table is not and this is what we’re doing at CoreShare.

# Data definition

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Meaning** | **Usage** | **Comment** |
| User | Actor | Use case scenarios | Anyone who uses the application and can share music, add friends, remove friends, comment, like and write posts |
| CoreCircle | Actor | Use case scenarios | Friends in your circle in which you both have accepted a friend request from |
| Friend | Actor | Use case scenarios | Friend that is included in the CoreCircle |
| Profile | User Interface | User Interface | Display the users’ favorite songs and posts |
| Home page | User Interface | User Interface | The main page displayed first seen by user |
| Register | Service | Site user service | User will create an account using an email and password. Will have access to all features |
| Login | Service | Site user service | Users can access their account using their email and password that is already registered into the system |
| Search | Service | Site user service | Allow user to look up other users and send friend requests |
| Post Music | Service | Site user service | Allow user to post a song and optionally include text |
| Comment | Service | Site user service | Allow user to reply to a post |
| Post Photo | Service | Site user service | Allow user to post an image |
| View Music | Service | Site user service | View what music the user’s friends are listening to including their history |
| Send Friend  Request | Service | Site user service | Requests another user to be added to their CoreCircle and vice versa |
| Accept Friend Request | Service | Site user service | Users are added to their friend’s CoreCircle. The user’s friend is added to their CoreCircle |
| Remove Friend  Request | Service | Site user service | Removes friend from your CoreCircle including theirs |
| Account | Data | Use case scenarios | Store users’ information |
| Post Video | Service | Site user service | Allows user to post a video |
| Post Image | Service | Site user service | Allows user to post an image |
| Send message | Service | Site user service | Send a message to other users |
| Messages | Data | Use case scenarios | Access messages sent or received by other users in their CoreCircle. |
| Name | Data | Use case scenarios | The users full name |
| Friends | Data | Use case scenarios | The list of friends in the users CoreCircle |
| Deny Friend Request | Service | Site user service | User cannot be added to CoreCircle |
| Email | Data | Use case scenarios | The user’s email for the website |
| Password | Data | Use case scenarios | The user’s password for the website |
| CoreShare | Domain name | Use case scenarios | Represents all web pages and website |

# Overview, scenarios and use cases

**Use Case – Create Account**

The first main use case of the product will be the ability for users to have their own account, and within that account users will be able to add people such as family, friends, and even people that they meet on the platform. Once the user has an account and friends linked to his account the user will be able to unlock other features such as being able to create a CoreCircle.

1. **Description:**

Describes how a user will visit the application and create an account.

1. **Actors:**
2. User
3. System
4. **Preconditions:**
5. User has an internet connection
6. System is functioning
7. **Primary Flow of Events:**
8. User visits the web page
9. User clicks on create an account
10. Account is created
11. Terminate use case: Create Account
12. **Alternate Flows:**
13. If their input details match an account that already exists, they will be notified
14. If their input details are invalid, such as using the wrong syntax/key, they will be notified

**Use Case – Send Friend Request**

The CoreCircle is a feature that allows the user to be able to view, listen, and communicate with a select group of people that the user had added in their CoreCircle that are considered the user’s closest friends. This feature will allow the user to view, listen, and communicate with their closest friends in a private manner where everyone will be aware of who is in their CoreCircle and whose CoreCircles they’re in too. The CoreCircle Request will be like any other friend request, except accepting it means, you’re letting me into your music world, such that if you send a CoreCircle request, the other person must accept it to add them to your CoreCircle, this adds yourself to their CoreCircle as well.

1. **Description:**

Describes how a user will send a friend request to another user.

1. **Actors:**
2. User
3. System
4. **Preconditions:**
5. User has an internet connection
6. System is functioning
7. User has an active account
8. User is logged in
9. **Primary Flow of Events:**
10. User visits the web page
11. User uses the search bar to search for another user
12. User clicks on the other user of interest
13. User clicks on send friend request
14. If user isn’t logged in, go to Alternative Flows part a
15. A friend request is sent to requested user
16. Terminate use case: Send Friend Request
17. **Alternate Flows:**

a. If the user isn’t logged in, a web prompt appears notifying the user to login or register.

**Use Case – Share Music**

User arrives to the home page and wants to share a specific song with everyone that is included in their CoreCircle

1. **Description:**

Describes how a user will share a song with everyone.

1. **Actors:** 
   1. User
   2. System
2. **Preconditions:**
   1. User has an internet connection
   2. System is functioning
   3. User has an active account
   4. User is logged in
3. **Primary Flow of Events:**
   1. User visits the web page and logs in or creates an account
   2. User shares a specific song
   3. The song is listed as a post shared amongst others
   4. Terminate use case: Share Music
4. **Alternate Flows:**

a. If the user isn’t logged in, a web prompt appears notifying the user to login or register.

**Use Case – Send Direct Message**

The user will have access to the Direct Message section, where they can communicate with a specific person, including regarding a specific song that they may have posted, or that they may want to send, and start a conversation.

1. **Description:**

Describes how a user will send a direct message.

1. **Actors:** 
   1. User
   2. System
2. **Preconditions:**
   1. User has an internet connection
   2. System is functioning
   3. User has an active account
   4. User is logged in
   5. User is friends with the person they are messaging
3. **Primary Flow of Events:**
   1. User visits the web page and logs in or creates an account
   2. User visits the Direct Messages tab
   3. User sends a message to another user that is included in their CoreCircle
   4. Terminate use case: Send Direct Message
4. **Alternate Flows:**
   1. If the user isn’t logged in, a web prompt appears notifying the user to login or register.
   2. If the person the user wants to send a message to is not part of their CoreCircle, they will be notified. The user cannot send a message and will have to send a friend request first.

**Use Case – Comment**

User arrives at the home page and wants to add a comment on another users’ post.

1. **Description:**

Describes how a user will add a comment onto a post.

1. **Actors:** 
   1. User
   2. System
2. **Preconditions:**
   1. User has an internet connection
   2. System is functioning
   3. User has an active account
   4. User is logged in
3. **Primary Flow of Events:**
   1. User visits the web page and logs in or creates an account
   2. User sees a post they want to add a comment to and clicks on the Comment button.
   3. A field displays so the user can enter a message.
   4. User submits the comment when they are finished.
   5. Terminate use case: Comment
4. **Alternate Flows:**

a. If the user isn’t logged in, a web prompt appears notifying the user to login or register.

**Use Case – Search**

User arrives at the home page and wants to search for another user.

1. **Description:**

Describes how a user will search for another user

1. **Actors:** 
   1. User
   2. System
2. **Preconditions:**
   1. User has an internet connection
   2. System is functioning
   3. User has an active account
   4. User is logged in
3. **Primary Flow of Events:**
   1. User visits the web page and logs in or creates an account
   2. User clicks on the search bar
   3. User types in the name of another user
   4. Web page displays the search results
   5. User clicks on that other user
   6. Terminate use case: Search
4. **Alternate Flows:**
   1. If the user isn’t logged in, a web prompt appears notifying the user to login or register.
   2. If the person that the user searches for does not exist, the user will be notified with a message.

# Initial list of high-level functional requirements

1. **Creating Account**
   1. Users can sign up for an account and their user ID, password, date of birth, first name, and last name will be stored.
2. **Sign In**
   1. User can log into account
3. **Search Friends**
   1. Users can friend other users by searching for them in the friend search bar
4. **Create a Post**
   1. User can post a song with or without a message for followers to see
5. **Delete a post**
   1. User can delete posts
6. **Send Friend Request**
   1. User can send their first CoreCircle request to create your CoreCircle
7. **Accept, Deny, Ignore Friend Requests**
   1. User can accept, deny, or ignore CoreCircle requests sent to them
8. **Added to CoreCircle**
   1. If user accepts friend’s CoreCircle request, then they are added to their friend’s CoreCircle & their friend gets added to their own CoreCircle
9. **Remove from CoreCircle**
   1. User can remove a friend from their CoreCircle, which removes themselves from their friend’s CoreCircle
10. **View CoreCircle Music**
    1. Users can view what their CoreCircle friends are listening to currently, if they are, or what they have listened to recently.
11. **Direct Message**
    1. Users can reply to their friend’s post by direct message

# List of non-functional requirements

**Performance Requirements:**

1. *Availability*: The very first requirement is that the service should be highly available. The users should be able to send songs and should be able to see other users’ songs posted on their home timeline without any downtime.

2. *Execution Speed*: Generating the timeline should be fast within half a second (at most)

3. *Consistency*: The system does not need to be strongly consistent--eventual consistency is fine.

* It will be okay to see other user songs post after some time

4. *Scalability*: The system should be scalable with the increasing data load presented on the backend.

* Increasing user
* Increasing song post

5. *Durability*: User data (user information and song post/ and friends) should be durable.

* It would not be good if a user login and find out that some of his/her songs post are gone or the users he was friends with, he is not long friend with them anymore

# High-level system architecture (UML)

1) DB tables:

1. “Users” -

* Userid - ID associated with user’s account
* Username - Username associated with user’s account
* Passwordhash - Password associated with user’s account
* Mobile - Mobile phone number associated with user’s account
* Email - Email associated with user’s account
* Registered - Date user registered with CoreShare
* Lastlogin - Date user last logged into CoreShare account

1. “Profile” -

* Proid - ID associated with user’s profile
* Userid - references Userid from “Users” table
* Details - Profile details

1. “User\_photos” -

* Photoid - ID associated with user’s photo post
* Proid - references Proid from “Profile” table

1. “Photos” -

* Pid - ID associated with photo post
* Userid - references Userid from “Users” table
* Photo\_title - title associated with photo post
* Photo\_filename - file name associated with photo post
* Caption - text caption associated with photo post

1. “Videos” -

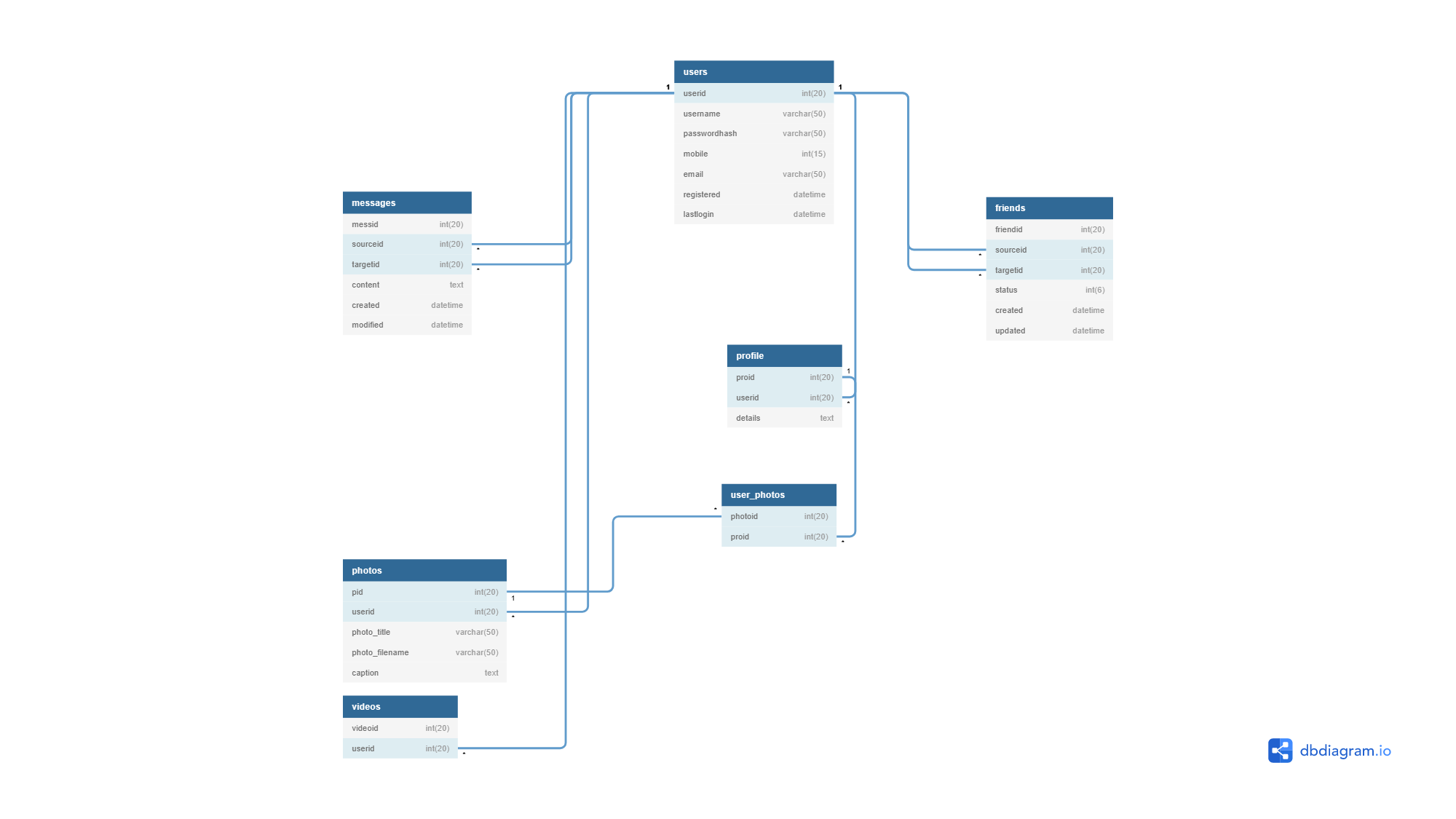
* Videoid - ID associated with user’s video post
* Userid - references Userid from “Users” table

1. “Friends” -

* Friendid - ID associated with user friend request
* Sourceid - ID associated with user who’s the source of friend request
* Targetid - ID associated with user who’s the target of friend request
* Status - status associated with existing friend requests (accepted, rejected, etc.)
* Created - date friend request created
* Updateid - date friend request status updated

1. “Messages” -

* Messid - ID associated with message
* Sourceid - ID associated with user who’s the source of message
* Targetid - ID associated with user who’s the target of message
* Content - text associated with message content
* Created - date message created/sent
* Modified - date message modified



2) Media storage

As we will be using multiple APIs, our media is mostly going to be on the system end.

We will mostly be using Audio and images, for the audio as they’re not going to be on the databases, once pulled from the API whichever format the audio was on their database, it will be the same format on our database. Usually, audio files are either .wav or .mp3.

We will store the profile picture on our database, as we will grab them from the users, same as their logins. The format for the picture will either be .jpg or .png.

We won't use videos as the platform is mainly for audio sharing, nor will we use GPSs.

Videos and photos will be stored with the website's file system. The file name will be stored in the database under the appropriate table.

3) Search/filter architecture and implementation: what will be the algorithm for

search: what DB terms will be searched, how it will be coded and organized

in the DB. Similarly, say what DB items will be filtered/sorted

For the Search Algorithm, we will have a simple search algorithm that will filter in artist name, song titles, album names, and usernames. We will grab all the information related to the artists,

Such as the songs, and albums from the APIs, but any user information will be grabbed from the API.

4) Your own APIs: Describe and define at high level any major APIs that you will

Create

We are aiming to create three majors APIs, which all well do the same thing, but it will be just from

Different platforms, just so we could give the user a wide range of choices.

We will have three different APIs that will all three do the same thing which is pulling

Songs from the targeted database and to infiltrate it, to our database.

One API will grab from Spotify

One API will grab from Soundcloud

One API will grab from YouTube.

5) Describe any significant non-trivial algorithm or process (like rating, ranking,

automatic prioritizing of items etc.)

One Main Trivial algorithm we were thinking of using, was a feature called “Share Charts”.

Like any other charts such as billboard charts, Spotify, apple music, tidal etc. The principles are always the same where the song with the most streams or the song people buy the most will be at the top, but with this chart our goal is to emphasize on the sharing feature. Instead of doing a top chart with the most stream or most listen we will do one based on the most shared songs.

The charts will be like any other charts except your songs need to be shared a lot by the users to make it to the top.

Lists of main software products:

* Visual Studio Code: We will use this to code the application.
* GitHub: Source control and collaboration.
* VMWare: needed to access server
* WinSCP: needed to access server
* MySQL: database construction

Languages:

* Web languages
  + HTML
  + CSS
  + JavaScript
  + PHP
* SQL

API

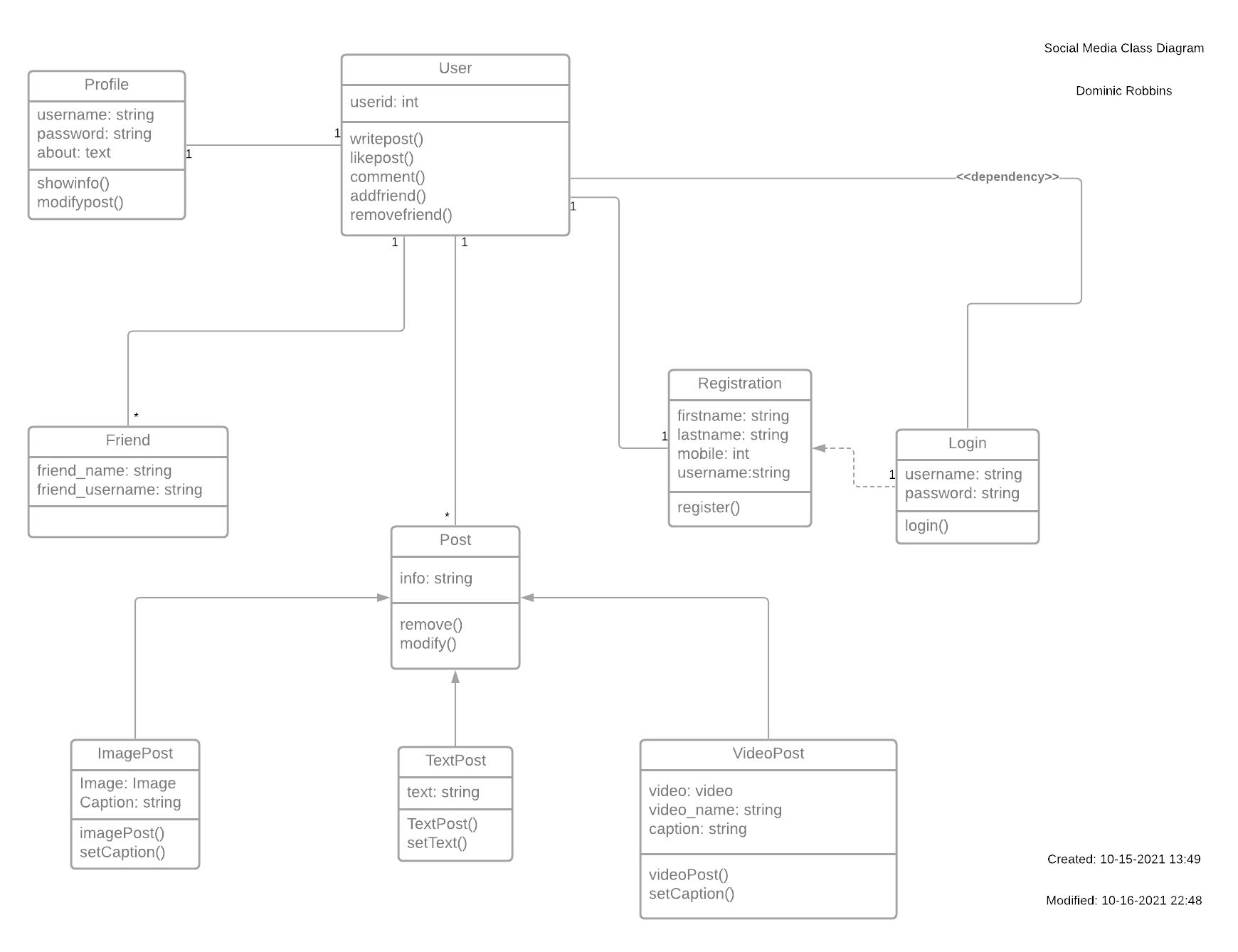
* Spotify API
* SoundCloud API

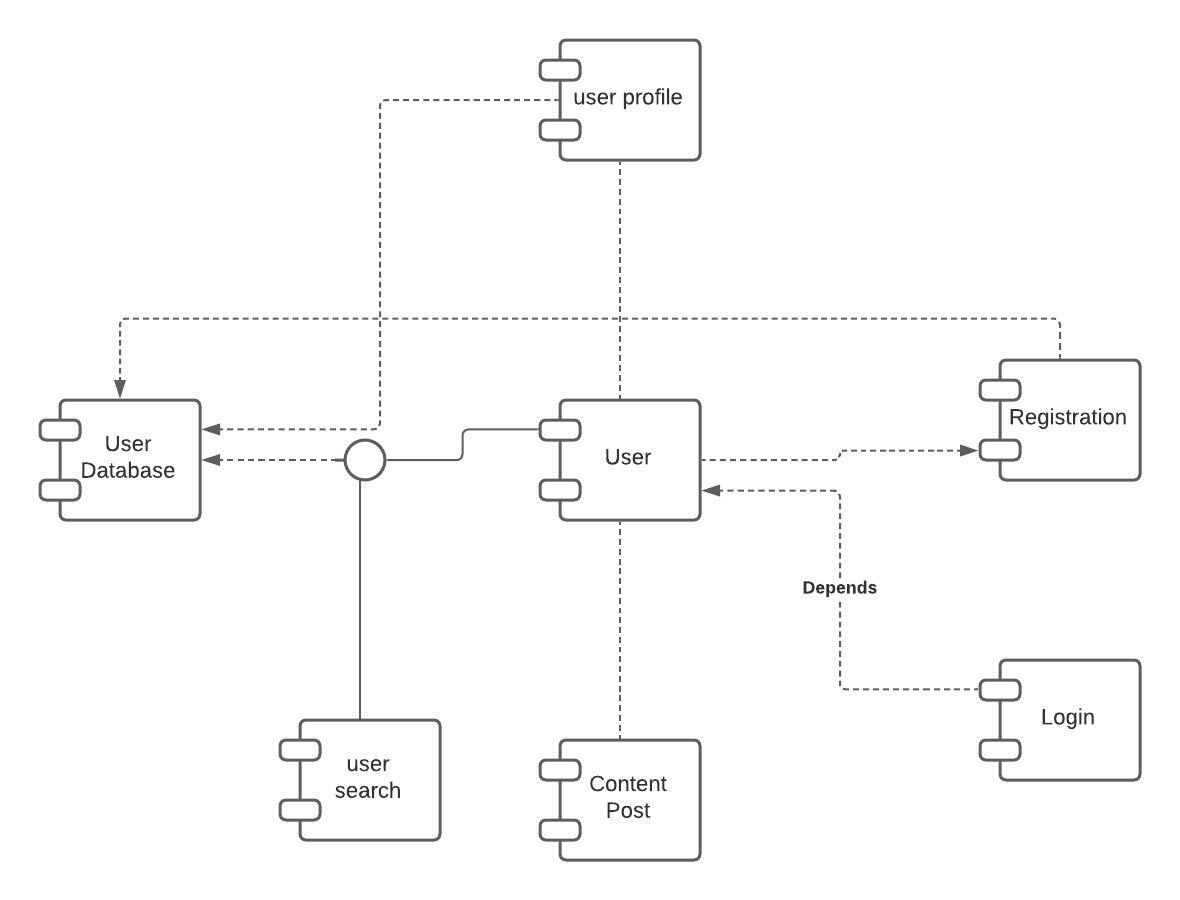
Supported browsers:

* Chrome
* Brave
* Edge
* Mozilla Firefox
* Opera

Frameworks:

* Bootstrap: UI design
* jQuery: JavaScript programming





# Identify risk and actions

**Skills Risk/Actions:**

Skills risks related to this project is that not all the members might be aware or knowledgeable of all the code that is being used throughout the whole project. A way to resolve this problem is to always keep our team updated when it comes to introducing any type of new technology into our project and making sure every team member is aware how most implementations are supposed to function.

**Schedule Risk/Actions:**

We do not have any schedule risks as we meet more than twice a week to discuss our project and every member has a flexible schedule to meet as needed.

**Technical Risk/Actions:**

Technical unknowns that we may encounter in this project is related to how the MySQL we are using is not the most up to date version and it may cause conflicts in the future. A way to circumvent this problem from happening is for the development team to always test things on the version we are using for the project instead of localhost for the team to have no problems when implementing the code live on the lamp server.

**Teamwork Risk/Actions:**

Issues that may arise from lack of teamwork is when different members might be working in different functionalities and that may cause a conflict in the overall code. A way to make sure we avoid this problem is to always state clearly what each member is doing and how that may involve another person’s code, and even if that still happens, we always have our GitHub Master to always make sure what is being uploaded and undo changes if necessary.

**Legal-Content Risk/Actions:**

Legal issues that we may encounter from this project is from using Spotify’s API to play songs. A way we found to deal with this issue was by reading the Spotify developer terms of service according to Section III: Licenses and Permissions, we are allowed to stream entire songs if the user is logged in into a premium account, and if the users is logged in, we are also allowed to use album art and other metadata from the API.

# Vertical demo

https://www.youtube.com/watch?v=fUjZM96bPCg