ADES DIT/FT/2B/03

Readdit

By Acosta Jan Jacob Domalanta (P2026271)

Tan Yong Rui (P2004147)

How Zu Kang Adam (P2026677)

TABLE OF CONTENTS

1. Introduction

1.1 Project Outline

Our team of three were tasked with the design and implementation of a 3-tier web application (Website, Backend, Database) with enough basic mechanisms to help solve the pains of our choice of problem statement.

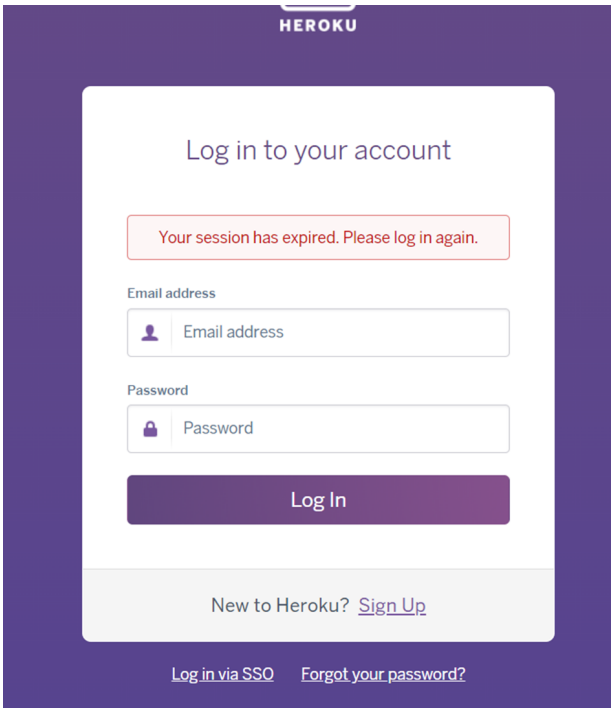
1.1.1 Project Idea

The idea that we chose to implement into a web server was to create a clone of the popular social media platform, Reddit.
http://reddit.com

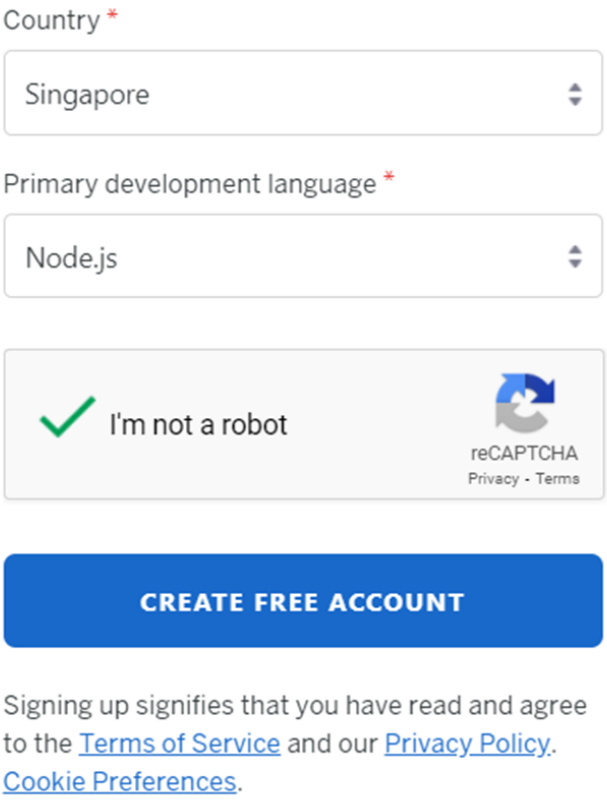
2. Application Hosting

2.1 Creating A Heroku Account

For hosting of our application for both frontend and backend, we have decided to use Heroku. In order to do that, you will first need an account.
Go to https://id.heroku.com/login and click on the **Sign Up** button shown below:



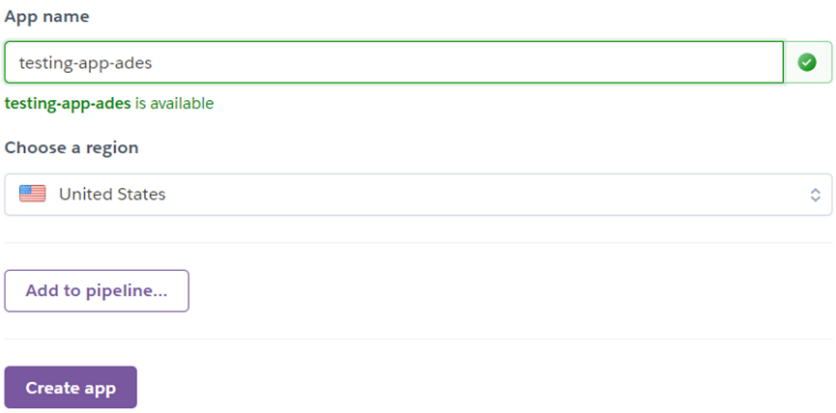
After clicking on Sign Up, fill in the Sign Up form with your details and click on **Create Free Account** shown below:



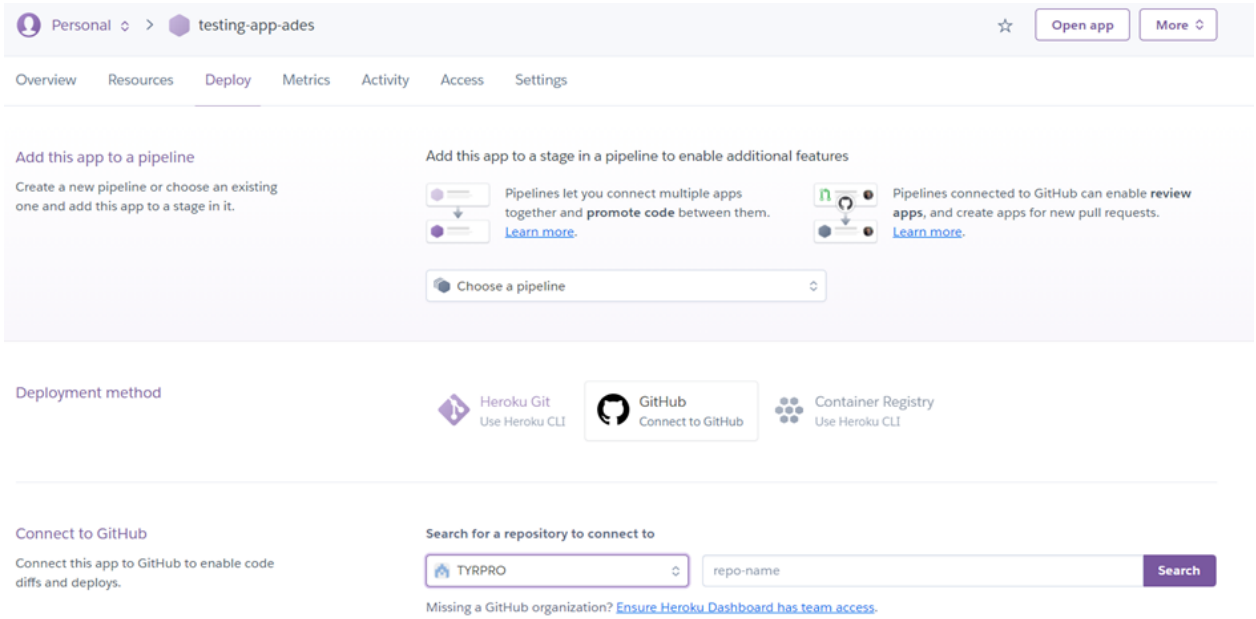
An email will be sent to you to verify your email account. After verification, head over to the Heroku dashboard.

2.2 Creating An Application

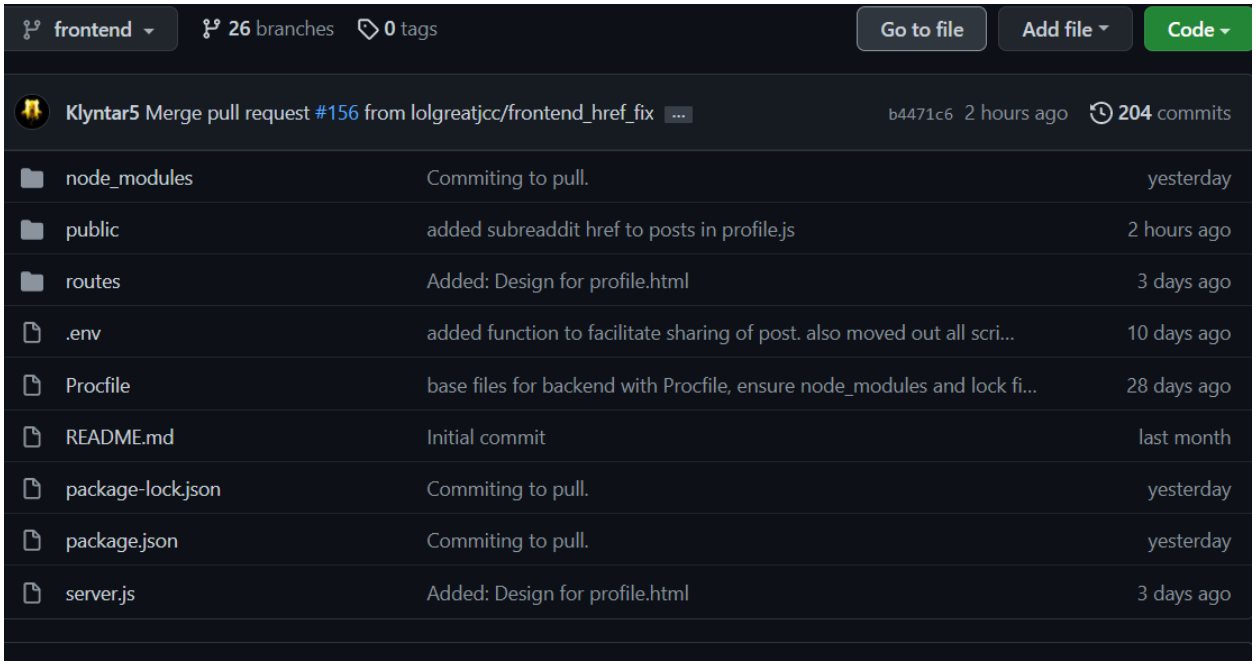
Head to the dashboard and create a new app after choosing an app name that isn't taken yet.



After creating an application, link the application to the github repository that has the code. You can search and select the branch that you want to deploy.



Ensure that your package.json, package-lock.json and server.js in the top directory:



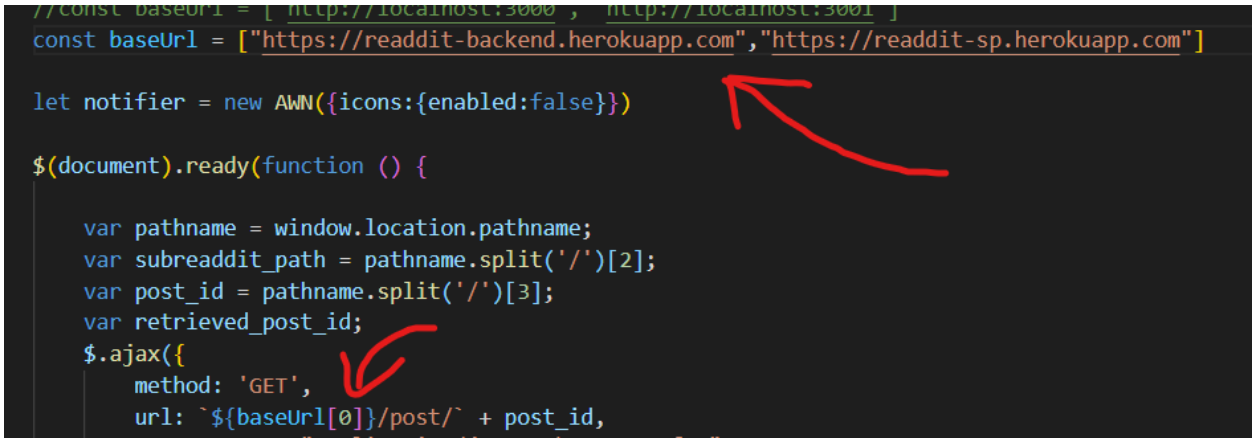
Also ensure that you have a file named Procfile (no extensions) to tell Heroku the type of application you are running and what command you want heroku to run to start the application:

In this case, I am telling Heroku I am running a web application, and run command “nodemon server.js” to start the server.

Do repeat step 2.2 again for either frontend or backend (depends on which one you did first).

2.3 Changes To Files To Take Note Of

Do remember to change your frontend calls to backend with the new Heroku backend application url.



There is no need to put the port number for the url as it is automatically defined by Heroku, but you still need to declare the port while starting the backend and frontend servers. To do this, you can use the value stored in process.env.PORT defined by Heroku:

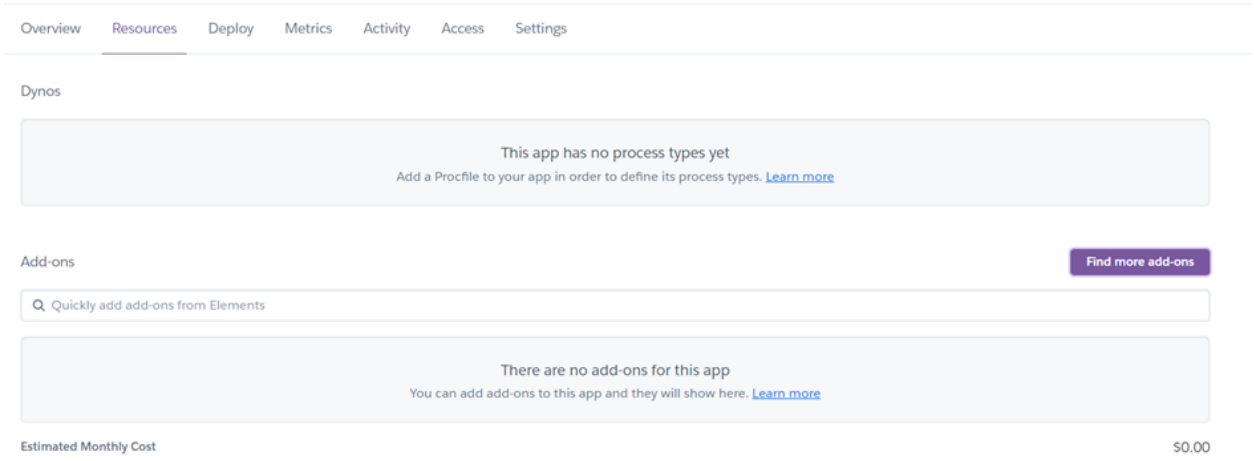
```
if (process.env.PORT != null){
    const port = process.env.PORT;
    app.listen(port,function(){
        console.log(`Server hosted on heroku!`);
    });
}
```

This port change applies to both frontend and backend.

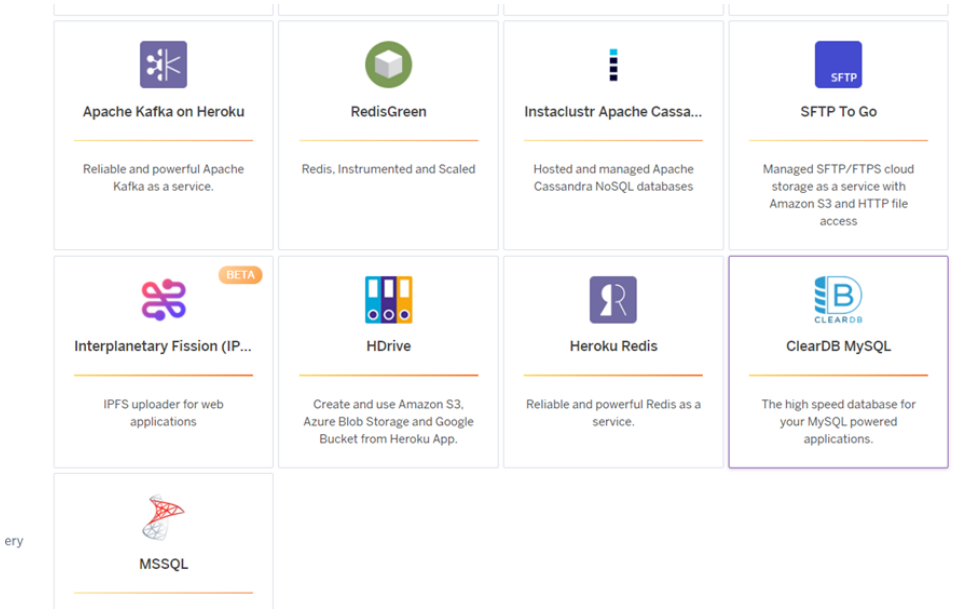
2.4 Creating and using ClearDB for cloud storage

Note: This step requires a debit or credit card. Please add a payment method to your Heroku account before continuing with this step. We recommend creating a virtual debit card online if you don't have a card. Rest assured that ClearDB does not cost.

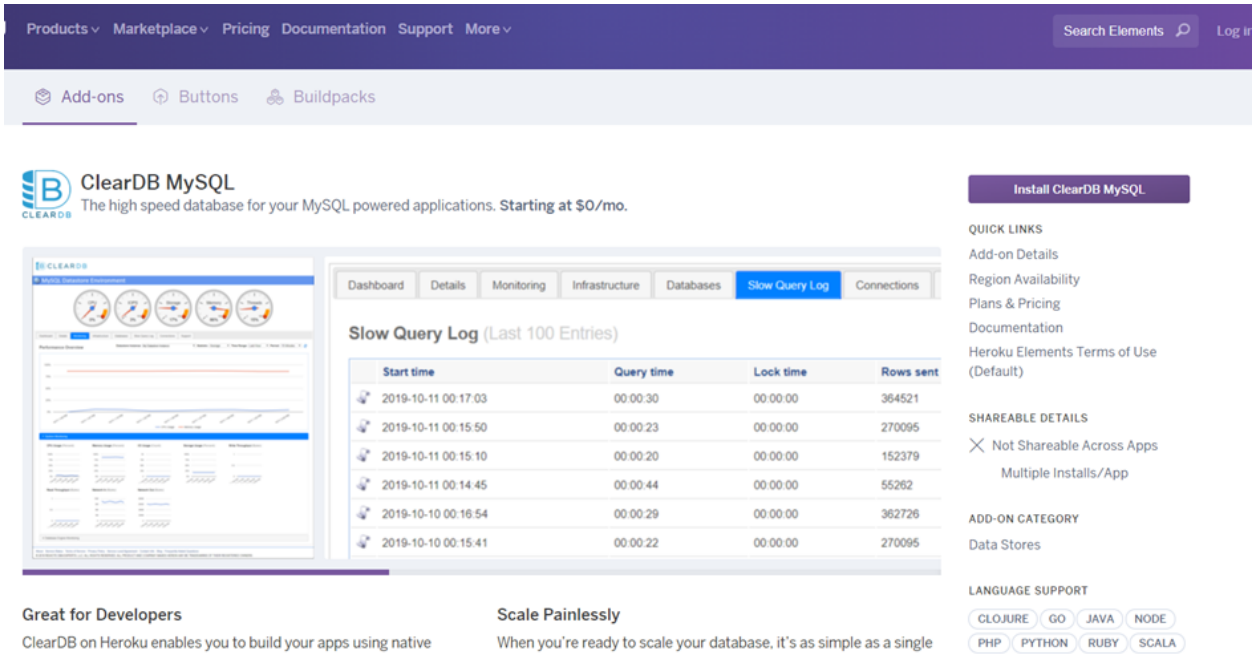
Go to your Heroku app dashboard and add new add-ons by clicking on **Find more add-ons**:



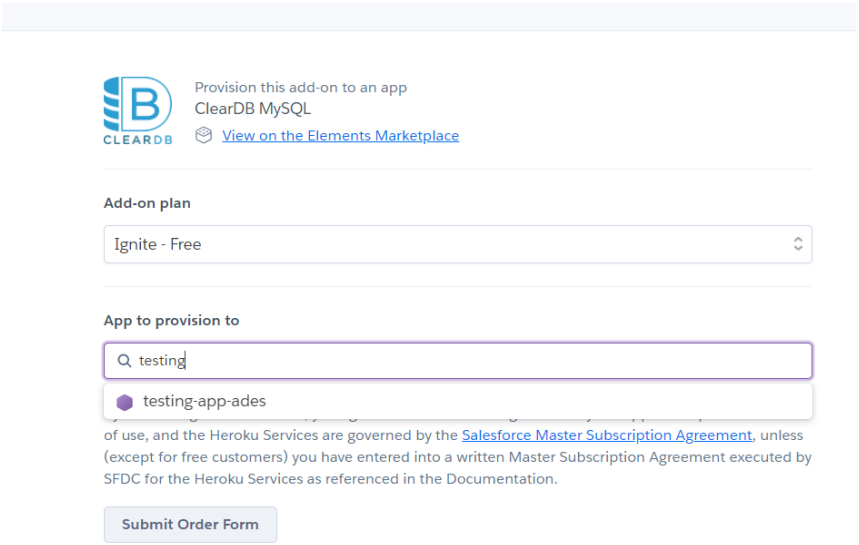
Under data stores, scroll down and find ClearDB:



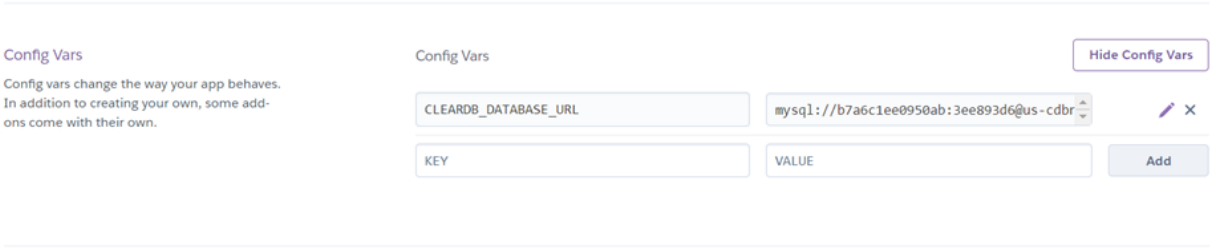
Click on **Install ClearDB MySQL**:



Type in the name of your Heroku App and click on the desired App, then click **Submit Order Form**:

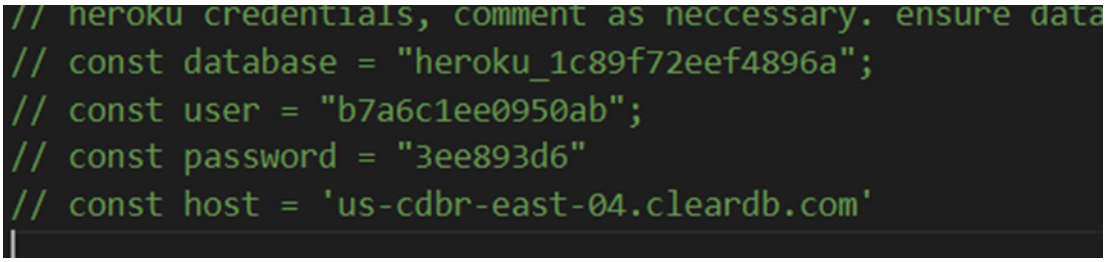


To get your ClearDB credentials, go to the Settings tab and scroll down to Config Vars, then click on **Reveal Config Vars** and copy the entire string:



Example of copied string:
mysql://b7a6c1ee0950ab:3ee893d6@us-cdbr-east-04.cleardb.com/heroku_1c89f72eef4896a?reconnect=true
The yellow highlight denotes the username, the green highlight denotes the password, the light blue highlight denotes the database URL/hostname and the orange highlight denotes the database name that **cannot** be changed. Ensure that your dump files that are going to be imported to ClearDB have their database names changed.

You can now replace your backend code's database connection details with ClearDB's hostname, username, password and database name:



3. Web Client to Backend

3.1 Web Client

To start, we used HTML and CSS to create the application's structure and design. More importantly, we used a combination of jQuery's Ajax and Axios to send HTTP requests. As per our requests to the backend, we used the four cardinal request methods (i.e. GET, PUT, POST, DELETE) to perform retrieve, edit, create and delete operations in the backend.

After the backend responds to the request, we then use Javascript to dynamically display and design information.
Our JavaScript files are placed in a separate folder than in the html as a script tag as users with itchy fingers can edit the scripts in the script tags, which can alter and grant access to buttons and views that the average user is not supposed to see. There are also checks in the JavaScript files to redirect a user away from views they are not supposed to access.

Finally, we utilized the Bootstrap framework to expedite the process of making aesthetically pleasing designs

3.2 Backend

For the backend, we built it using Node.js and the Express framework. Which is what we were taught and what we are most familiar with.

Even though the frontend has its Javascript files to prevent users from accessing pages they are not supposed to (such as administrative tools), it is also good practise to secure the backend as well for functions reserved for users with higher levels of authority, such as moderators or administrators. We will touch on more about our Role Based Access Control in Section 5.5.

Regarding our backend's structure, we tried our best to follow through with the following guidelines, Best Practices for REST API design (John and Ryan, 2020). Specifically following the points 'Use Nouns instead of verbs in Endpoint Path', 'Use Logical Nesting on Endpoints' and 'Handling Errors Gracefully'.

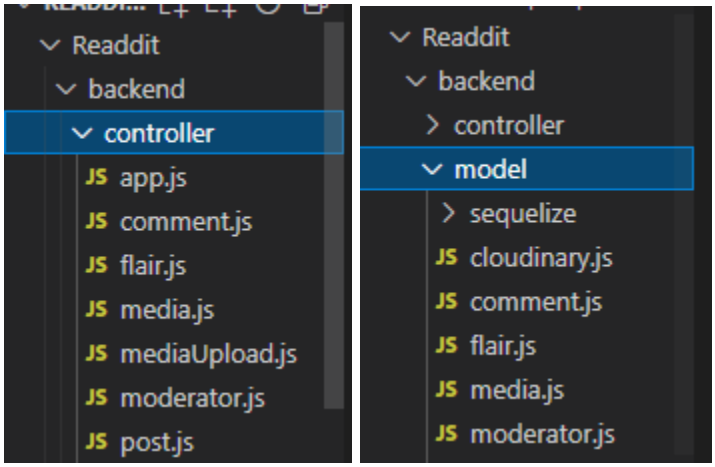
4. Backend to MySQL Database

4.1 Model, View, Controller (MVC Architecture)

For Readdit, we followed the MVC architecture.

As a short recap, MVC is a system that separates **an application into three main logical components**: the model, the view, and the controller. The Model handles all logic related to data (database and SQL statements). The View corresponds to the UI components that the user will see and interact with. And the Controller is responsible for handling requests received from the View and sending responses back to the view after applying business logic to data retrieved from the Model.

Below is an image of how we separated our View and Controller directories.

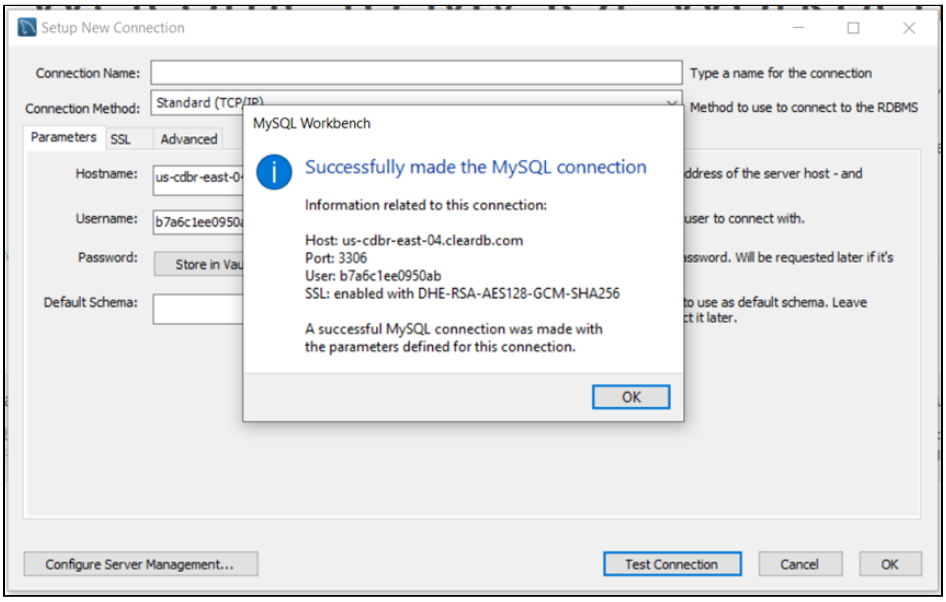


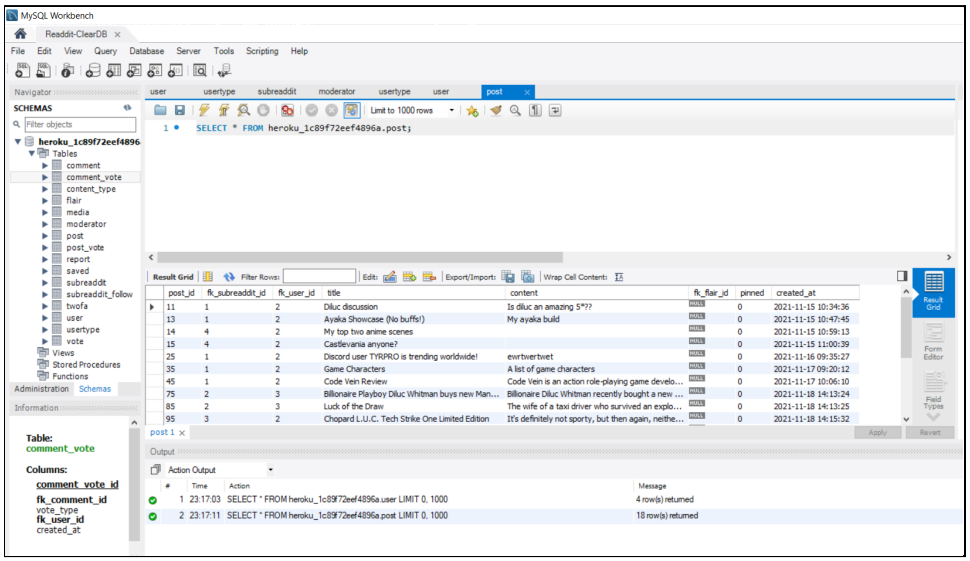
4.2 MySQL Database

For our database of choice, we chose to employ ClearDB MySQL Database. ClearDB is a free and online database that is hosted on a cloud and endorsed by Heroku.
Therefore, saving us the rigmarole of having to host and provision our own database on one of our personal servers.

We connected our created ClearDB database to our Heroku App to generate ClearDB's hostname, username, password and database name.

By entering the ClearDB Credentials into MySQL Workbench, we were able to access the data in the ClearDB Database via Mysql Workbench.





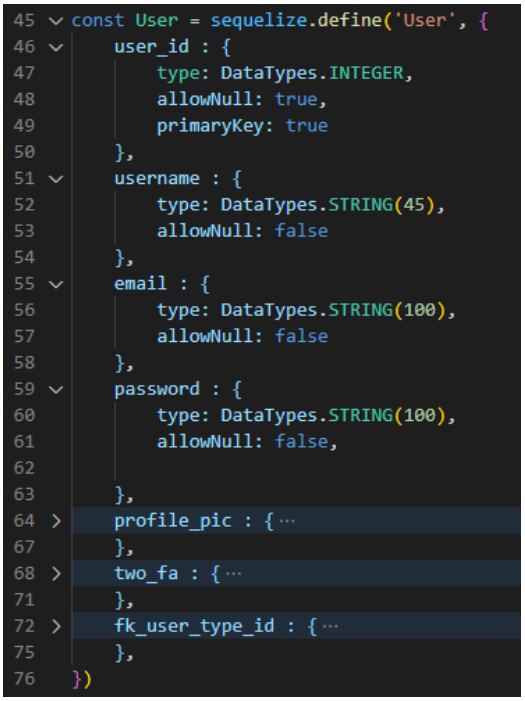
4.3 Employing Sequelize

Instead of the usual MySQL queries, we chose to implement Sequelize for the sending of queries from the backend to the database.

Sequelize is a promise-based Node.js Object-Relational Mapping tool. This is the link to their extensive documentation that we followed to use Sequelize. But in a nutshell, the core of Sequelize revolves around **models**. From what we understand, models are an abstraction of the tables inside the database, and we run methods/functions stemming from these models to interact with the database (similar to the JavaBeans that we learnt last semester).

```
//ADES CA1 Play2Win
const { Sequelize, Model, DataTypes, BOOLEAN } = require('sequelize');

//local credentials, comment as neccessary

// const database = "readdit";
// const user = "root";
// const password = "password"
// const host = 'localhost'

// heroku credentials, comment as neccessary. ensure database has data

const database = "heroku_1c89f72eef4896a";
const user = "b7a6c1ee0950ab";
const password = "3ee893d6"
const host = 'us-cdbr-east-04.cleardb.com'


const sequelize = new Sequelize(database, user, password, {
    host: host,
    dialect: 'mysql',

    pool: {
        max: 5,
        min: 0,
        acquire: 30000,
        idle: 1000
    },
    define: {
        freezeTableName: true,
        createdAt: false,
        updatedAt: false
    }
});
```

Image: Initializing Database Connection



```
const User = sequelize.define('User', {
    user_id : {
        type: DataTypes.INTEGER,
        allowNull: true,
        primaryKey: true
    },
    username : {
        type: DataTypes.STRING(45),
        allowNull: false
    },
    email : {
        type: DataTypes.STRING(100),
        allowNull: false
    },
    password : {
        type: DataTypes.STRING(100),
        allowNull: false,

    },
    profile_pic : {···
    },
    two_fa : {···
    },
    fk_user_type_id : {···
    },
})
```

Image: Our User model

The image on the left shows us initializing the sequelize object with our database credentials. While the image on the left is an example of us defining the User **model**.

While we do agree that Sequelize provides many benefits like speeding up development and securing the code (i.e. Sequelize protects against SQL injections as it uses parameterized queries and filters the data for us), using an ORM has introduced some shortcomings which we will talk about more in section 5.2.

5. Self-Directed Learning

5.1 ReactNative Mobile App

5.1.1 Introduction

ReactNative allows you to code in JavaScript and run your application on either iOS or Android (MacOS required for iOS). Designing the application is very akin to HTML and CSS, which will appeal to programmers who have experience in web development.

While developing an application for Readdit, we used back the same endpoints that were used in the web version of Readdit, which is great for code efficiency.

5.1.2 What We Learnt From ReactNative

Through ReactNative, we have learnt how to create an application with multiple screens that are able to load their content from external sources (CloudDB, Cloudinary).
We also learnt how to detect for specific key inputs, such as the enter button, which is used to submit a user's search request.

5.1.3 Difficulties Faced While Using ReactNative

Firstly, there is the issue of the application not waiting for the backend server to respond with data before trying to render it, which keeps throwing errors left and right. After hours of research, we found that we can tell ReactNative to render data conditionally:

```
<Text style={styles.postTitle}>
{isLoading ? <ActivityIndicator size="large"/> : item.title}
</Text>
```

Let's focus on the line of code in between the <Text></Text> tags. This line of code tells the application to check if isLoading is true or false. If it is, show an ActivityIndicator (loading spinner). If not, display the post's title. This checking of isLoading is widely used across the application as the majority of the content is loaded from online resources.

We also tried implementing the same image, GIF and video support, but only images were supported natively. GIF support natively was broken on the current

ReactNative version I was working on, hence we had to find another module to handle GIFs.
Fortunately, we found FastImage, which was able to support both images and GIFs, and was almost exactly used the same way as the native Image component in ReactNative.
However, we were not able to find any modules with working video playback support, with the large majority only working on Expo or iOS devices. This is one major drawback of ReactNative, where modules that promise functionality may not work as intended. Meanwhile, we will try our best to bring video playback support in CA2.

5.2 Sequelize

5.2.1 Introduction

After being recommended (by our lecturer) to attempt using an ORM for our application. We endeavoured to use Sequelize as our backend's daily driver for sending queries to the database.

To begin, although I would like to say that we learnt a lot from using ORMs - we learnt a lot about the ORM format and the usefulness of the model system as a form of database abstraction - we didn't get here without some difficulties.

5.2.2 What we learnt

First let's begin with the obvious, Sequelize is helpful as it speeds up development dramatically. We have to write less code comparatively and clears up the time needed to think up of an efficient and adequate SQL query.

As an example, here are two images.The one on top uses the base MySQL library while the one below uses Sequelize. Both images are functions attempting to retrieve all categories **but** the bottom one requires much less syntax (e.g. getting connection) and doesn't require us to come up with a SQL query.

```
getAllCategories: function (callback) {
    var conn=db.getConnection();
    conn.connect(function (err) {
        if(err) {
            console.log(err);
            return callback(err,null)
        }
        else {
            console.log("Connection Successful");
            var sqlQuery = `SELECT * FROM sp_games.category`
            conn.query(sqlQuery,function (err,result) {
                conn.end();
                if(err) {
                    console.log(err);
                    return callback(err,null);
                }
                else {
                    return callback(null,result);
                }
            })
        }
    })
}
```

Image: Base MySQL function

```
getAllCategories: function (callback) {
    Category.findAll().then(function (result) {
        callback(null,result)
    }).catch(function (err) {
        callback(err,null)
    })
}
```

Image: Sequelize Function

Next, Sequelize is much easier to update and maintain.
Have a look at this example:

```
284     getPost: function (post_id, callback) {
285         Post.findOne({
286             attributes: ['post_id', 'title', 'content', 'pinned', 'created_at'],
287             where: { post_id: post_id },
288 
289             include: [
290                 {
291                     model: User,
292                     attributes: ['username']
293                 },
294                 {
295                     model: Subreaddit,
296                     attributes: ['subreaddit_name', "subreaddit_id"]
297                 },
298                 {
299                     model: Flair,
300                     attributes: ['flair_name', "flair_colour"]
301                 },
302                 {
303                     model: Post_Vote,
304                     attributes: ['vote_type']
305                 },
306             ],
```

This is an image of a function that retrieves a post from our database.
Initially, we didn't have a rating system so we didn't have a Post_Vote table.
After implementing the rating system all we had to add was line 302 to line 305 in order for us to retrieve the votes associated with the post. In comparison, imagine attempting this via SQL query.
We would have to add an inner join and table1.col = table2.col to every single function. Not the most difficult, but very time consuming compared to the Sequelize method.

Lastly, another small benefit that Sequelize has is protection against SQL injection. Sequelize has an inbuilt parameter filter and also uses parameterized queries. I say that this is a small benefit as we're already used to using parameterized query and we also created our own middleware function to check users' input in our previous module.

```
savePost: function (post_id, user_id, callback) {
    Saved.create({
        fk_post_id: post_id,
        fk_user_id: user_id
    }).then(function (result) {
        return callback(result, null);
    }).catch(function (err) {
        return callback(null, err);
    })
},
```

5.2.3 Some difficulties

One of the drawbacks that we encountered was the initial loss in productivity due to the time taken to get over Sequelize's initial learning curve. Having to memorise the library's functions and syntax, then having to teach it to other group members took a lot of time. In our case, it took 2 - 3 weeks (out of 6 weeks) before we were confident in our ability to write Sequelize code. Although I'm sure this would help in the long run (for learning other ORMs), I feel like this process really hampered our ability to start writing code at the beginning.

Another drawback that I encountered was the loss of understanding of the SQL language after playing with Sequelize. Sequelize has the ability to 'super' simplify the query making process as we fully bypass the time and effort it takes to craft a custom made SQL query. While the simplification of query crafting is useful in expediting the process of development, I fear that using this tool has made us lose some essential time to practice writing SQL queries while we're still studying.
An analogy for this would be using a calculator before mastering mental arithmetic. Likewise, I think we should have more practice on writing SQL queries or finish our Data Engineering module before we're allowed to be using 'calculators' which seem to be using 'joins', 'sub-selects' and multiple-inserts.

```
Executing (default): SELECT `Saved`.`saved_id`, `Saved`.`fk_post_id`, `Post`.`post_id` AS `Post.post_id`, `Post`.`title` AS `Post.title`, `Post`.
`content` AS `Post.content`, `Post`.`pinned` AS `Post.pinned`, `Post`.`created_at` AS `Post.created_at`, `Post->User`.`user_id` AS `Post.User.use
r_id`, `Post->User`.`username` AS `Post.User.username`, `Post->Subreaddit`.`subreaddit_id` AS `Post.Subreaddit.subreaddit_id`, `Post->Subreaddit`
.`subreaddit_name` AS `Post.Subreaddit.subreaddit_name`, `Post->Post_Votes`.`post_vote_id` AS `Post.Post_Votes.post_vote_id`, `Post->Post_Votes`.
`vote_type` AS `Post.Post_Votes.vote_type` FROM `Saved` AS `Saved` LEFT OUTER JOIN `Post` AS `Post` ON `Saved`.`fk_post_id` = `Post`.`post_id` LE
FT OUTER JOIN `User` AS `Post->User` ON `Post`.`fk_user_id` = `Post->User`.`user_id` LEFT OUTER JOIN `Subreaddit` AS `Post->Subreaddit` ON `Post`
.`fk_subreaddit_id` = `Post->Subreaddit`.`subreaddit_id` LEFT OUTER JOIN `Post_Vote` AS `Post->Post_Votes` ON `Post`.`post_id` = `Post->Post_Vote
s`.`fk_post_id` WHERE `Saved`.`fk_user_id` = '1';
```

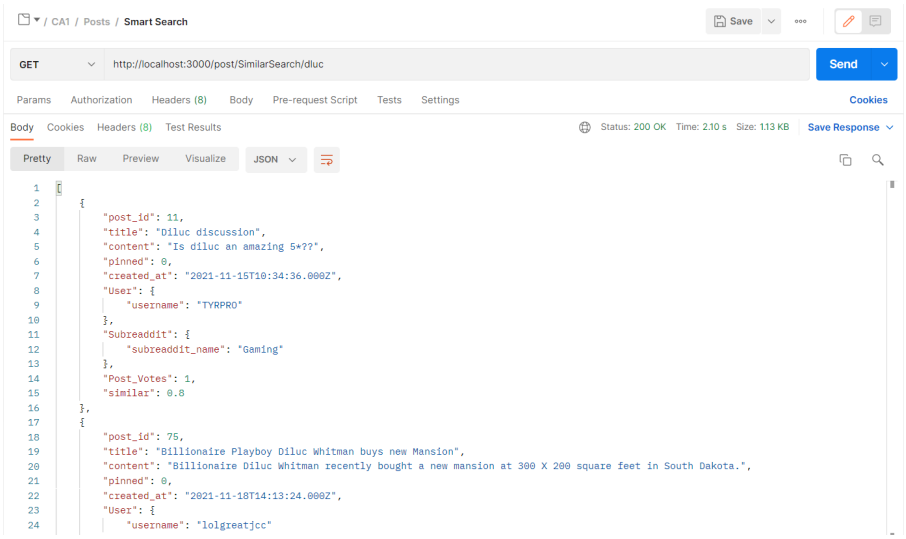
Image : Needlessly complicated query that we didn't expect

5.3 Similar Search

5.3.1 Introduction

One of our implemented features is an algorithm that searches our database for all posts with titles similar to our input string.

Example:



The basis of this algorithm is actually a math algorithm widely known as Levenshtein Distance or Edit Distance.

5.3.2 Levenshtein Distance

Levenshtein Distance was founded by the Russian scientist, Vladimir Levenshtein to calculate the similarities between two strings by calculating the number of ‘moves’ it takes to transform one string to the other.

Hence, the smaller the Levenshtein Distance, the more similar the strings are.

Example:

To transform ‘MAN’ into ‘WOMAN’, you would require two moves.
Hence the Levenshtein Distance between the strings will be 2.

To transform ‘MAN’ to ‘MANDATORY’, you would require six moves.
Hence the Levenshtein Distance between the Strings will be 6.

5.3.3 Calculating Percentage of Similarity

Now that we have a basic understanding of the Levenshtein Formula, let's get into how we applied it to our search algorithm. This is the formula we used:

```
var num = (longerLength - editDistance(longer,
shorter)) / parseFloat(longerLength);
```

This formula subtracts the Levenshtein Distance from the length of the Longer String and then calculates that number as a number of the Longer String's Length.

This will return a number from 0 - 1, 0 meaning that it is not similar at all (Since the Levenshtein Distance is = to the number of letters in the word) while 1 meaning that the strings are 100% identical (Since Levenshtein Distance is 0)

Example:

'DLUC' and "DILUC" will return a value of 0.8, meaning that it is 80% similar since the Levenshtein Distance of the strings is 1. Since only 1 character is removed from 'DILUC', which is 5 characters long. It means that ⅘ of the characters in 'DILUC' is untouched, which is also 80%.

5.3.4 Further Improvements

However, just this percentage alone is not enough, since it assumes that the compared strings are both single words. In that case, where a sentence or phrase is input, this percentage will be inaccurate.

Thus we resolved this issue by splitting the sentence into words using .split(" "), and then running the percentage function for each word, and finally the full sentence. If the score of the string is above 0.4 (At least 40% similar), we will add it to a results array.

Both strings will be split and we compare every occurrence of the split string for as many results to be added to the results array as possible.

Finally, we output the average of all scores in the results array as the final score of the string. Since the string will be compared to every post title in our database,

if the string's final score is above 0.4 (At least 40% similar), we add the corresponding record to our return endpoint to send to the frontend.

Example:

Comparing "Dog" to "I am a doggy dog"

"I am a doggy dog" → ['I','am','a','doggy','dog'], where "Dog" will be compared with every element in this array and finally compared with "I am a doggy dog" itself, totalling 6 rounds.

Only the scores of "doggy" (0.6) and "dog" (1) will be added to the results array. Thus the final score for the similarity between "Dog" and "I am a doggy dog" will be 0.8, allowing it to be added to the list of similar results.

5.3.5 Implementations

Although we have a list full of records similar to our input string, our job is still not over.

Since we're implementing this similar search along with a basic search, we will need to remove all occurrences from both searches from the similar search so that the similar search will only display results the basic search fails to retrieve.

Loops to remove duplicate results from similars array

```
for (var i = 0; i < similars.length; i++) {
    var duplicate = false;
    for (var count = 0; count < posts.length; count++) {
        if (similars[i].post_id == posts[count].post_id) {
            duplicate = true;
        }
    }
```

Next, we will also need to order the results in the similar post such that the most similar results will appear first, while the least similar results are at the bottom.

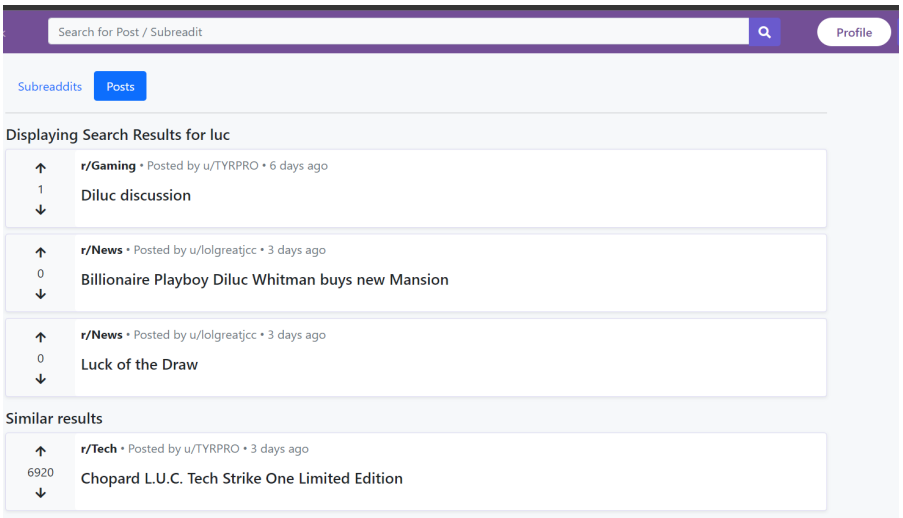
```
function arrangeSimilars(arr) {
    for (var i = 0; i < arr.length; i++) {
        if (i == arr.length - 1) {
            break;
        }
        if (arr[i].similar < arr[i + 1].similar) {
            var tmp = arr[i];
            arr[i] = arr[i + 1];
            arr[i + 1] = tmp;

            i = -1;
        }
    }
    console.log(arr);
    return arr;
}
```

This function passes in an array and rearranges them based on their similarity attribute in descending order.

```
success: function (data, status, xhr) {
    var similars = arrangeSimilars(data);
```

The function is called at the top of the success function in the ajax call.

With those two sorted out, we now have a working frontend similar search.

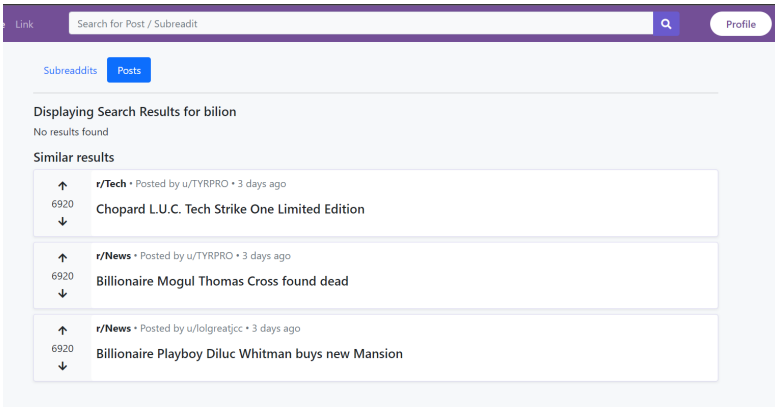


5.3.6 Limitations

However, this similar search is not without its limitations. Since the algorithm relies heavily on the Levenshtein Distance formula, the limitations of the Levenshtein Distance formula apply to the algorithm as well.

Example:
Our formula will view ‘BILION’ to be more similar to ‘EDITION’ than ‘BILLIONAIRE’.

Although ‘BILION’ is mathematically more similar to ‘EDITION’ than ‘BILLIONAIRE’, humans don’t perceive this similarity the same way. Thus this is one limitation of our Algorithm.



Thus results containing ‘EDITION’ are put on top of results containing ‘BILLIONAIRE’.

5.4 Sorting Algorithm

5.4.1 Reddit's Sorting System

Let me preface this by showing you the research we performed on Reddit's (our competitor) 'Hot' algorithm.

$$f(seconds, sign, n) = \log_{10} n + \frac{sign * seconds}{45000}$$

Where **n** is the total rating of the post (i.e. Upvote - Downvote) and **seconds** is the time in seconds since the first post. For simplicity's sake let us assume **sign** is equal to positive one.

First let us break down the formula into two parts, the part of the function before the plus and the part of the function after the plus.

5.4.1.1 Logarithmic Voting

As you can see from the function above Reddit's 'Hot' algorithm evaluates the total rating of the post logarithmically.

$$\log_{10} n$$

In essence, this means that the value/worth of votes reduces as the value of n grows. In layman terms this means that the first 10 upvotes have the same weight as the next 100 and the 100 upvotes hold the same weight as the next 1000.

Now, why is this bad? In our opinion, we think that votes should be created equal and shouldn't be worth less as more votes get added. Simply because we want to make users feel as if their votes actually matter and actually decide the sorting of posts.

5.4.1.2 Linear Time

Next, is the seconds divided by 45 000 or 12.5 hours. In essence, this just means that a post needs to have 10 times as many votes as another post which is 12.5 hours earlier. But this is besides the point, what I want to focus your attention on how it is calculated.

$$\frac{sign * seconds}{45000}$$

As seen from the image above, the confidence rating of a post is calculated linearly with respect to time. This means that the time it takes for the post in ‘Hot’ to drop is quite slow comparatively (i.e. the posts high up in the ‘Hot’ section will most likely be the same 8-12 hours later) . Of course, we could just change the value of 45 000 but we chose to go for a different approach.

5.4.2 Our Sorting Algorithm

So how did we solve the problems we had with Reddit’s algorithm? Based on our research and knowledge of Mathematics we have come up with our own sorting algorithm which attempts to solve our issues with Reddit's algorithm.

$$f(v, time) = \frac{v}{(time)^2}$$

Where **v** is equal to the total rating of the post (i.e. Upvote - Downvotes) and **time** is equal to the time in hours sine the post’s submission.

5.4.2.1 Linear Voting

As seen **above**, when calculating the rating of the post we take into account the total rating of the post linearly. This means that compared to Reddit's algorithm (which takes votes logarithmically), we instead take them into account linearly. This in essence solves our issue with their algorithm since every vote will be created equally since the first vote will have the same worth as the thousandth.

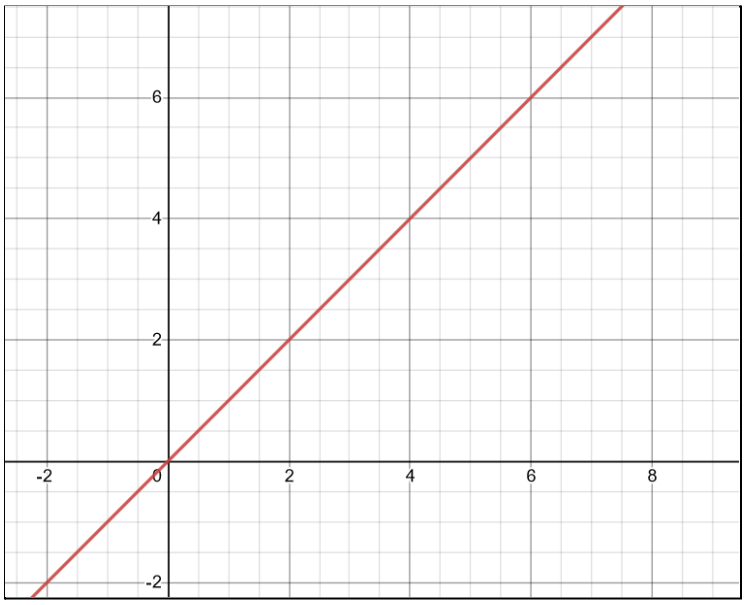


Image: y = x / (1)^2

The above graph shows how the post's confidence rating grows (in our formula) as the number of votes grows given that the variable of time doesn't change (i.e. one hour since post submission).

5.4.2.2 Quadratic Time

Regarding our issue with the speed by which posts in Reddit's 'Hot' section are rotated, we solved it by calculating the rating through quadratic time.

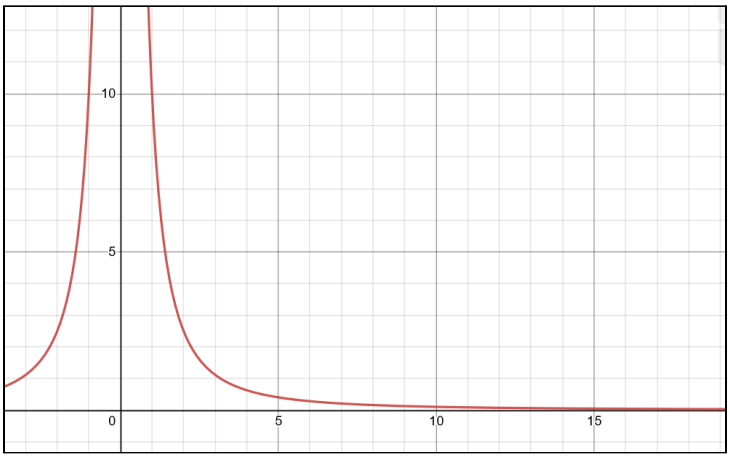


Image : y = 10 / (x)^2

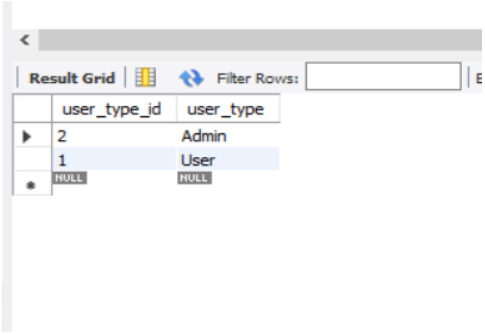
As you can see from the above graph as time goes on (given the same rating), the post's confidence rating drops quickly but later the gradient levels off.

We see this as the best possible solution since the post that users will see rotates quickly giving way for newer posts but still greatly values user's votes as after most posts have aged, higher voted posts will still generally have a higher confidence rating and thus be put higher up.

5.5 Role-Based Access Control

5.5.1 Introduction

We also implemented Role-Based Access Control to differentiate two types of users, Admin and User roles.



5.5.2 Token Creation

We make use of JWT tokens to help verify and authenticate our users. This token is generated on every login.

```
login: function (email, password, callback) {
    User.findAll({
        attributes: ['user_id', 'username', 'fk_user_type_id'],
        where: {

            [Op.and]: [
                { email: email },
                { password: password }
            ]
        }
    })
```

The login function gets the user info of the specified user using their email and password.

```
var token = jwt.sign(
    // (1) Payload
    {
        user_id: result[0].user_id,
        type: result[0].fk_user_type_id
    },
    // (2) Secret Key
    config.key,
    // (3) Lifretime of token
    {
        //expires in 24 hrs
        expiresIn: 86400
    }
);
```

If successful, a new token will be created by signing the values of the user id and user role, encoding it with a secret key and then setting its lifetime.

```
var secret='s12xyz00'; //your own secret key
module.exports.key = secret;
```

This secret key is defined in our config.js, and it will be called again whenever we want to extract information from the token.

5.5.3 Backend Restriction

We restricted the access of users from using admin only endpoints using our own middleware function, verify.js

Verify.js contains three functions for different uses, but the function we use for verifying admins is checkAdmin

```
checkAdmin : function (req, res, next){
  if (typeof req.headers.authorization !== "undefined") {
    // Retrieve the authorization header and parse out the
    // JWT using the split function

    let token = req.headers.authorization.split(" ")[1];
    //console.log('Check for received token from frontend : \n');
    //console.log(token);

    jwt.verify(token, config, (err, data) => {
      console.log("data extracted from token \n", data);
      if (err) {
        console.log(err);
        return res.status(403).send({ "message": "Unauthorized access", errCode: 1 });
      } else {
        if (data.type != 2){
          console.log("Not an admin!");
          return res.status(403).send({ "message": "Unauthorized access", errCode: 2 });
        }
        else{
            req.body.token_fk_user_type_id = data.type;
            next();
        }
        }
    });
  } else {
    res.status(401).send({ "message": "Unauthorized access" });
  }
},
```

The token we created contains the user's role id. Thus this function extracts the role id from the endpoint's Authorization Header and checks its value.

This is how we ensure that these endpoints are only used by those with the valid role id.

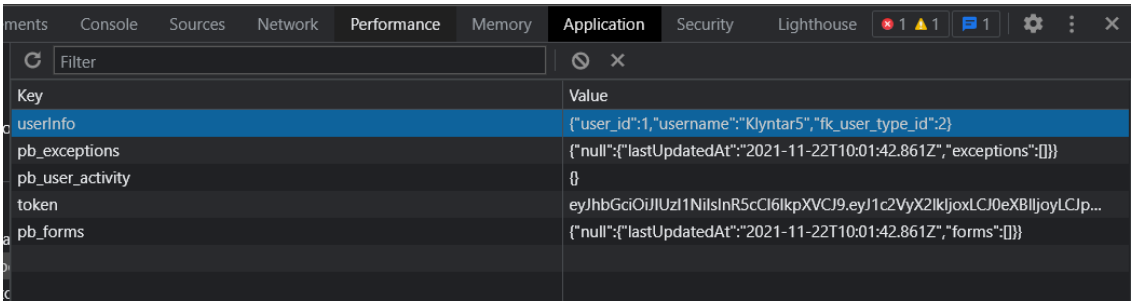
```
//delete user
app.delete('/users/:userid', verify.checkAdmin ,printDebugInfo, function (req, res) {
    var userid = req.params.userid;

    user.delete(userid, function (err, result) {
        if (!err) {
            res.status(204).send({"Result" : result});
        } else {
            res.status(500).send({"message":"Error deleting user." });
        }
    });

});
```

Delete user is an admin only endpoint, thus we secure it using verify.checkAdmin

5.5.4 Frontend Restriction

In the frontend, we restrict user's access to certain pages by checking their role id from the website's local storage.

When users log in, we store the created jwt token in the local storage.



Whenever the user accesses an admin only page, we send a call to the backend to verify the user's role using verify.checkAdmin

```
$(document).ready(function () {
    try {
        var userData = localStorage.getItem('userInfo');
        var token = localStorage.getItem("token")
        // userData = userData.slice(1,-1);

        var userJsonData = JSON.parse(userData);

        $.ajax({
            headers: { 'authorization': 'Bearer ' + token },
            url: `${baseUrl[0]}/verify`,
            type: 'GET',
            contentType: "application/json; charset=utf-8",
            dataType: 'json',
            success: function (data, textStatus, xhr) {
                console.log(data);
            },
            error: function (xhr, textStatus, errorThrown) {
                window.location.assign(`${baseUrl[1]}/home.html`);
            }
        });
    } catch (error) {
        window.location.assign(`${baseUrl[1]}/login.html`);
    }
```

If the user is not an admin, he will be redirected back to the home page. Thus this makes sure that only admins can access these pages.

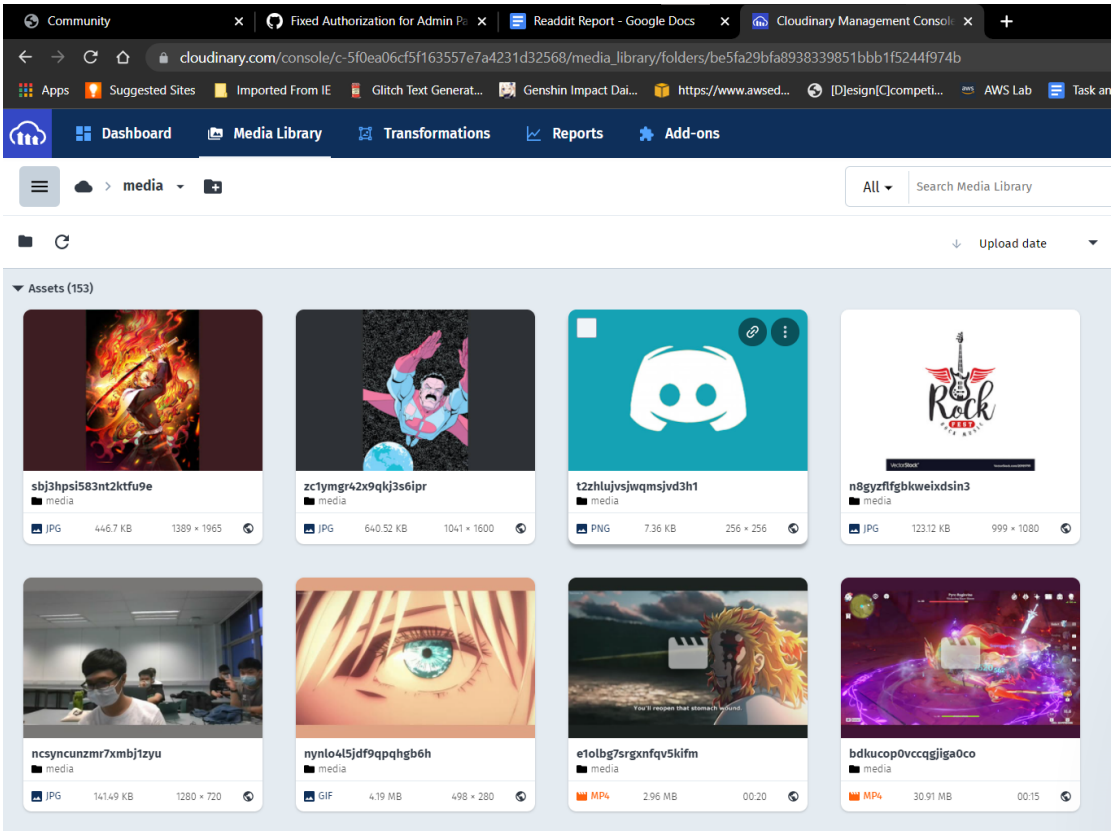
5.6 Cloudinary

5.6.1 Introduction

Cloudinary is an online media storage platform that is free for everyone to use with the free tier. We will be storing all of our media (Images, GIFs, Videos) in Cloudinary.

5.6.2 Takeaways From Cloudinary

Through using Cloudinary, I have learnt how to upload images, GIFs and videos to an online cloud platform. All of our application's media (save for the header logo) are stored in Cloudinary and fetched when needed)



I also learnt how to use new modules to process the media uploads, as well as to check for a file's type and size. This is important when verifying file uploads to ensure that the application doesn't try to display a PDF in a video or image HTML tag, or to ensure that the Cloudinary upload doesn't fail due to the file being too large.

I also learnt how to transform images using Cloudinary to return rounded images, which was useful for profile images.

Uploaded image in Cloudinary:

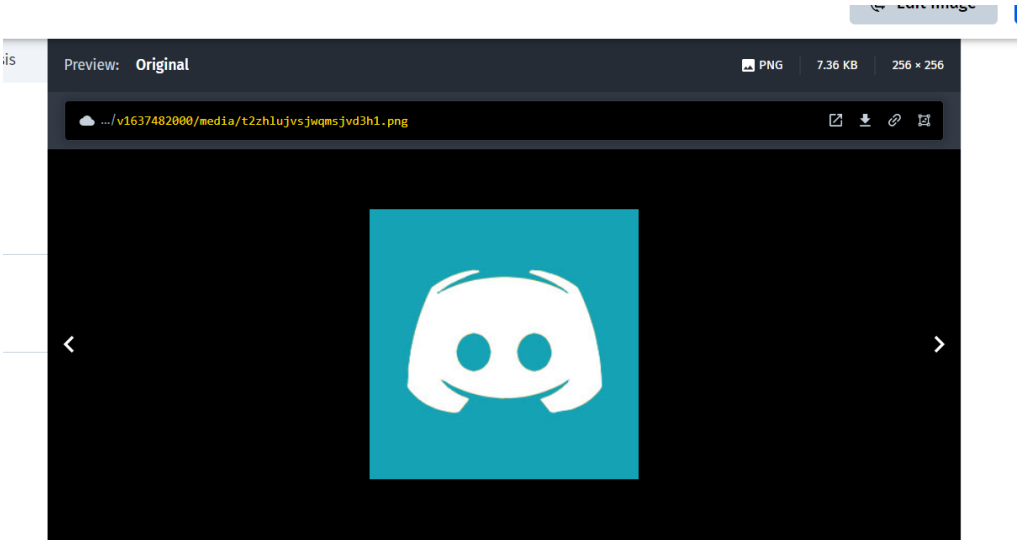
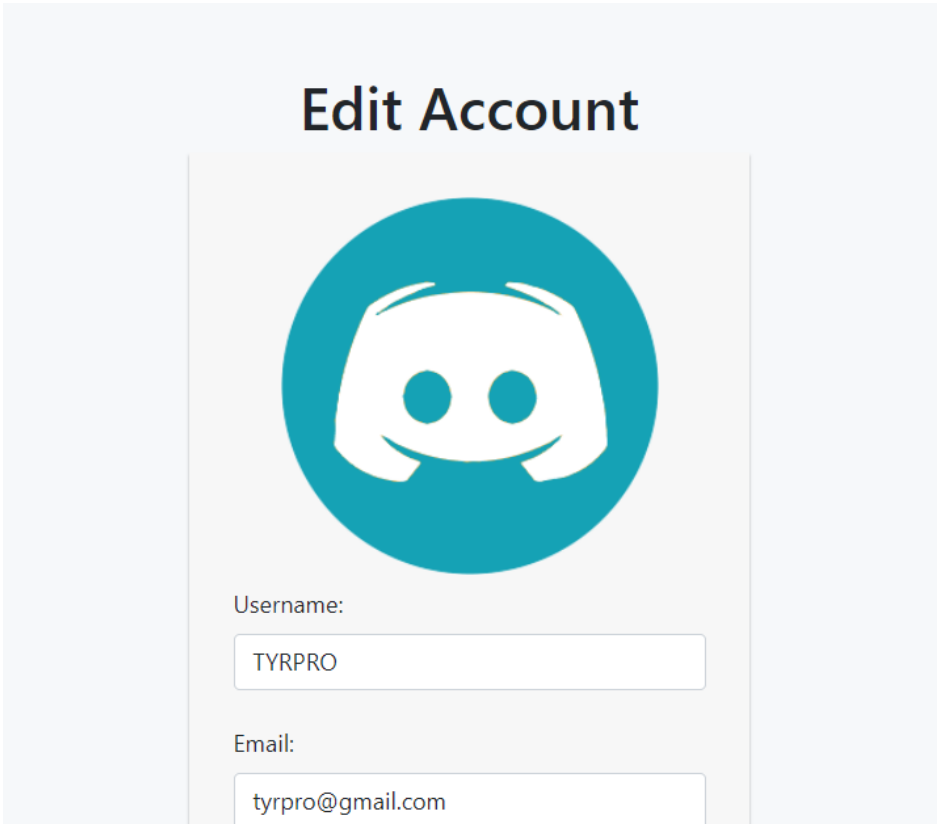


Image after Cloudinary transformation:



You could argue that the profile image could be formatted to achieve the same transformation via CSS, but having Cloudinary do it means that all the frontend client has to do is load the image, which saves some processing time.

5.6.3 Difficulties While Using Cloudinary

I needed to accept file uploads from the user, but files cannot be passed from the frontend to the backend using JSON data type. After doing some research, I settled on WebFormData, which allows me to pass user uploads to the backend server.

Even after passing the WebFormData to the backend, I was having trouble extracting the data from said WebFormData. Doing further research led me to find a npm module named multer, which was used to parse the WebFormData in the backend.
From these two incidents, I now know that WebFormData must be used in conjunction with multer for me to successfully implement file upload support.

A user might also accidentally submit a file type that is not a media type, such as PDFs and documents. They might also upload files that are too large for Cloudinary to support with the free tier. I ran into continuous issues of uploads failing as I did not know that Cloudinary had file size limitations.
After doing some research, I found that Cloudinary has a 10MB and 100MB max file size for images and videos respectively. In order to check for the file size, I had to use the fs (file system) import, which was foreign to me, to check for the file's size.
After some trial and error, checks for file size and file types were in place, and users were notified if the file size was too large or the file type was unsupported.

5.7 CSS Keyframes

5.7.1 Introduction

Due to our image uploading feature in our posting feature, one of the drawbacks we encountered is that the uploading of posts took a considerable amount of time.
Therefore we had to come up with a way to show users that their posts **were being uploaded**. And then a way to tell them once their post **has been uploaded**. Of course, we can add a simple text, which says "post being uploaded" and "post successfully posted" but that felt inelegant and visually boring. Therefore, I took this opportunity to learn how to animate using CSS and learn about keyframes. We will talk more about this in the presentation.

6. Teamwork

6.1 GitHub

Our project planning tool of choice was GitHub. Github enabled us to organize our work by allowing us to view the current progress of the project is, what everyone is currently working on, and what still needs to be done. We felt that our focus on Github and project planning increased our work efficiency significantly as we were always able to quickly comprehend the current status of the project. Below are some of Github's project planning tools that we used. Link to our GitHub repository.

6.1.1 Branches & Pull Requests

Firstly, we decided to use GitHub's branches (and pull request) system for any new features or issues that we were developing on. Using different branches instead of pushing to the central branch allows us to 'isolate' code and thus quickly figure out where an issue/bug stemmed from.

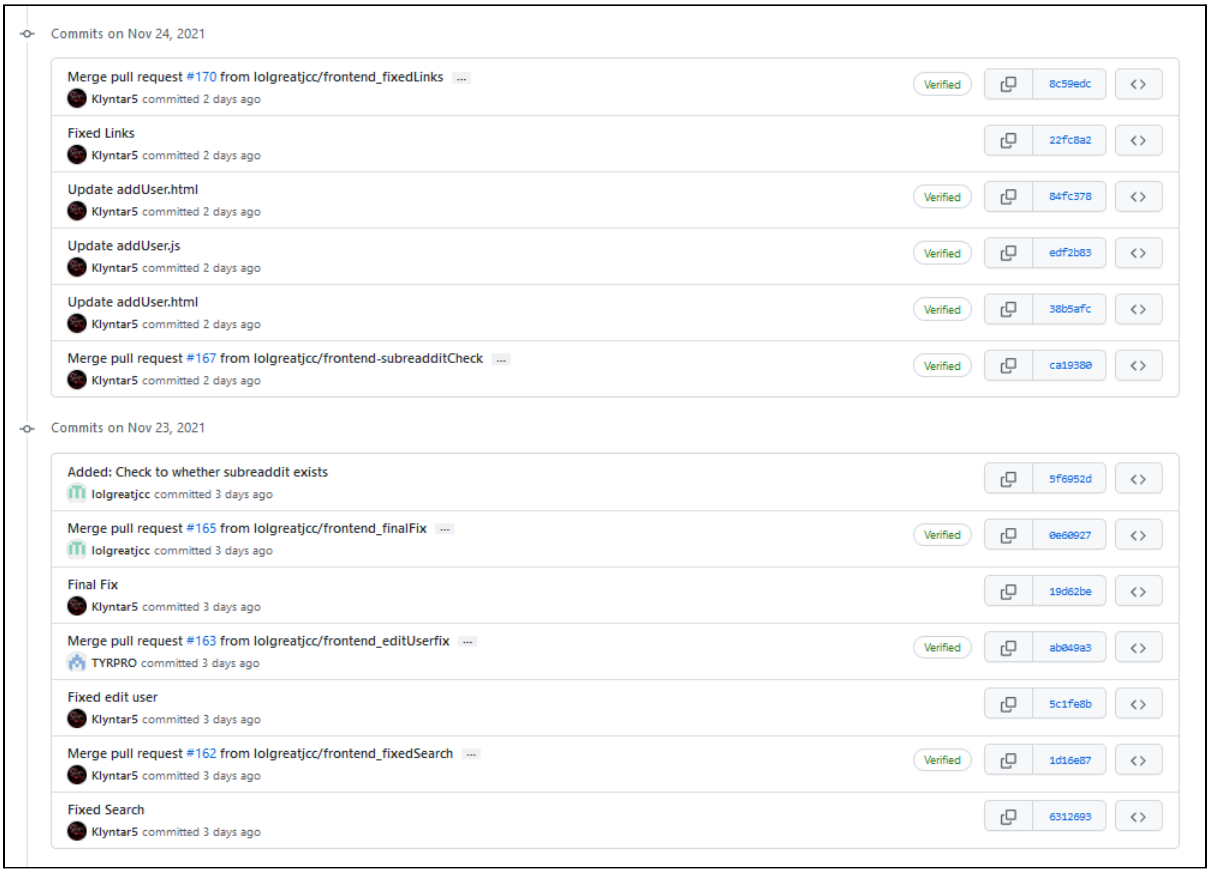


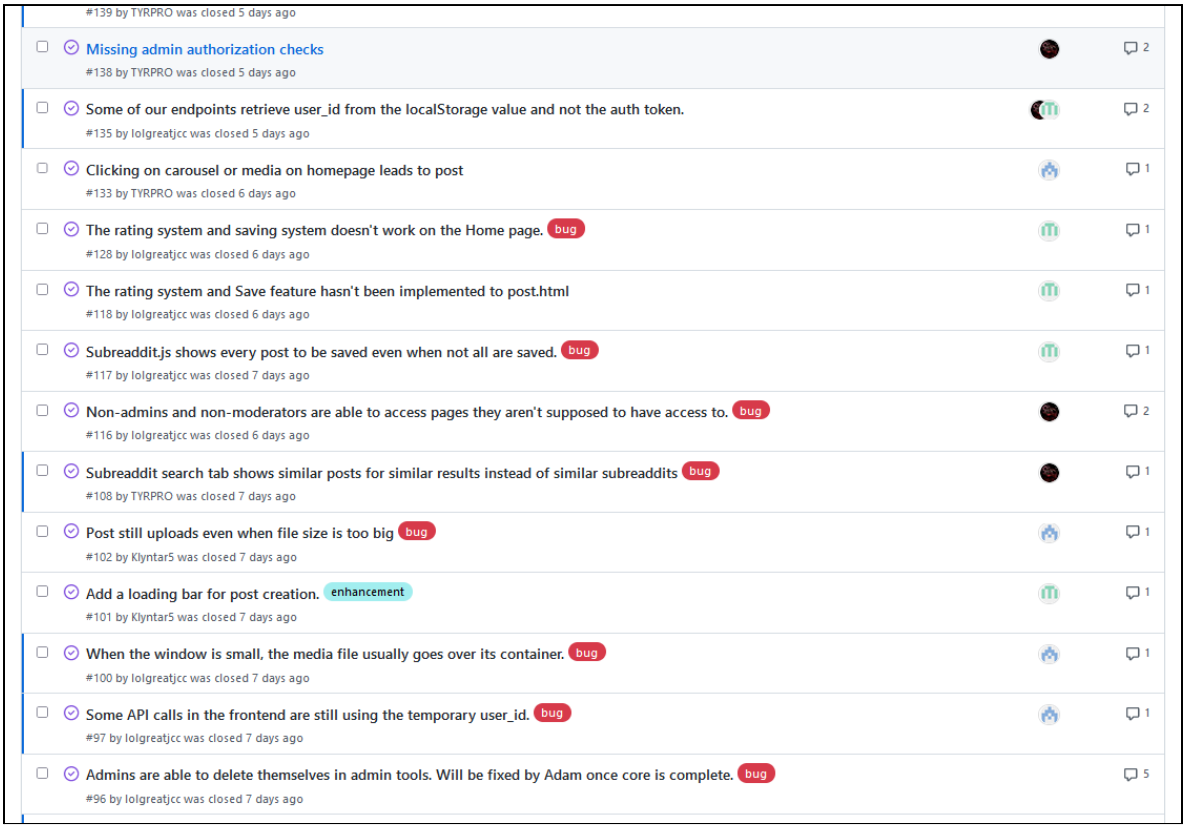
Image: Pull Request Commits

6.1.2 Project Board (Scrum Board)

Next, we decided to utilize GitHub's project boards as our Scrum board. Having a Scrum board was indispensable as it quickly allowed us to find out who to ask regarding the machinations of a specific feature. Therefore, accelerating our feature integration process. Furthermore, having a Scrum board helped us visualize the progress of the project. This allows us to quickly decide whether we need to speed up to reach the project deadline or slow down to produce higher quality work. Link to our Scrum Board.

6.1.3 Issues and Resolutions

We also heavily utilized GitHub's issues page. Using GitHub Issues increases the efficiency at which we fix bugs since all our application's bugs are able to be seen from one page. Another thing that helped us is its heavy integration with GitHub. GitHub Issues allows us to refer to pull requests and other users. We even incorporated a rule which states that once an issue is fixed have a short description of how the issue was solved and **refer to the pull request which implements the fix**.



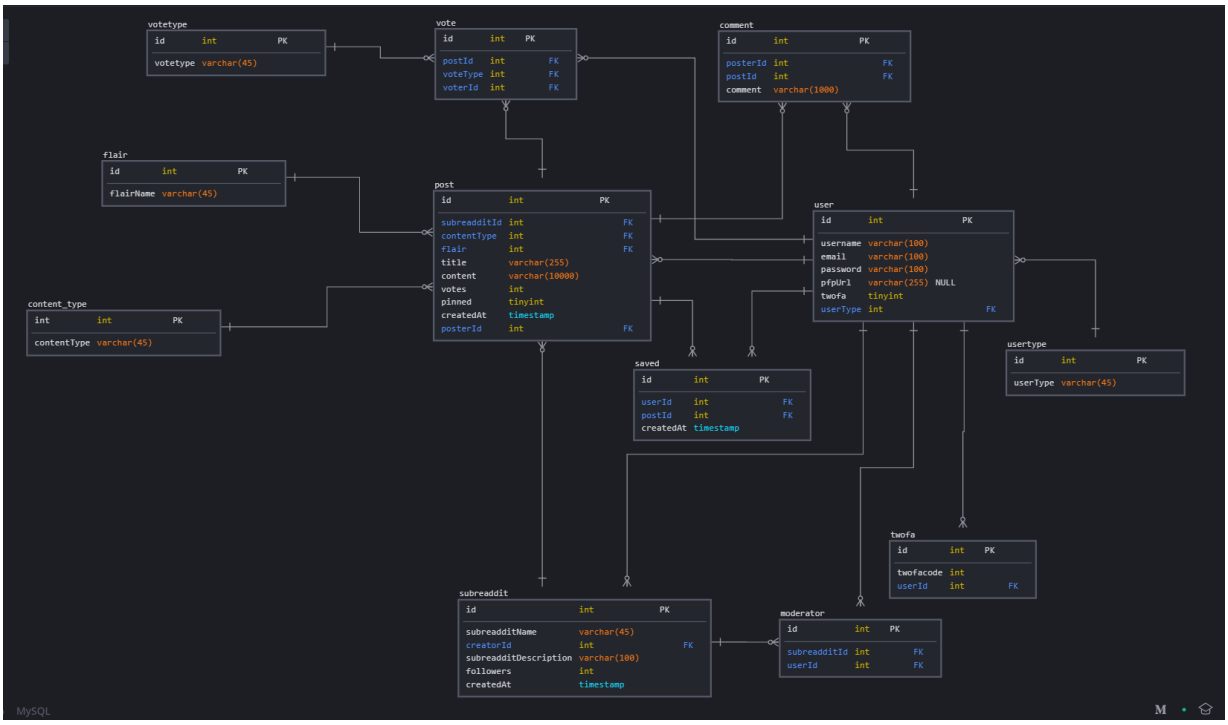
The assign feature was also very useful as at all times we could tell who was working on a specific feature so that two persons wouldn't implement the same fix.

6.2 Diagrams

Firstly, we'd like to start by stating that we know of the existence of other useful diagrams (e.g. UI Flow and Sequence Diagrams). But considering that our team is small and none of the other aspects was too complex, we decided that crafting other diagrams wasn't warranted.

6.2.1 ER Diagram

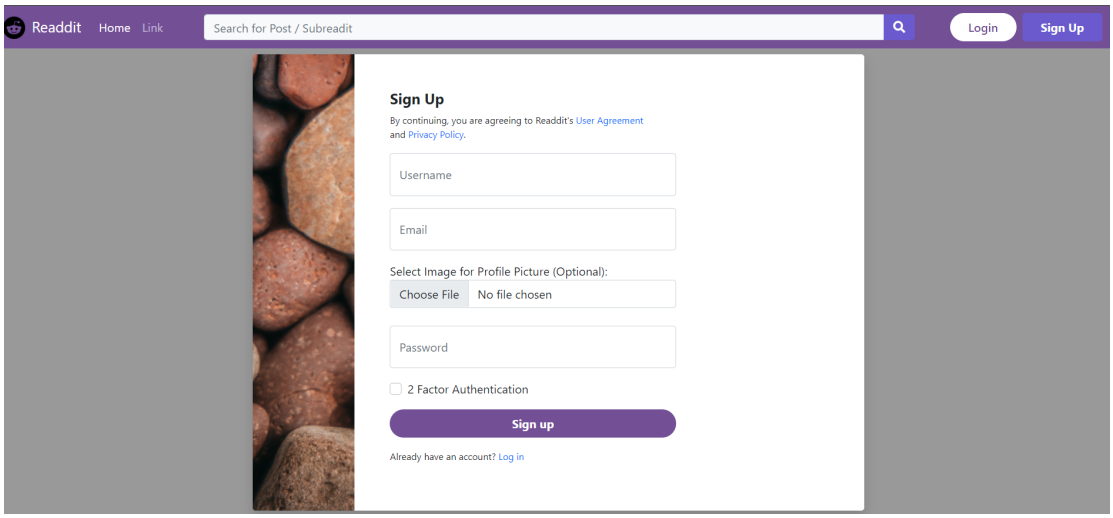
One of the aspects that did feel complex was our database. With our many tables and relationships, we felt compelled to create an ER diagram. Below is an image of our first ER diagram.



Creating an ER diagram made database creation less stressful since from the start of development we we're all on the same page on what the database should look like. Furthermore, the ER diagram is helpful as we are using the Sequelize ORM which meant that we had to make an abstraction of the database tables in Sequelize. Having the ER diagram to refer to expedited the model creation process.

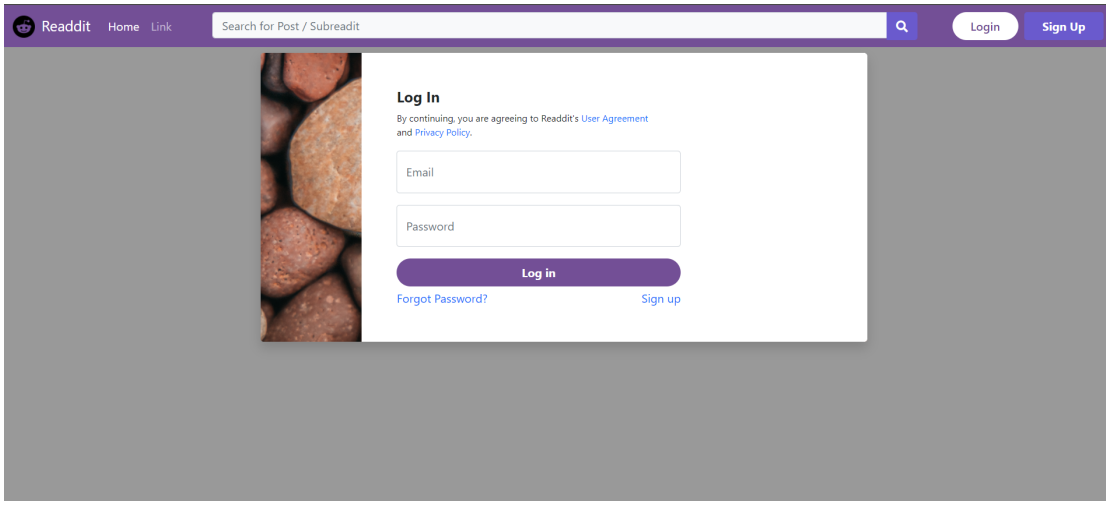
7. Readdit Features / Manual

7.1 Sign Up



Users can sign up for a readdit account here, where they are required to input a username, password, email and an optional profile picture.

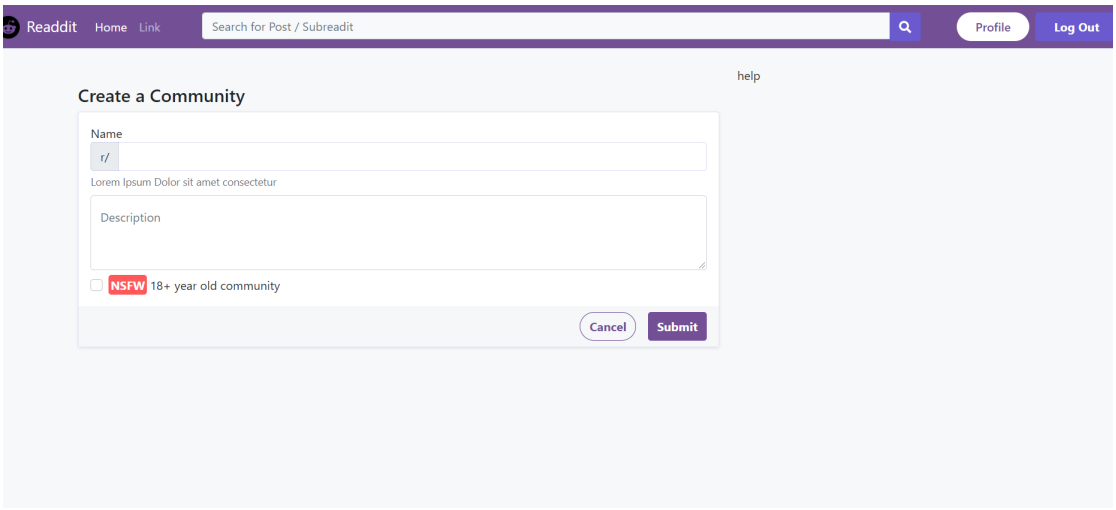
7.2 Login



Users who have signed up can log into their accounts using their created email and password.

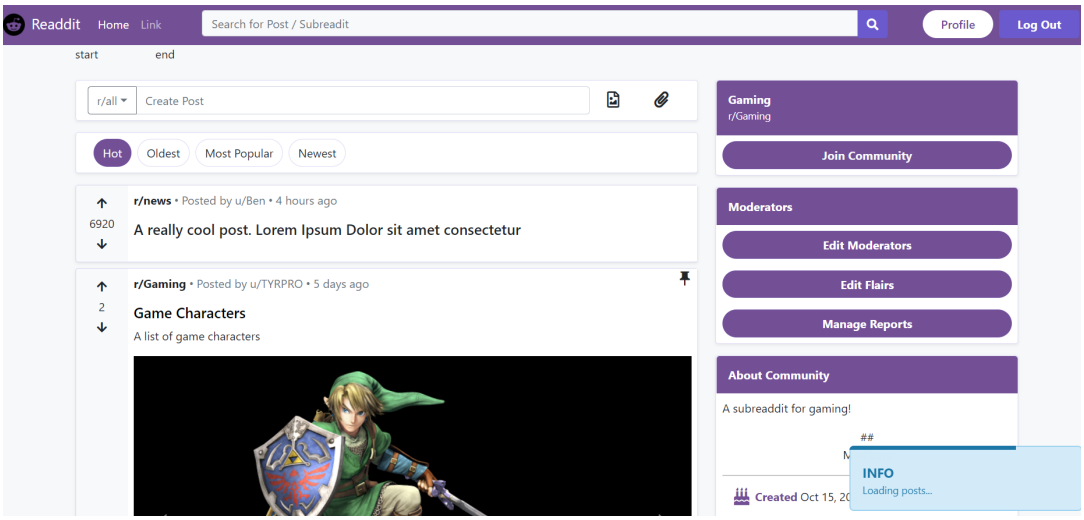
7.3 Subreaddits

7.3.1 Create Subreaddits



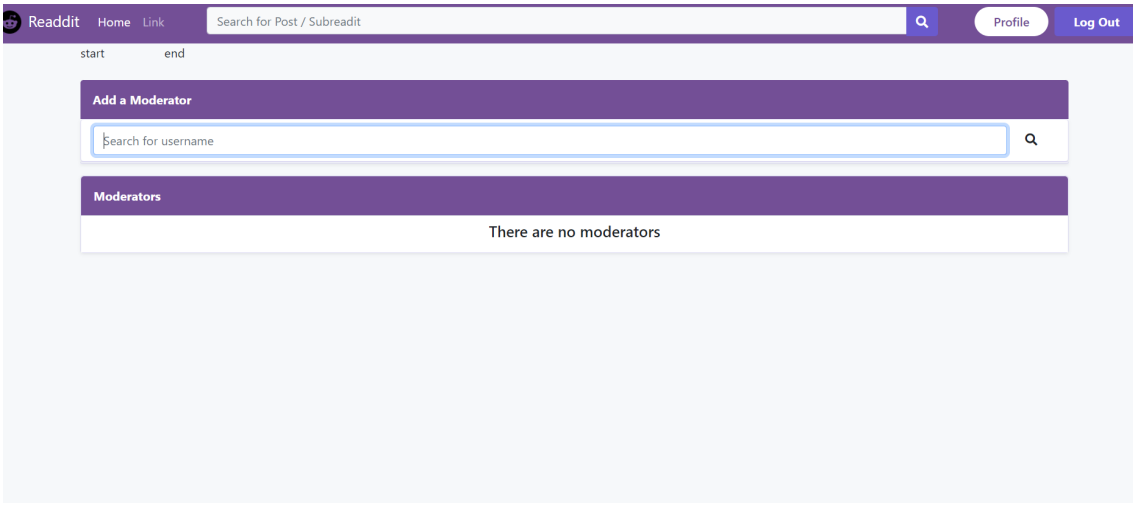
Users can create subreaddits for any category they want. Each subreaddit community requires a subreddit name and a subreddit description - describing what the subreaddit is about.

7.3.2 View Subreaddits



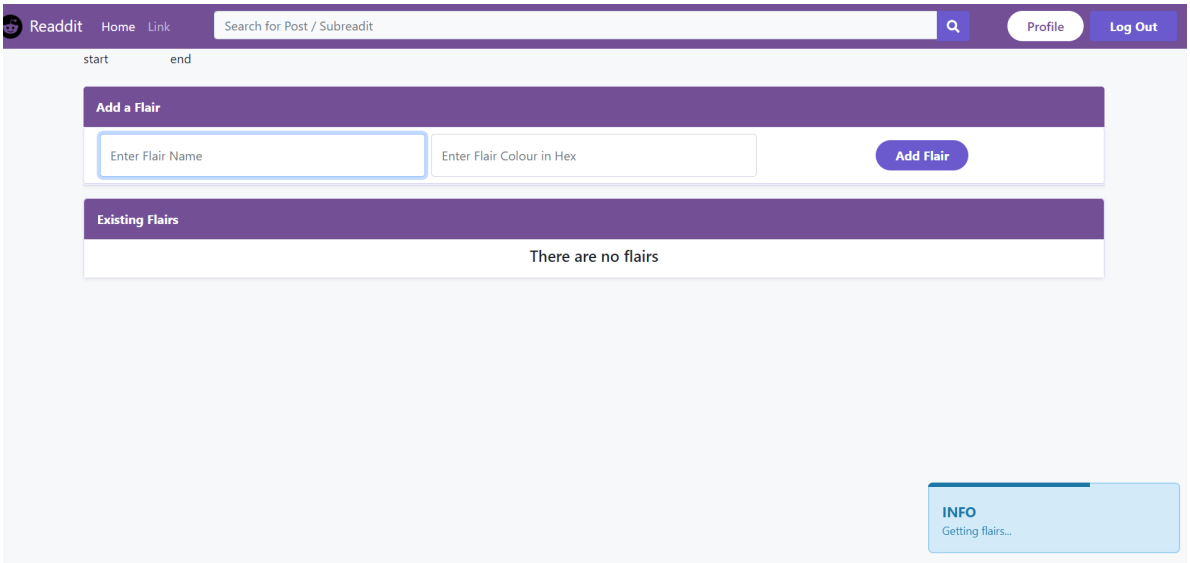
Users can view created subreaddits and all posts associated with that subreaddit. There is also a section on the right which shows the subreaddit's name, description and when the subreaddit was created.

7.3.3 Edit Moderators



Subreaddit owners can assign other users as Moderators.

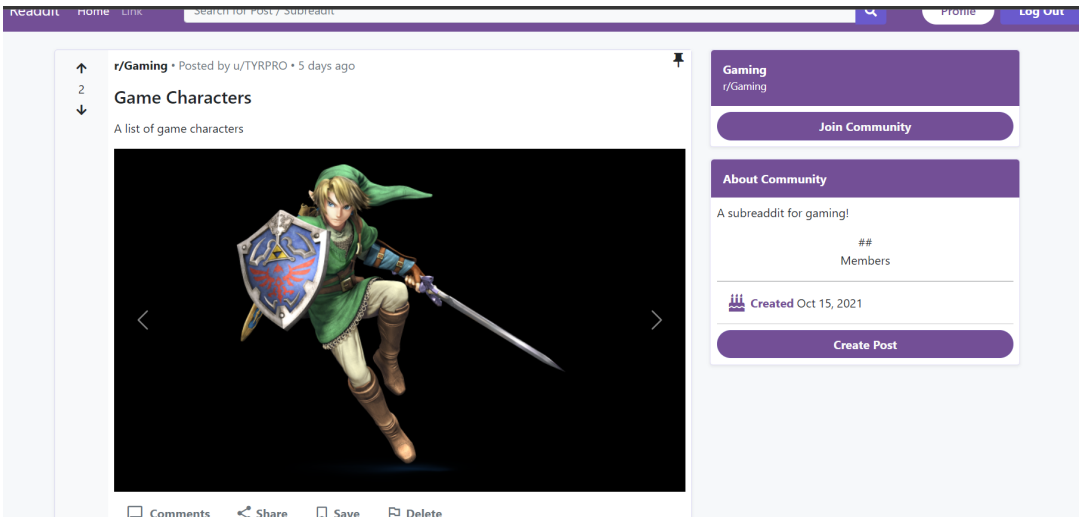
7.3.4 Edit Flairs



Moderators can create flairs for users to use in the subreaddit.

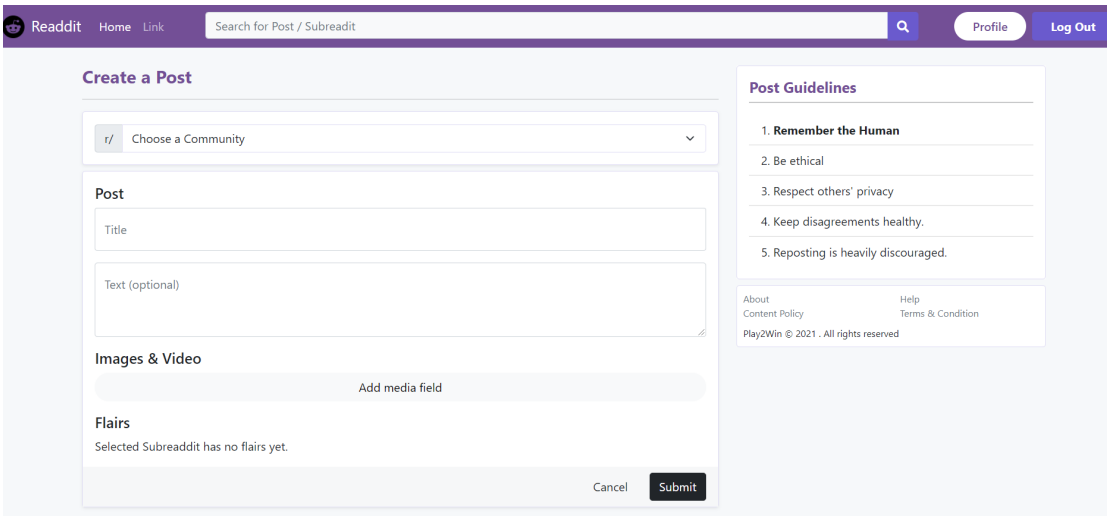
7.4 Posts

7.4.1 View Posts



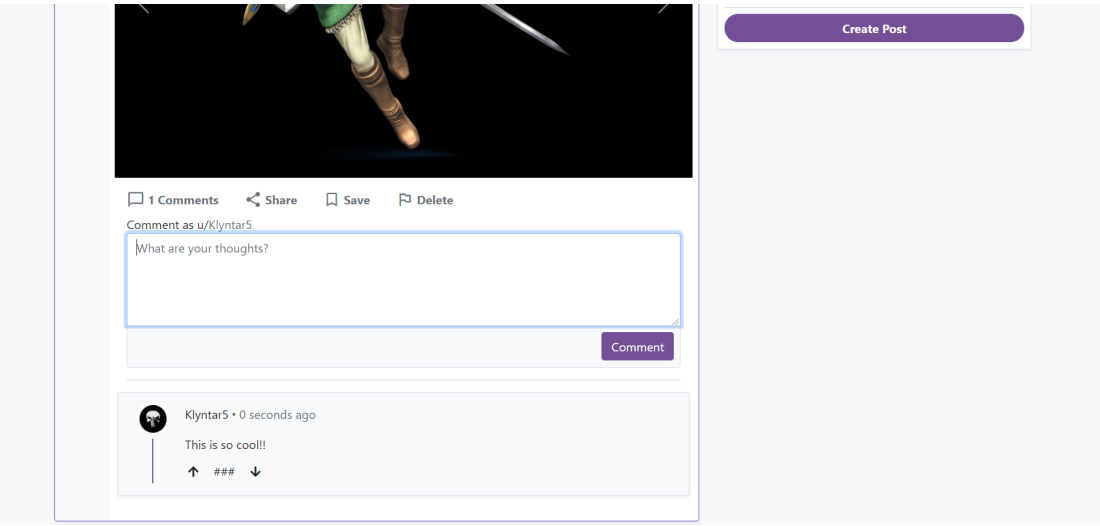
Users can view individual posts. Each post entry displays the post's title. Optionally each post may also display a description and media.
The entry will also show the post's author and how long ago the post was created.

7.4.2 Create Posts



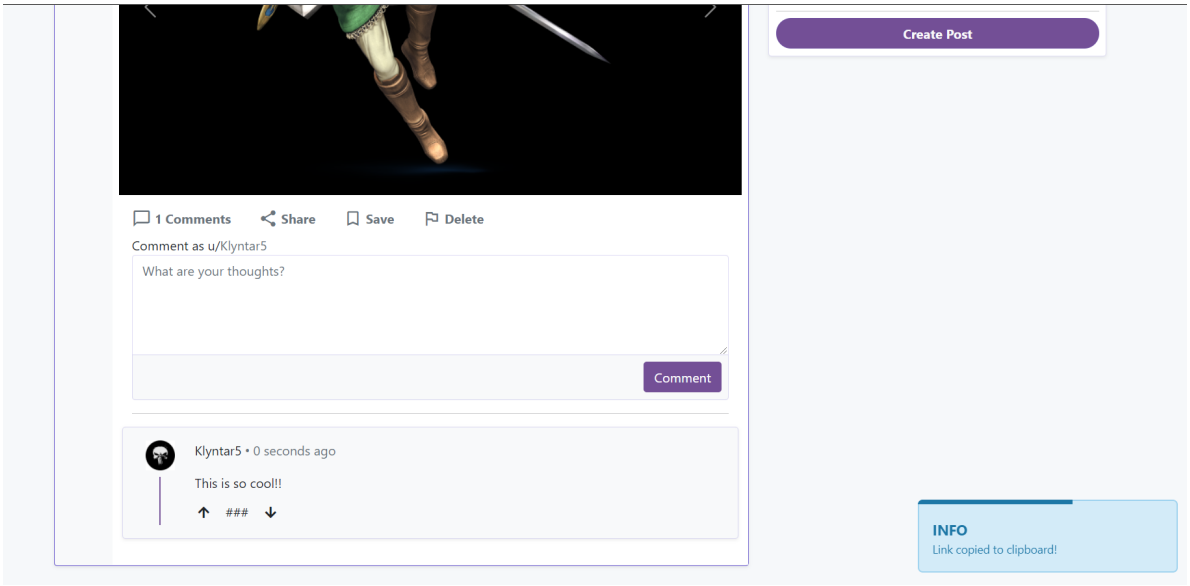
Users are able to create posts on the subreaddit of their choice. The post can contain GIFs, Videos, Images or a combination of any of those three. An optional description can also be added, but each post must be titled.

7.4.3 Comment on Posts



Users are able to comment on posts. Comments can be seen by everybody. And each comment entry shows who authored the comment and how long ago the post was made.

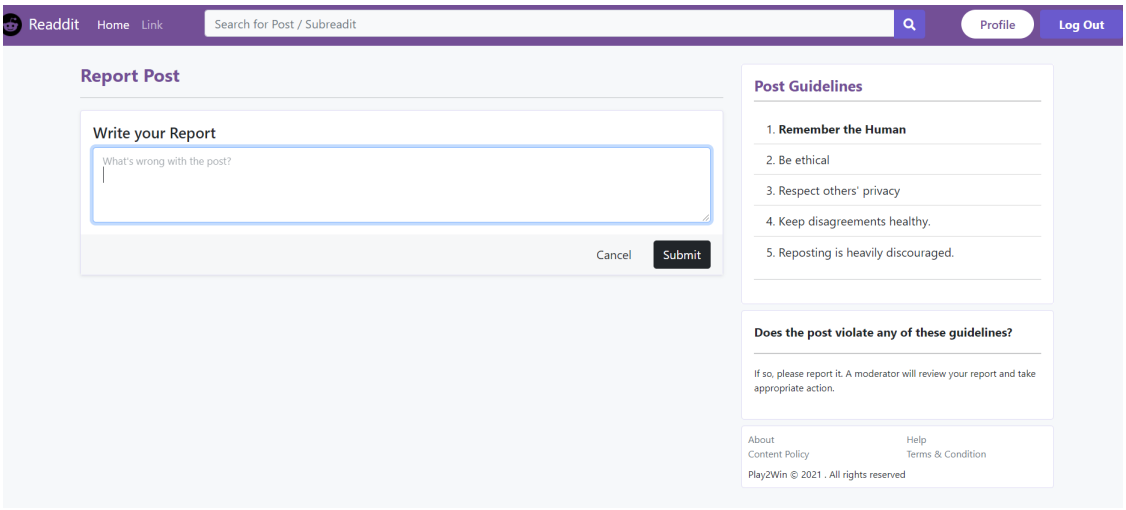
7.4.4 Share Posts



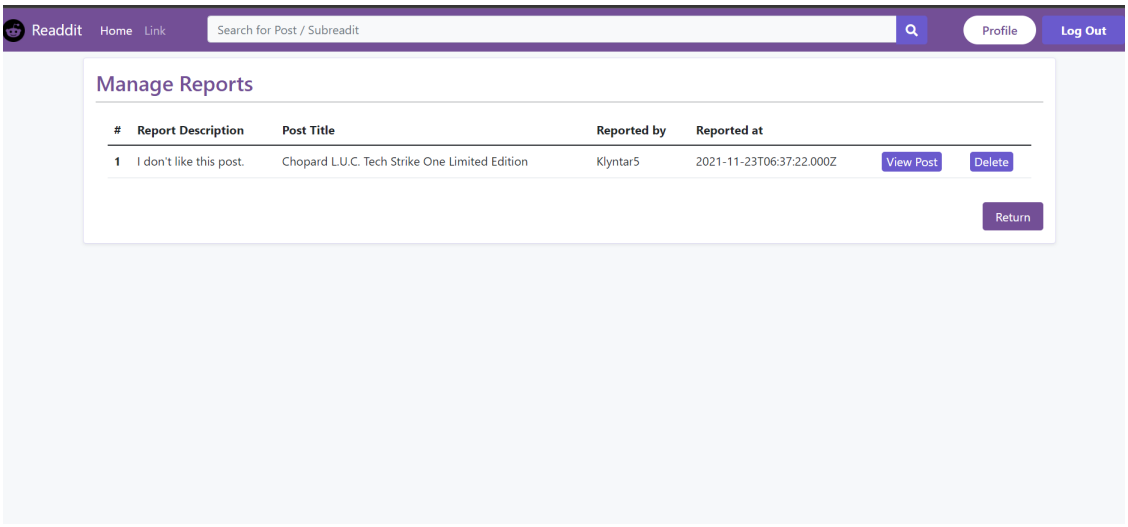
Users can share post by copying the post link when clicking the share button.

7.4.5 Report/Delete Posts

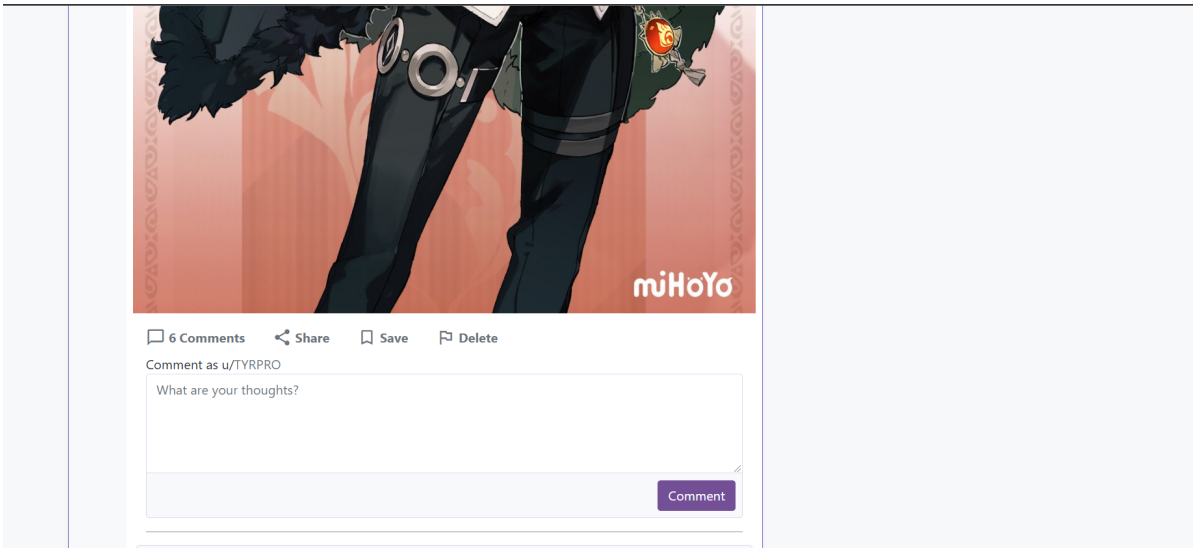
Report button



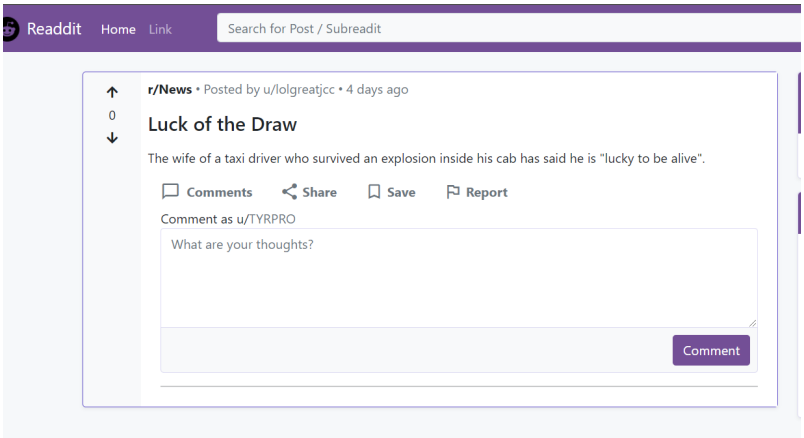
Users can report posts that they deem inappropriate.



Moderators of that subreaddit can view reports and either delete the post or delete the report.

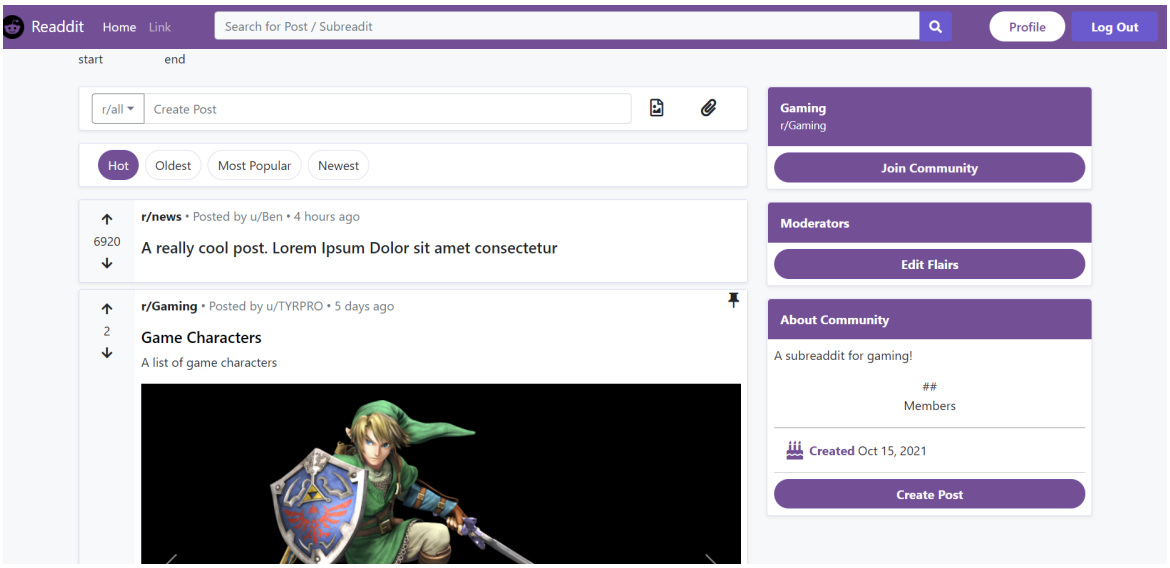


When viewing a post, moderators will be presented with a delete button



While normal users are presented with a report button.

7.4.6 Post Sorting



Subreaddits posts can be ordered by:

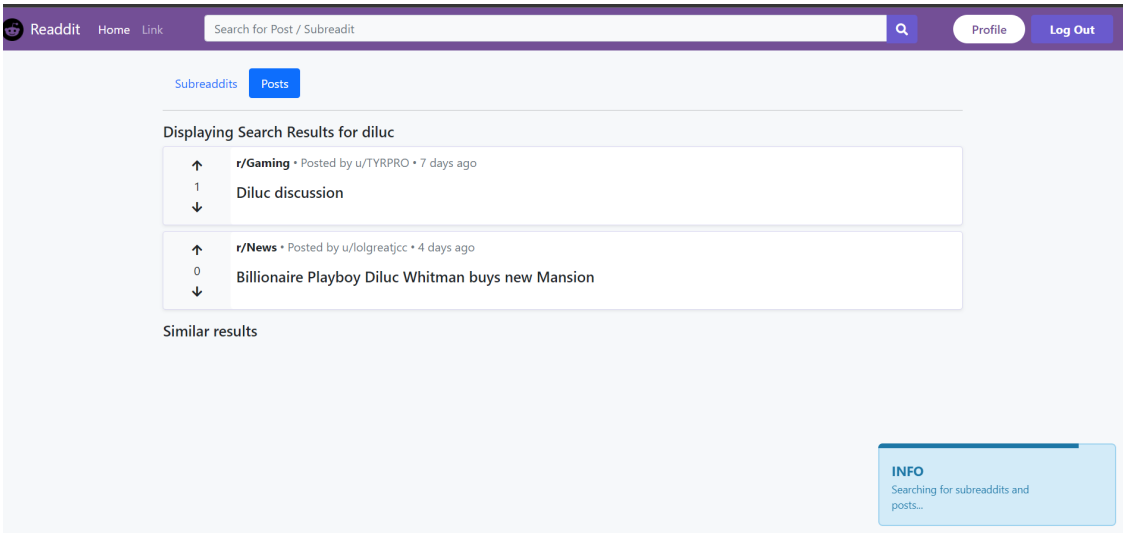
1. Hot. Sorts posts according to our sorting algorithm which takes into account the time since submission and votes.
2. Oldest. Sorts post by oldest posts first
3. Most Popular. Sorts post by number of votes
4. Newest. Sorts post by newest posts first

7.4.7 Save Posts

Users can save posts they like to view it again later in the profile page.

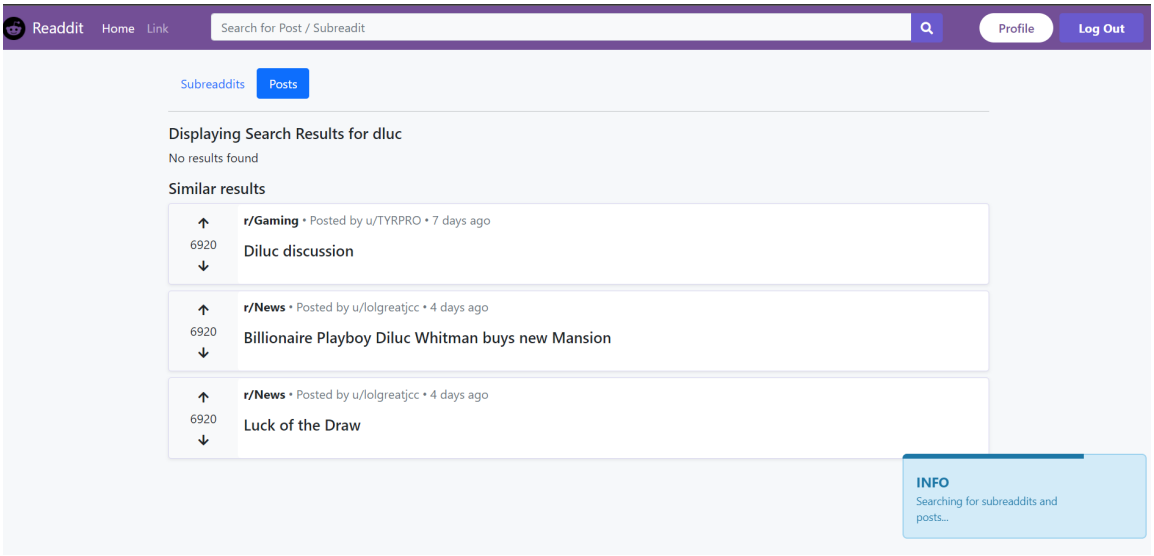
7.5 Search

7.5.1 Basic Search



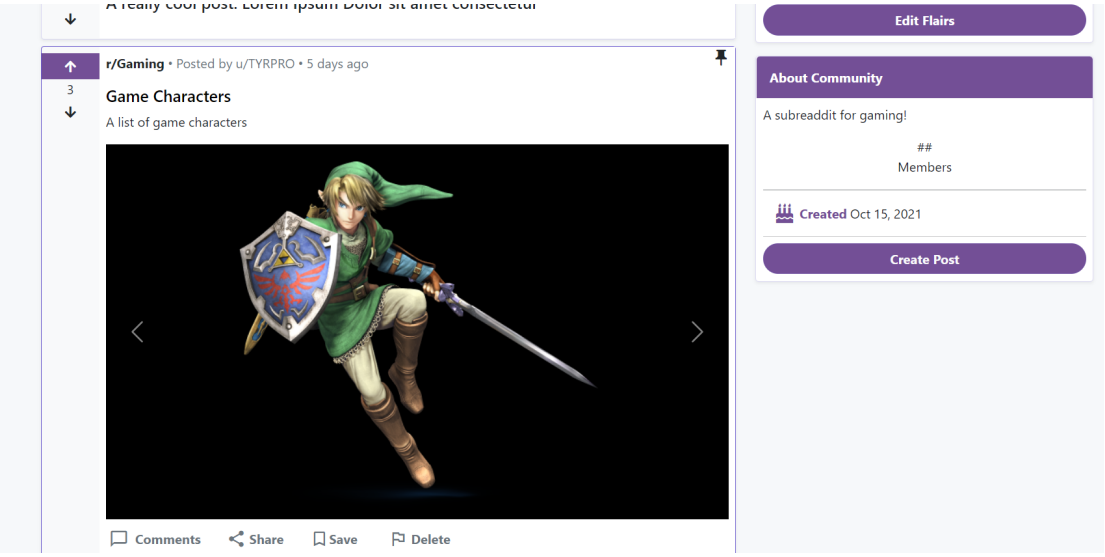
Users can search for posts or subreaddits by entering a keyword.

7.5.2 Similar Search



Similar Search displays results similar to the keyword that the basic search does not display. This is done using our own function that calculates the similarity between two strings.

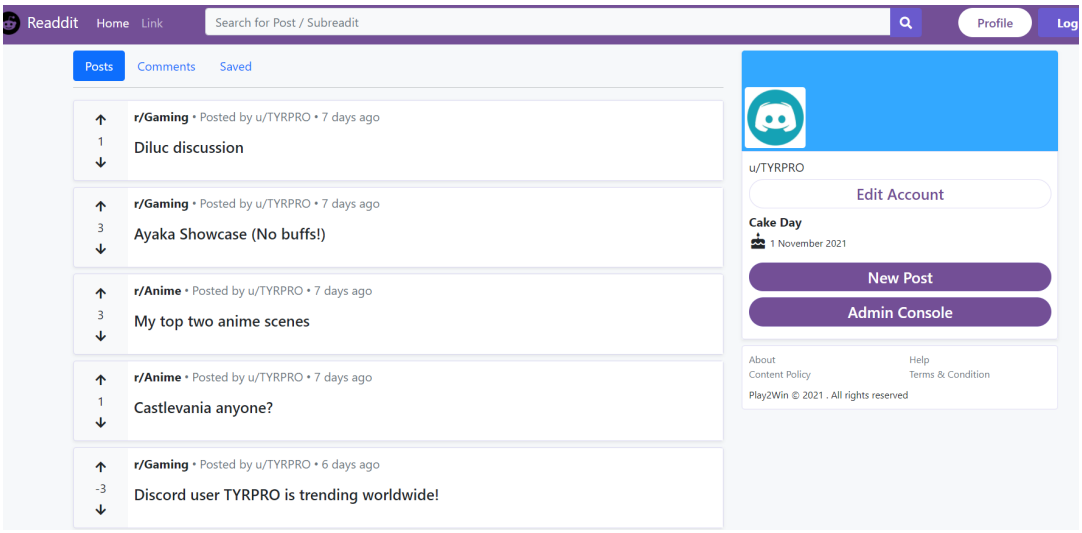
7.6 Rating System



Readdit's Rating system consists of upvotes and downvotes. The total number of votes for a post is the number of downvotes subtracted from the number of upvotes. As said previously this is used for our sorting algorithm.

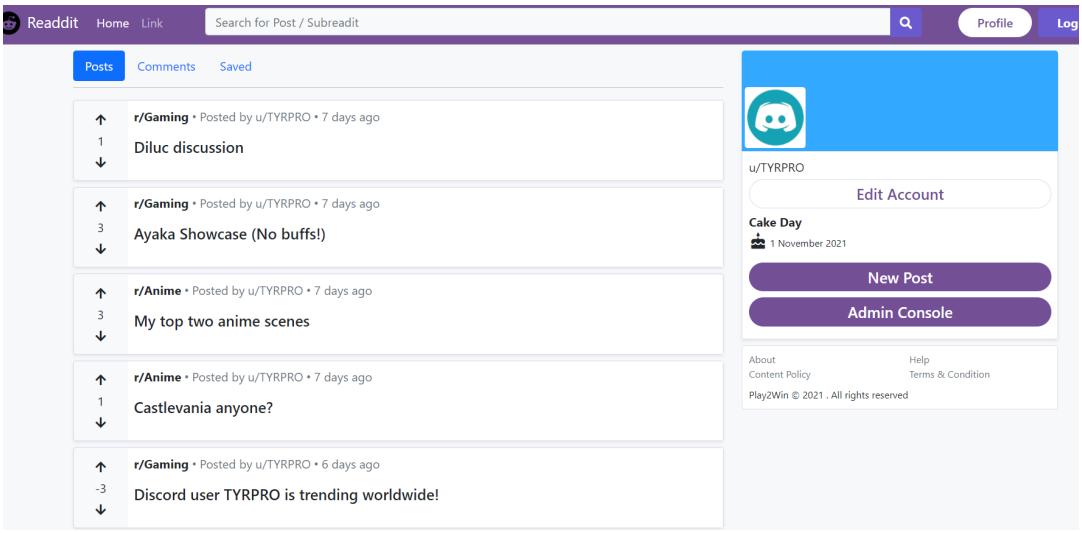
7.7 Profile

7.7.1 View User Information



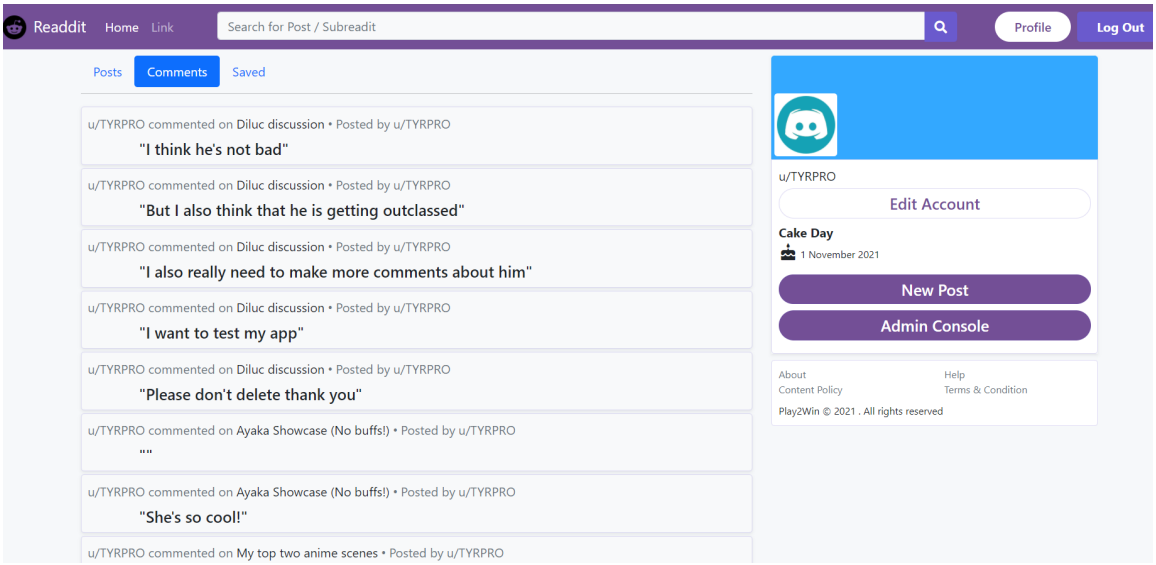
Users can view information about their account in their User Profile. (i.e. Username, Profile Picture and Account Creation Date)

7.7.2 View User Posts



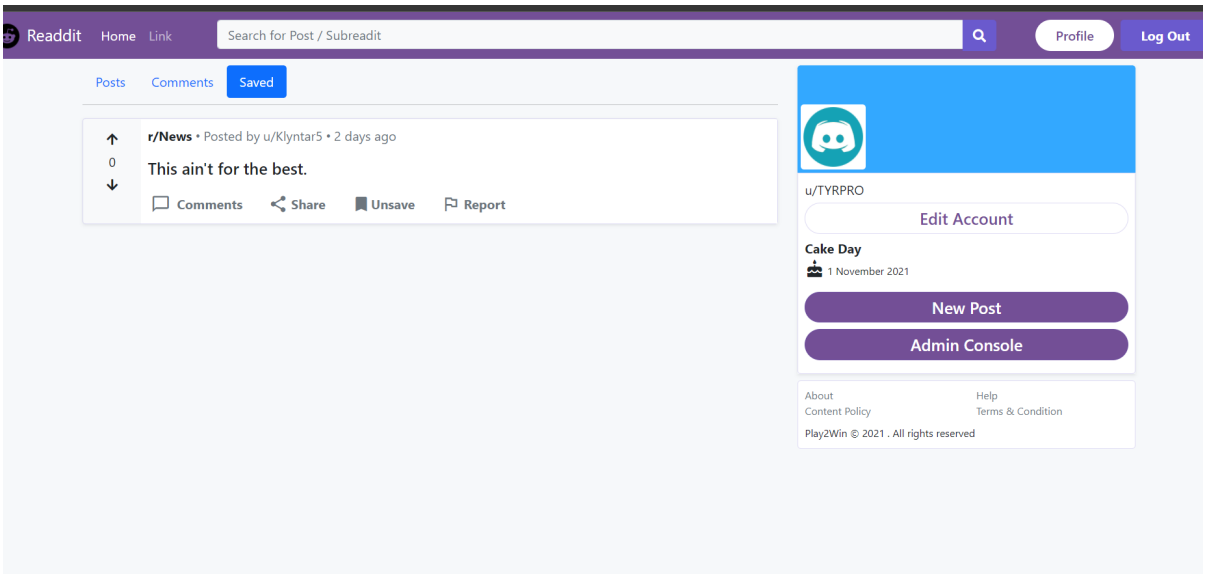
Users can also view posts they've created in User Profile.

7.7.3 View User Comments



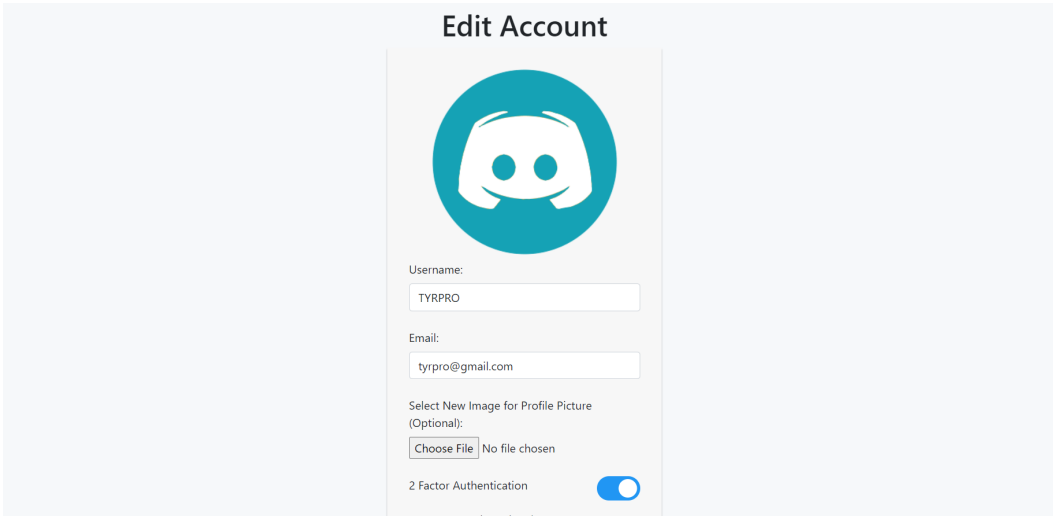
Users can view comments they've made on posts in User Profile

7.7.4 View User Saved Posts



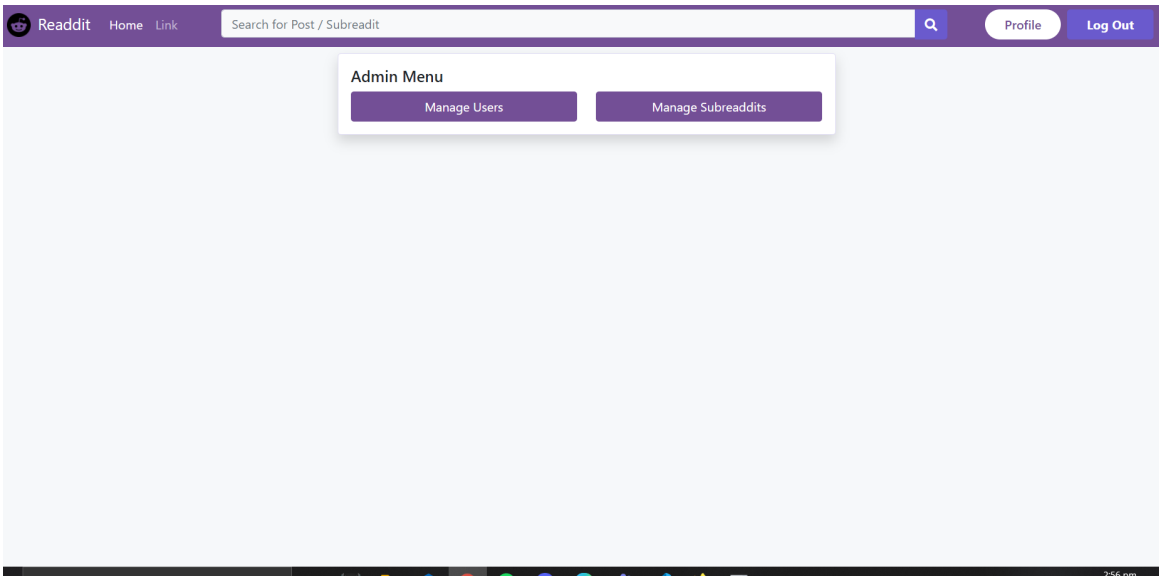
Users can also view posts they've saved in User Profile

7.7.5 Edit Account



Users can also edit their account information from the User Profile.

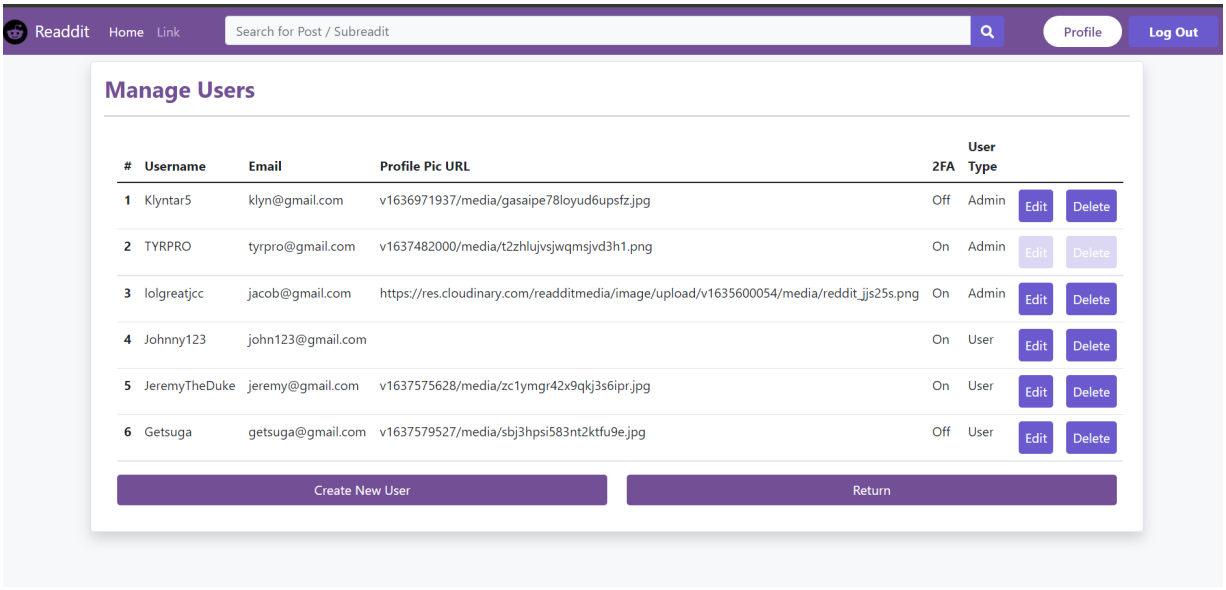
7.7.6 Access Admin Console (Admin Only)



If the User is an Admin, they will be able to access the Admin Console via their User Profile.

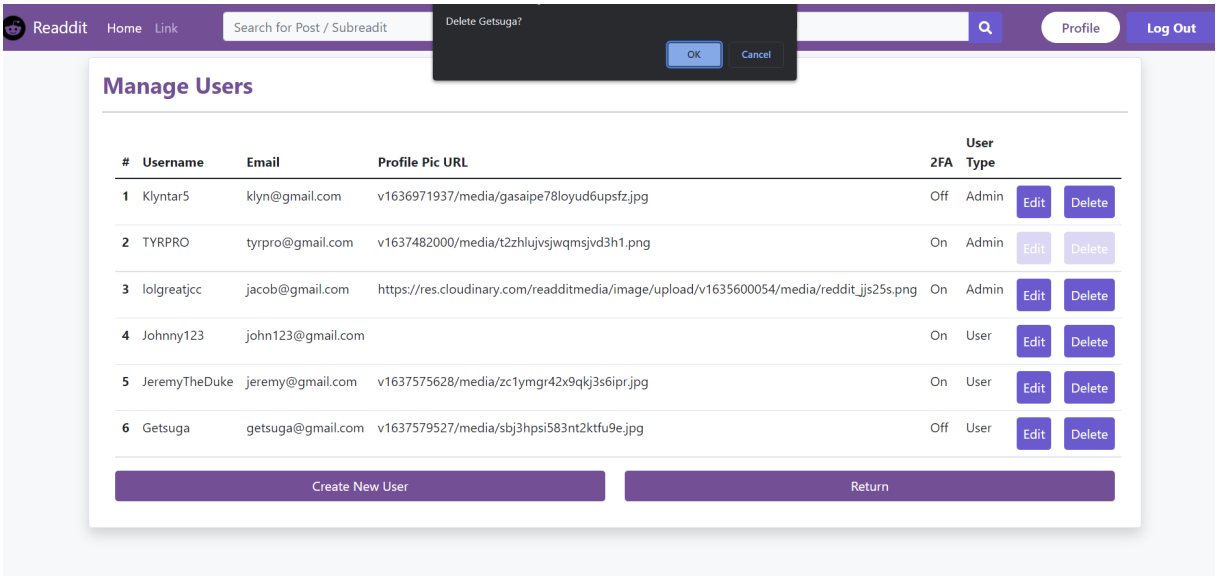
7.8 Admin Console

7.8.1 View all Users



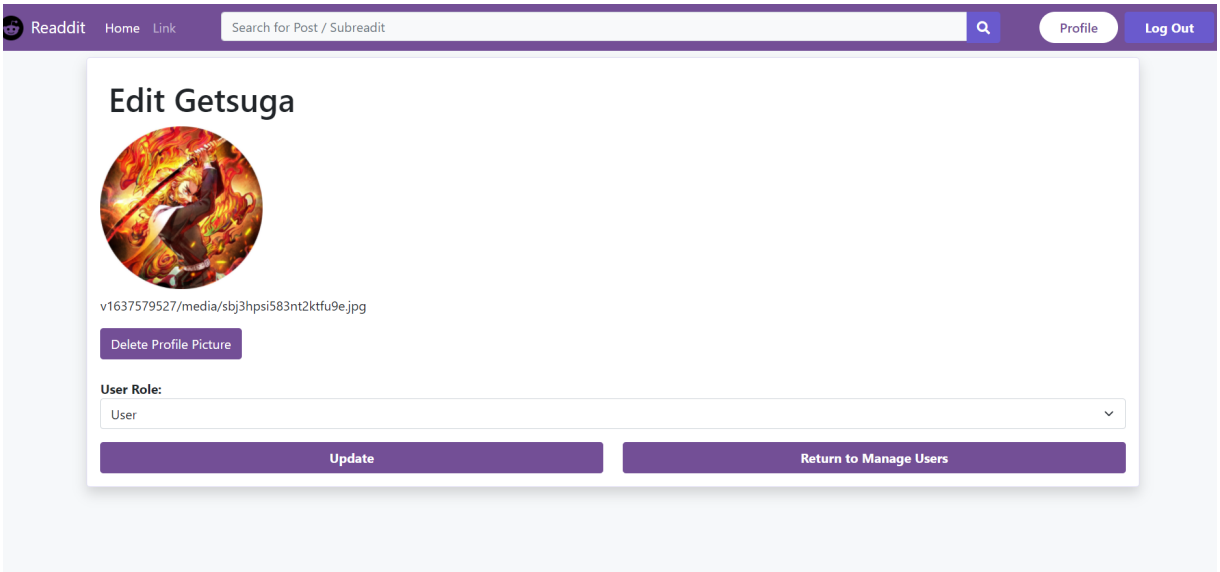
Admins are able to view all users from the database.

7.8.2 Delete User

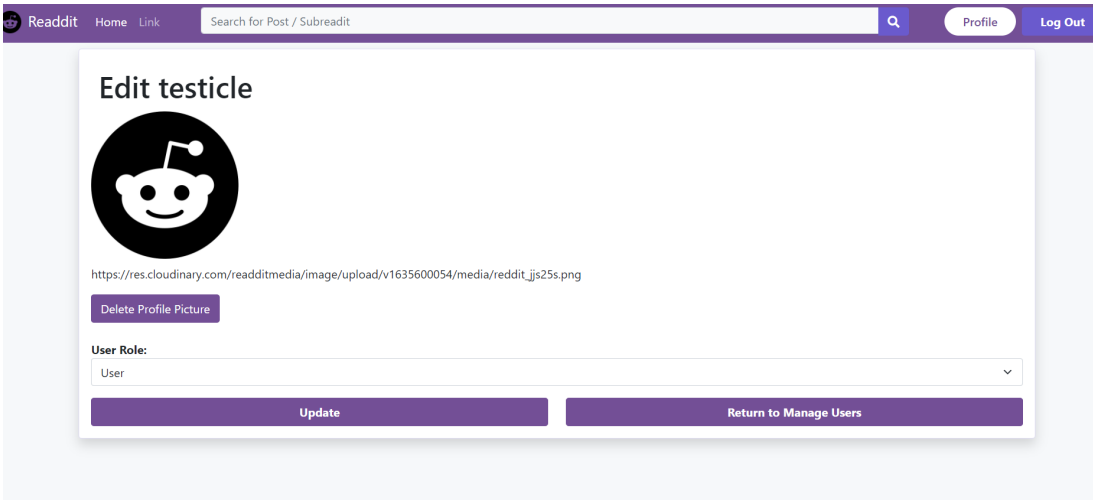


Admins are able to delete users.

7.8.3 Modify User

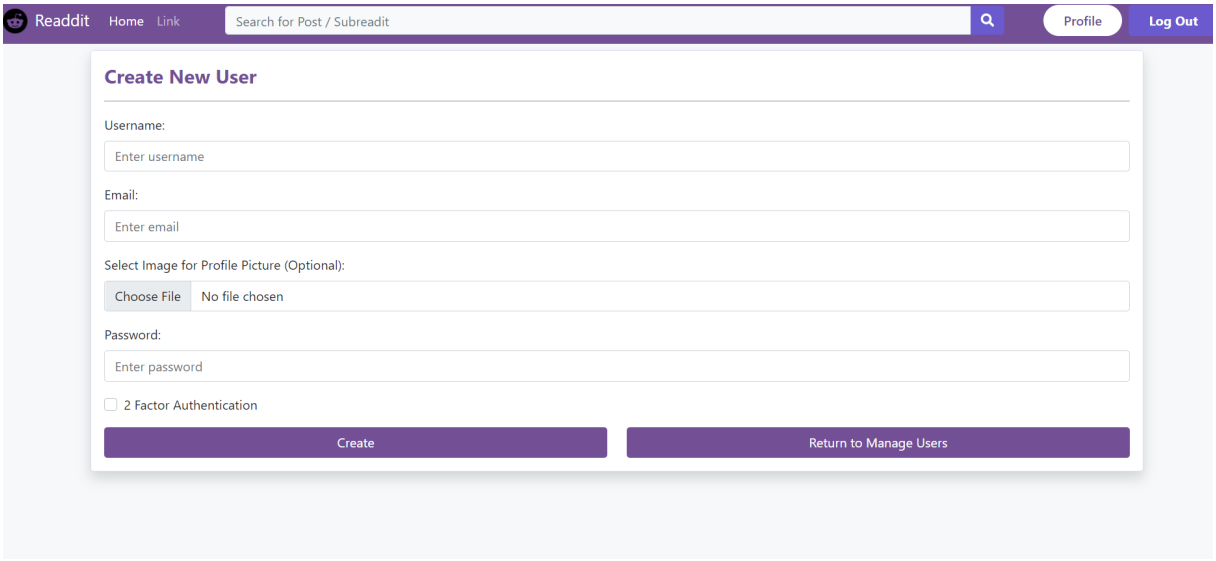


Admins are able to delete user profile pictures or change the role of a user



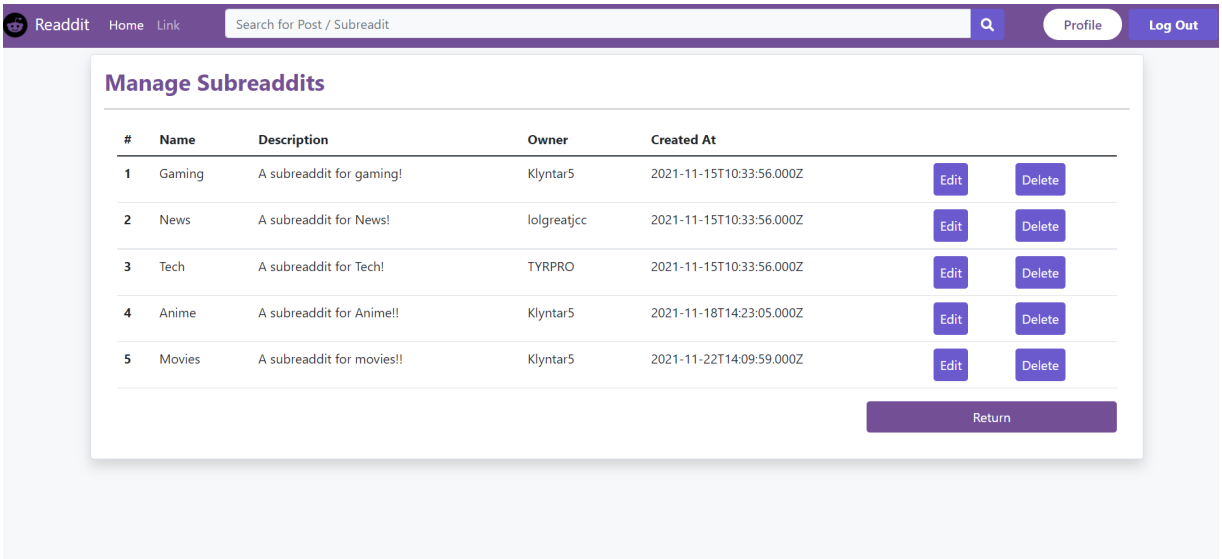
Users with deleted profile pictures will have the default profile picture.

7.8.4 Add User



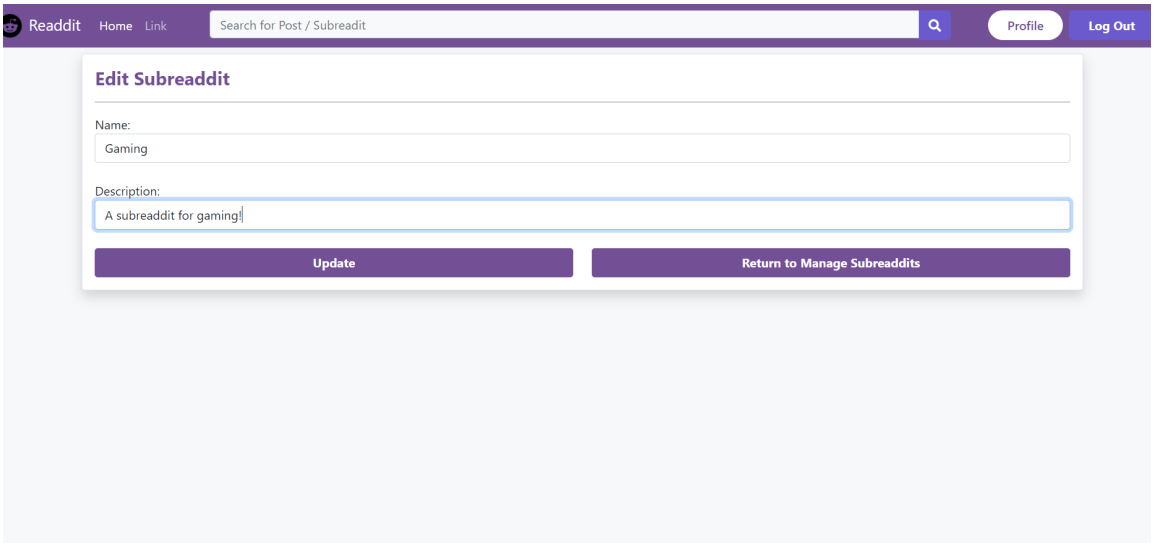
Admins are able to manually create a new user.

7.8.5 View Subreaddits



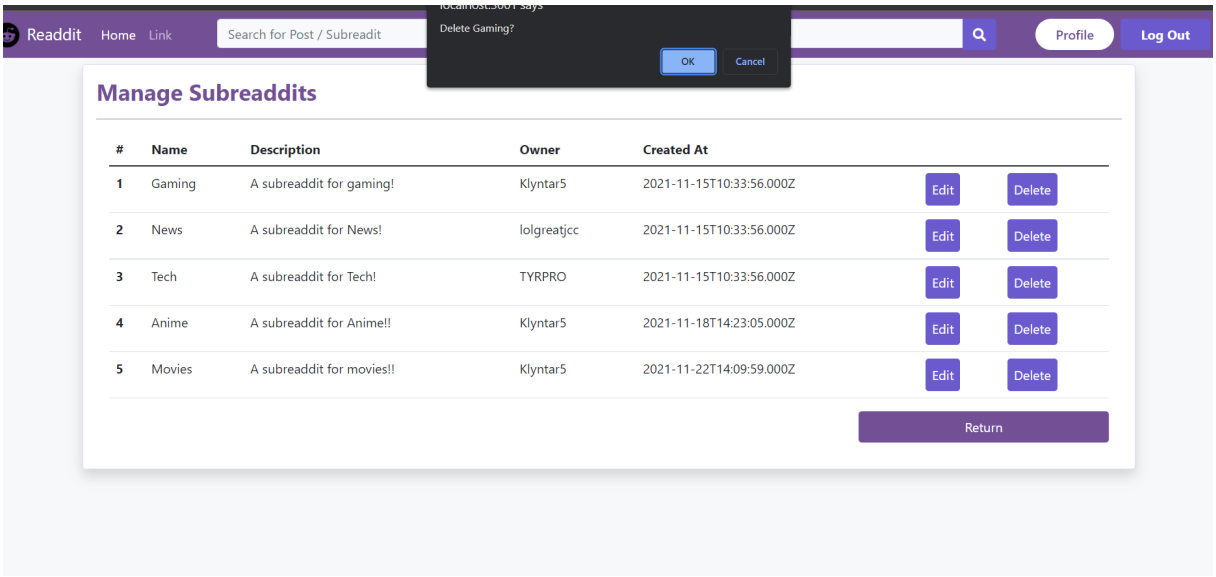
Admins are able to view all Subreaddits.

7.8.6 Modify Subreaddit



Admins are able to modify subreaddit details

7.8.7 Delete Subreaddit



Admins can delete subreaddits.

7.9 Mobile Application

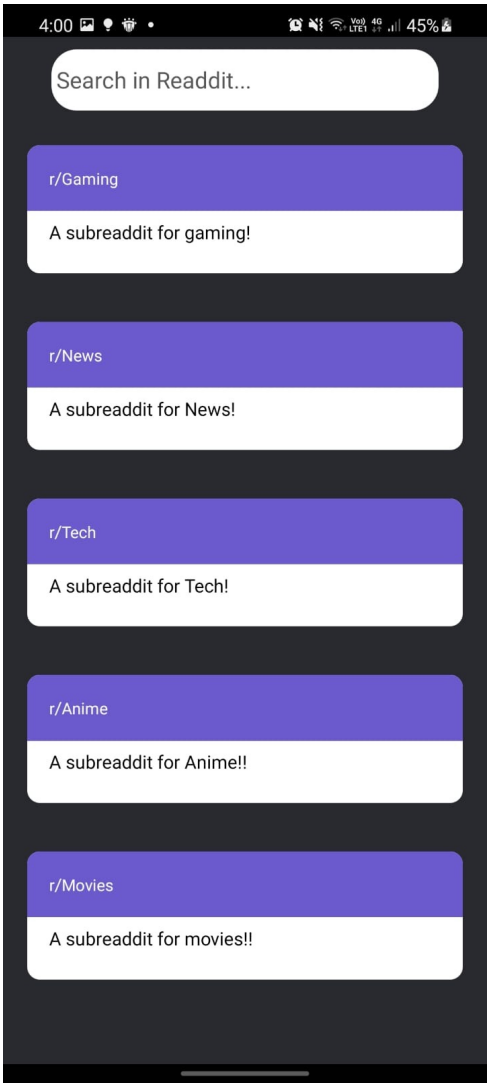
7.9.1 Homepage

When users open the application, they are greeted with the homepage. They can scroll through the same posts that are displayed on the web application's homepage:

4:00
45%
Search in Readdit...
r/Gaming • Posted By u/TYRPRO • 7 days ago
Diluc discussion
Is diluc an amazing 5*??
GENSHIN IMPACT
miHoYo
r/Gaming • Posted By u/TYRPRO • 7 days ago
Ayaka Showcase (No buffs!)
My ayaka build

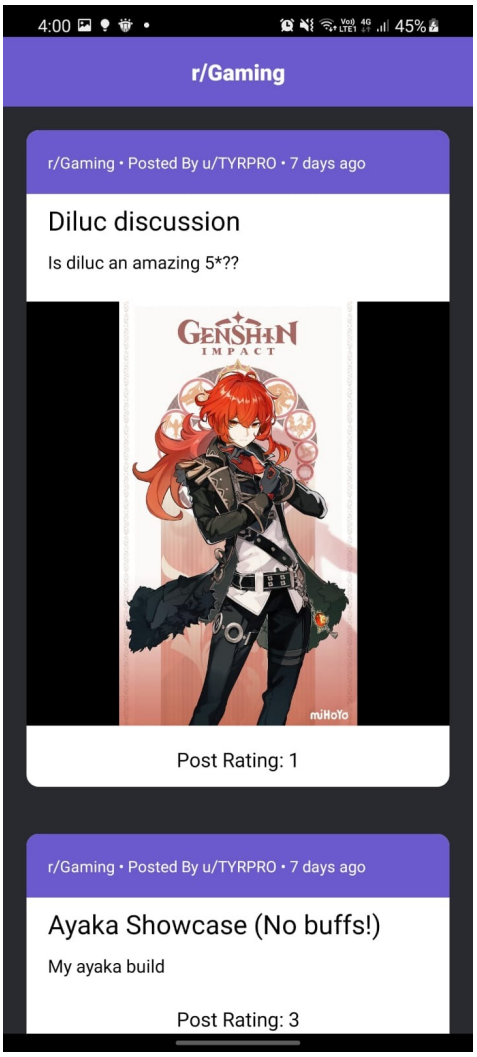
7.9.2 Search Function (Subreaddits)

Users can also search for Subreaddits in the search bar to return the Subreaddits that are similar to the Subreaddit names. Entering a blank search will return all Subreaddits:



7.9.3 Subreaddits

Clicking on a Subreaddit shows the posts within the Subreaddit. The user can tell which Subreaddit he/she is in by the header:



The posts shown in the Subreaddit and homepage will show the post title, post content, Subreaddit, posted by which user, how long ago the post was posted, the post rating, as well as any media that the post has. The post only has media support for images and GIFs at the moment.

7.9.4 Post Information

Clicking on the post shows the same content but also shows the comments: