Ryan Busk, Katie Kuenster, Tyler Sammons, Claire Sonderman

# Final Project Report: SimpleStream

**What We Accomplished:** In this project we accomplished many things. We have a simplified Reddit streaming site called SimpleStream that provides its users with their own personalized Reddit stream based on their preferences. We also created an email service for SimpleStream where users can choose the number of posts in the stream and how often they want to receive each email.

**Why SimpleStream is Useful:** This project is useful for both current Reddit users and novice reddit users. Current Reddit users can use SimpleStream to get away from all the clutter of the regular Reddit site. SimpleStream has no ads or any other irrelevant information. For each post the user only sees the post's title, score, thumbnail, author, subreddit, and the link. The experience as a whole is more streamlined and efficient. Also, if the user decides to subscribe to SimpleStream's email service, they can get their personalized reddit stream sent to them as often as they like, all without having to log on to the site each day. For many of the same reasons, SimpleStream is useful for people who have little to no experience with Reddit. Reddit can sometimes be overwhelming and confusing to someone who has never used it before. SimpleStream provides a user-friendly way to get the same Reddit posts in a more streamlined way.

**Our Data:** We got our data from Google BigQuery. We pulled roughly three months worth of Reddit Data and uploaded it to mysql via csv.

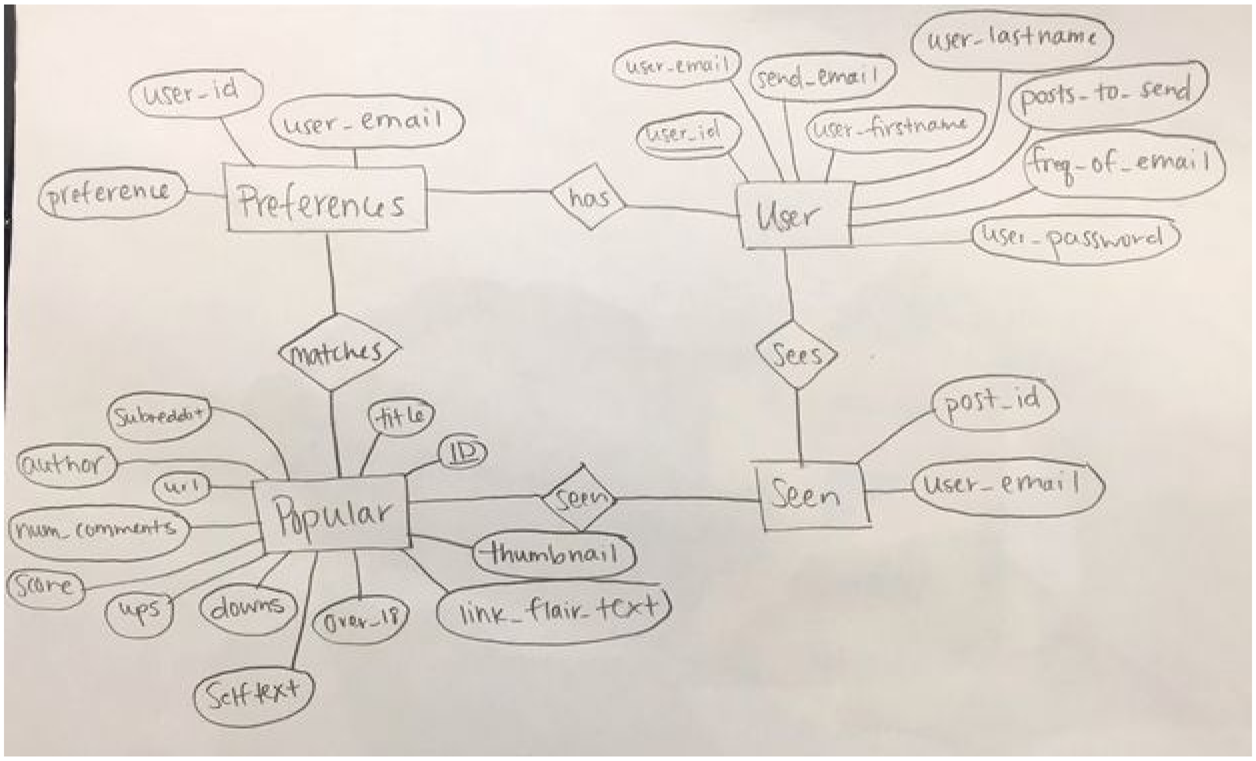

**Image 1: ER Diagram**

**User**(<u>user_id</u>, user_first_name, user_lastname, user_email, user_password, send_email, posts_to_send, freq_of_email)

**Preferences**(user_id, preference, user_email)

**Popular**(ID, subreddit, author, url, num_comments, score, ups, downs, title, selftext, over_18, thumbnail, link_flair_text)

**Seen**(post_id, user_email)

**Functionalities of SimpleStream:**

Create Account: You need to create an account to use SimpleStream. New users must enter a first name, last name, email, and password to create an account. Then a stored procedure is called, which is described below.

User Verification: If the user already exists, they log in when they first get to the SimpleStream page. We have a stored procedure that verifies the user email and password correspond with what we have in our User table in the database.

Delete Account: A user can delete their account. Using a stored procedure they will be removed from the User table, and this will also cascade to the Preferences and Seen tables.

Update Password: Each user can update their password. A stored procedure will update their password accordingly in the User table. Each password is also encrypted.

Update Account: Each user can update their account and email preferences. They can choose whether or not to receive emails, how often to receive them, and how many posts they want in each email.

Add Preferences: Each user can add preferences. A stored procedure will add this preference along with the user's email to the Preferences table.

Clear Preferences: Each user can also remove all their preferences, which removes each of their entries from the Preferences table.

View Recommended Posts: Based on the user's preferences, they can go to the home page of SimpleStream and view a new stream of Reddit posts that relate to their preferences.

View Top Posts: If the user does not have any preferences listed, they will see a stream of the top rated posts on their home stream instead.

Email Service: If the user chooses to, they can subscribe to an email service and receive their recommended or top posts straight to their inbox as often as they want.

**Basic Function:** Creating a User. Stored procedure given below.

```
procedure sp_createUser(in firstname varchar(45), in lastname varchar(45), in email
varchar(45), in password varchar(45))

begin

if (select exists (select * from User where user_email = email)) then

select 'User already exists';

else

insert into User (user_firstname, user_lastname, user_email, user_password) values (firstname,
lastname, email, AES_ENCRYPT(password, UNHEX('F3229A0B371ED2D9441B830D21A390C3')));

select count(*) from User where user_firstname = firstname and user_email = email and
user_lastname = lastname and user_password = AES_ENCRYPT(password,
UNHEX('F3229A0B371ED2D9441B830D21A390C3'));

end if;

end
```

Data Flow:

- User opens page
- User enters initial values.
- Python flask then enters initial values into mysql stored procedure.
- Stored procedure is run, adding user to User table
- Password is encrypted in stored procedure
- User is brought to main page.
- Popular posts are added to main page

**Advanced Functions:**

Get Popular Posts According to Preferences: This is a stored procedure in the teamnull database. The procedure takes in the user's email and the number of posts to return and returns that number of posts. The procedure takes all the posts from the Popular table that match preferences from the Preference table. It pulls posts from that and takes posts that have not been seen by the user already. It then sorts these posts by score and returns from the top as many that the user wants. If the user does not have preferences, it just returns the top unseen posts. This is advanced because it uses all the tables in our database and joins on the Seen and Popular tables.

Send Email with Popular Posts: This function uses the Popular posts and sends an email from our server to the user with the posts they want instead of the user having to log unto the site to see their recommended posts. The user sets preferences about the frequency of email and how many posts in the email on the site, and then receives emails to the email they entered when they signed up for the site. This is an advanced function because it required us to design and implement a schedule maker inside our server. We did this in the flask-python part of the backend.

**Technical Challenge:** The email service was challenging, specifically scheduling the emails to repeatedly send on time. We had to find a library to use or a set of commands in order to create a repeated event, but also to keep track of the ones being run to be able to stop them if the user were to unsubscribe from the emails. Originally we tried to run crontab commands, but in the end we couldn't get it to work. The final solution was to use the threading library and giving each user a thread, with a time on each thread.

**Original Plan vs. Final Product:** In our original plan, we were going to allow users to recommend posts to the Database. This would mean that not only would we have the post score, but also a separate Simple Stream score. This score wouldn't be seen by the users but would influence later post recommendations. We decided this wouldn't work for a streaming web app. We want all the input to be from reddit, and this just used as an overlay to the reddit posts.

Another thing we did not have is the ability to search exclusively by subreddit. We thought this feature wouldn't be used as much as just a general search. Also there would not be any difference between our subreddit stream and just going to that particular subreddit on reddit.

**Division of Labor:**

Claire Sonderman: Back End, Worked on the flask-python connection between the mysql database and the front end.

Tyler Sammons: Front End, Designed the page layout and helped Claire with connection.

Ryan Busk & Katie Kuenster: Back End, Designed mysql database and wrote required stored procedures in SQL, including the stored procedures necessary to return the recommended Reddit posts based on preferences.