Your first warm-up group collaboration:

- Look at the source code of our case study system SNOW, discuss and design a "retweet" functionality into the current system.
- Create another document together to explain/implement how you think the new function affects the current system.

Implementation Plan

Update the Database Schema:

- In Snow/backend/dao/models/flake.py
- we will add a retweet_of field to the Flake model under the Flake class. This field will point to another Flake instance that the current Flake is a retweet of. If retweet_of is None, the Flake is not a retweet.
- Under the same class, we will add a function called get_retweets which returns all the retweets for the flake.

Modify Backend Logic:

- In Snow/backend/api/impl/flake.py
- Under _post function: declare the variable retweet_of = None.
- If retweet_of in request.payload is true, then the post is a retweet and does not have anything in the image or reply_to variables. It will create a new flake pointing to the retweeted flake with the retweet_of field set to true.
- If retweet_of in request.payload, is false then the post is not a retweet, so no action needed.
-

Update the Frontend:

- Add a retweet button to the UI, which when clicked should call the post function with retweet already set to true and pointing to the original flake.
- The UI should also be updated to display the retweet by displaying who retweeted the post, that the post is a retweet, what time the retweet was made and display the original post under.

Impact on current system:

- The database schema needs to be updated to handle retweets.
- Will impact the database performance because more data are stored, and additional logic to handle retweets.
- The UI will be updated to allow retweeting and to display retweets.
- Retweets will change how content spreads across the platform, which can increase the visibility of certain flakes and allow more features to users.