Design Document

System Overview

This system is implemented in Python as a CLI application, with SQLite as the database, and a simulation of a social media platform. Users can interact with the system through commands to perform tasks such as posting tweets, replying to tweets, retweeting, viewing tweets, and managing hashtags. All data is stored in an SQLite database, and users can search tweets by keywords and view them sorted by date and time. The goal of this project is to provide a simple, CLI-driven tweet management system that simulates some basic functionalities of a social media platform . After login, they can:

- Post tweets
- Search for tweets or users
- View followers
- Logout

Files and Modules

The program is split into several modules, each of which handles a distinct functionality:

- **main.py**:
  - **Purpose**: Centralizes user authentication, system initialization, and command navigation.
  - **Key Features**: Handles user login, registration, and interaction with other modules.
- **follower_utils.py**:
  - **Purpose**: Manages followers and followee relationships.
  - **Key Features**: Allows users to view followers, follow/unfollow others, and explore follower details.
- **compose_tweet.py**:
  - **Purpose**: Facilitates the creation and posting of tweets.
  - **Key Features**: Handles tweet composition, inclusion of hashtags, and validation of tweet length.
- **tweet_search.py**:
  - **Purpose**: Enables searching for tweets based on keywords or hashtags.
  - **Key Features**: Implements efficient querying to return relevant tweets sorted by date and time.
- **search_users.py**:
  - **Purpose**: Allows users to find other users by specific search criteria.
  - **Key Features**: Provides functionalities to search and view user profiles.

Database Design

The system uses SQLite to store the following tables:

- users(usr, name, email, phone, pwd)
- follows(flwer, flwee, start_date)
- lists(owner_id, lname)
- include(owner_id, lname, tid)
- tweets(tid, writer_id, text, tdate, ttime, replyto_tid)
- retweets(tid, retweeter_id, writer_id, spam, rdate)
- hashtag_mentions(tid,term)

```
-- users

CREATE TABLE users

(

  usr  INT PRIMARY KEY,

  name  TEXT NOT NULL,

  email TEXT NOT NULL,

  phone INT NOT NULL,

  pwd  TEXT NOT NULL

);

-- follows

CREATE TABLE follows

(

  flwer  INT REFERENCES users ON DELETE CASCADE,

  flwee  INT REFERENCES users ON DELETE CASCADE,

  start_date DATE NOT NULL,

  PRIMARY KEY (flwer, flwee)

);

-- tweets

CREATE TABLE tweets

(

  tid     INT PRIMARY KEY,
```

```
  writer_id  INT NOT NULL REFERENCES users ON DELETE CASCADE,
  text     TEXT NOT NULL,
  tdate    DATE NOT NULL,
  ttime    TIME NOT NULL,
  replyto_tid INT REFERENCES tweets ON DELETE CASCADE
);
-- hashtag_mentions
CREATE TABLE hashtag_mentions
(
  tid  INT NOT NULL REFERENCES tweets ON DELETE CASCADE,
  term TEXT NOT NULL,
  PRIMARY KEY (tid, term)
);
-- list
CREATE TABLE lists
(
  owner_id INT REFERENCES users ON DELETE CASCADE,
  lname   TEXT,
  PRIMARY KEY (owner_id, lname)
);
-- include
CREATE TABLE include
(
  owner_id INT ,
  lname   TEXT,
  tid    INT NOT NULL REFERENCES tweets ON DELETE CASCADE,
  PRIMARY KEY (owner_id, lname, tid),
  FOREIGN KEY (owner_id, lname) REFERENCES lists ON DELETE CASCADE
```

```
);

-- retweets
CREATE TABLE retweets
(
  tid      INT NOT NULL REFERENCES tweets ON DELETE CASCADE,
  retweeter_id INT NOT NULL REFERENCES users ON DELETE CASCADE,
  writer_id   INT NOT NULL REFERENCES users ON DELETE CASCADE,
  spam     INT DEFAULT 0,
  rdate     DATE,
  PRIMARY KEY (tid, retweeter_id)
);
```

Table: users

- Columns
  - `usr`: Primary Key, unique identifier for users.
  - `name`, `email`, `phone`: User's personal details.
  - `pwd`: Password, stored securely.
- **Purpose**: Maintains user accounts.

Table: follows

- Columns
  - `flwer`: Follower ID.
  - `flwee`: Followee ID.
  - `start_date`: The date when the follow relationship started.
- **Purpose**: Tracks relationships between followers and followees.

Table: tweets

- Columns
  - `tid`: Tweet ID, Primary Key.
  - `writer_id`: User who wrote the tweet.
  - `text`: The tweet's content.
  - `tdate`, `ttime`: Date and time of posting.
  - `replyto_tid`: ID of the tweet being replied to, if any.
- **Purpose**: Stores all tweets, including replies.

Table: hashtag_mentions

- Columns
  - `tid`: Tweet ID.
  - `term`: Hashtag mentioned in the tweet.
- **Purpose**: Enables hashtag-based search and tracking.

Table: lists

- Columns
  - `owner_id`: User who owns the list.
  - `lname`: List name.
- **Purpose**: Supports user-created lists for categorization.

Table: include

- Columns
  - `owner_id`, `lname`: Combined foreign key referencing the lists table.
  - `tid`: Tweet ID.
- **Purpose**: Links tweets to specific user-defined lists.

Table: retweets

- Columns
  - `tid`: Original Tweet ID.
  - `retweeter_id`: User who retweeted.
  - `writer_id`: Original author.
  - `spam`: Flag for potential spam (0 or 1).
  - `rdate`: Date of retweet.
- **Purpose**: Tracks retweets and their metadata.

Module Details:

main.py Module:

1. **main()**
   - The main entry point of the program.
   - Checks if a database file path is provided as a command-line argument.
   - Establishes a connection to the database.
   - Initiates the login screen for the user to log in or sign up.

2. **connect(path)**

- Establishes a connection to the SQLite database using the given path.
- Opens a connection to the database and prepares a cursor for executing SQL commands.
- Ensures foreign key constraints are enabled for the database.

3. **login_screen()**
   - Displays a menu to allow users to choose between logging in as a registered user, signing up as a new user, or exiting the program.
   - Prompts the user for input to navigate to the appropriate action.
   - Handles input validation and ensures the user selects a valid option.

4. **registered_user()**
   - Handles the login process for registered users.
   - Prompts the user for a username and password.
   - Verifies the credentials against the database and logs the user in if successful.
   - If login is successful, the user's feed is displayed.

5. **unregistered_user()**
   - Allows new users to sign up by providing their name, email, phone number, and password.
   - Performs basic email format and phone number validation.
   - Inserts the new user into the database and assigns a unique user ID.
   - After sign-up, the user is given an option to go to the main menu or exit.

6. **system_functions(cursor, current_user_id)**
   - Displays a menu of system functionalities for the logged-in user to choose from.
   - Allows the user to search for tweets, search for users, compose a tweet, list followers, or log out.
   - Each option corresponds to a specific action (e.g., searching tweets or composing tweets).

7. **logout()**
   - Logs the current user out by clearing their user ID and redirecting them to the login screen.
   - Prompts the user to press Enter to return to the login screen or 'q' to quit the program.

compose_tweet.py Module:

1. **composeTweet(user_id, cursor)**
    - Validates the tweet's content for duplicates and empty text.
    - Identifies and processes hashtags within the tweet.
    - Inserts the tweet and its associated hashtags into the database.
    - Offers navigation options to view the feed, return to the main menu, or quit.

tweet_search.py Module

1. **search_Tweets(cursor, userId)**
    - Allows users to search for tweets using keywords or hashtags.
    - Displays tweets matching the search criteria.
    - Provides options to reply, retweet, or view replies for a selected tweet.

2. **reply_to_tweet(cursor, user_id, tweet_id)**
    - Prompts the user to compose a reply.
    - Validates the input to ensure no duplicate hashtags and non-empty text.
    - Allows users to reply to a specific tweet.

3. **retweet_tweet(cursor, user_id, tweet_id)**
    - Allows users to retweet a specific tweet.
    - Validates the existence of the original tweet.
    - Logs the retweet in the `retweets` table, associating it with the original tweet and writer.

4. **view_replies(cursor, tweet_id)**
    - Displays all replies to a specific tweet.
    - Fetches replies to the given tweet ID, including their content, date, time, and author information.

search_users Module

1. **search_users(cursor, current_user_id)**
    - allows searching for users and provides options for pagination, navigating back to the main menu, or displaying user details.
    - The `n` option for loading more users updates `offset` and fetches the next set of results.
    - If no more users are available, it handles this gracefully with a message.

2. **get_feed_tweets(keyword, offset=0, limit=5)**
    - This function fetches and displays tweets or retweets from users the current user follows.
    - Similar to `search_users`, it uses `offset` to load tweets incrementally.
3. **user_feed(cursor, current_user_id)**
    - Displays tweets and retweets from the users that the current user is following.
    - Fetches tweets from users the current user is following, with pagination to load more tweets.
    - Displays the tweet content, user name, and date.
    - Allows the user to move to the next set of tweets, exit, or return to the main menu
4. **get_users_list(keyword, offset=0, limit=5)**
    - Retrieves a list of users based on a search keyword. It fetches users from a database using the SQL query, filtering by user name.
    - Returns a paginated list of users that match the given keyword.
    - The query uses LIMIT and OFFSET for pagination, showing 5 users per page.

follower_utils.py Module:

1. **showFollowers(user_id, cursor)**:
    - This function shows a paginated list of followers for the given user.
    - It checks if there are more followers to display, offering the option to load more or view the details of a specific follower.
2. **getFollowerList(offset=0, limit=5)**:
    - Retrieves a list of followers for the current user using SQL queries.
    - Supports pagination with an offset and limit, displaying 5 followers at a time.
3. **showFollowerDetails(follower_id)**:
    - Displays detailed information about a specific follower, such as their tweets and contact information.
    - Allows interaction with the follower, including viewing more tweets or following/unfollowing them.

Group Work Breakdown Strategy

The work is divided among the team members as follows:

- **Luke Thomas:**
    - **Task:** Implement the `tweet_search.py` module, which handles searching tweets within the system. This includes query functionality, filtering by hashtags or keywords, and fetching relevant results for display. Also responsible for setting up the database and integrating the searching features with the database.

    - **Estimated Time:** 15 hours

- **Yuheng Li:**
    - **Task**: Were responsible for developing the `followers_utils` module, which manages the followers and followee relationships within the application. This includes implementing the functionality to follow and unfollow users, checking the follower status of a user, and providing necessary methods to retrieve lists of followers and the users a person is following.
    - **Estimated Time**: 15 hours

- **Anant Gupta:**
    - **Task:**
        - Tasked to create a module to implement compose tweet in main.py. I accomplished this by writing a function that asked the user for a tweet within input constraint and updating all the schemas.
        - Writing the logout function in main.py and optimized all the codes of different files.
        - Working along with Luke on tweet search to create a CLI to be more user friendly.
        - Develop the login and logout functionality (in `login_screen()` and `logout()` methods), handling user sessions, including error handling and input validation for both registered and unregistered users.
        - Work on the `search_users.py` module to implement user search functionalities, which includes searching for users based on input and returning relevant results.
        - Contribute to the creation and optimization of `main.py`, ensuring the overall structure is logical, smooth, and efficient for the system's flow.
        - Handle the integration and ensure the optimization of different parts of the application.
    - **Estimated Time:** 20 hours

- **Gurbaaz Gill:**
    - **Task**:
        - Implemented the system_functionalities and search_user after initially creating main.py.
        - Complete the login interface and user_feed that shows all the latest tweets. Moreover,
        - Modify a lot of follower_util to connect all the relational schemas together and create a smooth CLI.
        - Implement the `compose_tweet.py` module, responsible for the creation of tweets and ensuring the tweet composition flow works as expected.
        - Optimize different parts of the application, focusing on enhancing the performance and reducing redundancy where possible.
        - Ensure that the logout functionality is implemented and properly integrated with the main application.

- **Estimated Time:** 20 hours

Coordination Method:

The project uses Git for version control to manage code contributions. Team meeting was held weekly to discuss progress and resolve any issues.