Quacker

Omar Mahmoud, Nasif Qadri, Yousef Moussa

Design Document - CMPUT 291 Mini Project 1

2024-11-14

1 Overview and User Guide

1.1 System Overview

Quacker is a lightweight social media platform that enables users to share short posts (*Quacks*), follow other users, and interact with users.

1.2 Features

The core features of Quacker include:

- **User Authentication:** Users can log in or sign up for an account.
- **Posting Quacks:** Users can create new Quacks or reply to existing ones.
- **Feed Management:** Users can view a personalized feed containing Quacks from people they follow.
- **Search Functionality:** Users can search for other users or specific Quacks by keywords or hashtags.
- **Follow System:** Users can follow others to curate their feed.

1.3 User Guide

This section provides a step-by-step guide to using Quacker.

Getting Started

1. Launch the Quacker application.
2. On the welcome page, choose from the following options:
   - Press **1** to log in to your account.
   - Press **2** to sign up for a new account.
   - Press **3** to exit the application.

Creating an Account

1. Select **2** (Sign Up) on the start page.
2. Enter your details:
   - **Name:** Your full name.

- **Email:** A valid email address.
- **Phone Number:** A 10-digit phone number.
- **Password:** A secure password.

3. If successful, you will logged in.

Logging In

1. Select **1** (Log In) on the start page.
2. Enter your **User ID** and **Password**.

Using the Feed

1. View the latest Quacks from people you follow.
2. Navigate using these options:
   - Press **1** to see more Quacks.
   - Press **2** to see fewer Quacks.
   - Press **3** to search for users.
   - Press **4** to search for Quacks.
   - Press **5** to reply or repost a Quack from your feed.
   - Press **6** to list your followers
   - Press **7** to create a new Quack.
   - Press **8** to log out.

Creating a Quack

1. Navigate to the posting page by selecting **7** from the feed menu.
2. Enter the text of your Quack.
3. Press **Enter** to post the Quack.

Searching for Users or Quacks

1. Select **3** (Search for Users) or **4** (Search for Quacks) from the feed menu.
2. Enter a search term (e.g., a name or hashtag).
3. View the results and interact with users or Quacks as desired (follow, reply, etc...).

Logging Out

1. Select **8** from the feed menu to log out of your account.
2. You will be redirected to the start page.

Running the Application

1. **Compile the Application:**
   Use the provided *Makefile* to compile the code by running:

   ```
   make
   ```

2. **Start the Application:**
   Execute the application with a database filename:

   ```
   build/quacker <database_filename>
   ```

2 Software Design

The application consists of the following key components:

Quacker Class

The *Quacker* class is the main entry point of the application and manages user interactions. It provides functionality such as:

- **User Authentication:** Handles login and signup using methods *loginPage()* and *signupPage()*.
- **Feed Management:** Displays the user feed using the *mainPage()* function, and interacts with the *Pond* class for data.
- **Posting Quacks:** Allows users to post and reply to Quacks using the *postingPage()* and *replyPage()* methods.
- **Search Functionality:** Facilitates searching for users and Quacks using *searchUsersPage()* and *searchQuacksPage()*.

Pond Class

The *Pond* class handles all database interactions and encapsulates the data logic. It includes:

- **User Management:** Provides methods for adding users (*addUser()*) and validating login credentials (*checkLogin()*).
- **Quack Management:** Handles operations like adding new Quacks (*addQuack()*), replies (*addReply()*), and managing hashtags.
- **Feed Retrieval:** Generates personalized user feeds (*getFeed()*), combining tweets and requacks in chronological order.
- **Search Operations:** Performs database queries for searching users (*searchForUsers()*) and Quacks (*searchForQuacks()*).

Component Interactions

The application follows a structured flow of data between the *Quacker* and *Pond* classes:

1. The user interfaces with the *Quacker* class, which calls the relevant *Pond* methods for database interactions.
2. The *Pond* class processes SQL queries to retrieve, insert, or update data in the database.
3. Results from the *Pond* class are used to render user-friendly views in the *Quacker* class.

3 Testing Strategy

In testing this program the Python script *populate_db.py* was used repopulate the testing database with new unqiue data improving the manually typed tests ran on the project and increasing the number of tested edge cases. To generate simply run:

```
python3 test/populate_db.py
```

The script should then return the data of a randomly generated user. This is to make it easier to locate a valid login to use when entering the program. Example:

```
Random User Data:
User ID: 53
Name: Deborah Scott
Email: jasminchang@example.com
Phone: 4567624481
Password: E1AjR1ib*t
SUCCESS! Database populated with random test data.
```

The test script also allows you to easily change parameters regarding the generated data. These can be located in the defintion of *populate_db()* and are self-explanitory as seen below:

```python
def populate_db(db_name, user_count=100, tweet_count=500, list_count=200,
    follow_count=300):
```

When it came to testing the application, this was done manually due to the page centric format of our user interface. The felxibilty of testing manually allowed our team to catch unique edge cases that may have passed over an automated system.
We tested all the possible ways a user could interact with the program, making sure the program could handle erroneous inputs

4 Group Work Breakdown

We divided tasks based on strengths and collaboration needs. Below is a summary:

- **Nasif Qadri:** Responsible for backend development, including database design and implementation of the *Pond* class.
- **Yousef Moussa:** Focused on frontend development and the user interface within the *Quacker* class.
- **Omar Mahmoud:** Managed testing strategies, the data population script, debugging edge cases, and documenting/commenting code.

However, all of us ended up helping each other out with different parts of the project, so these these roles were not strictly defined