AggieBook Iteration 2 – Client Server

Team 25

Longxiang Li: 634005819 (individual)

Video Recording

Youtube link: https://youtu.be/9JBYYFNCAsE

Descriptions of user case

Use Case 1: User Registration

A new user wants to register an account on the AggieBook.

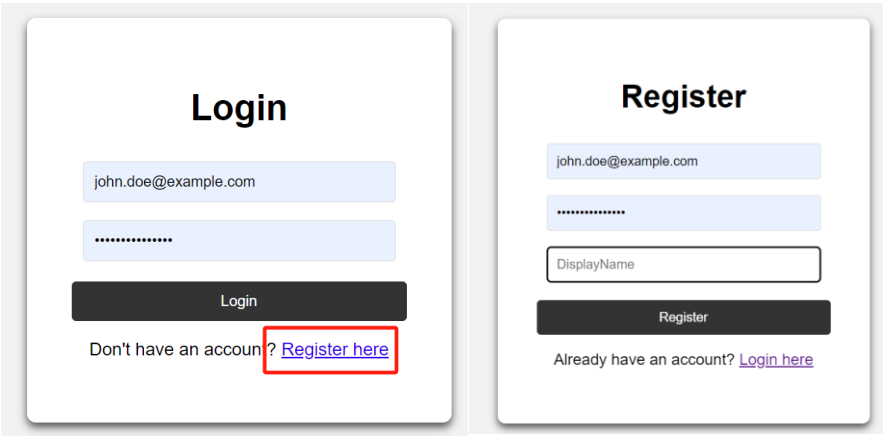
- **User Actions:**

1. User clicks on "Register here" button navigating from the login to register page.
2. User fills in the required fields: username, email, and password.
3. User clicks on "Register" button.

- **System Reactions:**

1. System validates the input data.
2. If the data is valid, the system creates a new account and notifies the user with a success message. (a “success” message makes a local navigation back)

UI Sketch:



Use Case 2: User Login

A registered user wants to log into the Aggiebook.

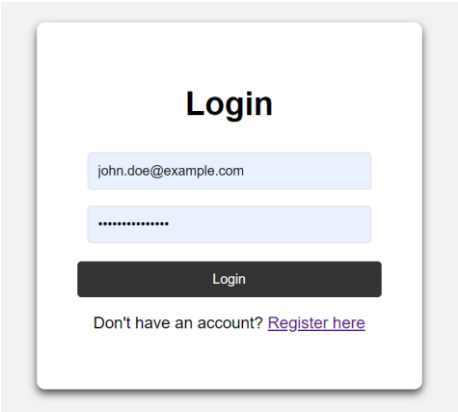
- **User Actions:**

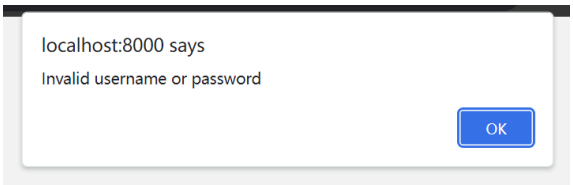
1. User enters username and password.
2. User clicks on "Login" button.

- **System Reactions:**

1. System verifies the username and password combination.
2. If the credentials are correct, the system logs the user in and return corresponding data (Userid, timeline, personal_info, following_counts, followed_counts) for local homepage construction.
3. If the credentials are incorrect, the system displays an error message.

UI Sketch:





Use Case 3: Modify Personal Information

A logged-in user wants to update their personal information.

- **User Actions:**

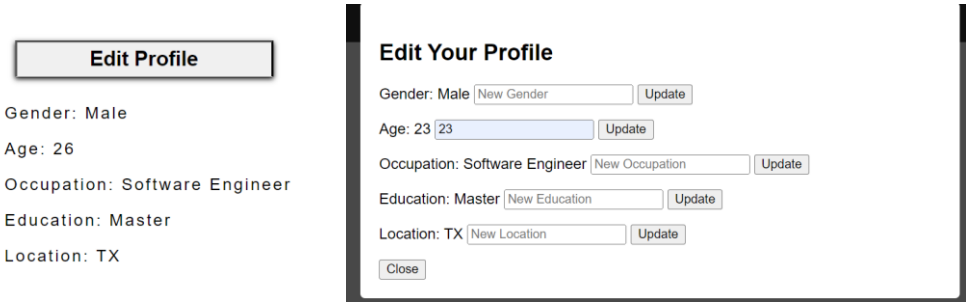
1. User navigates to the "homepage" section.
2. User clicks on "Edit Profile" button.
3. User modifies desired fields such as gender / age / etc.
4. User clicks on "update" button.

- **System Reactions:**

1. System validates the modified data.
2. If the data is valid, the system updates the user's profile and shows a success notification.
3. If the data is invalid, the system displays an error message.

4. Return the updated personal info to the frontend.

UI Sketch:



Use Case 4: Make New Story Post

A logged-in user wants to share a new story post.

- **User Actions:**

1. User clicks on "New Story" button in the homepage.
2. User enter title, contents.
3. User clicks on "complete" button.

- **System Reactions:**

1. System saves the story.
2. Add the post to the timelines of the current user and any user who follows the current user.
3. Return the updated timeline for the current user to the website.

UI Sketch:

Use Case 5: Search Friends

A user wants to search other users with display names.

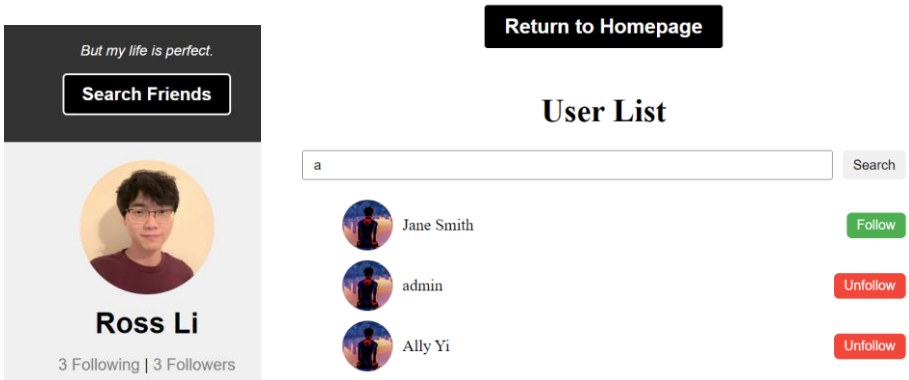
- **User Actions:**

1. User clicks on "search friend" button in the homepage to navigate to friend searching page.
2. User enters a token into the "Search" bar.
3. User clicks on the “search” button in the friend searching page.

- **System Reactions:**

1. System provides real-time search results as the user types.
2. System displays result to the web client.

UI Sketch:



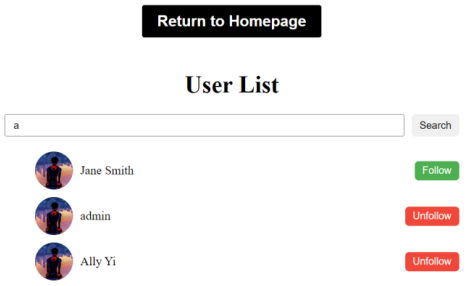
Use Case 6: Follow/Unfollow

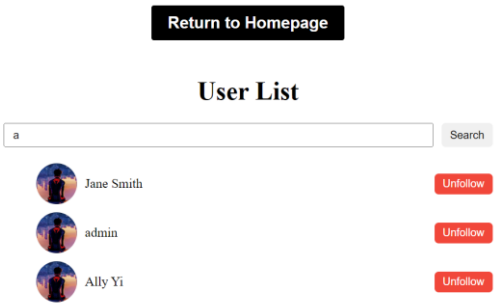
A user wants to follow or unfollow another user.

- **User Actions:**
1. User has done the friend searching to obtain the result.
2. User clicks on "Follow" button to start following or "Unfollow" to stop following the user.

- **System Reactions:**
1. If the user clicked "Follow", the system adds the profile to the user's following list and possibly notifies the followed user.
2. If the user clicked "Unfollow", the system removes the profile from the user's following list.
3. Refresh the friend searching page for next searching operation.

UI Sketch:





Use Case 7: Like a Post

A logged-in user wants to express appreciation for a post by liking it.

- **User Actions:**
1. User browses on the timeline of its homepage.
2. User clicks on the "Like" button beneath a post.

- **System Reactions:**
1. System increments the like count for the post.

UI Sketch:

Use Case 8: Comment on a Post

A logged-in user wants to leave a comment on a post.

- **User Actions:**
1. User browses on the timeline of its homepage.
2. User clicks on the "Comment" button beneath a post.
3. User types in their comment in the provided text box.
4. User clicks on "Add Comment" or a similar action button.

- **System Reactions:**
1. System adds the user's comment below the post.

UI Sketch:

Database design

1. Description of data entities and relationships as well as ER diagram: ./sql/databaseDescription.pdf.
2. SQL Code to design database: ./sql/databaseCreation.sql
3. SQL Code for sample data: ./sql/databaseInsertion.sql

Architecture design

Client (Frontend) Design

The client-side is developed using HTML, CSS, and JavaScript. It consists of four main webpages:

- **Login Page (./frontend-web/login.html)**: This is the entry point for users,

allowing them to input their credentials and gain access to their profile.

- **Registration Page (./frontend-web/register.html):** New users can create an account by providing the necessary information (username, password and display name).
- **Homepage (homepage.html):** After logging in, users are directed to the homepage where they can view and modify personal information, timeline posts, and generate a new story post.
- **Friend Page (friendpage.html):** Users can visit this page to search and follow/unfollow other users by part of their display names.

Server (Backend) Design

The server-side is powered by a Python HTTP Server which handles incoming client requests and responds accordingly. More specifically, it keeps listening to the local port 8000. Currently it won't record the information of client permanently. The customer data such as userID are stored in the ***localstorage*** in JS files.

- Request & Response (./backend-server/server.py): deal with the requests from the client with corresponding endpoints and data bodies.

- Data handling (./backend-server/dpOps.py): generally deal with CRUD operations on the database.

Communication Protocol

- Data Format: Data is typically exchanged in the JSON format.

- Endpoints: Depending on the webpage and functionality, the client sends requests to different server endpoints.

The defined data format are shown below. Note: the "status" and "message" fields are skipped for the Output data in the table.

Endpoint	Method	Input (post-body JSON)	Output	Description
/ or /login	GET		Redirect to the login page ./frontend-web/login.html	
/login	PUT	Username, password	Userid, timeline, personal_info,	

			following_counts, followed_counts	
/register	GET		Redirect to the register page ./frontend-web/register.html	
/register	POST	Username, password, displayname		
/newstory	POST	Userid, title, content	Timeline	
/unfollow	POST	Userid, targetUserID	Following_counts, Followed_counts, Timeline	
/follow	POST	Userid, targetUserID	Following_counts, Followed_counts, Timeline	
/searchUser	POST	Userid, token	**Users**	
/like	POST	Userid, postid	Timeline	
/comment	POST	Userid, postid, content	Timeline	
/userinfo	PUT	Attr (string), value, userid	Personal_info	Update user info
/follower	PUT	Userid	Following	Get list of followings
/following	PUT	Userid	Followers	Get list of followers
/{.js/.css/.jpg/.html}	GET		Redirection to the corresponding local addres	

Note:
Json format for Timeline: {postid, title, content, likes (count), isLiked (bool), **comments**}

Json format for Comments: {userid, content}

Json format for **Users** (in searchUser): {userid, displayname, is_following}

Json format for Personal_Info: {displayname, gender, age, occupation, education, location}

Json format for Following: {userid, displayname}

Json format for Followers: {userid, displayname}