



AggieBook Iteration 2 – Client Server

Team 25

Longxiang Li: 634005819 (individual)

Note: Given that I'm undertaking this project individually and the subject matter is relatively complex, I've decided, with my teacher's approval, to omit the desktop-client component in the third iteration and concentrate solely on developing the backend.

Re: Request for Personal Modification in Final Project Scope

 **Nguyen, Tung** <tung@tamu.edu>
2023/12/4 0:56 

收件人: Longxiang, Li

Yes, because you work on an 1-person team, you can work on half of the workload. So what you propose is acceptable. Please describe this clearly in your documentation.

Tung

From: Longxiang Li (SDS, 118010144) <lazarussybil@tamu.edu>
Sent: Sunday, December 3, 2023 10:39 PM
To: Nguyen, Tung <tung@tamu.edu>
Subject: Request for Personal Modification in Final Project Scope

Dear Professor Tung,

I hope this message finds you well. I'm reaching out to discuss my final project, where I'm currently working solo on a social networking site. Due to the extensive interactive features, the workload is kind of challenging for one person.

As my career interest lies primarily in backend development, I'm wondering if I could focus on the web-based client and backend server, omitting the desktop-based frontend. This would allow me to better align the project with my professional goals and manage the workload more effectively.

Your understanding and guidance on this matter would be greatly appreciated. Thank you for your time.

Best regards,
Longxiang Li

Video Recording

Youtube link: <https://youtu.be/Kmzz0KsY92Q>

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:

The image displays two side-by-side UI forms. The left form is titled 'Login' and contains two input fields: the first has the placeholder 'john.doe@example.com' and the second has a masked password '*****'. Below these is a dark 'Login' button. At the bottom, it says 'Don't have an account? [Register here](#)', with the link highlighted by a red rectangle. The right form is titled 'Register' and contains three input fields: the first has 'john.doe@example.com', the second has a masked password '*****', and the third is labeled 'DisplayName'. Below these is a dark 'Register' button. At the bottom, it says 'Already have an account? [Login here](#)'.

Use Case 2: User Login

A registered user wants to log into the Aggiebook.

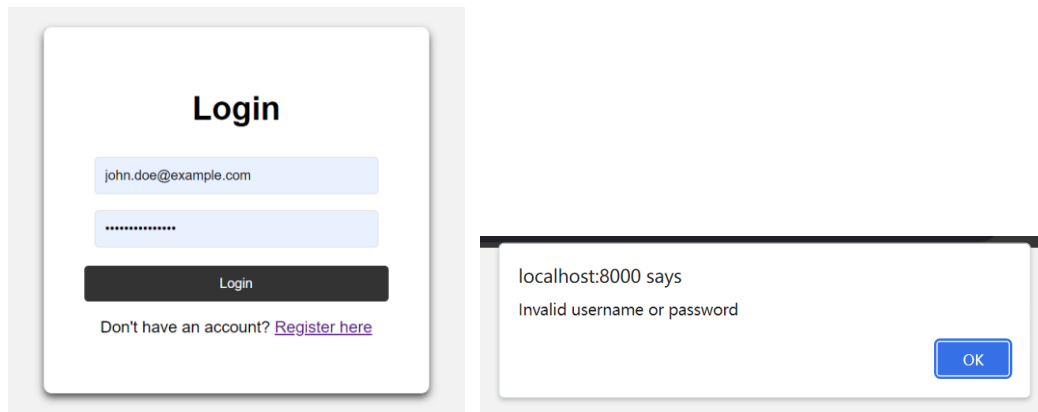
- **User Actions:**

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:

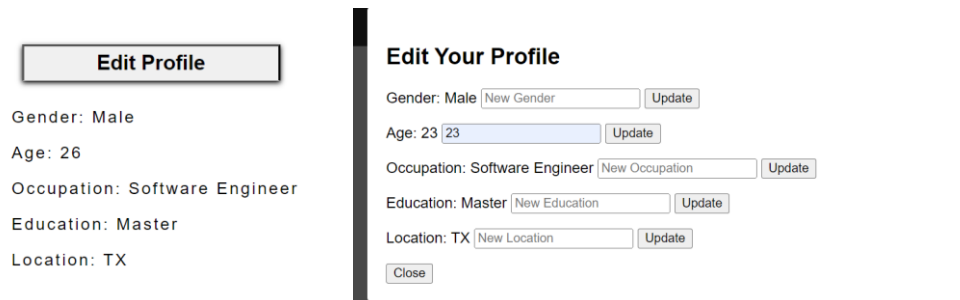
1. User navigates to the "homepage" section.
2. User clicks on "Edit Profile" button.
3. User modifies desired fields such as gender / age / etc.
4. User clicks on "update" button.

- 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:



The UI sketch shows two versions of the 'Edit Profile' form. On the left is a simplified version with a single 'Edit Profile' button and a list of current user details: Gender: Male, Age: 26, Occupation: Software Engineer, Education: Master, and Location: TX. On the right is a more detailed version titled 'Edit Your Profile'. It contains input fields for each attribute, each with a current value and a 'New' field, followed by an 'Update' button. The fields are: Gender (Male, New Gender), Age (23, 23), Occupation (Software Engineer, New Occupation), Education (Master, New Education), and Location (TX, New Location). A 'Close' button is at the bottom left of the detailed form.

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:



The UI sketch shows a single button labeled 'new story'.



The image shows a UI sketch for a 'Write a New Story' form. The form is contained within a white rectangular box with a thin black border. At the top left of the box, the title 'Write a New Story' is written in a bold, black, sans-serif font. Below the title, there is a label 'Title:' followed by a single-line text input field. Underneath the title field is a large, empty rectangular area for the story content. To the left of this area is the label 'Content:'. At the bottom of the form, there are two buttons: 'Complete' and 'Cancel', both with a light gray background and a thin black border. The entire form is set against a dark gray background.

Use Case 5: Search Friends

A user wants to search other users with display names.

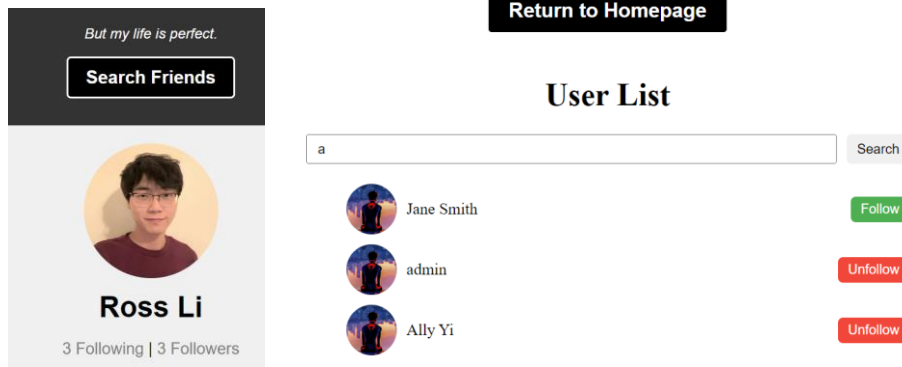
- User Actions:

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.

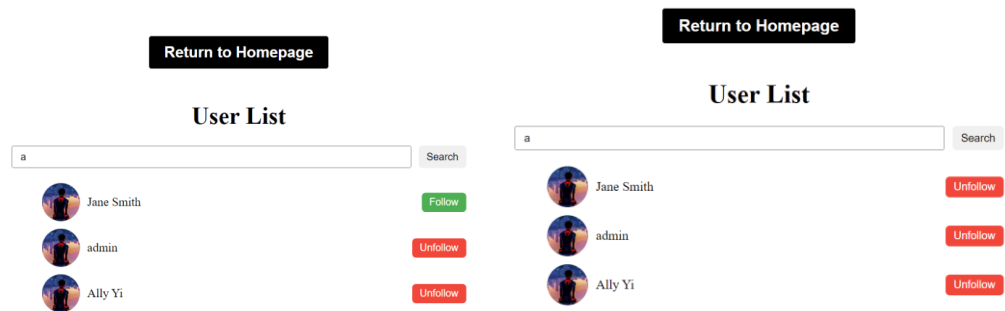
- User Actions:

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:

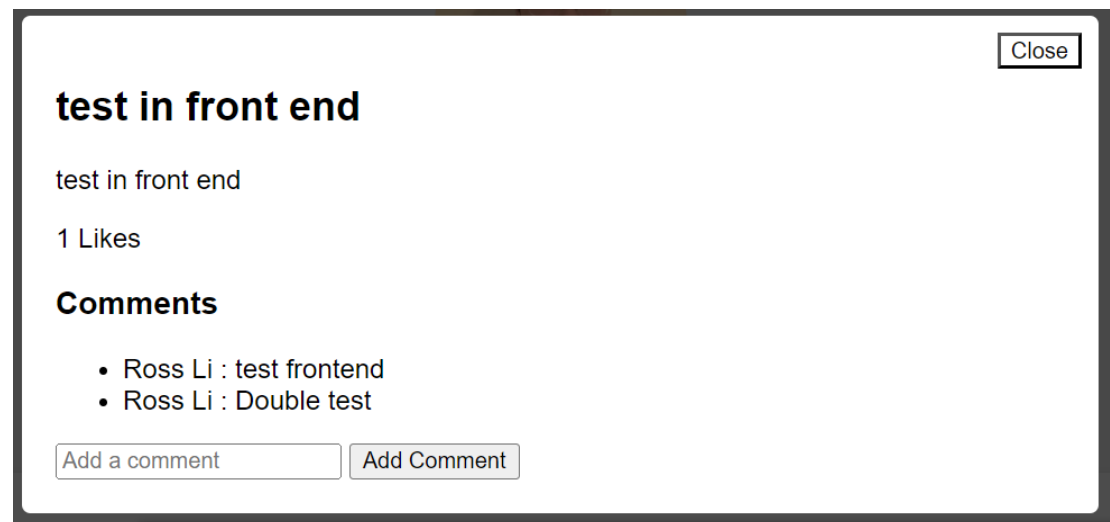
1. User browses on the timeline of its homepage.

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 (SQLite) deprecated

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`

Database design (Redis & MongoDB)

The database design utilizes two distinct types of NoSQL databases to store and manage data for the social media application: Redis, a key-value store, and

MongoDB, a document-oriented database.

Redis

User Credentials:

- **Key:** `user:{username}`
- **Value:** `{hashed password}`
- **Example:** user: john.doe@example.com -> `hashed("passwordForJohn")`

User Relationships:

- **Key:** `following:{followerID}:{followingID}`
- **Value:** temporarily null. In future it can be any JSON string describing the relationship, such as closeness rate.
- **Example:** following: 657115646cf49170e02325da:
6571152e6cf49170e02325d1

MongoDB

MongoDB acts as our primary data store for complex, schema-less document storage.

Note: Due to account restrictions, users of the free version can only create one database. Consequently, I've configured the **user_server** and **post_server**, two separate microservices, to access the *userinfo* collection and *posts* collection, respectively. This approach aligns with the assignment requirement of "Each microservice should use an independent database." In other words, for MongoDB, I used independent collections as a substitute for independent databases.

Userinfo Collection:

- Stores comprehensive user profiles with personal information attributes. A timeline array is also stored here for homepage posts presentation generation.
- Sample data:

```

{
  "userID": "6571152e6cf49170e02325d1",
  "display_name": "lazarussybil",
  "gender": "M",
  "age": "24",
  "occupation": "Developer",
  "education": "B.Sc. Computer Science",
  "location": "New York"
  "timeline": [
    "657115446cf49170e02325d5",
    "6571334af664b777c7202bd7",
    "6571336af664b777c7202bda"
  ]
}

```

Posts Collection:

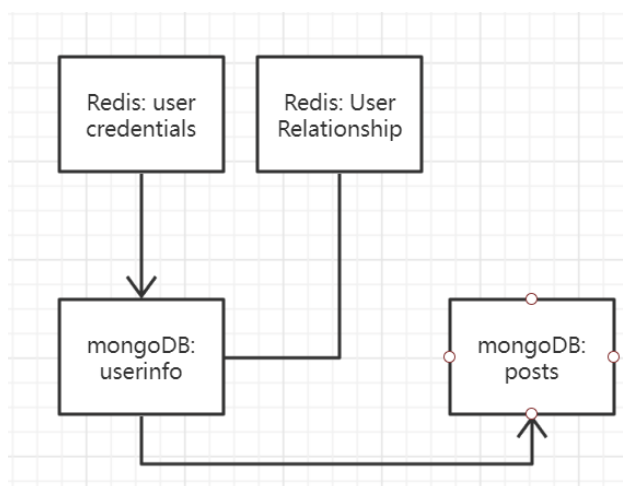
- Stores user-generated content, including posts, comments, and likes.
- Sample data:

```

// posts
{
  "postID": "657115446cf49170e02325d5",
  "userID": "6571152e6cf49170e02325d1",
  "title": "My First Post",
  "content": "This is the content of my first post.",
  "post_time": "2023-12-06T12:00:00Z",
  "comments": [
    {
      "userID": "6571152e6cf49170e02325d1",
      "content": "Nice post!",
      "comment_time": "2023-12-06T12:05:00Z"
    }
  ],
  "likes": [
    { "userID": "6571152e6cf49170e02325d1" }
  ]
}

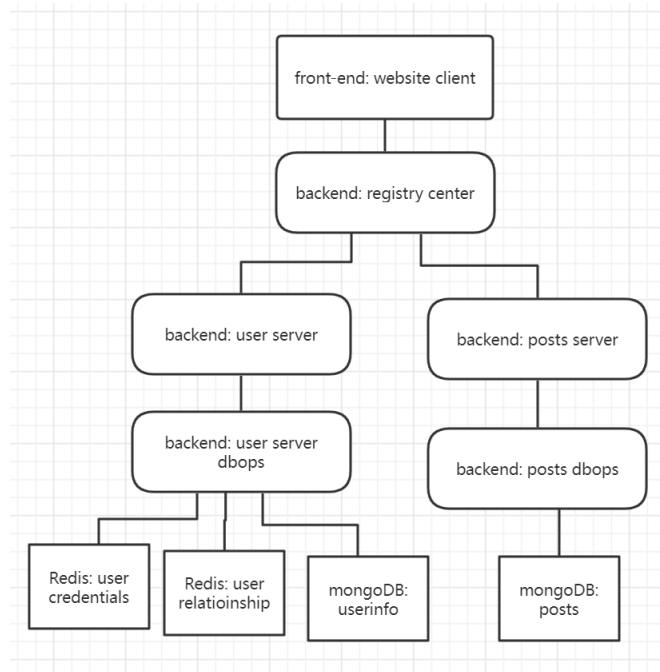
```

ER-diagram



Architecture design

Overview diagram



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, read 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 (Microservice Breakdown)

1. Registry Center (port 8000):

- Responsible for service discovery and management.
- Tracks the availability of the User Server and Post Server.
- Provides information about these services to other parts of the system.
- **Forward request: when receiving a request from a client, based on the path implication forwarding the original request to corresponding microservice server.**

2. User Server (port 8001):

- After setup, **register itself** in the registry center.
- Manages user-related operations (registration, login, follow/unfollow, user information updates).
- Interacts with both **Redis** (for simple data like user credentials and follow relationships) and **MongoDB *userinfo*** collection (for detailed user information).

3. Post Server (port 8002):

- After setup, **register itself** in the registry center.
- Handles operations related to posts (creating posts, commenting, liking, fetching timeline).
- Primarily interacts with **MongoDB *posts*** collection to manage posts and related data like comments and likes.

4. Communication

- Each microservice exposes RESTful APIs for communication.
- The Registry Center uses RESTful APIs to provide information about available services.

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.

1. Registry Center related endpoints protocol

Page navigation actions

Endpoint	Method	Input (post-body JSON)	Output	Description
/ or /login	GET		Redirect to the login page ./frontend-web/login.html	
/register	GET		Redirect to the register page ./frontend-web/register.html	
/ { .js/.css/.jpg/.html }	GET		Redirection to the corresponding local address	
/inner-service	POST	action (string:[“register”/”unregister”]), service_name (string), service_address (string)		For microservice servers to register or unregister

2. User server related endpoints protocol

/users/register	POST	Username, password,		
-----------------	------	------------------------	--	--

		displayname		
/users/unfollow	POST	Userid (string), targetUserID (string)	Following_counts (int), Followed_counts, Timeline	
/users/follow	POST	Userid (string), targetUserID (string)	Following_counts, Followed_counts, Timeline	
/users/searchUser	POST	Userid, token	Users	
/users/userinfo	PUT	Attr (string), value, userid	Personal_info	Update user info
/users/follower	PUT	Userid (string)	Following	Get list of followings
/users/following	PUT	Userid	Followers	Get list of followers
/users/login	PUT	Username, password	Userid, timeline, personal_info, following_counts, followed_counts	

3. Post server related endpoints protocol

/posts/newstory	POST	Userid(string), title(string), content(string)	Timeline	
/posts/like	POST	Userid, postid	Timeline	
/posts/comment	POST	Userid, postid, content	Timeline	

Explanation for complete data structure in Protocol

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

Json format for **Comments**: {userid, content}

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

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

Json format for **Following**: {userid, displayname}

Json format for **Followers**: {userid, displayname}

Operation Manual

Sees in README.