Home Page

The User component displays the user's information and posts. The API requests are made using the useEffect hook. There are two useEffect hooks: one to fetch the user's data and the other to fetch their posts based on the user's \_id. The component uses the fetched data to display the user's information and posts.

Questions:

1. What, if any DATA is required from the backend to render the wireframe? Generate documentation for any required API. If no dynamic data is required, indicate that.

The User component requires dynamic data from the backend to render the wireframe. It requires data for the following:

* User's information: first\_name, last\_name, birthday, location, occupation, and status
* User's posts: content, comments, and likes

API Documentation:

To fetch the user's information:

* Method: GET
* URL: http://localhost:5000/session/${session\_id}
* Request Header: Authorization: Bearer ${session\_id}
* Response Body:
* json
* Copy code
* { "data": { "user": { "\_id": "user id", "first\_name": "first name", "last\_name": "last name", "birthday": "birthday", "location": "location", "occupation": "occupation", "status": "status" } } }

To fetch the user's posts:

* Method: GET
* URL: http://localhost:5000/post/${user.\_id}
* Response Body:
* json
* Copy code
* { "data": [ { "\_id": "post id", "content": "post content", "comments": [ { "\_id": "comment id", "content": "comment content" } ], "likes": 5 } ] }

1. What, if any, ACTIONS is this wireframe responsible for? Button clicks and form submissions often trigger logic that leverages the API layer. Generate documentation for any required API. If no actions are required, indicate that.

The wireframe does not have any actions that trigger an API call.

Registration Page

The wireframe appears to be responsible for submitting user registration data to the backend. The API endpoint for this action is expected to be POST http://localhost:5000/user/register, and the wireframe sends a JSON object in the body of the request containing the user's first name, last name, email, birthday, password, occupation, and location.

Therefore, the backend should implement a route handler to accept and process the POST request to /user/register. The expected request body should be a JSON object containing the following fields:

* first\_name (string): the user's first name
* last\_name (string): the user's last name
* email (string): the user's email address
* birthday (string): the user's birthday in the format YYYY-MM-DD
* password (string): the user's password
* occupation (string): the user's occupation
* location (string): the user's location

The backend should respond to the request with a status code indicating whether the user registration was successful or not. If the registration was successful, the response should contain a message indicating that the user was registered successfully. If the registration was not successful, the response should contain an error message.

No dynamic data is required from the backend to render the wireframe.

Login Page

This is a React component for a Login page that communicates with a backend API.

Data required from the backend:

* None, the form only collects data from the user and sends it to the backend for authentication.

Actions triggered by the wireframe:

* When the user clicks the "Login" button, the handleSubmit function is called to submit the form data to the backend API for authentication.
* If the authentication is successful, the user is redirected to the home page. If not, the user remains on the login page.
* There is also a link to the registration page (/registration) for users who do not yet have an account.

API documentation:

* POST /user/login - This API endpoint expects a JSON body with email and password properties. It returns a JSON response with a type property indicating if the login was successful or not, and a data property containing the user's information if the login was successful.
* POST /session - This API endpoint expects a JSON body with a user property containing the user's information. It returns a JSON response with a session\_id property containing the user's session ID.

Blog Post

This is a React component that displays a list of posts on a network view page. It relies on two useEffect hooks to retrieve data from the backend API:

1. The first useEffect hook retrieves the user ID from the server by making a GET request to /session/${token} endpoint with the user's token, which is retrieved from a cookie using the getCookie function from the react-use-cookie library. The user ID is then stored in the component's state using the setUserId function. If there is an error or the response type is "error", the user is redirected to the login page using the navigate function from the react-router-dom library.
2. The second useEffect hook retrieves a list of posts from the server by making a GET request to the /posts endpoint. The response data is stored in the component's state using the setPosts function. If there is an error or the response type is "error", the error is logged to the console.

The component uses the PostList function to render a list of PostCard components for each post in the posts state array. The PostCard component displays the post content and metadata, such as the author's name, profile picture, and timestamp.

This component does not have any actions that require communication with the backend API beyond the initial data retrieval, so no additional API documentation is required beyond the endpoints used in the useEffect hooks.

React component named PostCard displays a post, and its comments, and allows the user to like, unlike, and add comments to the post. The component has several state variables, including post and user, both initialized to empty objects using useState. The post state variable is updated using the setupPost function, which modifies the updatedAt property of the post and then sets the state of the post to the updated post object. The useEffect hook is used to fetch the user data associated with the post from the server and update the user state variable.

The comment list function returns a list of comment items to be displayed on the postcard. The handleLike and handleUnlike functions handle the user's liking and unliking of the post, respectively, by sending a PUT request to the server to update the post's like count. The handlePostComment function handles the user's posting of a comment to the post by sending a POST request to the server with the comment content and the user and post IDs.

Blog Admin

This is a React component that manages a list of posts and displays them in a table. It uses Bootstrap components for styling and pagination. The component fetches data from a backend API and displays it using the Post component. The user can delete a post by clicking the "Delete" button next to each post.

The Post component is a modal that appears when the user clicks the "Delete" button. It asks the user to confirm the deletion of the post and then calls the deletePost function passed as a prop to the component. The deletePost function sends a DELETE request to the backend API to delete the post and updates the state of the posts variable by removing the deleted post.

The PostManagerView component fetches the list of posts from the backend API using the getPostsPaginated function. It also fetches the total number of posts using the getPostCount function to calculate the number of pages needed for pagination. The handlePaginationClick function updates the current page of the pagination when the user clicks on a page number.

The postList function maps the posts array to the Post component and returns it. It also filters the posts array based on the current page and the limit variable.

Network View

**Since this is part of the logged in views we need to fetch the session or return the user to the log in screen**

Api:

*get /session/{session\_id}*

-Validates the session the session and returns a response with the user’s information.

**We also need all users information besides the user that is logged in.**

Api:

*get /users/{user\_id}*

-Returns a response containting the information of all users except the one that is logged in.

**Witihin the cards, we need the latest post from that user.**

Api:

*get /post/latest/{user\_id}*

-Returns a response containting the latest post of the given user id.

Admin View

No dynamic data is needed for this view because all we need is to display cards that will navigagte to different views of the project in the future.