JUKE User Registration Flow – Detailed Explanation

This document explains the JavaScript implementation for creating a new user in JUKE, including the core registration function, form handling, key components, and the overall flow.

---

1. Core Registration Function

File: `auth.js`
Lines: 468–502

```
async function register(username, email, password, firstName, lastName) {
    try {
        const result = await postAuthJson(
            '/auth/register',
            { username, email, password, firstName, lastName },
            function (data) { return !!(data && data.token && data.user); }
        );
        const data = result.data;
        if (result.ok) {
            // Store token and user data
            localStorage.setItem('juke_token', data.token);
            localStorage.setItem('juke_user', JSON.stringify(data.user));
            currentUser = data.user;
            // Redirect to feed
            window.location.hash = '#/feed';
            return { success: true };
        } else {
            return { success: false, error: data.error };
        }
    } catch (error) {
        return { success: false, error: 'Network error. Please try again.' };
    }
}
```

Explanation:

- Sends a POST request to `/auth/register` using `postAuthJson`.
- Includes user data: `username`, `email`, `password`, `firstName`, `lastName`.
- Validates that the response contains both `token` and `user` information.
- On success:
- Stores authentication token in `localStorage`.
- Stores user information as a JSON string in `localStorage`.
- Updates the `currentUser` variable.
- Redirects the user to the feed page (`#/feed`).

- Handles errors by returning an appropriate message for network or server issues.

---

2. Registration Form Handler

File: `auth.js`
Lines: 625–678

```
function setupRegisterForm() {
    const form = document.getElementById('registerForm');
    if (!form) return;
    form.addEventListener('submit', async (e) => {
        e.preventDefault();

        // Get form values
        const username = document.getElementById('username').value.trim();
        const email = document.getElementById('email').value.trim();
        const password = document.getElementById('password').value;
        const confirmPassword = document.getElementById('confirm-
password').value;
        const firstName = document.getElementById('firstName').value.trim();
        const lastName = document.getElementById('lastName').value.trim();

        // Validate passwords match
        if (password !== confirmPassword) {
            // Show error message
            return;
        }

        // Attempt registration
        const result = await register(username, email, password, firstName,
lastName);

        if (!result.success) {
            // Show error message
        }
    });
}
```

Explanation:

- Listens for the `submit` event on the registration form.
- Prevents default form submission to handle it via JavaScript.
- Retrieves user input values from form fields.
- Checks that the password and confirm password fields match before sending data.
- Calls the `register` function with the collected data.
- Handles errors returned by the registration function (e.g., invalid input, network issues).

---

3. Key Components

1. **API Call**

```
const result = await postAuthJson('/auth/register', userData,
validator);
```

2. Sends user data to the server endpoint `/auth/register`.
3. Waits for the server response.
4. Validates that the response includes a token and user data.
5. **Data Storage**

```
localStorage.setItem('juke_token', data.token);
localStorage.setItem('juke_user', JSON.stringify(data.user));
```

6. Stores the authentication token for session persistence.
7. Saves user information to maintain logged-in state.
8. **Form Validation**

```
if (password !== confirmPassword) { return; }
```

9. Ensures the user enters matching passwords.
10. Prevents invalid submissions.
11. Allows showing user-friendly error messages.
12. **Error Handling**

```
try { ... } catch (error) { return { success: false, error: 'Network
error...' }; }
```

13. Catches network or server errors.
14. Returns user-friendly messages instead of breaking the application.

---

4. Flow Summary

1. User fills out the registration form.
2. JavaScript captures the input values.
3. Client-side validation ensures passwords match.
4. Data is sent to the server via `postAuthJson`.

5. Server returns a token and user data.
6. Data is stored in `localStorage` for authentication persistence.
7. User is redirected to the feed page.

This sequence ensures a smooth registration experience with proper validation, error handling, and session management.