

Week 2 Assignment: Social Media Login Service

Code & Implementation Guide

Repo on Git: <https://github.com/psu-edu/social-media-login.git>

Implementation guide included on README.md on Git

The screenshot displays the GitHub repository page for 'social-media-login' by 'psu-edu'. The repository is private and has 0 stars, 0 forks, and 0 watchers. The main content area shows a list of files and their commit history:

File	Commit Message	Time Ago
models	added user and auth js files	6 hours ago
public	adjusted directory structure to move static html and css into ...	35 minutes ago
routes	added user and auth js files	6 hours ago
README.md	updated readme	1 minute ago
client_secret_805207106585-fsn2g6fop53lhb9...	updated following instructions again	2 hours ago
environment.env	updated index and environment	yesterday
node.js	updated index and added node.js	8 hours ago
package.json	added environment variables and installed dependencies	2 days ago
server.js	adjusted directory structure to move static html and css into ...	35 minutes ago

The sidebar on the right contains the following sections:

- About:** Social Media Login Service Assignment. Includes links to Readme, Activity, Custom properties, stars (0), watching (0), and forks (0).
- Releases:** No releases published. Link to [Create a new release](#).
- Packages:** No packages published. Link to [Publish your first package](#).
- Deployments:** 24 deployments. The latest deployment is 'github-pages' 9 minutes ago. Link to [+ 23 deployments](#).
- Languages:** A bar chart showing the language distribution: HTML (57.5%), JavaScript (37.0%), and CSS (5.5%).

Login Service Implementation

URL for login service implementation: <https://vigilant-barnacle-nr32lo5.pages.github.io/>

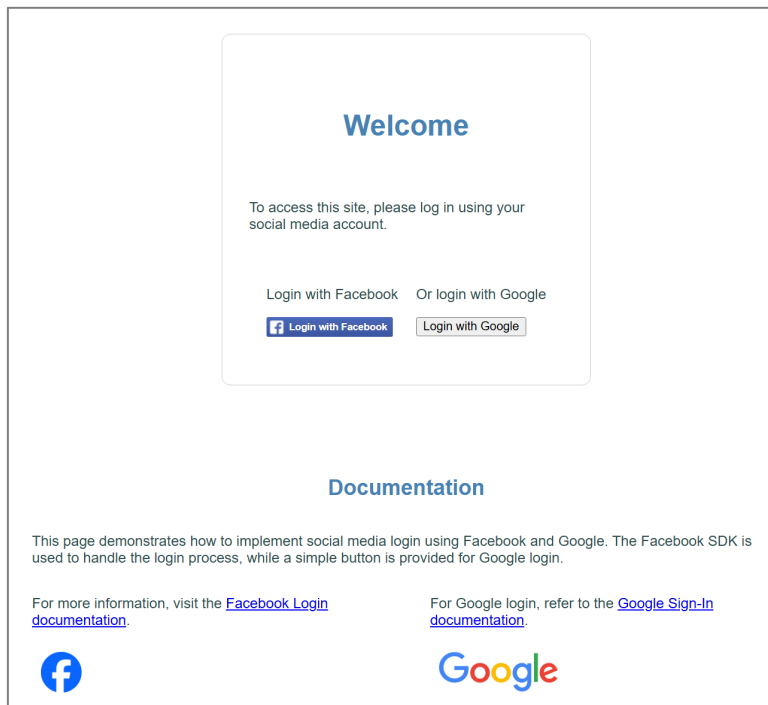
Detailed Report

This report covers details for implementing this assignment with the overall goal to design and implement a login service using social media authentication.

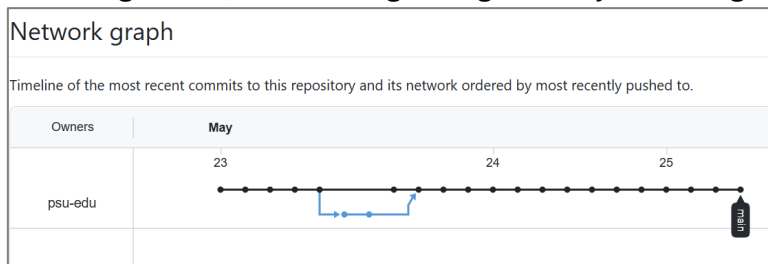
Frontend: HTML, CSS and JavaScript

My frontend implementation includes my index.html file with internal CSS and JavaScript. I started without an external stylesheet.css file since I was anticipating only to use a single page

for this project. I later found this not to be true once I added a callback.html page, so I pulled the CSS out into a stylesheet.css. This is client-side/user facing and displays login buttons and handles redirects. Much of the JS code was provided by Google and Facebook's SDK as both had detailed instructions to initiate the OAuth login.



Lastly, I was also able to get more familiar with interactions between VS Code and GitHub during this assignment, maintaining change history and merging to remote main throughout my work.



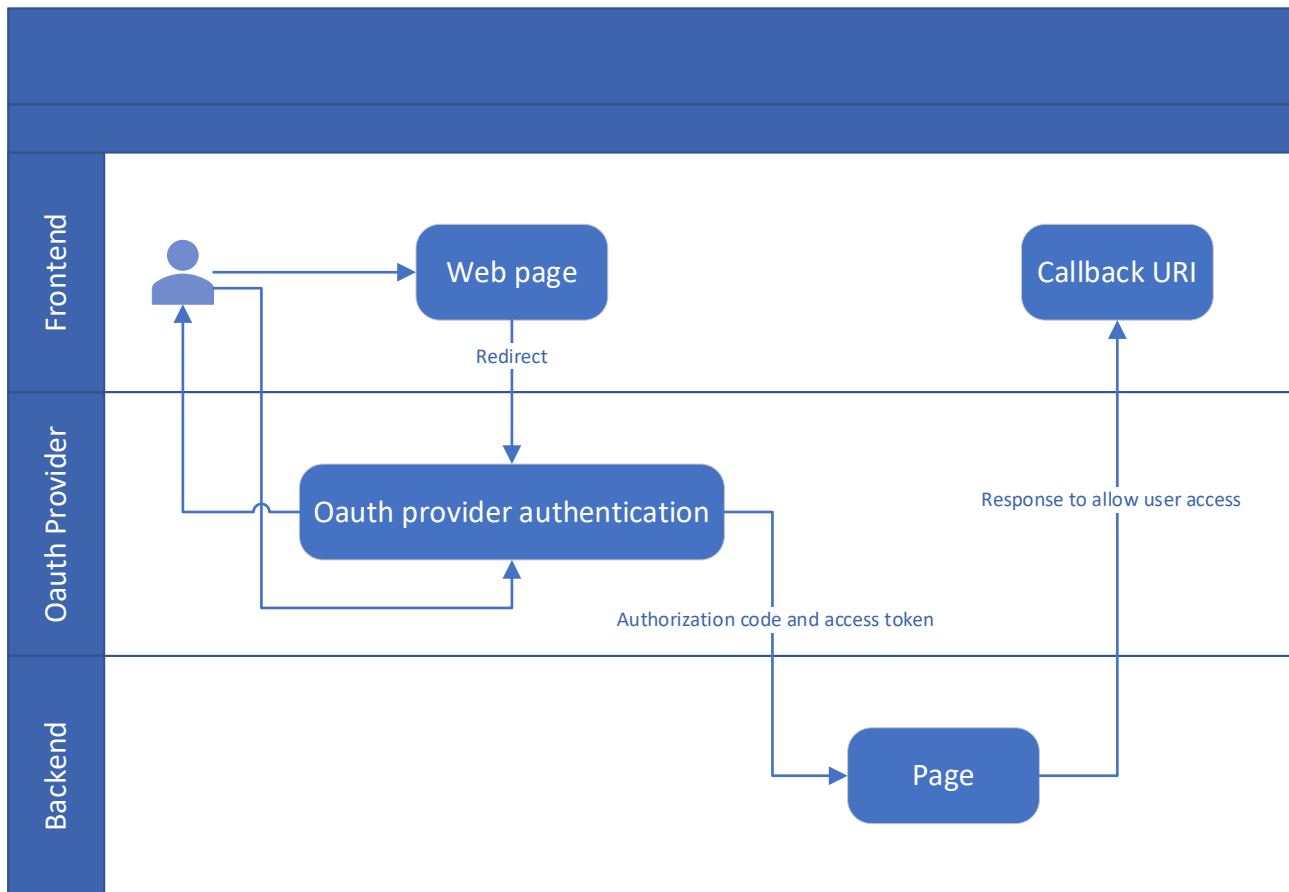
Backend: Server (Node.js)

My main backend server file is set up to handle Google and Facebook tokens securely and handle user data. I also set up an environment.env file to store details like my client secret and redirect URI for each service. This way, I am keeping these values from being exposed to users.

Deployment

Deployed by setting my GitHub repo to public and then deploying using render. During deployment, I was able to add my environment.env file and client_secret.json file which contained both google and facebook ID and secret ID.

Social Media Authentication Flow



Summary of steps

1. Frontend (client-side) web page displays buttons for user to login on website using their Google or Facebook account
2. Web browser redirects to OAuth provider (either Facebook or Google)
3. User login credentials are handled by provider
4. Provider redirects user to callback.html
5. Backend (server-side) handles code exchange and token validation and stores user info

Security considerations and measures taken

- **Verification of token** used to confirm access was provided by Facebook or Google as expected and not tampered with by checking token's signature as recommended in the SDK.
- **Rotating secrets:** Available in the SDK for google and allows the developer to use client secret rotation where you can add a new secret to your OAuth client configuration, migrate to the new secret while the old secret is still usable, and disable the old secret afterwards. This also ensures good security practices (*Google*).
- **Leveraged HTTPS** protocol for greater security by encrypting communications to safeguard the privacy of information exchanged.

- **CSRF protection** was added with code examples provided by Google's developer documentation (*Google*).

```

1 public > < index.html > < html > < body
2   <html xmlns="http://www.w3.org/1999/xhtml" >
3   <body>
4     <script async defer crossorigin="anonymous">
5       <script>
6         function testAPI() {
7           FB.api('/me', function(response) {
8             // If you are logged in, automatically get your name and email address, your public profile information
9             FB.login(function(response) {
10               if (response.authResponse) {
11                 console.log('Welcome! Fetching your information.... ');
12                 FB.api('/me', {fields: 'name, email'}, function(response) {
13                   document.getElementById('profile').innerHTML = "Good to see you, " + response.name + ". I see you";
14                 });
15               } else {
16                 // If you are not logged in, the login dialog will open for you to login asking for permission to get
17                 console.log('User cancelled login or did not fully authorize. ');
18               }
19             });
20           }
21         }
22       </script>
23     </script>
24   </body>
25 </html>

```

References

Setting up OAuth 2.0 - API Console Help. *Google*.

<https://support.google.com/googleapi/answer/6158849?hl=en> Accessed May 2025.

Facebook Login for the Web with the JavaScript SDK. *Meta*.

<https://developers.facebook.com/docs/facebook-login/web> Accessed May 2025.

<https://developers.facebook.com/docs/development/create-an-app/facebook-login-use-case>

Geeks for Geeks. (2024). Implementing an OAuth Server With NodeJS and Express.

<https://www.geeksforgeeks.org/implementing-an-oauth-server-with-nodejs-and-express/#>

Sandoval, F. (2023). Google OAuth2 for Dummies. *DEV Community*.

<https://dev.to/fredysandoval/google-oauth2-for-dummies-4bl1>

AI Tools Leveraged

Microsoft Copilot (web based) was used early on as a way to ideate on the right design approach for this project.

Prompt: I'm designing a website with a feature that allows users to login using Facebook or Google OAuth. I have already set up the required developer accounts and have a html file to display the buttons based on instructions from the developer documentation. i'm looking for a recommended design approach that includes secure handling of user data during user authentication that leverages html and javascript.

Output

To ensure secure handling of user data during authentication using HTML and JavaScript, here's a recommended design approach that balances usability, security, and maintainability:

1. Use OAuth Flow Securely: Both Google and Facebook recommend using the OAuth 2.0 Authorization Code Flow with PKCE for client-side apps. This ensures tokens are not exposed to the browser directly.
2. Frontend Architecture (HTML + JS): Your HTML should only handle the UI and redirect to the OAuth provider. After authentication, the provider redirects back to your site with an authorization code.
3. Backend Involvement (Highly Recommended)
Even if you're using HTML/JS, you should involve a backend server to:
 - a. Exchange the authorization code for access and ID tokens.
 - b. Validate the ID token (signature, expiration, issuer).
 - c. Store user session securely (e.g., using HTTP-only cookies).
 - d. Avoid exposing tokens to the frontend.
4. Token Handling Best Practices
 - a. Never store tokens in localStorage or sessionStorage (vulnerable to XSS).
 - b. Use HTTP-only, Secure cookies for storing session identifiers.
 - c. Validate tokens on the server using the provider's public keys (JWT validation).
5. Libraries and Tools
Use trusted libraries to simplify and secure the process:
 - a. Frontend: oidc-client-ts, @auth0/auth0-spa-js
 - b. Backend: passport.js, express-session, jsonwebtoken
6. Additional Security Measures
 - a. Use HTTPS always.
 - b. Implement CSRF protection for any state-changing requests.
 - c. Use Content Security Policy (CSP) headers to mitigate XSS.
 - d. Rate-limit login attempts to prevent abuse.