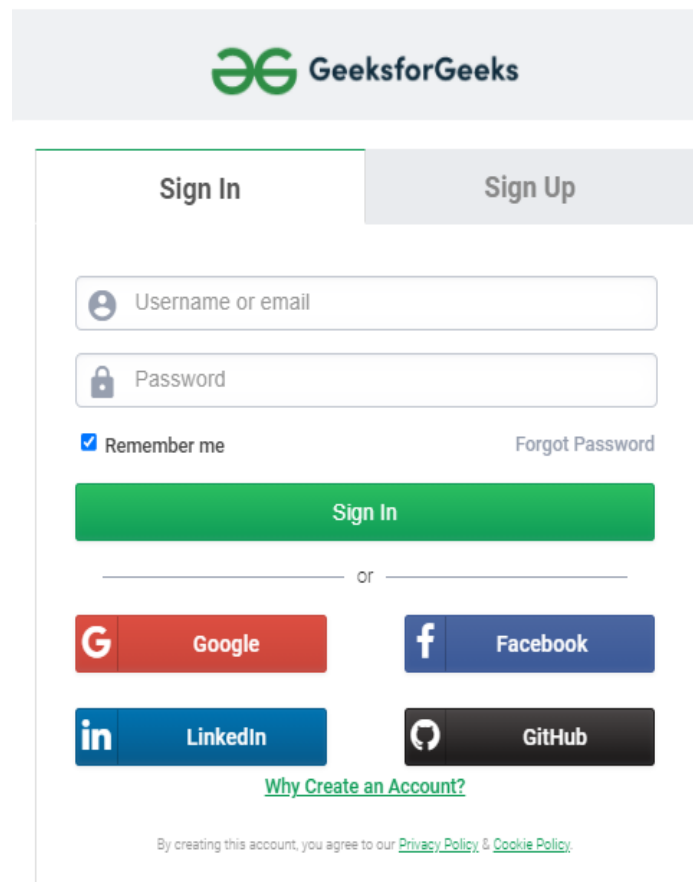


Contents

- Introduction to OAuth2
- OAuth 2.0 concepts
- Workflow of OAuth 2.0
- Scope and Consent:
- Github OAuth2 with Spring Boot

Introduction to OAuth2 :

OAuth2.0 is an Open industry-standard authorization protocol that allows a third party to gain limited access to another HTTP service, such as Google, Facebook, and GitHub, on behalf of a user, once the user grants permission to access their credentials.



The image shows a web form for GeeksforGeeks. At the top is the GeeksforGeeks logo. Below it are two tabs: 'Sign In' (active) and 'Sign Up'. The 'Sign In' tab contains a form with a 'Username or email' field, a 'Password' field, a 'Remember me' checkbox, and a 'Forgot Password' link. A green 'Sign In' button is below these fields. Below the button is a horizontal line with 'or' in the center. Underneath are four social media login buttons: Google (red), Facebook (blue), LinkedIn (blue), and GitHub (black). Below these is a link 'Why Create an Account?'. At the bottom, a small text line says 'By creating this account, you agree to our [Privacy Policy](#) & [Cookie Policy](#)'.

Example of OAuth2 :

Let us now discuss OAuth. OAuth is an open-standard authorization framework that enables third-party applications to gain limited access to user's data. Essentially, OAuth is about delegated access.

Delegation is a process in which an owner authorizes a service provider to perform certain tasks on the owner's behalf. Here the task is to provide limited access to another party.

OAuth 2.0 concepts

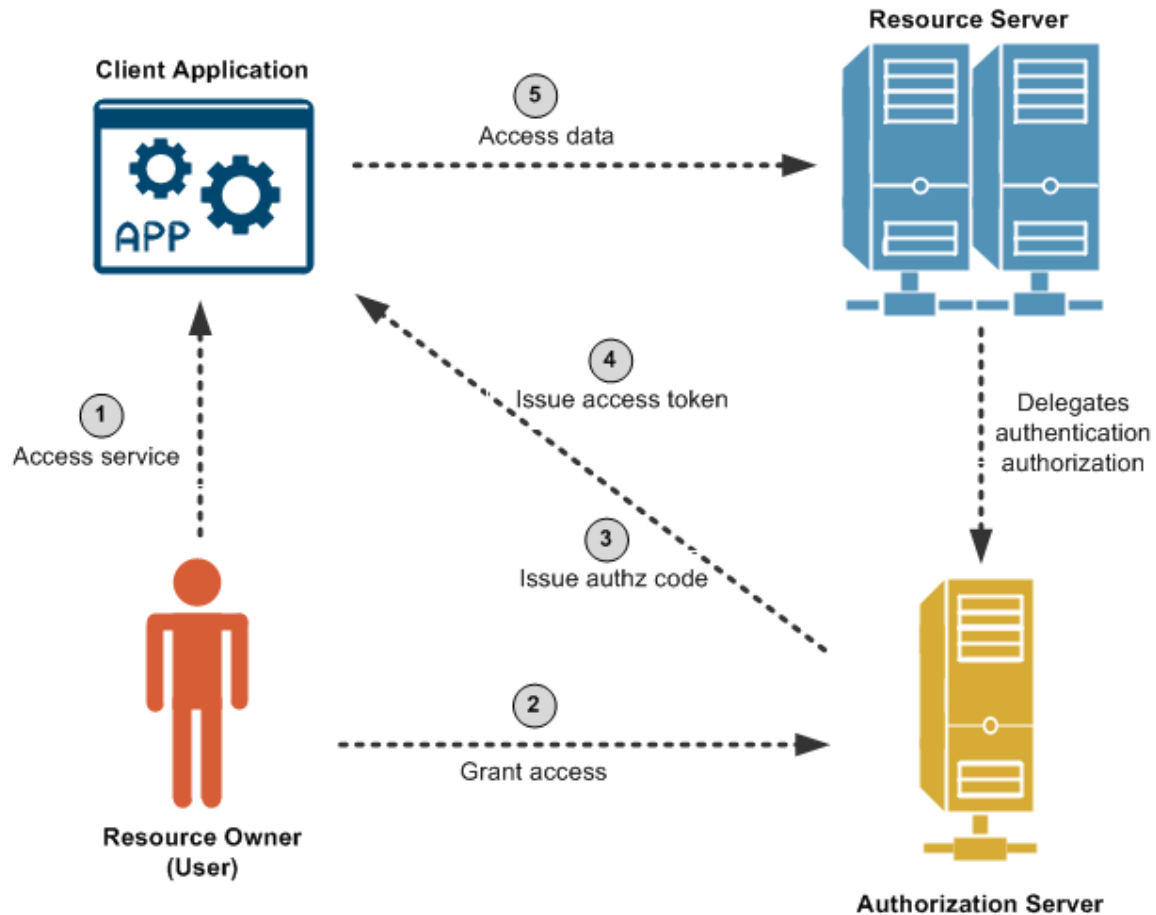
Resource Owner (Github User): An entity capable of granting access to a protected resource. When the resource owner is a person, it is referred to as an end user.

Resource Server (Spring Boot Application/Database Server): The server hosting the protected resources, and which is capable of accepting and responding to protected resource requests using access tokens. In this case, the API Gateway acts as a gateway implementing the Resource Server that sits in front of the protected resources.

Client Application (Spring Boot Application): A client application making protected requests on behalf of the resource owner and with its authorization.

Authorization Server (Github): The server issuing access tokens to the client application after successfully authenticating the Resource Owner and obtaining authorization. In this case, the API Gateway acts both as the Authorization Server and as the Resource Server.

Scope: Used to control access to the Resource Owner's data when requested by a client application. You can validate the OAuth scopes in the incoming message against the scopes registered in the API Gateway. An example scope is userinfo/readonly.



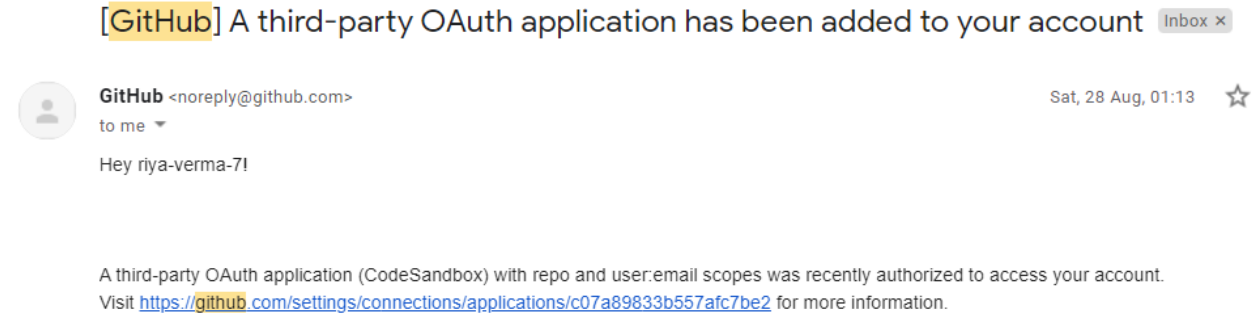
Scope and Consent:

The scopes define the specific actions that apps can perform on behalf of the user. They are the bundles of permissions asked for by the client when requesting a token.

For example, we can share our LinkedIn posts on Twitter via LinkedIn itself. Given that it has write-only access, it cannot access other pieces of information, such as our conversations.

On the Consent screen, a user learns who is attempting to access their data and what kind of data they want to access, and the user must express their consent to allow third-party access to the requested data. You grant access to your IDE, such as CodingSandbox, when you link your GitHub account to it or import an existing

repository. The Github account you are using will send you an email confirming this.



What is a token?

A token is a piece of data containing just enough information to be able to verify a user's identity or authorize them to perform a certain action.

We can comprehend access tokens and refresh tokens by using the analogy of movie theaters. Suppose you (resource owner) wanted to watch the latest Marvel movie (Shang Chi and the Legends of the Ten Rings), you'd go to the ticket vendor (auth server), choose the movie, and buy the ticket(token) for that movie (scope). Ticket validity now pertains only to a certain time frame and to a specific show. After the security guy checks your ticket, he lets you into the theatre (resource server) and directs you to your assigned seat.

If you give your ticket to a friend, they can use it to watch the movie. An OAuth access token works the same way. Anyone who has the access token can use it to make API requests. Therefore, they're called "Bearer Tokens". You will not find your personal information on the ticket. Similarly, OAuth access tokens can be created without actually including information about the user to whom they were issued. Like a movie ticket, an OAuth access token is valid for a certain period and then expires. Security personnel usually ask for ID proof to verify your age, especially for A-rated movies. Bookings made online will be authenticated by the app before tickets are provided to you.

So, **Access tokens are credentials used to access protected resources.** Each token represents the scope and duration of access granted by the resource owner and enforced by the authorization server. The format, structure, and method of utilizing access tokens can be different depending on the resource server's security needs.

Session 16



A decoded access token, that follows a JWT format.

```
{ "iss": "https://YOUR_DOMAIN/",  
  "sub": "auth0|123456",  
  "aud": [ "my-api-identifier", "https://YOUR_DOMAIN/userinfo" ],  
  "azp": "YOUR_CLIENT_ID", "exp": 1474178924, "iat": 1474173924,  
  "scope": "openid profile email address phone read:meetings" }
```

Now that your showtime has expired and you want to watch another movie, you need to buy a new ticket. Upon your last purchase, you received a Gift card that is valid for three months. You can use this card to purchase a new ticket. In this scenario, the gift card is analogous to Refresh Tokens. **A Refresh token is a string issued to the client by the authorization server and is used to obtain a new access token when the current access token becomes invalid.**

They do not refresh an existing access token, they simply request a new one. The expiration time for refresh tokens tends to be much longer than for access tokens. In our case, the gift card is valid for three months, while the ticket is valid for two hours. Unlike the original access token, it contains less information.

Workflow of OAuth 2.0



<https://www.geeksforgeeks.org/workflow-of-oauth-2-0/>

Github OAuth2 with Spring Boot

We have seen InMemory authentication and UserDetailsService db authentication .Now we will see OAuth2 where we don't need to take care of Authentication .We just have to take care of Authorization as OAuth2 means we are logging in through a third party organization.

Challenge 1: What is OAuth2 Server(provider) and OAuth2 Client ?

HINT : Github is an OAuth2 provider and geeksforgeeks is an OAuth2 client.

Sign in to GitHub
to continue to GeeksforGeeks

Username or email address

Password [Forgot password?](#)

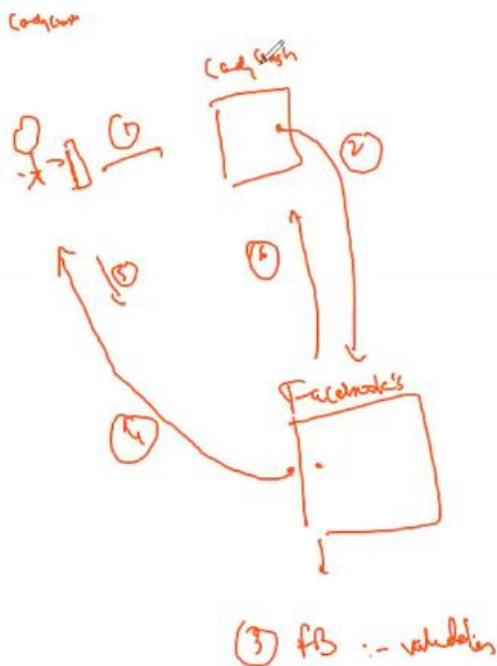
Sign in

New to GitHub? [Create an account.](#)

Challenge 2: Does OAuth2 with Google means Google will share your password with the application you are trying to sign on with Google ?

Challenge 3: Explain the concept of OAuth2 with an example ?

HINT : Signup on CandyCrush using Facebook



Session 16



TASK : Lets login to geeksforgeeks using Github when you are already logged in to github.

TASK : Lets login to geeksforgeeks using Github when you are not logged in to github.

Note : Maven dependency spring-boot-starter-oauth2-client internally depends on spring security framework .

Please open the Spring Boot Github OAuth2 documentation :
<https://spring.io/guides/tutorials/spring-boot-oauth2/>

Also please clone the official repository from Spring Boot to demonstrate github OAuth2
<https://github.com/spring-guides/tut-spring-boot-oauth2>

Let's have a go at a few challenges before we register the project . Start the server and hit localhost:8080/

Challenge 4 : What is the use of `.anyRequest().authenticated()` ?

HINT : If API request is not authenticated , then it cause an exception which is caught by

```
.exceptionHandling(e -> e.authenticationEntryPoint(new  
HttpStatusEntryPoint(HttpStatus.UNAUTHORIZED))  
)
```

This is present in our configure function and shows 401 error . / api leads to calling /user but as /user is not authenticated , it never calls the function mapped to the request .Thus , we get 401 and the callback function never executes and we stay in the logout state .

Challenge 5 : If we remove `.anyRequest().authenticated()` and hit / , what will happen ?

HINT : Which exception will we get ?

Challenge 6: What will happen if we keep everything else but remove the below :

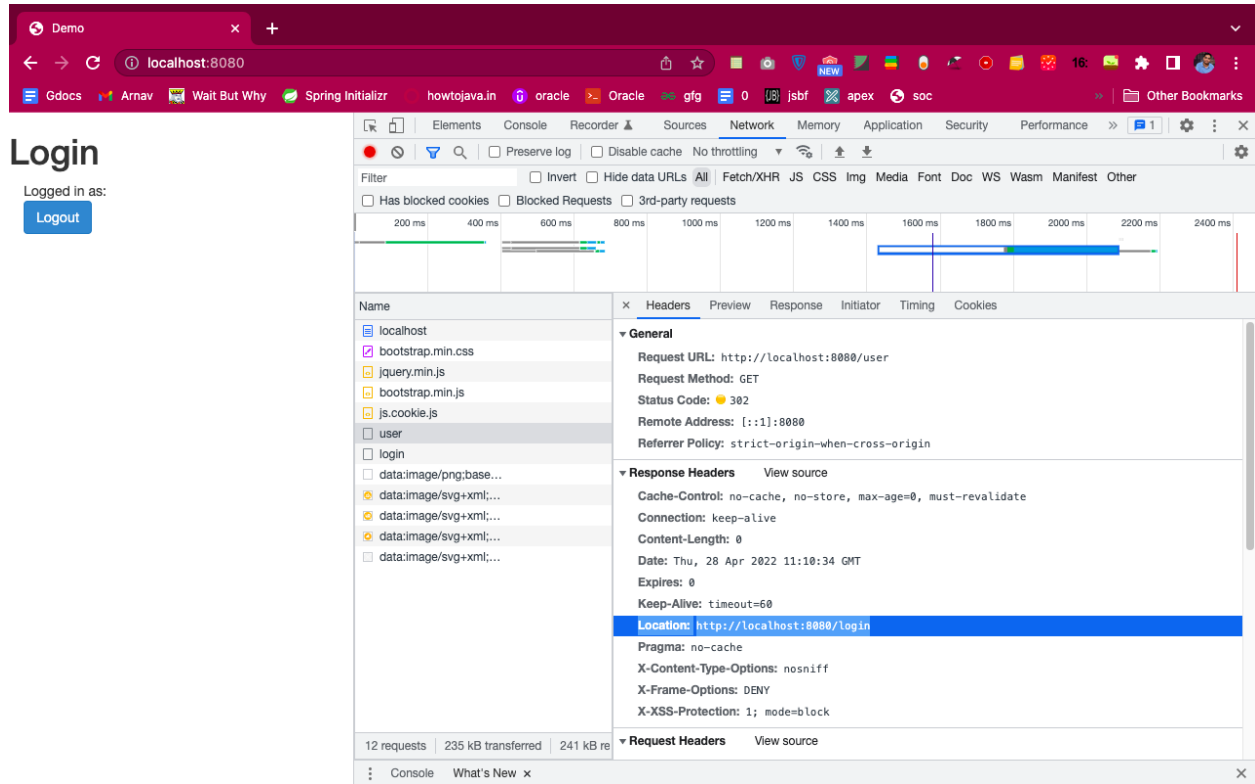
```
.exceptionHandling(e -> e  
  
                                .authenticationEntryPoint(new  
HttpStatusEntryPoint(HttpStatus.UNAUTHORIZED))  
  
                                )
```

HINT : When / will be hit , it will load html which will in turn load js written in script tag .
Now in the script tag , we are hitting /user with below .

```
$.get("/user", function(data) {  
    console.log('hello2')  
    $("#user").html(data.name);  
    $(".unauthenticated").hide();  
    $(".authenticated").show();  
});
```

The callback function is responsible for hiding the unauthenticated div and showing the authenticated div .But callback will only be called with successful response 200 from /user . Now something interesting happens ...

.anyRequest().authenticated() requires /user to be authenticated so it gets redirected to /login which is coming from spring security . Now we are not handling authentication in our project , so it returns a 200 response . This causes the /user to call the callback creating a wrong impression that we are logged in but we are not as can be confirmed from the below image .

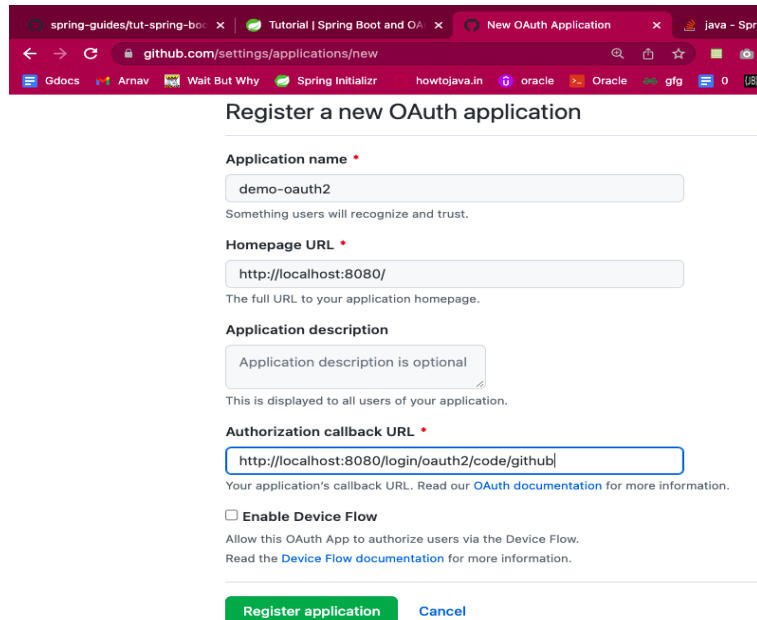


The screenshot shows a web browser at `localhost:8080` with a 'Login' page. The page has a 'Logout' button. The Chrome DevTools Network tab is open, showing a list of requests. The selected request is for `http://localhost:8080/login`. The 'Headers' panel is expanded, showing the following details:

- General:**
 - Request URL: `http://localhost:8080/user`
 - Request Method: `GET`
 - Status Code: `302`
 - Remote Address: `:::1:8080`
 - Referrer Policy: `strict-origin-when-cross-origin`
- Response Headers:**
 - Cache-Control: `no-cache, no-store, max-age=0, must-revalidate`
 - Connection: `keep-alive`
 - Content-Length: `0`
 - Date: `Thu, 28 Apr 2022 11:10:34 GMT`
 - Expires: `0`
 - Keep-Alive: `timeout=60`
 - Location: `http://localhost:8080/login` (highlighted)
 - Pragma: `no-cache`
 - X-Content-Type-Options: `nosniff`
 - X-Frame-Options: `DENY`
 - X-XSS-Protection: `1; mode=block`
- Request Headers:** (empty)

At the bottom of the Network tab, it shows 12 requests, 235 kB transferred, and 241 kB received.

Lets now strictly follow the steps in the documentation and register the cloned app with Github . Post that lets start the server and hit the `localhost:8080/ api` .



The screenshot shows the GitHub 'Register a new OAuth application' form. The fields are filled as follows:

- Application name:** `demo-oauth2`
- Homepage URL:** `http://localhost:8080/`
- Application description:** (empty)
- Authorization callback URL:** `http://localhost:8080/login/oauth2/code/github`

The 'Enable Device Flow' checkbox is unchecked. At the bottom, there are two buttons: 'Register application' (green) and 'Cancel' (blue).

Now let us understand the flow . When we start our app and hit localhost:8080/ , it loads the html file which inturn loads the javascript and /user gets hit . Now , as /user is unauthenticated and according to our configure function , only authenticated and public permit all calls are allowed . So it gets caught by exceptionhandler() in the configure function and /user returns 401 which does not allow callback to execute and we see the login screen .

The moment we click on login with github link , it hits /oauth2/authorization/github which redirects to

https://github.com/login/oauth/authorize?response_type=code&client_id=<client_id>&scope=read:user&state=C28&redirect_uri=http://localhost:8080/login/oauth2/code/github

This api inturn redirects to

<http://localhost:8080/login/oauth2/code/github?code=0792&state=C2YDf> which redirects to <http://localhost:8080/> and grants access token to the our application.

<http://localhost:8080/> api loads the html file which inturn loads the javascript and /user gets hit . Now , as /user is authenticated this time , so /user returns 200 response and

Session 16



map with principal details . This map gets passed as an argument to the callback function which gets called as soon as /user returns 200 response .

Click on the logout button . It will log us out ,clear the session and invalidate the cookie.

Logout Functionality

Challenge 7: Have we written a POST request for /logout? If not , then how is our project able to perform logout functionality?

HINT : Check the inbuilt functionality provided by spring security in the configure function .

Challenge 8: We are not using csrf().disable() but still /logout POST request is able to function successfully . How are we dealing with csrf requests here ?

Further General Challenges about OAuth2 :

Challenge 9: yaml and yml files are the same . Why do we have 2 extensions even when they are exactly the same ?

Challenge 10: Will my app start if I give an incorrect client id or password ?

Challenge 11: I have authorized Github once . Now once my server stops , why does Github not ask for consent to use my github profile for login when I click login with github ?

HINT : user token is stored .Check the client's oauth2 github dashboard.

Challenge 12: Now lets analyze the properties/yml file and see where the properties are coming . Are these custom properties or already defined properties ?

Challenge 13: In the config class , we only have authorization function , not the authentication function . Why so ?

Session 16



HINT : We have outsourced the authentication part to oauth2 provider which in our case is Github.

Challenge 14: What will happen if we do formLogin() instead of oauth2Login() ?

Challenge 15: From where is the name coming in the front end ?

Challenge 16: In the frontend script tag we can see /user is being hit . Where is the controller mapping for this endpoint written ?

Challenge 17: Let's suppose earlier I did not have a set company in my github . Now if I set it and reload <https://localhost:8080> , will it start showing the company ? If I logout in the app and again login with github in the app , will it start showing company ? Or will it need a restart of the server ?

Challenge 18: How can all of us access an application whose server is running in my machine ?

HINT : NGROK (./ngrok http <port>)

References :

<https://www.geeksforgeeks.org/types-of-authentication-protocols/>

<https://www.geeksforgeeks.org/difference-between-ldap-and-oauth-2/>

<https://www.geeksforgeeks.org/oauth2-authentication-with-spring-and-github/>

<https://owasp.org/www-community/attacks/csrf>

