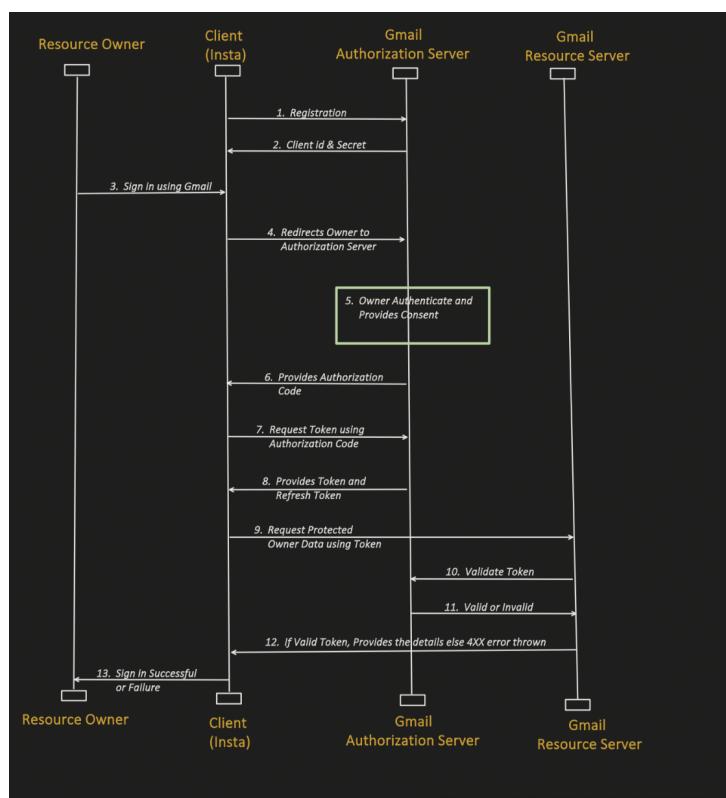


As explained in previous video:



- What is OAuth:  
It's an Open Authorization framework, enables secure third party access to user protected data.
- And its different Grant Types like
  - Authorization Code Grant,
  - Implicit Grant,
  - Client Credentials Grant etc....

#### **Quick Recap of "Authorization Code Grant" flow:**



**So, before we proceed with User Authentication implementation using OAuth framework,**

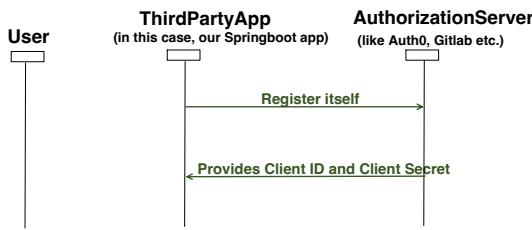
**Lets understand difference between OAuth and OIDC**

S.N 0	Title	OAuth2	OIDC
1.	Full Form	Open Authorization	OpenID Connect
2.	Purpose Used for	Authorization • <b>Grant</b> secure third party access to <b>user protected data</b> like third party app showing my google calendar data	Authentication - Layer built on top of OAuth2 and enables third party app to <b>verify the identity</b> of the user by an Authorization server
3.	Token generated	Access token (Opaque or JWT) - used by Third Party App, to call resource server API's to access user protected data.	ID_Token (JWT) + Access Token - ID_Token is a <b>JWT</b> token, which is meant for Third party app, and can be used to

		<ul style="list-style-type: none"> <li>- Many times, these access token could be <b>Opaque</b>, means only Authorization server interpret and validate them.</li> </ul>	<p style="text-align: center;">Authenticate user. (this JWT token contains minimal user info just required for Authentication)</p>
4.	Scope	<p>read/write: request access to read or write to resources.</p> <p>profile: request access to basic profile info like name, profile picture etc.</p> <p>email: request access to user email address</p> <p>etc..</p>	<p>openid (must) : it indicates that third party app is requesting a ID TOKEN (to authenticate the user)</p> <p>Now, this ID TOKEN (JWT) itself will contain some minimal info which can be used to authenticate user. But if required some additional info. We can use scope= openid, profile</p> <p>so this JWT now also has some profile related info too.</p>

### Step1:

- Third party app registration to Authorization server.



#### GitLab Registration:

User settings / Applications

Edit application

Name: My SpringBoot-App

Redirect URI: http://localhost:8080/login/oauth2/code/gitlab

Scopes

- api Grants complete read/write access to the API, including all groups and projects, the container registry, the dependency proxy, and the package registry.
- read\_api Grants read access to the API, including all groups and projects, the container registry, and the package registry.
- read\_user Grants read-only access to your profile through the User API endpoint, which includes username, public\_email, and full\_name. Also grants access to read-only API endpoints under /users.
- create\_runner Grants create access to the runners.
- manage\_runner Grants access to manage the runners.
- kbs\_proxy Grants permission to perform Kubernetes API calls using the agent for Kubernetes.
- read\_repository Grants read-only access to repositories on private projects using Git-over-HTTP or the Dependent File API.
- write\_repository Grants read-write access to repositories on private projects using Git-over-HTTP (not using the Dependent File API).
- read\_registry Grants read-only access to container registry images on private projects.
- write\_registry Grants write access to container registry images on private projects. You need both read and write access to push images.
- read\_virtual\_registry Grants read-only access to container images through the dependency proxy in private projects.
- write\_virtual\_registry Grants read, write, and delete access to container images through the dependency proxy in private projects.
- read\_observability Grants read-only access to GitLab Observability.
- write\_observability Grants write access to GitLab Observability.
- ai\_features Grants access to GitLab Duo related API endpoints.
- sudo Grants permission to perform API actions as any user in the system, when authenticated as an admin.
- admin\_mode Grants permission to perform API actions as an administrator, when Admin Mode is enabled.
- read\_service\_ping Grant access to download Service Ping payload via API when authenticated as an admin user.
- openid Grants permission to authenticate with GitLab using OpenID Connect. Also gives read-only access to the user's profile and group memberships.
- profile Grants read-only access to the user's profile data using OpenID Connect.
- email Grants read-only access to the user's primary email address using OpenID Connect.

Save application

#### Auth0 Registration:

Basic Information

Name \*: My SpringBoot App

Domain: dev-g4t3m706jqpcif6.us.auth0.com

Client ID: p03Hg12bXTL1Yb8PwP20Wqfek6P1Ex

Client Secret: The Client Secret is not base64-encoded.

Description: Add a free text description in less than 140 characters.

A free text description of the application. Max character count is 140.

Allowed Callback URLs: http://localhost:8080/login/oauth2/code/auth0

JSON Web Token (JWT) Signature Algorithm: RS256

Specify the algorithm used to sign the JSON Web Token: HS256: JWT will be signed with your client secret. RS256: JWT will be signed with your private signing key and they can be verified using your public signing key (see Certificates - Signing Certificate section).

Trust Token Endpoint IP Header:

Trust that the IP specified in the 'auth0-forwarded-for' header is the end-user's IP for brute-force-protection on token endpoint.

OIDC Conformant:

Applications flagged as OIDC Conformant will strictly follow the OIDC specification. Turning on this flag can introduce breaking changes to this application. If you have any questions you can contact support.

**After registration, we get Client id and Secret**

Application: My-SpringBoot-App

Application ID	e224307910bf0d7b087b34076cd6d11488bd829da99ff9c2dea3cd7516b87e45	
Secret	.....	Renew secret
This is the only time the secret is accessible. Copy the secret and store it securely.		
Callback URL	http://localhost:8080/login/oauth2/code/gitlab	
Confidential	Yes	
Scopes	openid (Authenticate using OpenID Connect)	

### pom.xml

```
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-oauth2-client</artifactId>
</dependency>
```

### application.properties

```
#OAuth configurations
##GitLab
spring.security.oauth2.client.registration.gitlab.client-id=e24307910bf0d7b087b34076cd6d11488bd829da99ff9c2dea3cd7516b87e45
spring.security.oauth2.client.registration.gitlab.client-secret=gloss-4d80d8c0b97e7807e36270769e5da47b2467af9395fb418bd745177ec9c81d18
spring.security.oauth2.client.registration.gitlab.scope=openid
spring.security.oauth2.client.registration.gitlab.authorization-grant-type=authorization_code
spring.security.oauth2.client.registration.gitlab.redirect-uri=http://localhost:8080/login/oauth2/code/gitlab
spring.security.oauth2.client.provider.gitlab.authorization-uri=https://gitlab.com/oauth/authorize
spring.security.oauth2.client.provider.gitlab.token-uri=https://gitlab.com/oauth/token
spring.security.oauth2.client.provider.gitlab.issuer-uri=https://gitlab.com
spring.security.oauth2.client.provider.gitlab.jwk-set-uri=https://gitlab.com/oauth/discovery/keys

##Auth0
spring.security.oauth2.client.registration.auth0.client-id=pCDh6Li2bXTLlyb0PwFzGMVgFek6PiEx
spring.security.oauth2.client.registration.auth0.client-secret=cSWyt6ubVJ_NJ1jb8GX4mBUlnYb622ejCpnqYxQ1JRYEpo7IC5wpDM8E6bG6t
spring.security.oauth2.client.registration.auth0.scope=openid, profile
spring.security.oauth2.client.registration.auth0.authorization-grant-type=authorization_code
spring.security.oauth2.client.registration.auth0.redirect-uri=http://localhost:8080/login/oauth2/code/auth0
spring.security.oauth2.client.provider.auth0.authorization-uri=https://dev-q4t3m70i6jqdc16i.us.auth0.com/authorize
spring.security.oauth2.client.provider.auth0.token-uri=https://dev-g4t3m70i6jqdc16i.us.auth0.com/oauth/token
spring.security.oauth2.client.provider.auth0.issuer-uri=https://dev-q4t3m70i6jqdc16i.us.auth0.com/
spring.security.oauth2.client.provider.auth0.jwk-set-uri=https://dev-g4t3m70i6jqdc16i.us.auth0.com/.well-known/jwks.json
```

### SecurityConfig.java

```
@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http.authorizeHttpRequests(auth -> auth
                .anyRequest().authenticated()
                .csrf(csrf -> csrf.disable())
                .oauth2Login(Customizer.withDefaults());
        return http.build();
    }
}
```

### Basic Controller class, just for testing

```
@RestController
public class UserDetailsController {

    @GetMapping("/")
    public String defaultHomePageMethod(){
        return "hello, you are logged in";
    }

    @GetMapping("/users")
    public String getUsersDetails(){
        return "fetched the details of successfully";
    }
}
```

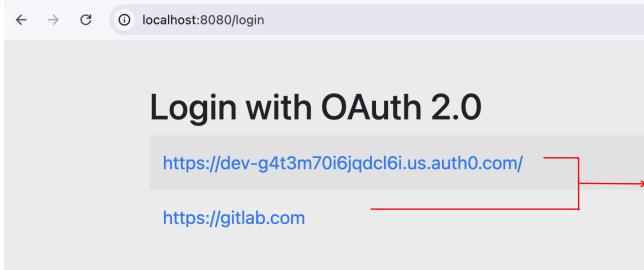
### Let's try:

Notice one thing that, I have not created any User in our Springboot app.

### Started the application server :

```
Started the application server :  
[INFO] [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8080 (http) with context path /  
[INFO] [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Started SpringBootApplication in 2.87 seconds (process /  
[INFO] [main] o.s.c.c.CachedResource : Initializing Spring DispatcherServlet 'dispatcherServlet'  
[INFO] [main] o.s.c.c.CachedResource : Initializing Servlet 'dispatcherServlet'  
[INFO] [main] o.s.c.c.CachedResource : Completed initialization in 1 ms
```

Typed the [localhost:8080](http://localhost:8080) url, it takes us to /login endpoint



When I clicked the Auth0 authorization server, it redirects me to Auth0 login page

Welcome

Log in to dev-g4t3m70i6jqdcl6i to continue to  
My SpringBoot App.

Email address\*

Password\*

[Forgot password?](#)

[Continue](#)

Once I provided the sign in at Auth0 and provided the consent, I am able to logged in

localhost:8080

hello, you are logged in

Now, if I try to access, any other API, I don't have to sign in again and access will be given.

localhost:8080/users

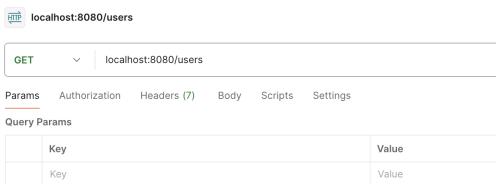
fetched the details of successfully

What? With just pom.xml, application.properties and SecurityConfig.java changes, we able to run the complete OAuth2 flow.

Answer is Yes, Springboot Security framework provides the compete functionality of OAUTH2 protocol, we don't have to code anything.

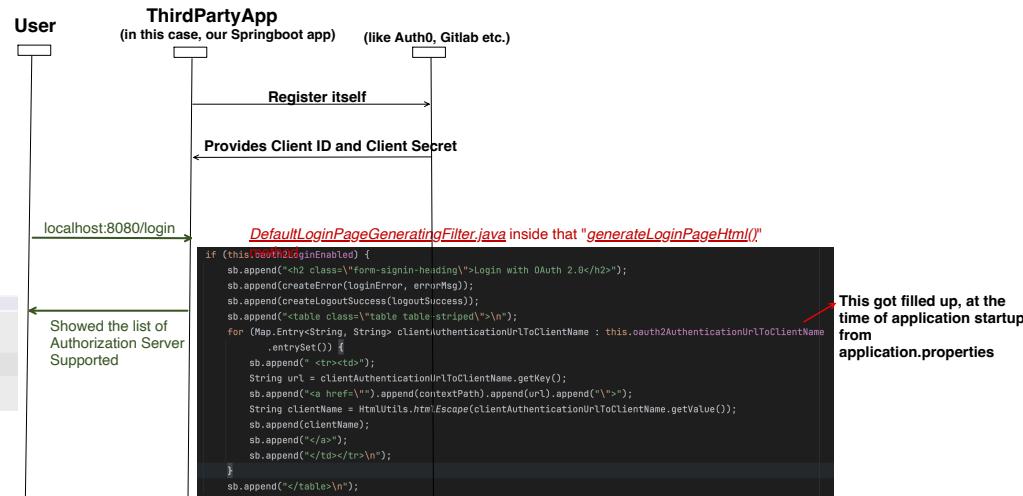
But here is the twist:

When I tried to access the "/users" API through postman (not through browser), it takes me back to Login page, why?



Because, Springboot assume that, OAuth2 login will be done on a browser, so by-default it creates SESSION.

### AuthorizationServer



User click on one of the authorization server links, then /oauth2/authorization/{registration\_id} will get invoked.  
In this registration\_id is either auth0 or gitlab, what we have configured in application.properties

```
public class OAuth2AuthorizationRequestRedirectFilter extends AbstractHttpConfigurer<OAuth2AuthorizationRequestRedirectFilter> {
    @Override
    public void init(OAuth2AuthorizationRequestRedirectFilter oAuth2AuthorizationRequestRedirectFilter) {
        super.init(oAuth2AuthorizationRequestRedirectFilter);
        oAuth2AuthorizationRequestRedirectFilter.setRegistrationId("auth0");
        oAuth2AuthorizationRequestRedirectFilter.setClientAuthenticationMethod("client_secret_basic");
        oAuth2AuthorizationRequestRedirectFilter.setScope("openid profile");
        oAuth2AuthorizationRequestRedirectFilter.setResponseType("code");
        oAuth2AuthorizationRequestRedirectFilter.setAuthorizationGrantType("authorization_code");
        oAuth2AuthorizationRequestRedirectFilter.setAuthorizationEndpoint("https://auth0.com/oauth/authorize");
        oAuth2AuthorizationRequestRedirectFilter.setTokenEndpoint("https://auth0.com/oauth/token");
        oAuth2AuthorizationRequestRedirectFilter.setIssuer("https://auth0.com");
        oAuth2AuthorizationRequestRedirectFilter.setDiscovery("https://auth0.com/.well-known/openid-configuration");
        oAuth2AuthorizationRequestRedirectFilter.setJwkSet("https://dev-g4t3m70l6jqdcl6i.us.auth0.com/.well-known/jwks.json");
    }
}
```

*OAuth2AuthorizationRequestRedirectFilter.java*

```

@Override
protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain)
    throws ServletException, IOException {
try {
    OAuth2AuthorizationRequest authorizationRequest = this.authorizationRequestResolver.resolve(request);
    if (authorizationRequest == null) {
        this.sendRedirectForAuthorization(request, response, authorizationRequest);
        return;
    }
}

```

authorizationRequest = (OAuth2AuthorizationRequest@14004)

- additionalParameters = (Collections\$UnmodifiableMap@14012)
- authorizationUri = "https://gitlab.com/oauth/authorize"
- authorizationGrantType = (AuthorizationGrantType@12119)
- responseType = (OAuth2AuthorizationResponseType@14007)
- clientId = "e224307910bf7d7b087b3d676cd6d11488bd029da99ff9c2d; e43cd75160874e58scope=openIdState; e478hs5EKUkfW5sDJss3VgKEW4qT9aE0lqL3w8RxwM;"
- redirectUri = "http://localhost:8080/login/oauth2/code/gitlab"
- scopes = (Collections\$UnmodifiableSet@14010) size = 1
- state = "g78hs5EKUkfW5sDJss3VgKEW4qT9aE0lqL3w8RxwM;"
- authorizationRequestURI = "https://gitlab.com/oauth/authorize?response\_type=code&client\_id=e224307910bf7d7b087b3d676cd6... View
- attributes = (Collections\$UnmodifiableMap@14014) size = 2
- registration\_id -> "gitlab"

URL (decoded) Raw

[https://gitlab.com/oauth/authorize?response\\_type=code&client\\_id=e224307910bf7d7b087b3d676cd6d11488bd029da99ff9c2d; e43cd75160874e58scope=openIdState; e478hs5EKUkfW5sDJss3VgKEW4qT9aE0lqL3w8RxwM;](https://gitlab.com/oauth/authorize?response_type=code&client_id=e224307910bf7d7b087b3d676cd6d11488bd029da99ff9c2d; e43cd75160874e58scope=openIdState; e478hs5EKUkfW5sDJss3VgKEW4qT9aE0lqL3w8RxwM;)

https://dev-g4t3m70i6jqdc16i.us.auth0.com/authorize  
or  
https://gitlab.com/oauth/authorize

Invokes authorization-uri of the specific registration\_id

Welcome

Log in to dev-g4t3m70i6jqdc16i.us.auth0.com to continue to My SpringBoot App.

Email address:

Password:

Forgot password?

Continue

GitLab.com

Username or primary email:

Password:

Forgot your password?

Remember me

Sign in

http://localhost:8080/login/oauth2/code/gitlab  
or  
http://localhost:8080/login/oauth2/code/auth0

Redirect API got invoked by authorization server, with Authorization Code present in the response

authorizationResponse = (OAuth2AuthorizationResponse@17855)

- redirectUri = "http://localhost:8080/login/oauth2/code/gitlab"
- state = "g78hs5EKUkfW5sDJss3VgKEW4qT9aE0lqL3w8RxwM;"
- code = "7353f4ccfc6510ef0439428a3eccb579143c2e6b1a8bd0f063c68062efa4d781"
- error = null

```

@Override
public Authentication attemptAuthentication(HttpServletRequest request, HttpServletResponse response)
    throws AuthenticationException {
    OAuth2AuthorizationResponse authorizationResponse = OAuth2Utils.convert(params,
        redirectUri);
    Object authenticationDetails = this.authenticationDetailsSource.buildDetails(request);
    OAuth2LoginAuthenticationToken authenticationRequest = new OAuth2LoginAuthenticationToken(clientRegistration,
        new OAuth2AuthorizationExchange(authenticationRequest, authorizationResponse));
    authenticationRequest.setDetails(authenticationDetails);
    OAuth2LoginAuthenticationToken authenticationResult = (OAuth2LoginAuthenticationToken) this
        .getAuthenticationManager()
        .authenticate(authenticationRequest);
    OAuth2AccessToken oauth2Authentication = this.authenticationResultConverter
        .convert(authenticationResult);
    Assert.notNull(oauth2Authentication, message: "authentication result cannot be null");
    oauth2Authentication.setDetails(authenticationDetails);
    OAuth2AuthorizedClient authorizedClient = new OAuth2AuthorizedClient(
        authenticationResult.getClientRegistration(), oauth2Authentication.getName(),
        authenticationResult.getAccessToken(), authenticationResult.getRefreshToken());
    this.authorizedClientRepository.saveAuthorizedClient(authorizedClient, oauth2Authentication, request, response);
    return oauth2Authentication;
}

```

Creates an Authentication object

Calls Authentication Manager, in this case, since our scope is "openid" and Authentication object is OAuth2LoginAuthenticationToken, OidcAuthorizationCodeAuthenticationProvider.java will handle this request.

OidcAuthorizationCodeAuthenticationProvider.java

```

@Override
public Authentication authenticate(Authentication authentication) throws AuthenticationException {
    OAuth2LoginAuthenticationToken authorizationCodeAuthentication = (OAuth2LoginAuthenticationToken) authentication;
    // Section 3.1.2.1 Authentication Request -
    // https://openid.net/specs/openid-connect-core-1_0.html#AuthRequest
    // scope
    // REQUIRED. (p)enID Connect requests MUST contain the "openid" scope value.
    if (!authorizationCodeAuthentication.getAuthorizationExchange()
        .getAuthorizationRequest().getScopes()
        .contains(OidcScopes.OPENID)) {
        // This is NOT an OpenID Connect Authentication Request so return null
        // and let OAuth2LoginAuthenticationProvider handle it instead
        return null;
    }
    OAuth2AuthorizationRequest authorizationRequest = authorizationCodeAuthentication.getAuthorizationExchange()
        .getAuthorizationRequest();
    OAuth2AuthorizationResponse authorizationResponse = authorizationCodeAuthentication.getAuthorizationExchange()
        .getAuthorizationResponse();
    if (authorizationResponse.getStatusError()) {
        throw new OAuth2AuthenticationException(authorizationResponse.getError(),
            authorizationResponse.getError().toString());
    }
    if (!authorizationResponse.getState().equals(authorizationRequest.getState())) {
        OAuth2Error oauth2Error = new OAuth2Error(INVALID_STATE_PARAMETER_ERROR_CODE);
        throw new OAuth2AuthenticationException(oauth2Error, oauth2Error.toString());
    }
    OAuth2AccessTokenResponse accessTokenResponse = getResponse(authorizationCodeAuthentication);
    ClientRegistration clientRegistration = authorizationCodeAuthentication.getClientRegistration();
    Map<String, Object> additionalParameters = accessTokenResponse.getAdditionalParameters();
    if (!additionalParameters.containsKey(OidcParameterNames.ID_TOKEN)) {
        OAuth2Error invalidIdTokenError = new OAuth2Error(INVALID_ID_TOKEN_ERROR_CODE,
            description: "Missing (required) ID Token in Token Response for Client Registration: "
                + clientRegistration.getRegistrationId(),
            uri: null);
        throw new OAuth2AuthenticationException(invalidIdTokenError, invalidIdTokenError.toString());
    }
    OidcIdToken idToken = createOidcToken(clientRegistration, accessTokenResponse);
}

```

Invokes token URI of the Authorization server

From Response, it fetches the Access and Id\_Token, and return back to the OAuth2LoginAuthenticationFilter.java

https://dev-g4t3m70i6jqdc16i.us.auth0.com/oauth/to  
or

ken  
<https://gitlab.com/oauth/token>

Returns Access Token and ID\_TOKEN

**Now, OAuth2LoginAuthenticationFilter.java stores this tokens and creates a HttpSession**

**Login Successful home page is called with Session id in cookie**

localhost

Request URL:	http://localhost:8080/
Request Method:	GET
Status Code:	200 OK
Remote Address:	[::]: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:	40
Content-Type:	text/html; charset=UTF-8
Date:	Mon, 14 Apr 2025 10:17:43 GMT
Expires:	0
Keep-Alive:	timeout=60
Pragma:	no-cache
X-Content-Type-Options:	nosniff
X-Frame-Options:	DENY
X-Xss-Protection:	0
Request Headers	
Accept:	text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Accept-Encoding:	gzip, deflate, br, zstd
Accept-Language:	en-GB,en-US;q=0.9,en;q=0.8
Connection:	keep-alive
Cookie:	SESSION=ZWU2YnNNGEINWxOC00NDE5Lk5ZTUHOTZhOTk0ZAA2mNi

Notice, access token is not a JWT (jwt has 3 parts separated by '!'). So this must be Opaque token

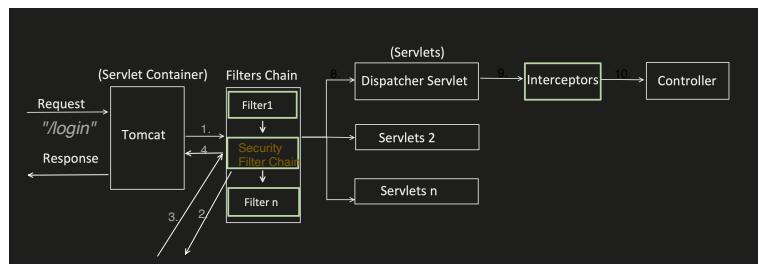
```

accessTokenResponse = (OAuth2AccessTokenResponse@13583)
  accessToken = (OAuth2AccessToken@7345)
    tokenType = (OAuth2AccessToken$TokenType@17355)
    scopes = (Collections$UnmodifiableSet@17359) size = 1
    tokenValue = "ce8ac80d0ba062838d55d7d029d4452ea009af418cee3d3593de4581d988b0d"
    issuedAt = (Instant@17358) "2025-04-14T00:18:18.180819Z"
    expiresAt = (Instant@17359) "2025-04-14T12:08:18.180819Z"
    refreshToken = (OAuth2RefreshToken@17346)
  additionalParameters = (Collections$UnmodifiableMap@13585) size = 2
    created_at = (Integer@17352) 1744625177
    id_token = (String@17351) "eyJ0eXAiOiJKV1QiLCJraWQiOiJuZXdpUXE5amIDODRddNzSiPQ1ONKE4V0ZMU1YyMEIiLYk3SWxXRFNRIiwYWxnjciUiMyNTYifQeyJpc3MiOiJodHRw... View"
  
```

Id\_token is in JWT form and when checked its payload data, we see all the data present which is required for Authentication

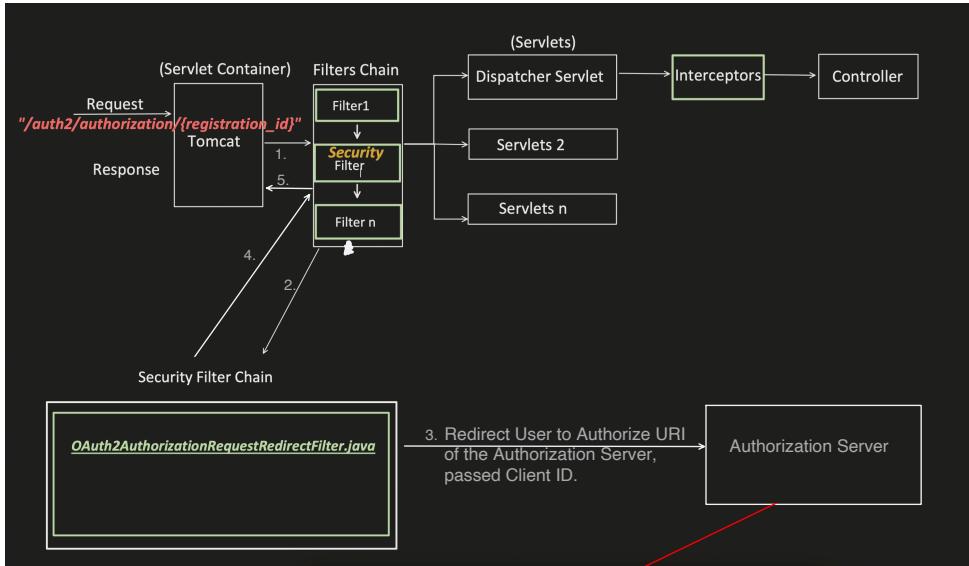
```

idToken = (IdToken@17606)
  claims = (Collections$UnmodifiableMap@17612) size = 14
    sub = "1305204"
    groups_direct = (ArrayList@12636) size = 0
    profile = "https://gitlab.com/shrayansh"
    iss = "https://gitlab.com"
    preferred_username = "shrayansh8"
    nonce = "KYMoeavRYoJfJlQLSuDVICoYrt0oxYPkfFGJ-I"
    picture = "https://secure.gravatar.com/avatar/ff704974a3a9ad29148e871109055ec60718b9739aca3519bc310efc128727?s=80&d=identicon"
    aud = (ArrayList@17648) size = 1
    auth_time = (Instant@17650) "2025-04-13T14:40:38Z"
    name = "shrayansh Jain"
    nickname = "shrayansh8"
    sub_legacy = "0616f3fd985fe5ca5fda0b57b8cc8950ea58967e45a36dc885be6a24ed28cc2"
    exp = (Instant@17614) "2025-04-14T00:08:18Z"
    iat = (Instant@17613) "2025-04-14T00:06:18Z"
  
```

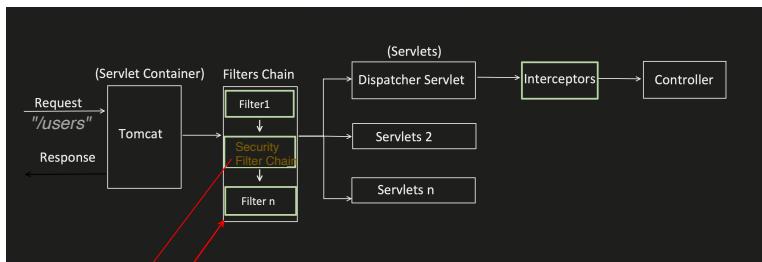
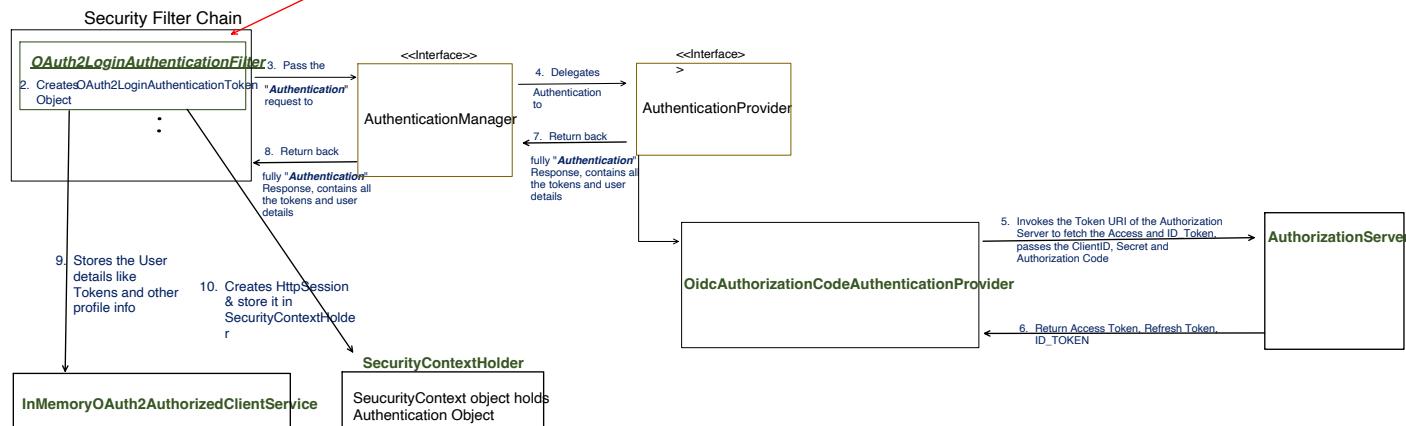


**DefaultLoginPageGeneratingFilter**

(returns the list of authorization servers supported)



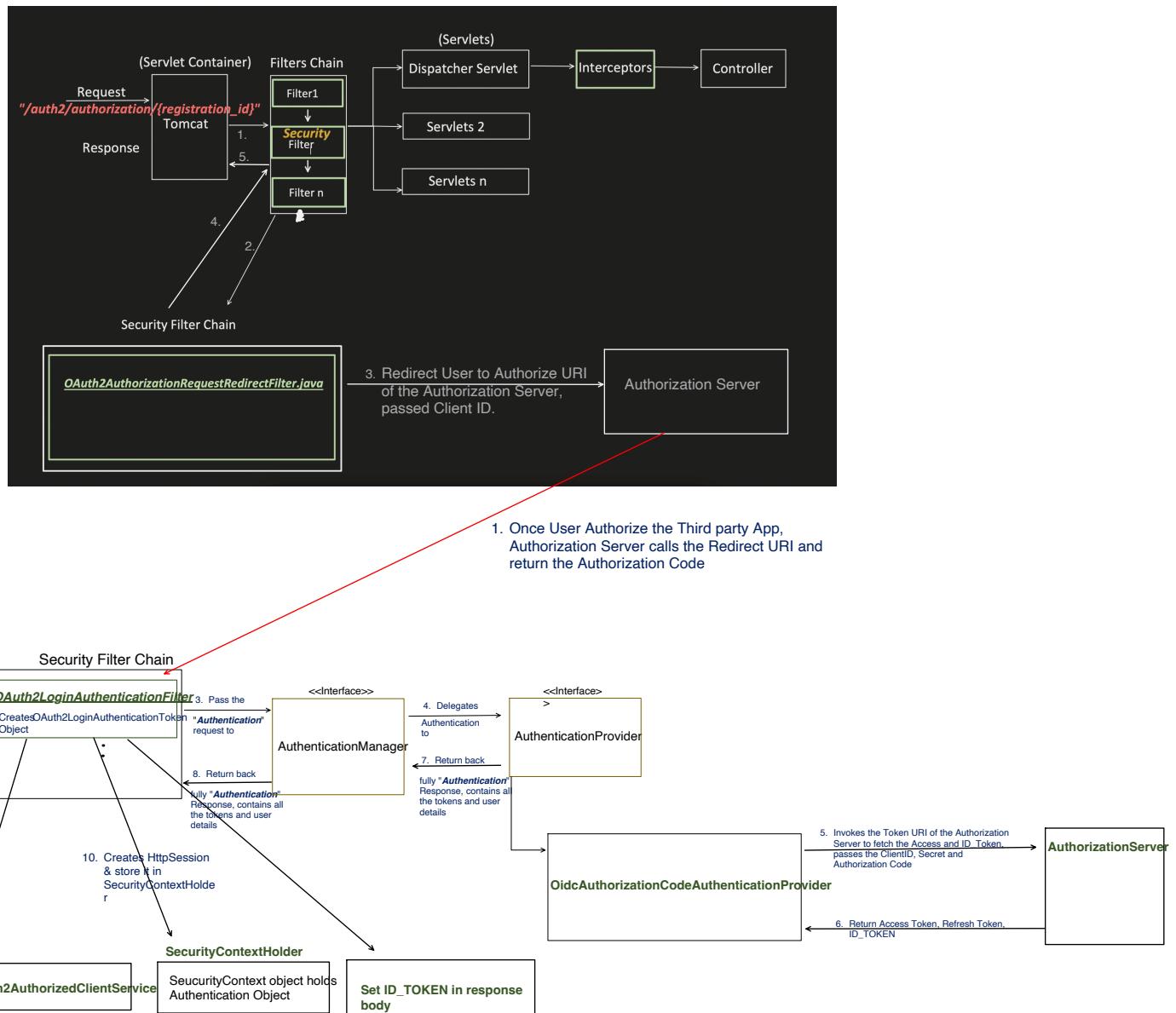
1. Once User Authorize the Third party App,  
Authorization Server calls the Redirect URI and  
return the Authorization Code



- That's why, when we hit the request from Postman, it again asked for Login, because Session id is not set in the cookie.
- Also, when session is set, it might not validate the token with each request, till Session is valid. So it might be a possible scenario that ID Token becomes invalidated but Session is still active.
- And this also makes OAuth Stateful.

So, how to fix this ?

- Lets make it STATELESS
- And return the ID\_TOKEN in the response
- With each request, client will pass the Token and we will validate the token.



OAuth2LoginAuthenticationFilter parent class has one method "onAuthenticationSuccess()" at last, which by default do some cleanup task once authentication process completes, I have overwrite that method and set the ID\_TOKEN in the response body.

```

@Component
public class CustomOAuth2SuccessHandler implements AuthenticationSuccessHandler {

    private final OAuth2AuthorizedClientService clientService;

    @Autowired
    public CustomOAuth2SuccessHandler(OAuth2AuthorizedClientService clientService) {
        this.clientService = clientService;
    }

    @Override
    public void onAuthenticationSuccess(HttpServletRequest request, HttpServletResponse response
                                         Authentication authentication) throws IOException {
        OAuth2AuthenticationToken authToken = (OAuth2AuthenticationToken) authentication;
        OAuth2AuthorizedClient client = clientService.loadAuthorizedClient(
            authToken.getAuthorizedClientRegistrationId(), authToken.getName());

        if (client != null) {
            String idToken = null;
            if (authToken.getPrincipal() instanceof OidcUser) {
                OidcUser oidcUser = (OidcUser) authToken.getPrincipal();
                idToken = oidcUser.getIdToken().getAccessTokenValue();
            }

            // Send the access token in the response (JSON)
            response.setContentType("application/json");
            response.getWriter().write(s: "{ \"id_token\": \"\\" + idToken + "\" }");
            response.getWriter().flush();
        } else {
            response.sendError(HttpServletRequest.SC_UNAUTHORIZED, s: "Authorization failed");
        }
    }
}

```

```

@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http,
                                                   CustomOAuth2SuccessHandler successHandler) {
        http.authorizeHttpRequests(auth -> auth
                                    .anyRequest().authenticated())
            .sessionManagement(session -> session
                            .sessionCreationPolicy(SessionCreationPolicy.STATELESS))
            .csrf(csrf -> csrf.disable())
            .oauth2Login(oauth -> oauth
                         .successHandler(successHandler));
        return http.build();
    }
}

```

Start the application

```

Hibernate:
  drop table if exists user_register cascade
Hibernate:
  create table user_register (
    id bigint generated by default as identity,
    password varchar(255) not null,
    role varchar(255),
    username varchar(255) not null unique,
    primary key (id)
)
2025-04-14T18:40:51.827+05:30 INFO 31908 --- [           main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2025-04-14T18:40:53.200+05:30 WARN 31908 --- [           main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database queries may be slow in some applications.
2025-04-14T18:40:53.242+05:30 INFO 31908 --- [           main] r$InitialUserDetailsManagerConfigurer : Global AuthenticationManager configured with UserDetailsService bean with name 'userDetailsService'
2025-04-14T18:40:53.487+05:30 INFO 31908 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8080 (http) with context path '/'
2025-04-14T18:40:53.492+05:30 INFO 31908 --- [           main] c.c.l.SpringbootApplication          : Started SpringbootApplication in 2.985 seconds (process running for 3.125)
2025-04-14T18:40:57.746+05:30 INFO 31908 --- [nio-8080-exec-1] o.a.c.c.C.[localhost].[]           : Initializing Spring DispatcherServlet 'dispatcherServlet'
2025-04-14T18:40:57.746+05:30 INFO 31908 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet      : Initializing Servlet 'dispatcherServlet'
2025-04-14T18:40:57.747+05:30 INFO 31908 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet      : Completed initialization in 1 ms

```

After Authorizing from the Authorization server:

```

Pretty print □
{
  "id_token": "eyJ0eXAiOiJKV1QiLCJraWQiOiIjrzXdpUXE5amlD0DRdIlnzSllPoi10nKE4V0ZMU1YyME1iLxk3SwxXRfnriiwiYwxiJoijUIMyNTYif0.eyJpc3MIo1JodHrwczovL2dpdGxhYi5jb20iLCJzdWIi0iIxMzA1MjA0IiwiYXVKiJoiZT1yNDMwNzkxMGlxZmQ3YjAN421znjA3NmNkmNx0MT040GjK0D15ZGE50wZm0MyZGVhMzNkNzUxNmI4N2U0NSisImV4cIGMtC0NDY2NjM4MSwiAf0TjoxNz00NjM2MjYxLCub25jZS16IkNWVUVdztobtJ1bnB5MFd1dn14ZFZUD1JXQ1BNdE1MOXV2c2g5sulswMGMiLClhdXro3RpblU0j0E3ND01NTUyMzgsIxN1y19s2wdhY3ki0iNjE2Z3NzDk4NWh2ly2F1mWzK2Yt1lNTd10Gn10dk10MvhNt5hj4ndVhMzK2YzgjNwJ1nmeEyTR1ZD14Y2MyIiwiibmFzZS161nNcm5Fy5zsaCe0y1uivibm1j25hbwU01j2ahJnewFuc241iwiCh1ZmVycmVX3Vz2XjUyW11jioic2hyXLhbN0CIsInByb2ppGu0i0jodHrwczovL2dpdGxhYi5jb20v2chYXlhbnNoOCIsInBpY3R1cmU0i0j0dhrWczovL3N1Ly3VzY2S5ncmf2YXRhc15j2b29YXzhdfGfyl22mhzNa00Tc0YTNh0WFkMj1mDh0GU30DExMDkzMTVLY2yMv2Y4jK3M2lhY2E2NTES5yMzTB1ZmUxMjg3Mj_cz04MCZKpwIKzW50aNvbii5imdyb3Vwc19kaXJ1Y3Qio1tdf0_i0f6d0ehofTPB_-BjEZag5BnYkg0sJr0CpXzwJrcINS16nJA47n0A2N0o0bhw_cNs_EMnkRRL6_Kh-wqA9SN9TCiN9Eq4PxYLK0yVcz16y1vRKAI62nWv7Yt-byTyc0vLe2NNEPjjdLX2x_w_9Np4Bmsvz2zTafp7y8_rnSV1rLvsV4VB23pAy6318H7izVm5eCnQn61vUxf88LvyL7PK3wdDt_YyVg0rNHaPexbdGjQAPFGheer7XLk2vzb_OrxMvu5qGjigbEqXLPui5pVKqabCLKTE6X2v-gyEBi0lQ267PfrnhqUuvPVpXKhsS1NKbps771HDrV1rLvsV4VB23pAy6318H7izVm5eCnQn61vUxf88LvyL7PK3wdDt_YyVg0rNHaPexbdGjQAPFGheer7XLk2vzb_OrxMvu5qGjigbEqXLPui5pVKqabCLKTE6X2v-zTkhbCsZUKqm28PrGUxWv0i10g6yemk5ue4US1yMth_asVoXHEpA7Z1DqghTrsagNLYgxPKuU6DsRb-jYy4jJ80oA60nGtmnDnS9NXYeqR6Ctbs_rjOn25ui0Lpo-302cb5kxmW5h5xjDeelioSA94H5ye5Kyb0n0x0UWh8rqFwD_kFNF8cFtPs_0083cjFjQDIBiv@BNL_14KMPAMWRBQUGHzD9qd1zrUKi7XA8zQkfpw0BfdtrRq2uysRk0YBUpWtbt1fp1SmHX43RHU1JdJrwQDarnWnLpbSvpvo"
}
```

Now, only 1 task left, now Client will pass this TOKEN with every request and we have to verify it.

GET  Send

Params Authorization  Headers (8) Body Scripts Settings

Auth Type

Bearer Token

Token

eyJ0eXAiOiJKV1QiLCJraWQiOiJrZXdpUXE5...

Cookies

Created New Filter to Validate the token

```
public class OAuthValidationFilter extends OncePerRequestFilter {

    private final OAuthTokenValidatorUtil tokenValidatorUtil;

    @Autowired
    public OAuthValidationFilter(OAuthTokenValidatorUtil tokenValidatorUtil) {
        this.tokenValidatorUtil = tokenValidatorUtil;
    }

    @Override
    protected void doFilterInternal(HttpServletRequest request,
                                    HttpServletResponse response,
                                    FilterChain filterChain) throws ServletException, IOException {

        String token = extractJwtFromRequest(request);
        if (token != null) {

            String username = tokenValidatorUtil.isTokenValid(token);
            if (StringUtil.isNullOrEmpty(username)) {
                response.sendError(HttpServletRequest.SC_UNAUTHORIZED, "Invalid or expired token");
                return;
            }
            Authentication auth = new UsernamePasswordAuthenticationToken(username, null, List.of());
            SecurityContextHolder.getContext().setAuthentication(auth);
        }
        filterChain.doFilter(request, response);
    }

    private String extractJwtFromRequest(HttpServletRequest request) {
        String bearerToken = request.getHeader("Authorization");
        if (bearerToken != null && bearerToken.startsWith("Bearer ")) {
            return bearerToken.substring(7);
        }
        return null;
    }
}
```

Utility Class, just to validate the token

```
@Component
public class OAuthTokenValidatorUtil {

    public String isTokenValid(String accessToken) {

        String iss = getIssuerIdFromToken(accessToken);
        JwtDecoder decoder = JwtDecoders.fromIssuerLocation(iss);
        Jwt jwt = decoder.decode(accessToken);
        if(jwt != null) {
            return (String) jwt.getClaims().get("sub");
        }
        return null;
    }

    public static String getIssuerIdFromToken(String jwtToken) {
        try {
            String[] parts = jwtToken.split("\\.");
            if (parts.length < 2) {
                throw new IllegalArgumentException("Invalid JWT token");
            }

            String payloadJson = new String(Base64.getUrlDecoder().decode(parts[1]));
            ObjectMapper mapper = new ObjectMapper();
            Map<String, Object> payloadMap = mapper.readValue(payloadJson, Map.class);
            String iss = (String) payloadMap.get("iss");
            return iss;
        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }
}
```

```
@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Autowired
    private OAuthTokenValidatorUtil tokenValidatorUtil;

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http,
                                                   CustomOAuth2SuccessHandler successHandler) throws Exception {
        http.authorizeHttpRequests(auth -> auth
                .anyRequest().authenticated()
                .sessionManagement(session -> session
                        .sessionCreationPolicy(SessionCreationPolicy.STATELESS))
                .csrf(csrf -> csrf.disable())
                .oauth2Login(oauth -> oauth
                        .successHandler(successHandler))
                .addFilterBefore(new OAuthValidationFilter(tokenValidatorUtil), UsernamePasswordAuthenticationFilter.class));

        return http.build();
    }
}
```

GET localhost:8080/users

Params Authorization Headers (8) Body Scripts Settings

Auth Type: Bearer Token

The authorization header will be automatically generated when you send the request. Learn more about [Bearer Token](#) authorization.

Token: eyJ0eXAiOiJKV1QiLCJraWQiOjRzXdpUXE5amIDDRDdlNzIIPQ10NE4v0ZMU1YyME1iLXk3SWxxRFNRliwiWxnjiouiMyNTYiFQeyJpc3MlOjodHrwczovL2dpjdGxhYi5jb20lCJzdWliOixMzA1MjA0liwiXVkjlioiZTlvNDMwNzKxMGIxZmQ3YjA4Nl2zNjA3NmNkhNmQxMTQ4OGJkODISZGE5OWmWzMyZGvhM2NKnzUxNmI4N2U0NSlsm4Cc16MtcoNDYzNg4NyviaWF0joNxQONJM3ny3LCJub25jZl6jldRaIQ3XdiQkNCMGZr2kobtZ2bGrZlfYk1UekFpOHUzS29fxzU4c7QilCJhdxRox3RpBWUj0e3NDQ1NTUyMzglnNtY19sZWdhY3kiOliwNjE22jNzDk4NWZlY2FfNWZkYtliNTdiOGnjODK10WVvhNTg5NjdiNDvhMzzkYzg4N

Body Cookies (1) Headers (11) Test Results

200 OK

Raw Preview Visualize

fetched the details of successfully