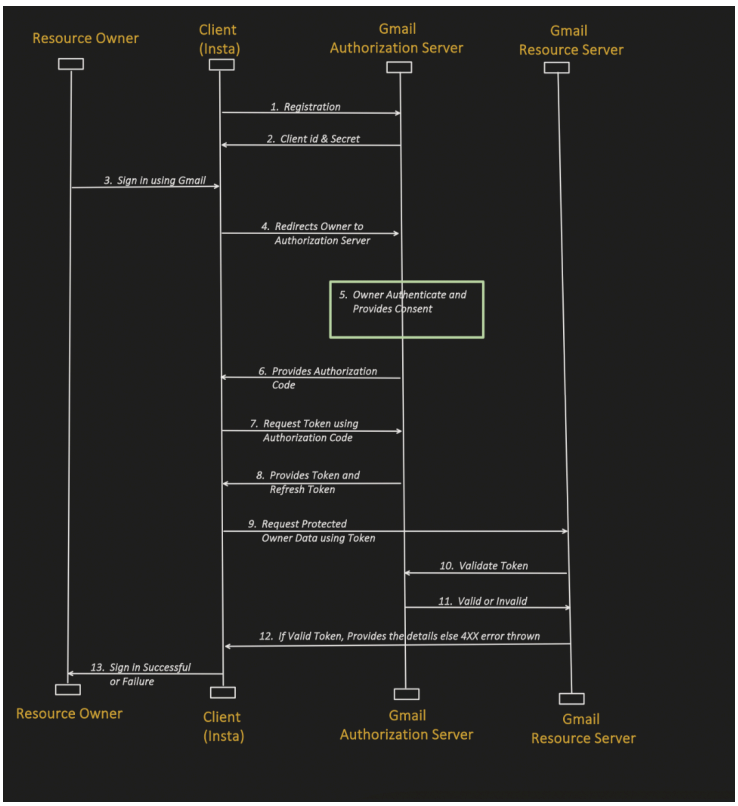
As explained in previous video:

- 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,

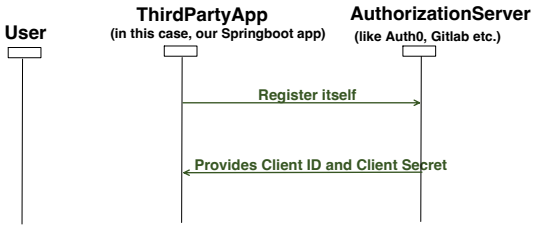
Lets understand difference between OAUTH and OIDC

S.No	Title	OAuth2	OIDC
1.	Full Form	Open Authorization	OpenID Connect
2.	Purpose Used for	Authorization • **Grant** secure third party access to **user protected data** like third party app showing my google calendar data	Authentication - Layer built on top of OAuth2 and enables third party app to **verify the identity** 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 JWT token, which is meant for Third party app, and can be used to

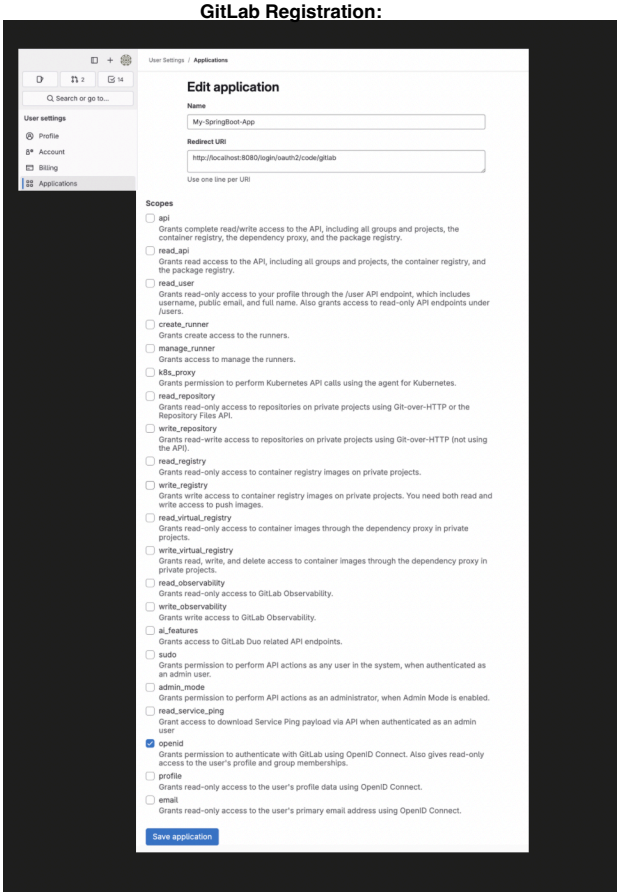
		- Many times, these access token could be Opaque, means only Authorization server interpret and validate them.	Authenticate user. *(this JWT token contains minimal user info just required for Authentication)* - Also has Access token, we can call resource server API's to access more user restricted data, if required.
4.	Scope	read/write: request access to read or write to resources. profile: request access to basic profile info like name, profile picture etc. email: request access to user email address etc..	openid (must) : it indicates that third party app is requesting a ID TOKEN (to authenticate the user) 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 so this JWT now also has some profile related info too.

Step1:

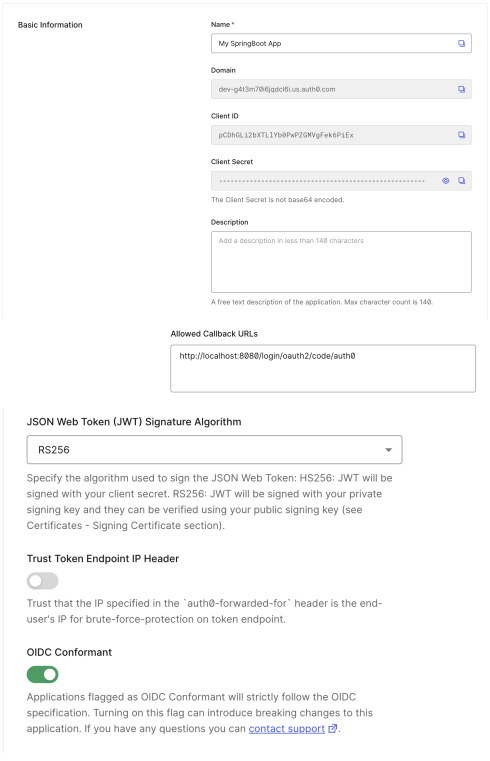
- **Third party app registration to Authorization server.**



GitLab Registration:



Auth0 Registration:



After registration, we get Client id and Secret

Application: My-SpringBoot-App

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=e224307910b1fd7b087b36076cd6d11488bd829da99ff9c2dea3cd7516b87e45
spring.security.oauth2.client.registration.gitlab.client-secret=gloas-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=pCDhGLi2bXTLlYb0PwPZGMVgFek6PiEx
spring.security.oauth2.client.registration.auth0.client-secret=crSWtyt6uBYJ_NJ1jbN8GX4mBULnYB622ejCnnzYxQ1JRyEpo7IC5wpDM8E6bG4t
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-g4t3m70i6jqdcl6i.us.auth0.com/authorize
spring.security.oauth2.client.provider.auth0.token-uri=https://dev-g4t3m70i6jqdcl6i.us.auth0.com/oauth/token
spring.security.oauth2.client.provider.auth0.issuer-uri=https://dev-g4t3m70i6jqdcl6i.us.auth0.com/
spring.security.oauth2.client.provider.auth0.jwk-set-uri=https://dev-g4t3m70i6jqdcl6i.us.auth0.com/.well-known/jwks.json
```

SecurityConfig.java

```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();
    }
}
```

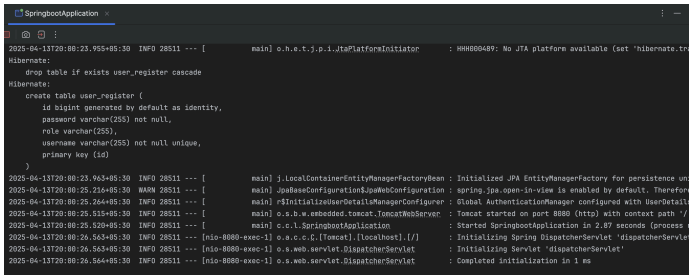
Basic Controller class, just for testing

```java
@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";
    }
}
```

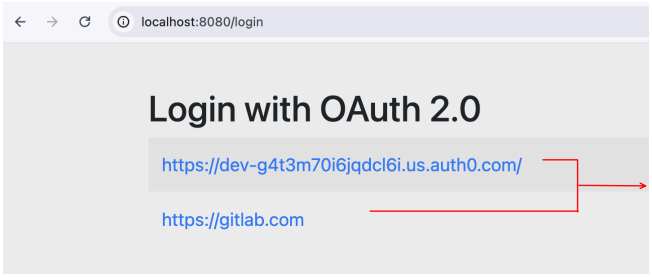
Let's try:

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

Started the application server :

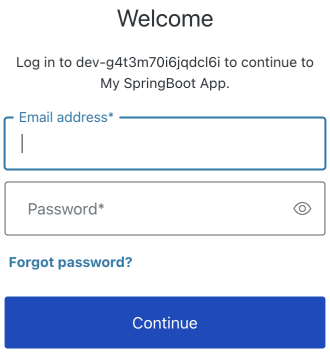


Typed the *localhost:8080* url, it takes us to /login endpoint

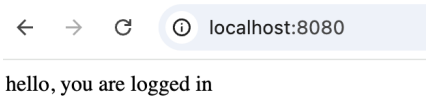


Two Authorization servers, Which we have added in application.properties.

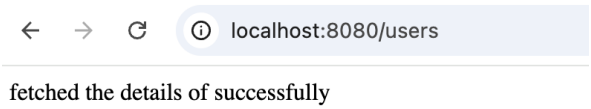
When I clicked the Auth0 authorization server, it redirects me to Auth0 login page



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



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

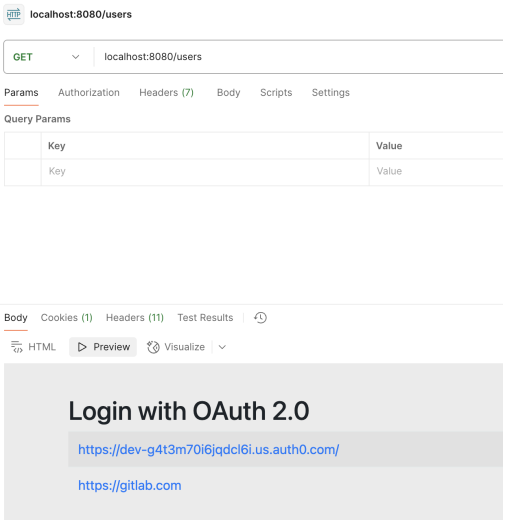


What? With just pom.xml, application.properties and SecurityConfig.java changes, we able to run the complete OAuth2 flow.

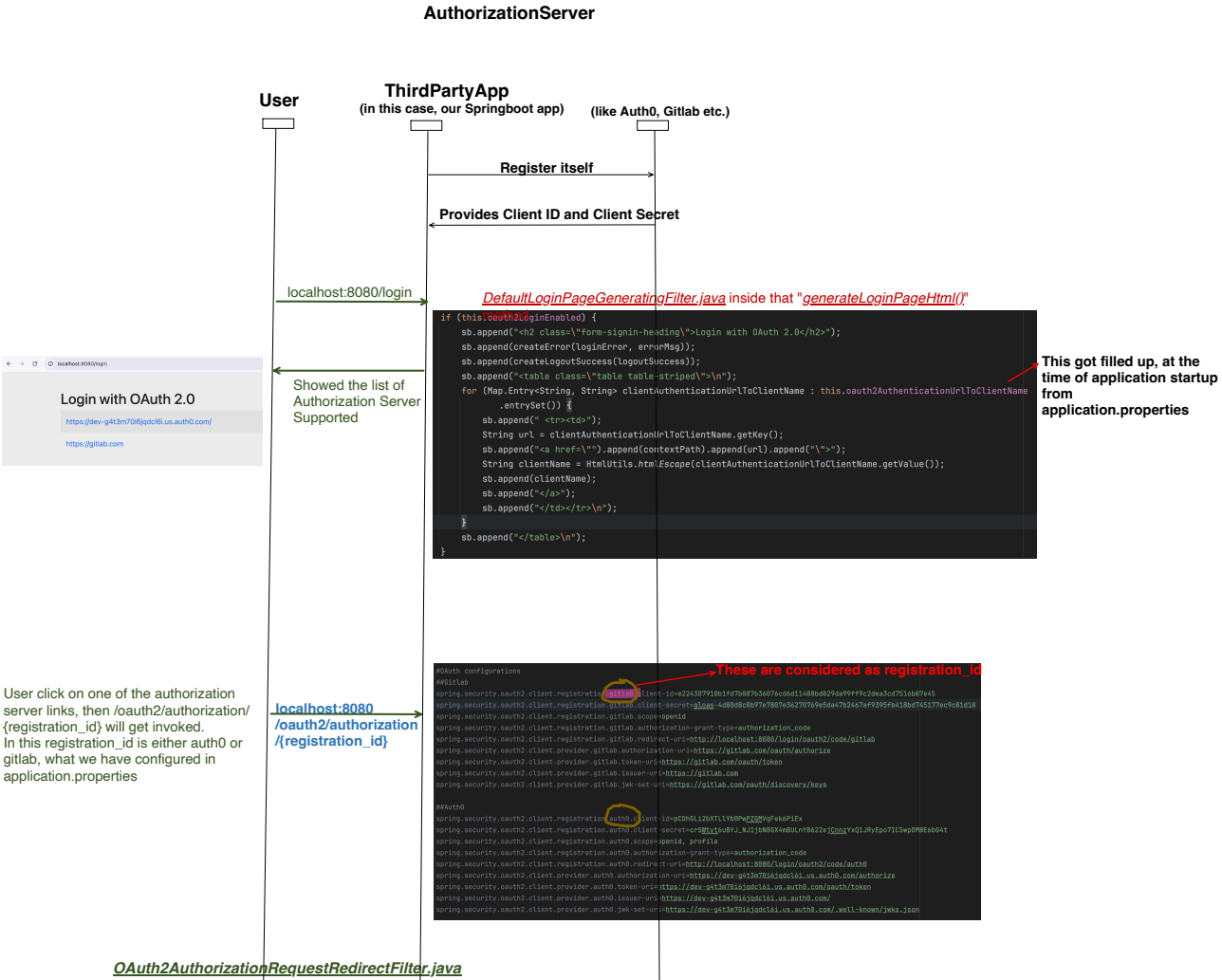
Answer is Yes, Springboot Security framework provides the compete functionality of OAUTH2 protocol, we don't have to code anything.

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.



```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;
        }
    }
```

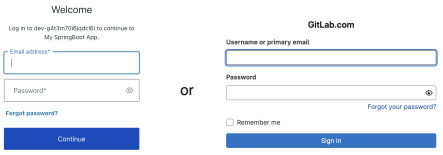
```
authorizationRequest = {OAuth2AuthorizationRequest@14004}
  additionalParameters = {Collections$UnmodifiableMap@14012} size = 1
  authorizationUri = "https://gitlab.com/oauth/authorize"
  authorizationGrantType = {AuthorizationGrantType@12119}
  responseType = {OAuth2AuthorizationResponseType@14007}
  clientId = "e224307910b1fd7b087b36076cd6d11488bd829da99ff9c2d"
  redirectUri = "http://localhost:8080/login/oauth2/code/gitlab"
  scopes = {Collections$UnmodifiableSet@14010} size = 1
  state = "g78hs5E5KUkfW5sDJSs3Vg5KEW4qT9aEOiqL3w8RxwM="
  authorizationRequestUri = "https://gitlab.com/oauth/authorize?response_type=code&client_id=e224307910b1fd7b087b36076cd6... View
  attributes = {Collections$UnmodifiableMap@14014} size = 2
    "registration_id" -> "gitlab"

URL (decoded)  Raw
https://gitlab.com/oauth/authorize?response_type,
=code&client_id,
=e224307910b1fd7b087b36076cd6d11488bd829da99ff9c2d,
=ea3cd7516b87e45&scope=openid&state,
=g78hs5E5KUkfW5sDJSs3Vg5KEW4qT9aEOiqL3w8RxwM,
=&redirect_uri=http://localhost:8080/login/oauth2,
=/code/gitlab&nonce=y7Bfhui5GUs3LUjfKq5BSRs_,
=G6HU5SXa6TDlAi3sa8
```

https://dev-g4t3m70i6jqdcl6i.us.auth0.com/authorize

or

https://gitlab.com/oauth/authorize

Invokes authorization-uri of the specific registration_id



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 = "g78hs5E5KUkfW5sDJSs3Vg5KEW4qT9aEOiqL3w8RxwM="
  code = "7353f4ccf6c510ef0439428a3eccb579143c2e6b1a6bdf063c680662efa4d781"
  error = null
```

OAuth2LoginAuthenticationFilter.java

```java
@Override
public Authentication attemptAuthentication(HttpServletRequest request, HttpServletResponse response)
        throws AuthenticationException {
    OAuth2AuthorizationResponse authorizationResponse = OAuth2AuthorizationResponseUtils.convert(params,
            redirectUri);
    Object authenticationDetails = this.authenticationDetailsSource.buildDetails(request);
    OAuth2LoginAuthenticationToken authenticationRequest = new OAuth2LoginAuthenticationToken(clientRegistration,
            new OAuth2AuthorizationExchange(authorizationRequest, authorizationResponse));
    authenticationRequest.setDetails(authenticationDetails);
    OAuth2LoginAuthenticationToken authenticationResult = (OAuth2LoginAuthenticationToken) this
            .getAuthenticationManager()
            .authenticate(authenticationRequest);
    OAuth2AuthenticationToken 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.jav

```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. OpenID Connect requests MUST contain the "openid" scope value.
    if (!authorizationCodeAuthentication.getAuthorizationExchange() OAuth2AuthorizationExchange
            .getAuthorizationRequest() OAuth2AuthorizationRequest
            .getScopes() Set<String>
            .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.statusError()) {
        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-g4t3m70i6jqdcl6i.us.auth0.com/oauth/to

or

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

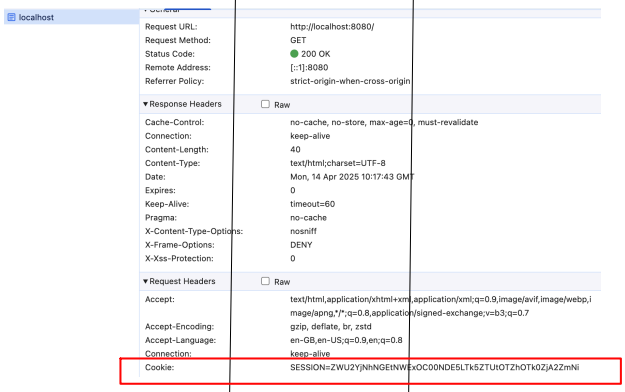
Returns
Access Token and
ID_TOKEN

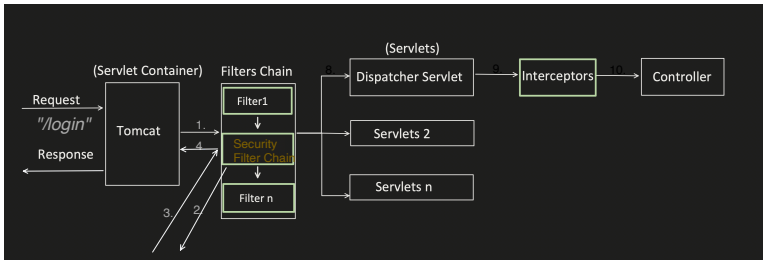
Notice, access token is not a JWT (jwt has 3 parts separated by '.' So this must be Opaque token

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

Now, OAuth2LoginAuthenticationFilter.java stores this tokens and creates a HTTPSession

Login Successful home page is called with Session id in cookie

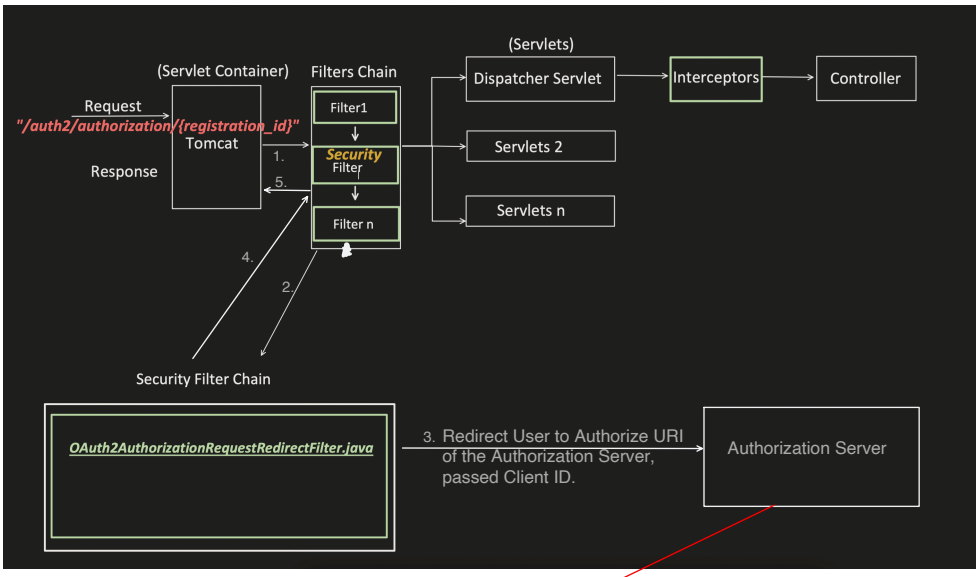




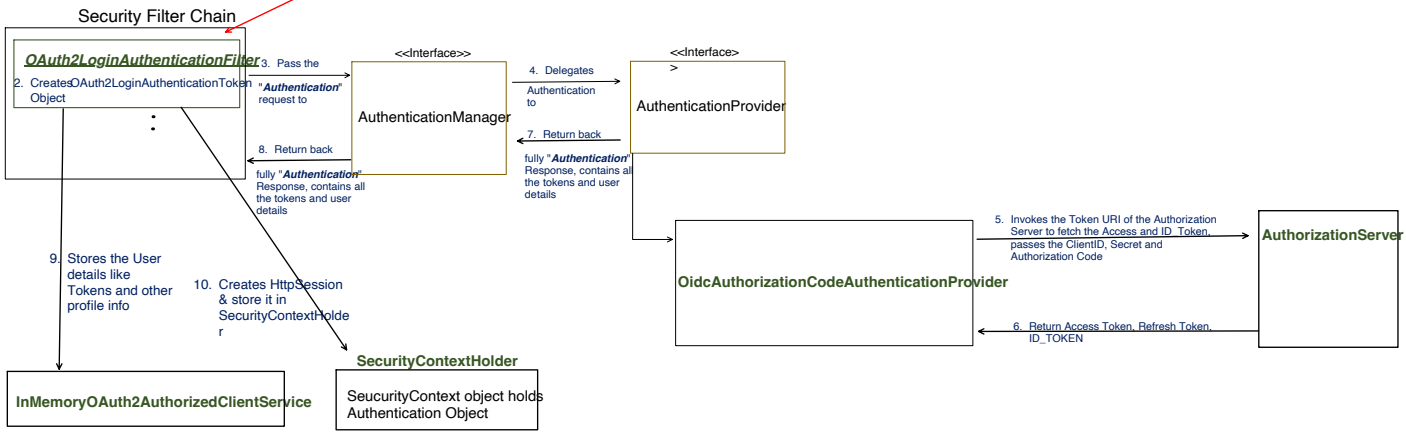
Security Filter Chain

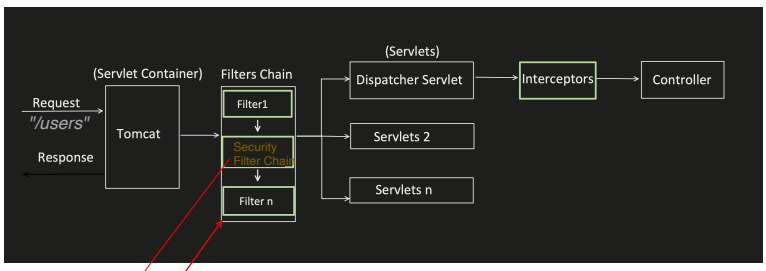
DefaultLoginPageGeneratingFilter

(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



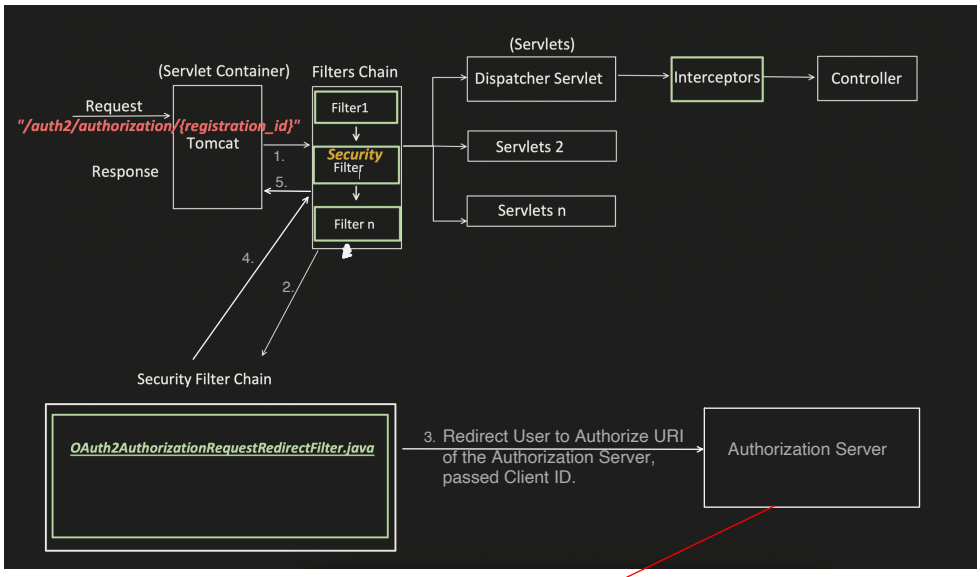


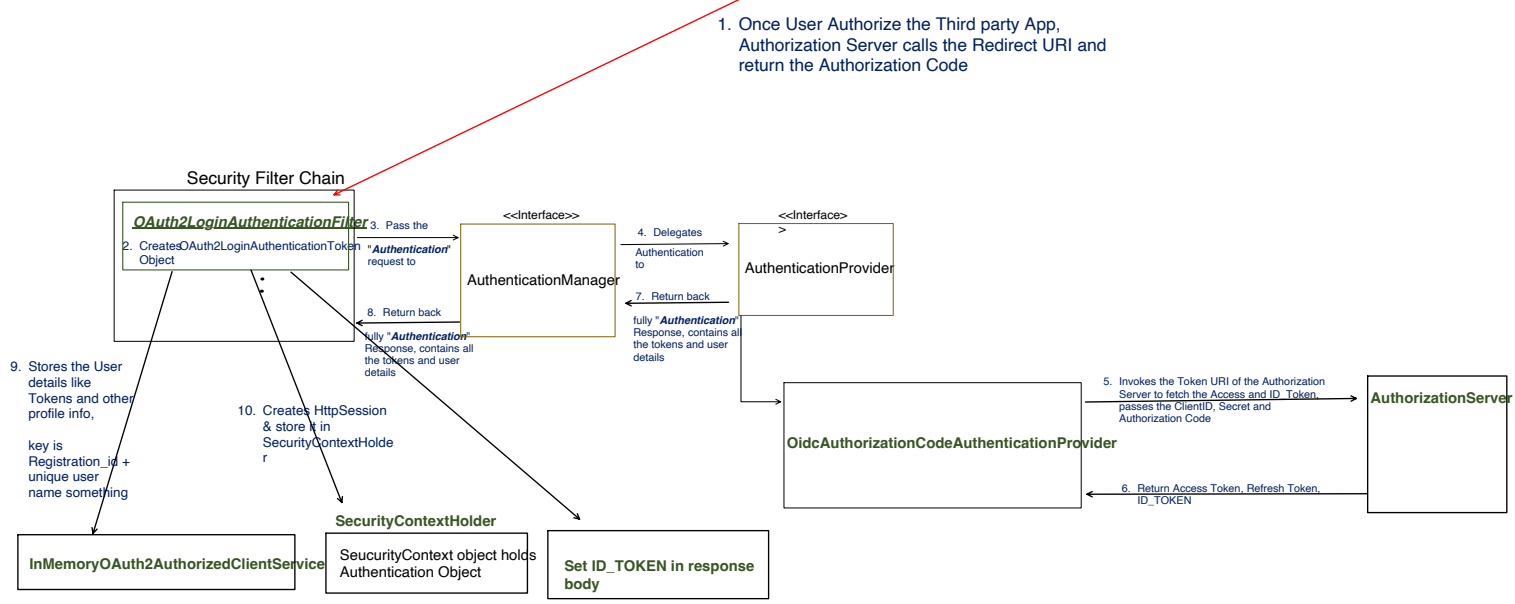
- That's why, when we hit the request from Postman, it again asked for Login, because Session id is not set in the cookie.
- Also, when session is set, it might not validate the token with each request, till Session is valid. So it might be a possible scenario that ID Token becomes invalidated but Session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

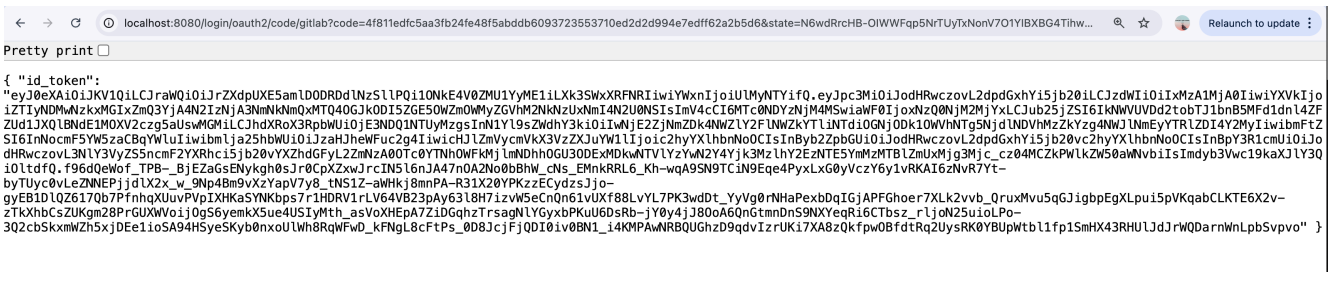
```java
@Component
public class CustomOAuth2SuccessHandler implements AuthenticationSuccessHandler {

    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().getTokenValue();
            }

            // 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(HttpServletResponse.SC_UNAUTHORIZED,  s: "Authorization failed");
        }
    }
}
```

```java
@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http,
                                                   CustomOAuth2SuccessHa
        http.authorizeHttpRequests(auth -> auth
                        .anyRequest().authenticated())
                .sessionManagement(session -> session
                        .sessionCreationPolicy(SessionCreationPolicy.STA
                .csrf(csrf -> csrf.disable())
                .oauth2Login(oauth -> oauth
                        .successHandler(successHandler));
        return http.build();
    }
}
```

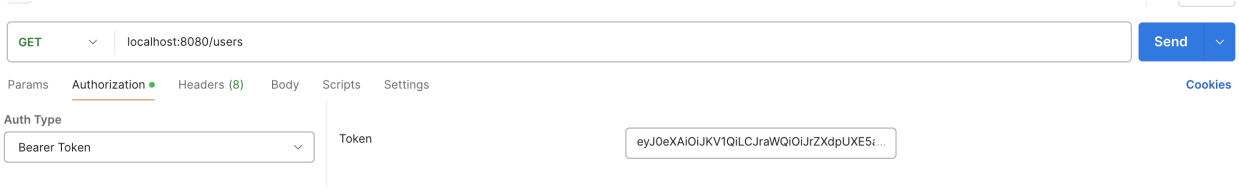
Start the application

```
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$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database queries may
2025-04-14T18:40:53.242+05:30  INFO 31908 --- [          main] r$InitializeUserDetailsManagerConfigurer : Global AuthenticationManager configured with UserDetailsService bean with name
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.[Tomcat].[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:



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



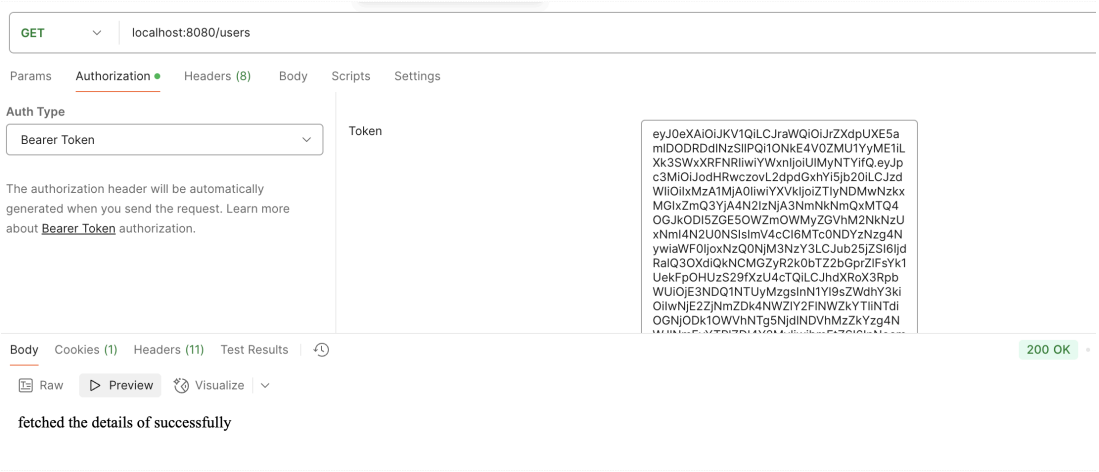
Created New Filter to Validate the token

```java
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(HttpServletResponse.SC_UNAUTHORIZED, s: "Invalid or expired token");
                return;
            }
            Authentication auth = new UsernamePasswordAuthenticationToken(username, credentials: null, List.of());
            SecurityContextHolder.getContext().setAuthentication(auth);
        }
        filterChain.doFilter(request, response);
    }

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

Utility Class, just to validate the token

```java
@Component
public class OAuthTokenValidatorUtil {

    public String isTokenValid(String accessToken) {

        String issu = getIssuerIdFromToken(accessToken);
        JwtDecoder decoder = JwtDecoders.fromIssuerLocation(issu);
        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( regex: "\\.");
            if (parts.length < 2) {
                throw new IllegalArgumentException("Invalid JWT to
            }

            String payLoadJson = new String(Base64.getUrlDecoder()
            ObjectMapper mapper = new ObjectMapper();
            Map<String, Object> payloadMap = mapper.readValue(payL
            String iss = (String) payloadMap.get("iss");
            return iss;
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }
}
```

```java
@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
eyJ0eXAiOiJKV1QiLCJraWQiOiJrZXdpUXE5amlDODRDdlNzSlIPQi1ONkE4V0ZMU1YyME1iLXk3SWxXRFNRliwiYWxnIjoiUlMyNTYifQ.eyJpc3MiOiJodHRwczovL2dpdGxhYi5jb20iLCJzdWIiOiIxMzA1MjA0liwiYXVkIjoiZTIyNDMwNzkxMGIxZmQ3YjA4N2IzNjA3NmNkNmQxMTQ4OGJkODI5ZGE5OWZmOWMyZGVhM2NkNzUxNmI4N2U0NSIsImV4cCI6MTc0NDYzNzg4NywiaWF0IjoxNzQ0NjM3NzY3LCJub25jZSI6IjdRalQ3OXdiQkNCMGZyR2k0bTZ2bGprZlFsYk1UekFpOHUzS29fXzU4cTQiLCJhdXRoX3RpbWUiOjE3NDQ1NTUyMzgsInN1YiI9sZWdhY3kiOilwNjE2ZjNmZDk4NWZlY2FlNWZkYTliNTdiOGNjODk1OWVhNTg5NjdlNDVhMzZkYzg4N
Body
Cookies (1)
Headers (11)
Test Results
200 OK
Raw
Preview
Visualize
fetched the details of successfully