Description of issue

This behavior is strange from my point of view. If I enable the refresh token

```
.tokenSettings(t -> {
t.enableRefreshTokens(true);
})
```

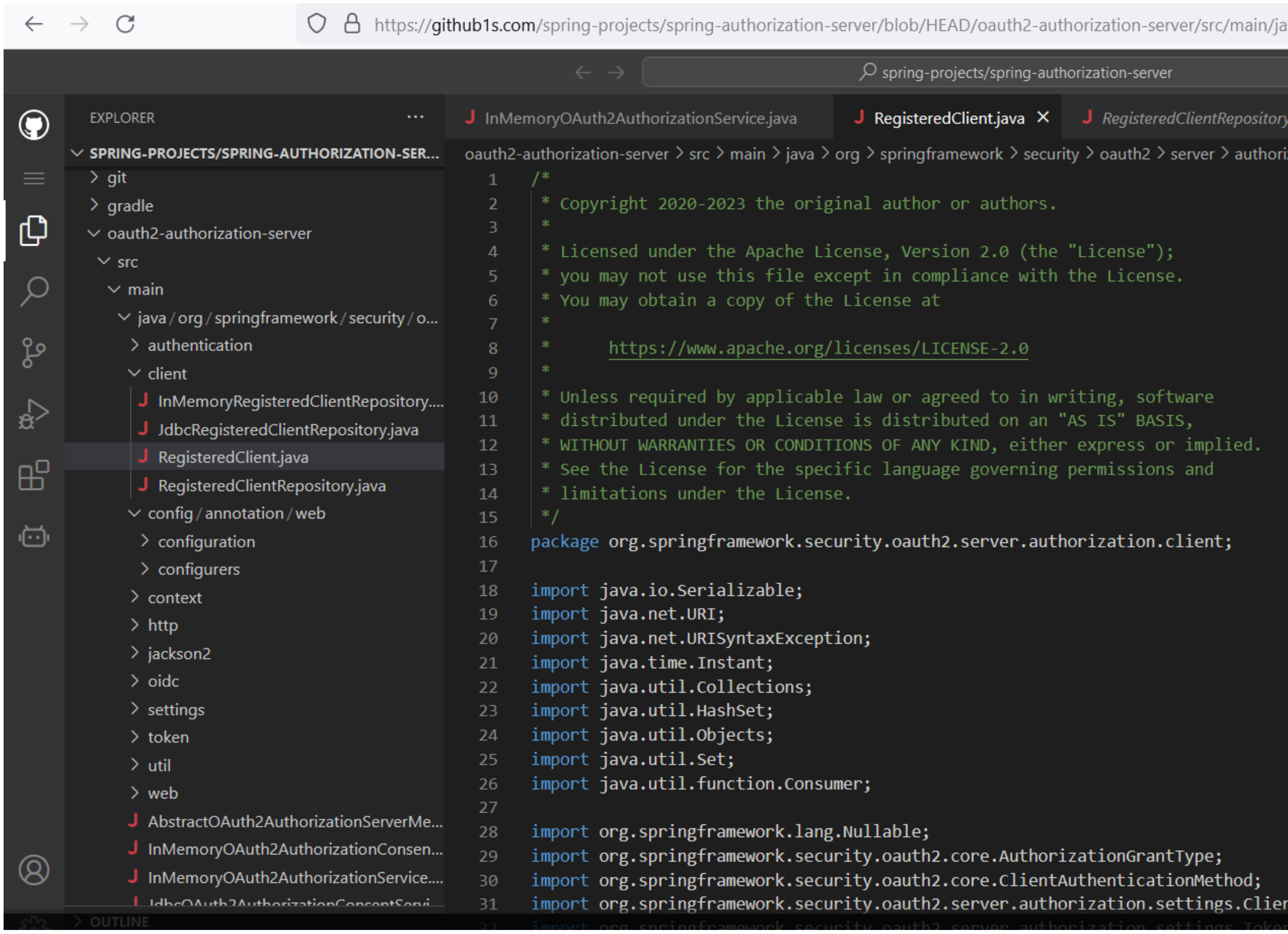
Which is also the default out of my observations. Even if the client doesn't support the REFRESH_TOKEN grant type, I get a refresh token in the response. Now, of course, I can never use that value because the client doesn't support this grant type. So why do I need to get the value at all then?

Maybe it's just my opinion, but I find this behavior strange. Shouldn't these two configurations be somehow coordinated?

Solution of issue

1. Dont enable the refresh tokens

2. Use the public client instead

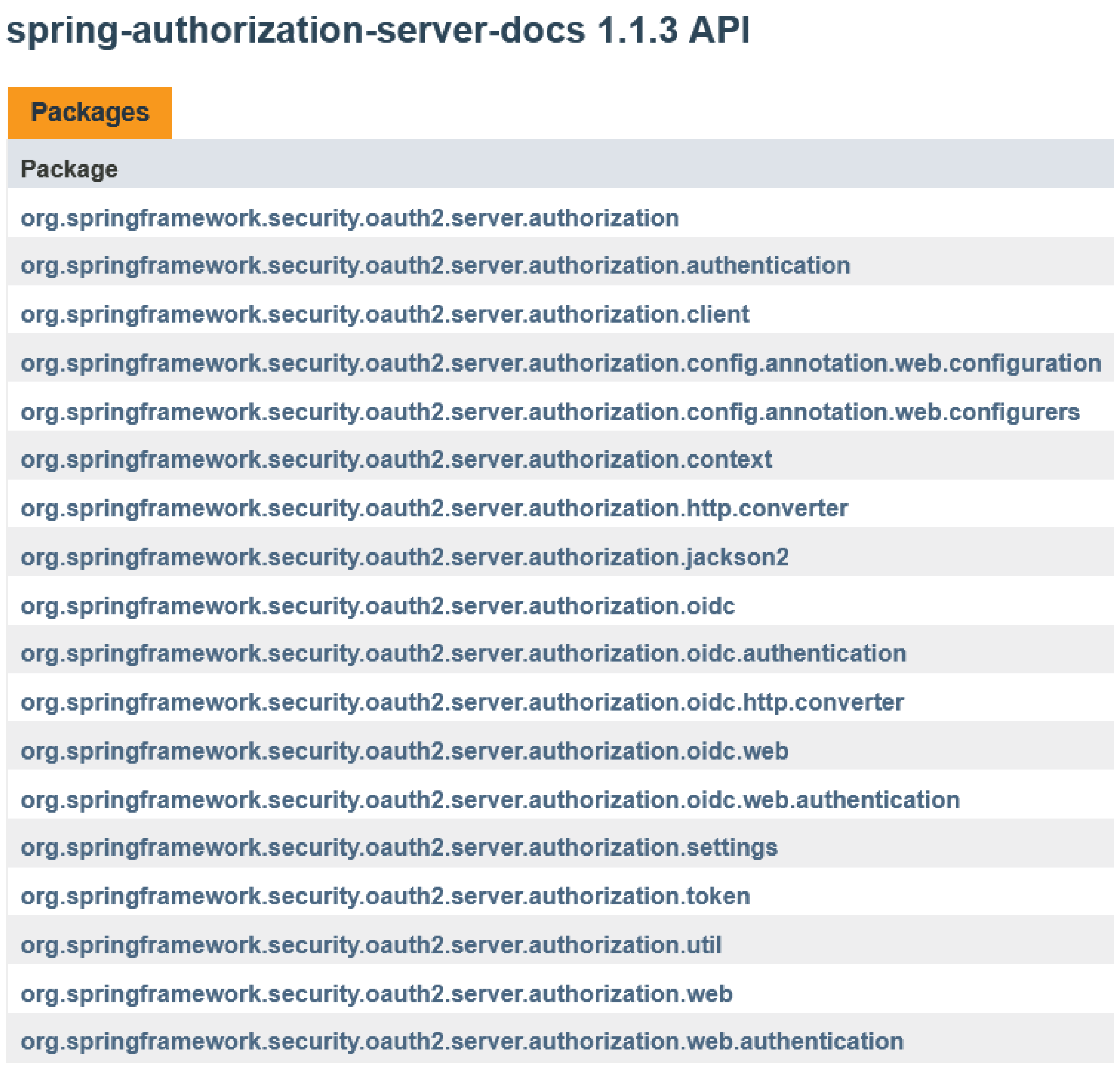

https://github1s.com/spring-projects/spring-authorization-server/blob/HEAD/oauth2-authorization-server/src/main/ja

spring-projects/spring-authorization-server

EXPLORER
SPRING-PROJECTS/SPRING-AUTHORIZATION-SER...
git
gradle
oauth2-authorization-server
src
main
java / org / springframework / security / o...
authentication
client
InMemoryRegisteredClientRepository....
JdbcRegisteredClientRepository.java
RegisteredClient.java
RegisteredClientRepository.java
config / annotation / web
configuration
configurers
context
http
jackson2
oidc
settings
token
util
web
AbstractOAuth2AuthorizationServerMe...
InMemoryOAuth2AuthorizationConsen...
InMemoryOAuth2AuthorizationService....
OUTLINE
InMemoryOAuth2AuthorizationService.java
RegisteredClient.java
RegisteredClientRepository
oauth2-authorization-server > src > main > java > org > springframework > security > oauth2 > server > authori

```java
/*
 * Copyright 2020-2023 the original author or authors.
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      https://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package org.springframework.security.oauth2.server.authorization.client;

import java.io.Serializable;
import java.net.URI;
import java.net.URISyntaxException;
import java.time.Instant;
import java.util.Collections;
import java.util.HashSet;
import java.util.Objects;
import java.util.Set;
import java.util.function.Consumer;

import org.springframework.lang.Nullable;
import org.springframework.security.oauth2.core.AuthorizationGrantType;
import org.springframework.security.oauth2.core.ClientAuthenticationMethod;
import org.springframework.security.oauth2.server.authorization.settings.Clier
```

As the RefreshToken is used almost everywhere , we take a look at the particular code and try to figure out the apis and inner workings



spring-authorization-server-docs 1.1.3 API

Packages

Package
org.springframework.security.oauth2.server.authorization
org.springframework.security.oauth2.server.authorization.authentication
org.springframework.security.oauth2.server.authorization.client
org.springframework.security.oauth2.server.authorization.config.annotation.web.configuration
org.springframework.security.oauth2.server.authorization.config.annotation.web.configurers
org.springframework.security.oauth2.server.authorization.context
org.springframework.security.oauth2.server.authorization.http.converter
org.springframework.security.oauth2.server.authorization.jackson2
org.springframework.security.oauth2.server.authorization.oidc
org.springframework.security.oauth2.server.authorization.oidc.authentication
org.springframework.security.oauth2.server.authorization.oidc.http.converter
org.springframework.security.oauth2.server.authorization.oidc.web
org.springframework.security.oauth2.server.authorization.oidc.web.authentication
org.springframework.security.oauth2.server.authorization.settings
org.springframework.security.oauth2.server.authorization.token
org.springframework.security.oauth2.server.authorization.util
org.springframework.security.oauth2.server.authorization.web
org.springframework.security.oauth2.server.authorization.web.authentication

Here is the description of which api file performs which work

READ IT LIKE THIS
When I want to authenticate a token , I would make changes to
TOKEN AND AUTHENTICATE file.

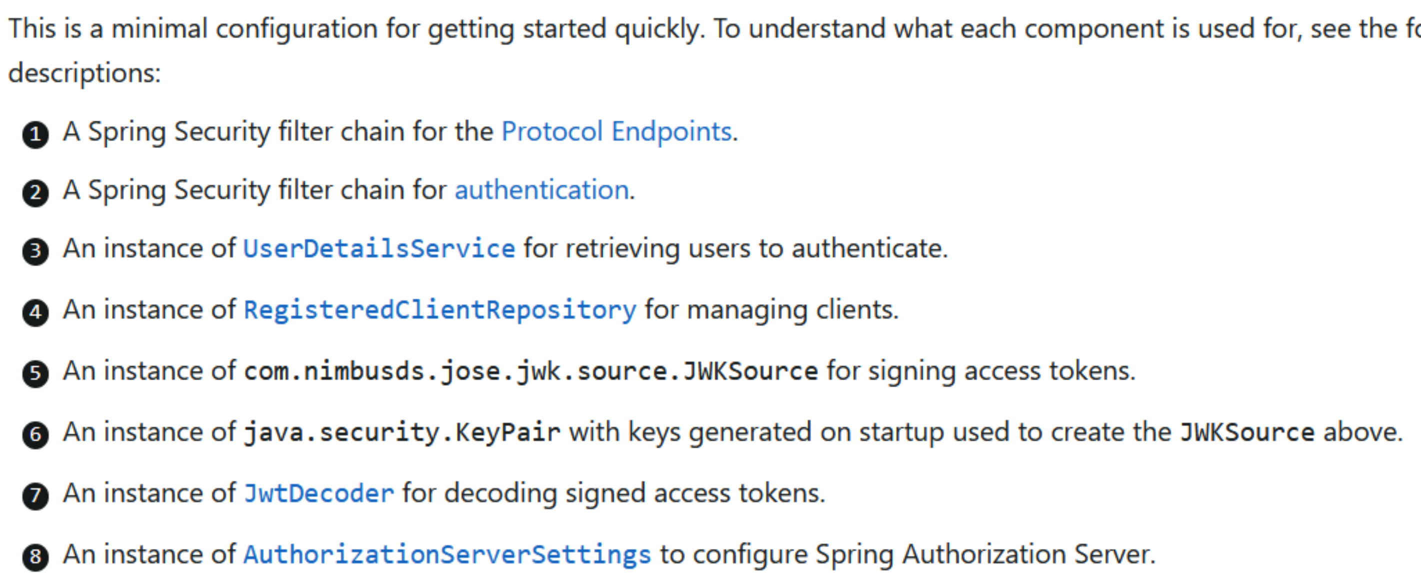
This is how you see the documentation

// We can see that it is written in the format of writting a authentication filter

```
@Configuration
@EnableWebSecurity
public class SecurityConfig {

        @Bean (1)
        @Order(1)
// Create a SecurityfilterChain
        public SecurityFilterChain authorizationServerSecurityFilterChain(HttpSecurity http)
                        throws Exception {
                OAuth2AuthorizationServerConfiguration.applyDefaultSecurity(http);
                http.getConfigurer(OAuth2AuthorizationServerConfigurer.class)
                        .oidc(Customizer.withDefaults());       // Enable OpenID Connect 1.0
// Call the oidc method of OAuth2AuthorizationServerConfigurer class
                http
                        // Redirect to the login page when not authenticated from the
                        // authorization endpoint
                        .exceptionHandling((exceptions) -> exceptions
                                .defaultAuthenticationEntryPointFor(
                                        new LoginUrlAuthenticationEntryPoint("/login"),
                                        new MediaTypeRequestMatcher(MediaType.TEXT_HTML)
                                )
                        )
//  When the login is not passed and the page is not released and otherwise all exceptions are
passed
                        // Accept access tokens for User Info and/or Client Registration
                        .oauth2ResourceServer((resourceServer) -> resourceServer
                                .jwt(Customizer.withDefaults()));

                return http.build();
        }


        @Bean (2)
        @Order(2)
        public SecurityFilterChain defaultSecurityFilterChain(HttpSecurity http)
                        throws Exception {
                http
```

```
                        .authorizeHttpRequests((authorize) -> authorize
                                .anyRequest().authenticated()
                        )
                        // Form login handles the redirect to the login page from the
                        // authorization server filter chain
                        .formLogin(Customizer.withDefaults());

                return http.build();
        }

        @Bean (3)
        public UserDetailsService userDetailsService() {
                UserDetails userDetails = User.withDefaultPasswordEncoder()
                                .username("user")
                                .password("password")
                                .roles("USER")
                                .build();

                return new InMemoryUserDetailsManager(userDetails);
        }

    //
        @Bean (4)
        public RegisteredClientRepository registeredClientRepository() {
// Write the oidcClient of type RegisteredClient and write all settings
                RegisteredClient oidcClient =
RegisteredClient.withId(UUID.randomUUID().toString())
                                .clientId("oidc-client")
                                .clientSecret("{noop}secret")

.clientAuthenticationMethod(ClientAuthenticationMethod.CLIENT_SECRET_BASIC)
// Attach the authorization method of    CLIENT_SECRET_BASIC
.authorizationGrantType(AuthorizationGrantType.AUTHORIZATION_CODE)
// Attach the authorization grant type of AUTHORIZATION_CODE
.authorizationGrantType(AuthorizationGrantType.REFRESH_TOKEN)
                                .redirectUri("http://127.0.0.1:8080/login/oauth2/code/oidc-client")
                                .postLogoutRedirectUri("http://127.0.0.1:8080/")
                                .scope(OidcScopes.OPENID)
                                .scope(OidcScopes.PROFILE)
// Attach all the
.clientSettings(ClientSettings.builder().requireAuthorizationConsent(true).build())
                                .build();

                return new InMemoryRegisteredClientRepository(oidcClient);
        }
```

```
// Create a bean having a class of SecurityContext type having keyPair, publickey , privatekey and
rsa key as its members
@Bean (5)
	public JWKSource<SecurityContext> jwkSource() {
		KeyPair keyPair = generateRsaKey();
		RSAPublicKey publicKey = (RSAPublicKey) keyPair.getPublic();
		RSAPrivateKey privateKey = (RSAPrivateKey) keyPair.getPrivate();
		RSAKey rsaKey = new RSAKey.Builder(publicKey)
				.privateKey(privateKey)
				.keyID(UUID.randomUUID().toString())
				.build();
		JWKSet jwkSet = new JWKSet(rsaKey);
		return new ImmutableJWKSet<>(jwkSet);
	}
//  generateRsaKey() is called  in the above keyPair
	private static KeyPair generateRsaKey() { (6)
		KeyPair keyPair;
		try {
			KeyPairGenerator keyPairGenerator =
KeyPairGenerator.getInstance("RSA");
			keyPairGenerator.initialize(2048);
			keyPair = keyPairGenerator.generateKeyPair();
		}
		catch (Exception ex) {
			throw new IllegalStateException(ex);
		}
		return keyPair;
	}

	@Bean (7)
	public JwtDecoder jwtDecoder(JWKSource<SecurityContext> jwkSource) {
		return OAuth2AuthorizationServerConfiguration.jwtDecoder(jwkSource);
	}

	@Bean (8)
	public AuthorizationServerSettings authorizationServerSettings() {
		return AuthorizationServerSettings.builder().build();
	}

}
```

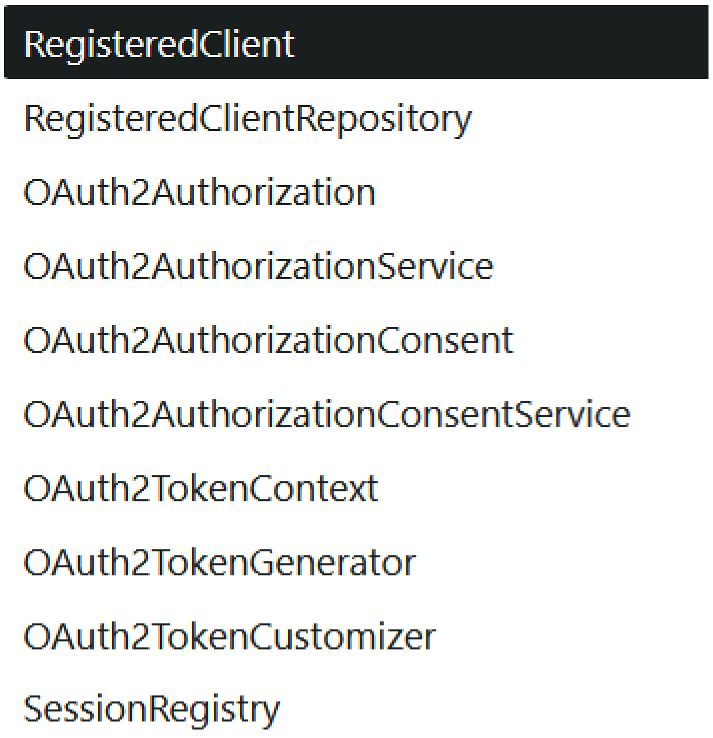


This is a minimal configuration for getting started quickly. To understand what each component is used for, see the fo
descriptions:

1. A Spring Security filter chain for the Protocol Endpoints.
2. A Spring Security filter chain for authentication.
3. An instance of `UserDetailsService` for retrieving users to authenticate.
4. An instance of `RegisteredClientRepository` for managing clients.
5. An instance of `com.nimbusds.jose.jwk.source.JWKSource` for signing access tokens.
6. An instance of `java.security.KeyPair` with keys generated on startup used to create the `JWKSource` above.
7. An instance of `JwtDecoder` for decoding signed access tokens.
8. An instance of `AuthorizationServerSettings` to configure Spring Authorization Server.

Try to fit this instances in various parts of application - Whenever you find something similar , write the sentence

Identify the core components or key players - entities or functions that have been frequently being used in the entire codebase


RegisteredClient

RegisteredClientRepository

OAuth2Authorization

OAuth2AuthorizationService

OAuth2AuthorizationConsent

OAuth2AuthorizationConsentService

OAuth2TokenContext

OAuth2TokenGenerator

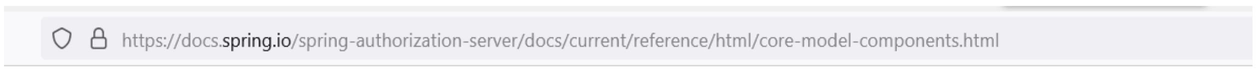
OAuth2TokenCustomizer

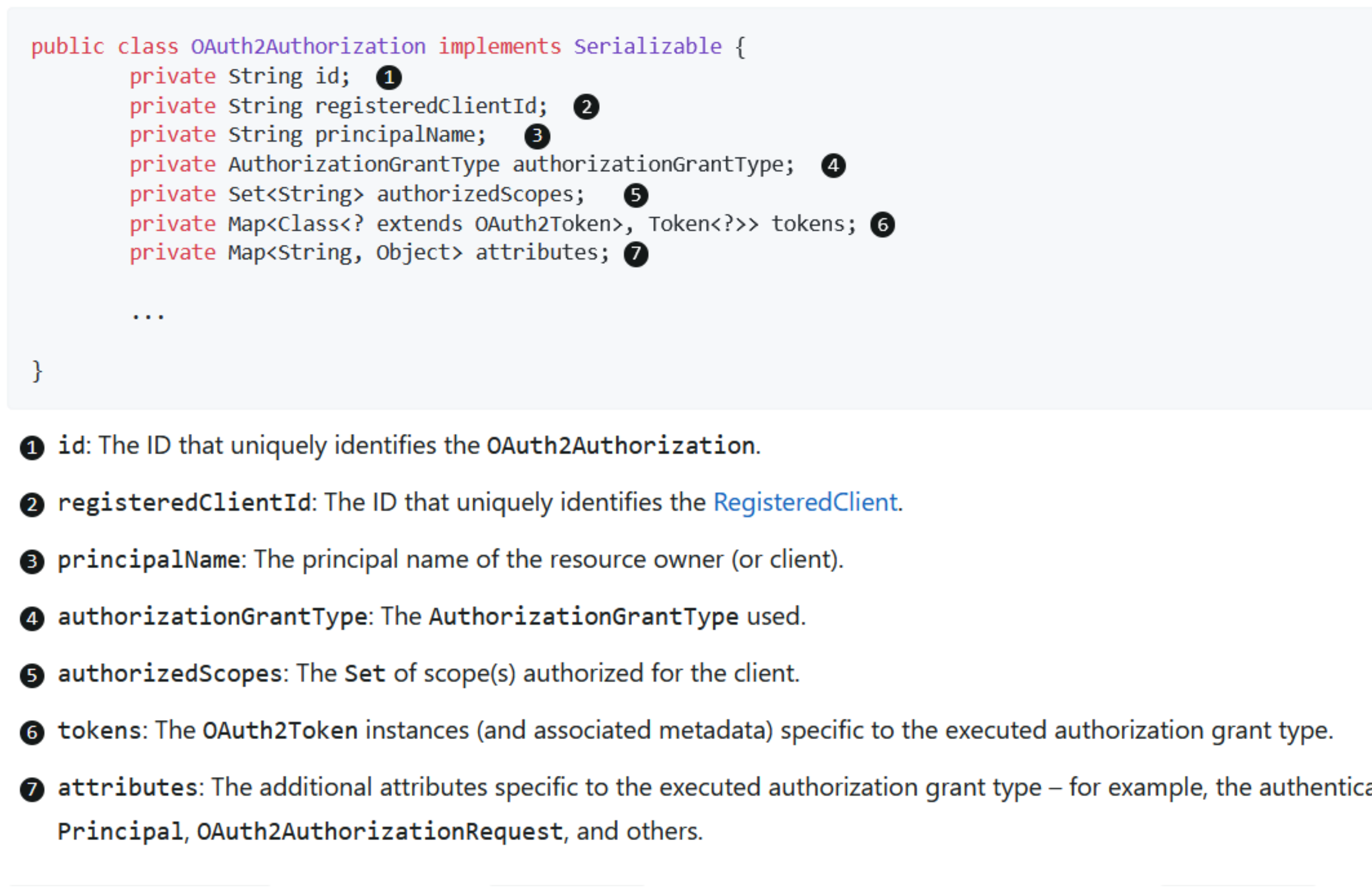
SessionRegistry

Sorted by decreasing order of frequency of usage

If we write all the methods first one would have more methods used , followed by second one and then third one

Macro version - code written above is reused in this classes and is handled as functionality.


https://docs.spring.io/spring-authorization-server/docs/current/reference/html/core-model-components.html



```java
public class OAuth2Authorization implements Serializable {
        private String id;  ❶
        private String registeredClientId;  ❷
        private String principalName;   ❸
        private AuthorizationGrantType authorizationGrantType;  ❹
        private Set<String> authorizedScopes;   ❺
        private Map<Class<? extends OAuth2Token>, Token<?>> tokens; ❻
        private Map<String, Object> attributes; ❼

        ...

}
```

❶ `id`: The ID that uniquely identifies the `OAuth2Authorization`.

❷ `registeredClientId`: The ID that uniquely identifies the RegisteredClient.

❸ `principalName`: The principal name of the resource owner (or client).

❹ `authorizationGrantType`: The `AuthorizationGrantType` used.

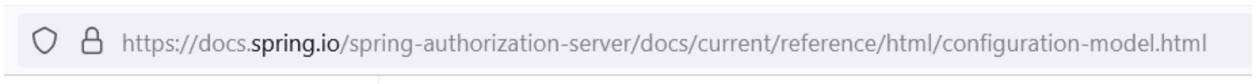
❺ `authorizedScopes`: The `Set` of scope(s) authorized for the client.

❻ `tokens`: The `OAuth2Token` instances (and associated metadata) specific to the executed authorization grant type.

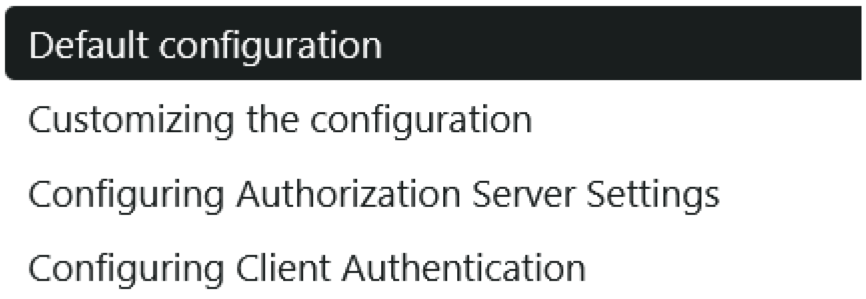
❼ `attributes`: The additional attributes specific to the executed authorization grant type – for example, the authentica
`Principal`, `OAuth2AuthorizationRequest`, and others.

This is where you can find the secondary usage of the variables

This is how you think when you find the root cause of the search



https://docs.spring.io/spring-authorization-server/docs/current/reference/html/configuration-model.html

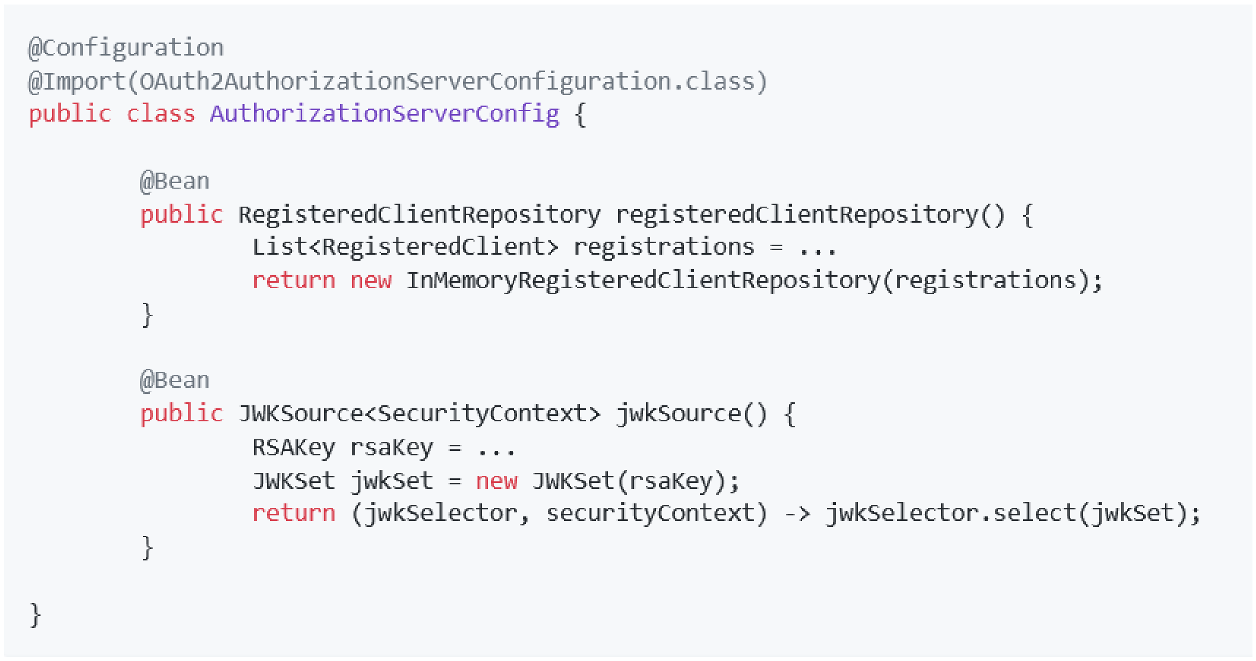


Default configuration

Customizing the configuration

Configuring Authorization Server Settings

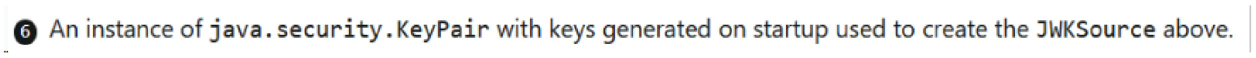
Configuring Client Authentication



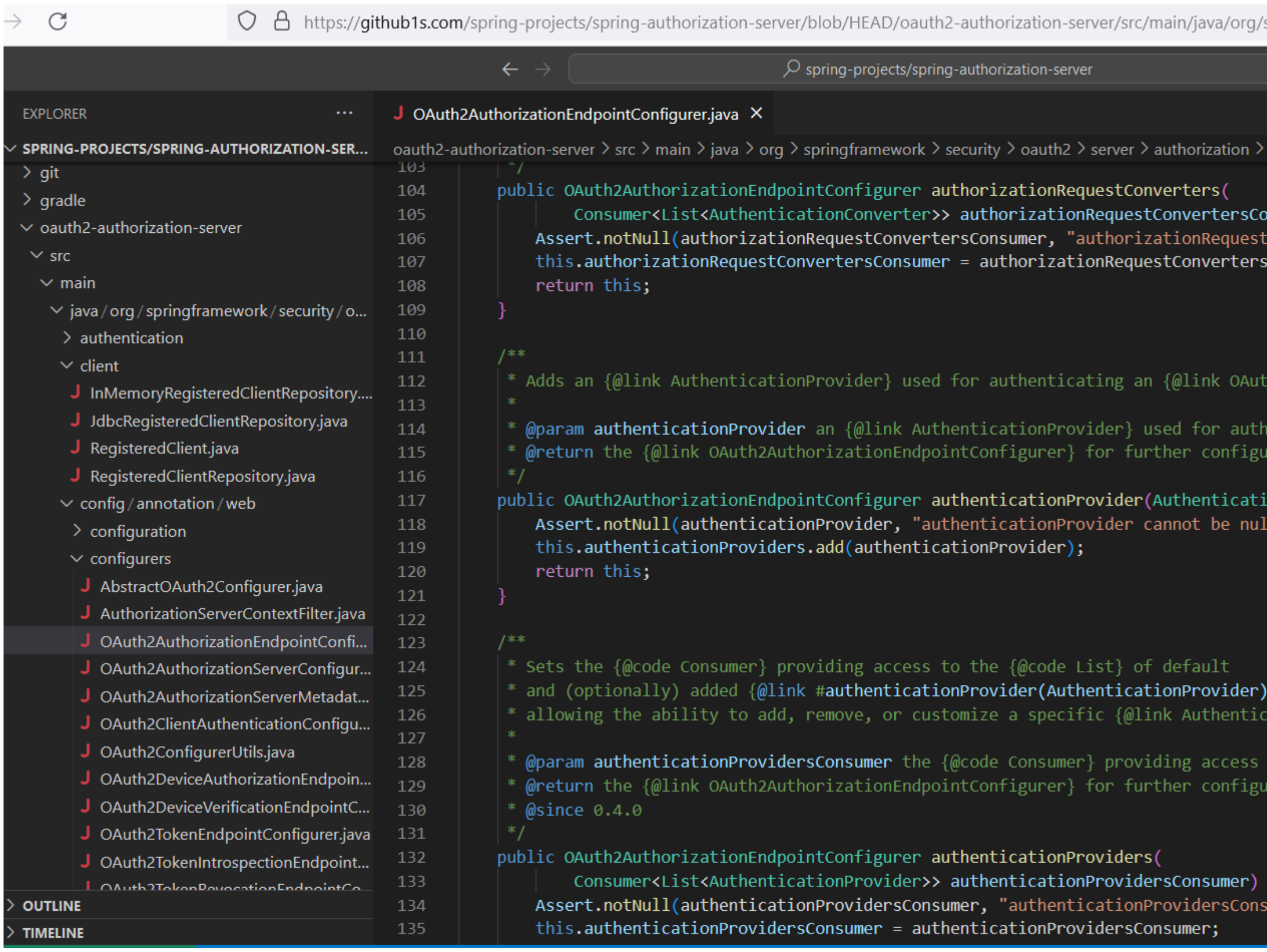
```java
@Configuration
@Import(OAuth2AuthorizationServerConfiguration.class)
public class AuthorizationServerConfig {

        @Bean
        public RegisteredClientRepository registeredClientRepository() {
                List<RegisteredClient> registrations = ...
                return new InMemoryRegisteredClientRepository(registrations);
        }

        @Bean
        public JWKSource<SecurityContext> jwkSource() {
                RSAKey rsaKey = ...
                JWKSet jwkSet = new JWKSet(rsaKey);
                return (jwkSelector, securityContext) -> jwkSelector.select(jwkSet);
        }

}
```

Whenever you see this code or any other code found in the same url , understand that it is some type of configuration that has been done

We can see this format has been used in the above having rsa key and jwkSet.
This is the 6th point in the second part.



6 An instance of `java.security.KeyPair` with keys generated on startup used to create the `JWKSource` above.

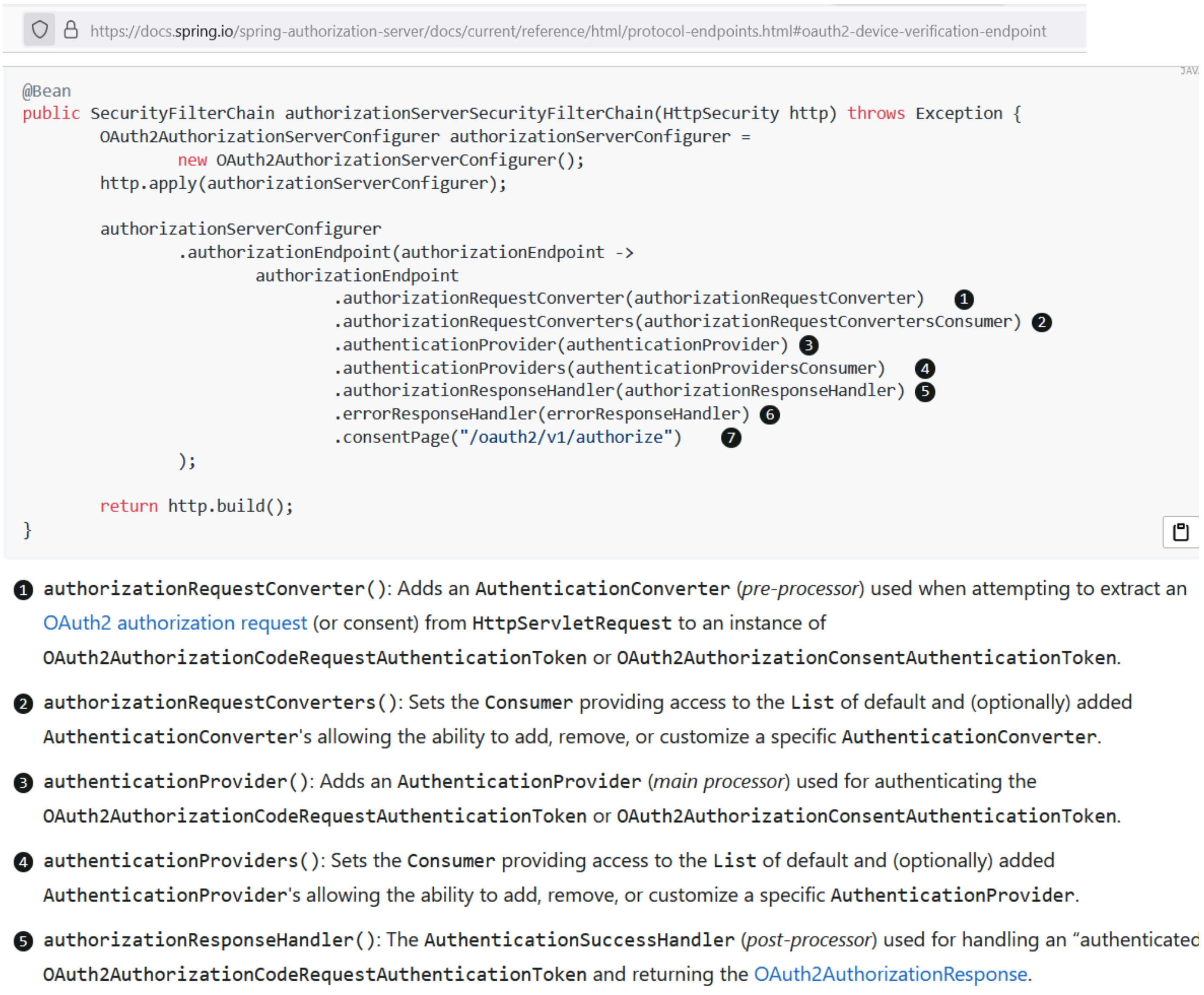

EXPLORER
SPRING-PROJECTS/SPRING-AUTHORIZATION-SER...
git
gradle
oauth2-authorization-server
src
main
java / org / springframework / security / o...
authentication
client
InMemoryRegisteredClientRepository....
JdbcRegisteredClientRepository.java
RegisteredClient.java
RegisteredClientRepository.java
config / annotation / web
configuration
configurers
AbstractOAuth2Configurer.java
AuthorizationServerContextFilter.java
OAuth2AuthorizationEndpointConfi...
OAuth2AuthorizationServerConfigur...
OAuth2AuthorizationServerMetadat...
OAuth2ClientAuthenticationConfigu...
OAuth2ConfigurerUtils.java
OAuth2DeviceAuthorizationEndpoin...
OAuth2DeviceVerificationEndpointC...
OAuth2TokenEndpointConfigurer.java
OAuth2TokenIntrospectionEndpoint...
OUTLINE
TIMELINE
spring-projects/spring-authorization-server
OAuth2AuthorizationEndpointConfigurer.java
oauth2-authorization-server > src > main > java > org > springframework > security > oauth2 > server > authorization >

https://github1s.com/spring-projects/spring-authorization-server/blob/HEAD/oauth2-authorization-server/src/main/java/org/s

```java
	 */
	public OAuth2AuthorizationEndpointConfigurer authorizationRequestConverters(
			Consumer<List<AuthenticationConverter>> authorizationRequestConvertersCo
		Assert.notNull(authorizationRequestConvertersConsumer, "authorizationRequest
		this.authorizationRequestConvertersConsumer = authorizationRequestConverters
		return this;
	}

	/**
	 * Adds an {@link AuthenticationProvider} used for authenticating an {@link OAut
	 *
	 * @param authenticationProvider an {@link AuthenticationProvider} used for auth
	 * @return the {@link OAuth2AuthorizationEndpointConfigurer} for further configu
	 */
	public OAuth2AuthorizationEndpointConfigurer authenticationProvider(Authenticati
		Assert.notNull(authenticationProvider, "authenticationProvider cannot be nul
		this.authenticationProviders.add(authenticationProvider);
		return this;
	}

	/**
	 * Sets the {@code Consumer} providing access to the {@code List} of default
	 * and (optionally) added {@link #authenticationProvider(AuthenticationProvider)
	 * allowing the ability to add, remove, or customize a specific {@link Authentic
	 *
	 * @param authenticationProvidersConsumer the {@code Consumer} providing access
	 * @return the {@link OAuth2AuthorizationEndpointConfigurer} for further configu
	 * @since 0.4.0
	 */
	public OAuth2AuthorizationEndpointConfigurer authenticationProviders(
			Consumer<List<AuthenticationProvider>> authenticationProvidersConsumer)
		Assert.notNull(authenticationProvidersConsumer, "authenticationProvidersCons
		this.authenticationProvidersConsumer = authenticationProvidersConsumer;
```

Method Name is given in yellow
Method Type is given in green
What has been passed is given in blue with curly brackets
The type of variable to be passed is given in green

Trying to find out in the entire codebase - search the term in vscode and figure out all cases where it has been written


https://docs.spring.io/spring-authorization-server/docs/current/reference/html/protocol-endpoints.html#oauth2-device-verification-endpoint

```java
@Bean
public SecurityFilterChain authorizationServerSecurityFilterChain(HttpSecurity http) throws Exception {
	OAuth2AuthorizationServerConfigurer authorizationServerConfigurer =
		new OAuth2AuthorizationServerConfigurer();
	http.apply(authorizationServerConfigurer);

	authorizationServerConfigurer
		.authorizationEndpoint(authorizationEndpoint ->
			authorizationEndpoint
				.authorizationRequestConverter(authorizationRequestConverter)   ❶
				.authorizationRequestConverters(authorizationRequestConvertersConsumer) ❷
				.authenticationProvider(authenticationProvider) ❸
				.authenticationProviders(authenticationProvidersConsumer)   ❹
				.authorizationResponseHandler(authorizationResponseHandler) ❺
				.errorResponseHandler(errorResponseHandler) ❻
				.consentPage("/oauth2/v1/authorize")    ❼
		);

	return http.build();
}
```

❶ `authorizationRequestConverter()`: Adds an `AuthenticationConverter` (*pre-processor*) used when attempting to extract an OAuth2 authorization request (or consent) from `HttpServletRequest` to an instance of `OAuth2AuthorizationCodeRequestAuthenticationToken` or `OAuth2AuthorizationConsentAuthenticationToken`.

❷ `authorizationRequestConverters()`: Sets the `Consumer` providing access to the `List` of default and (optionally) added `AuthenticationConverter`'s allowing the ability to add, remove, or customize a specific `AuthenticationConverter`.

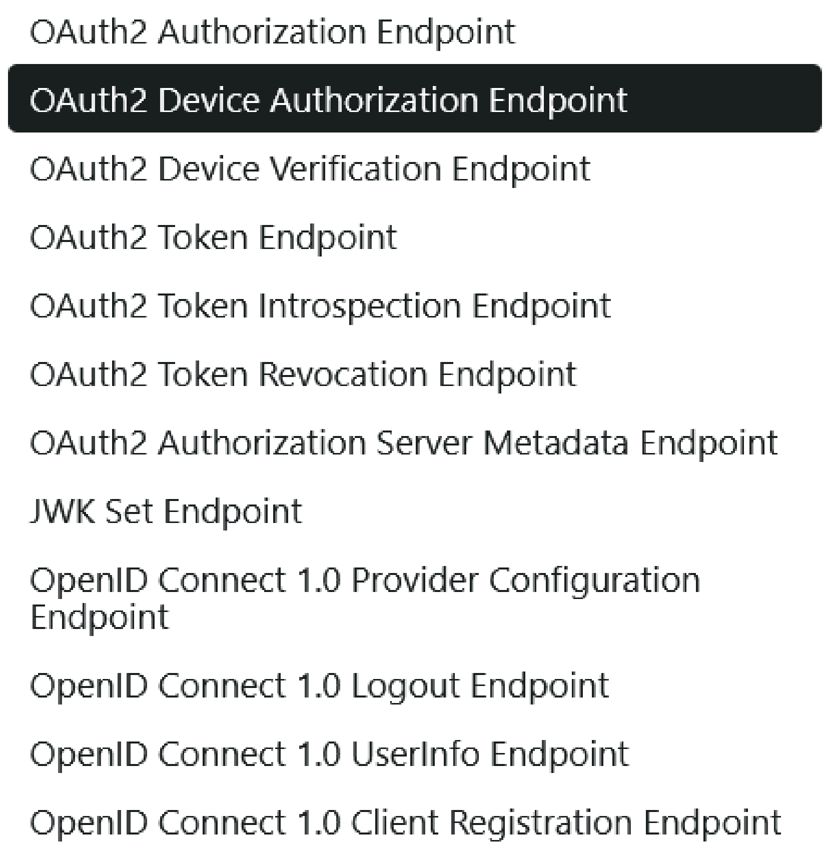
❸ `authenticationProvider()`: Adds an `AuthenticationProvider` (*main processor*) used for authenticating the `OAuth2AuthorizationCodeRequestAuthenticationToken` or `OAuth2AuthorizationConsentAuthenticationToken`.

❹ `authenticationProviders()`: Sets the `Consumer` providing access to the `List` of default and (optionally) added `AuthenticationProvider`'s allowing the ability to add, remove, or customize a specific `AuthenticationProvider`.

❺ `authorizationResponseHandler()`: The `AuthenticationSuccessHandler` (*post-processor*) used for handling an "authenticated `OAuth2AuthorizationCodeRequestAuthenticationToken` and returning the OAuth2AuthorizationResponse.

This tells us how to use particular function

This particular type has been explained in the documentation - this will be the code that we will be mainly fixing - and it will use all classes that has been explained above.



OAuth2 Authorization Endpoint

OAuth2 Device Authorization Endpoint

OAuth2 Device Verification Endpoint

OAuth2 Token Endpoint

OAuth2 Token Introspection Endpoint

OAuth2 Token Revocation Endpoint

OAuth2 Authorization Server Metadata Endpoint

JWK Set Endpoint

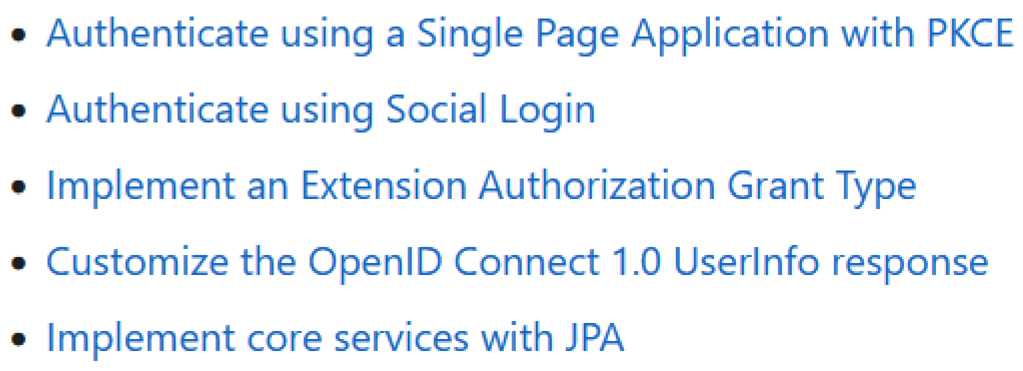
OpenID Connect 1.0 Provider Configuration Endpoint

OpenID Connect 1.0 Logout Endpoint

OpenID Connect 1.0 UserInfo Endpoint

OpenID Connect 1.0 Client Registration Endpoint

All these functions are going to be used in code - thats why we should remember it by name.



- Authenticate using a Single Page Application with PKCE
- Authenticate using Social Login
- Implement an Extension Authorization Grant Type
- Customize the OpenID Connect 1.0 UserInfo response
- Implement core services with JPA

This is how you write the name of issues using such verbs.

How-to: Implement core services with JPA

Define the data model

- Client Schema
- Authorization Schema
- Authorization Consent Schema

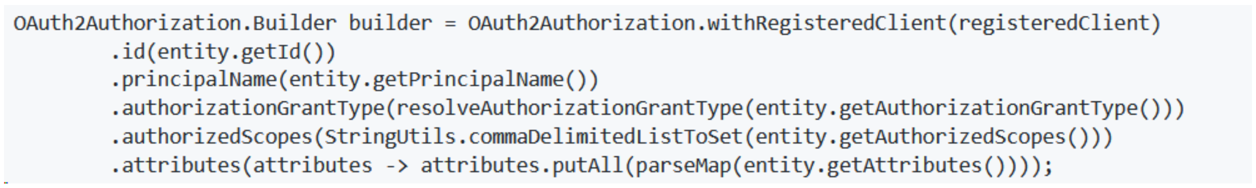
Create JPA entities

- Client Entity
- Authorization Entity
- Authorization Consent Entity

Create Spring Data repositories

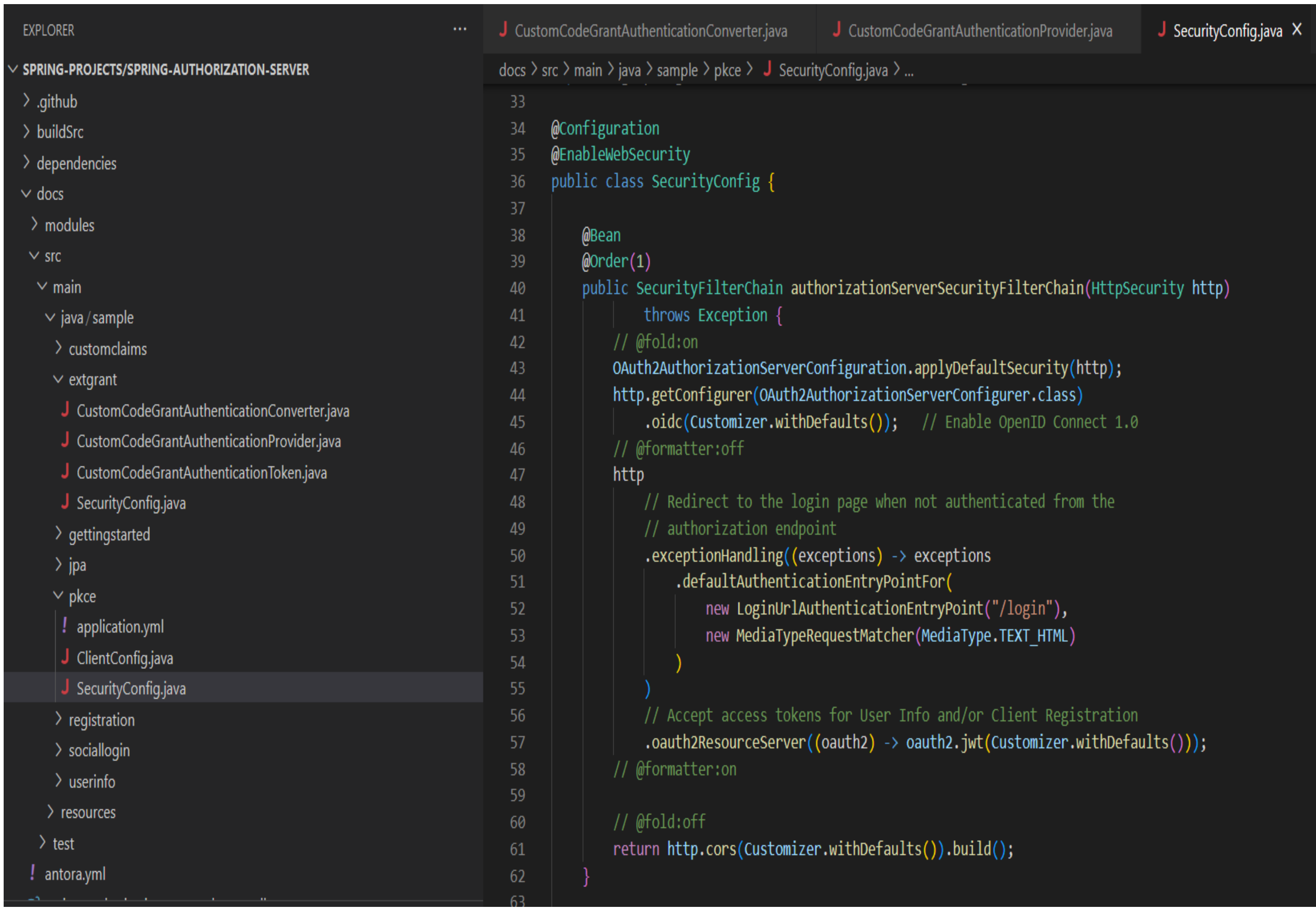
- Client Repository
- Authorization Repository
- Authorization Consent Repository

Implement core services

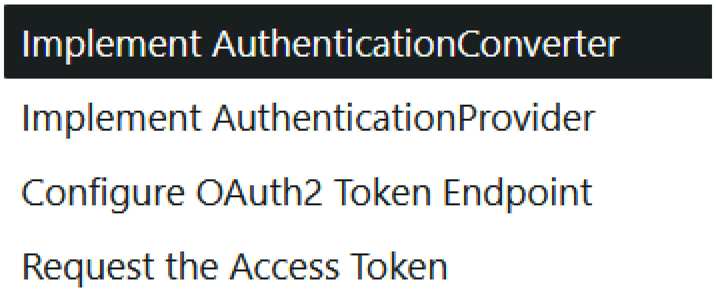


```
OAuth2Authorization.Builder builder = OAuth2Authorization.withRegisteredClient(registeredClient)
        .id(entity.getId())
        .principalName(entity.getPrincipalName())
        .authorizationGrantType(resolveAuthorizationGrantType(entity.getAuthorizationGrantType()))
        .authorizedScopes(StringUtils.commaDelimitedListToSet(entity.getAuthorizedScopes()))
        .attributes(attributes -> attributes.putAll(parseMap(entity.getAttributes())));
```

An instance of RegisteredClient being intialized with parameters
This comes from 4 of the great 8 points written


EXPLORER
SPRING-PROJECTS/SPRING-AUTHORIZATION-SERVER
.github
buildSrc
dependencies
docs
modules
src
main
java / sample
customclaims
extgrant
CustomCodeGrantAuthenticationConverter.java
CustomCodeGrantAuthenticationProvider.java
CustomCodeGrantAuthenticationToken.java
SecurityConfig.java
gettingstarted
jpa
pkce
application.yml
ClientConfig.java
SecurityConfig.java
registration
sociallogin
userinfo
resources
test
antora.yml
CustomCodeGrantAuthenticationConverter.java
CustomCodeGrantAuthenticationProvider.java
SecurityConfig.java
docs > src > main > java > sample > pkce > SecurityConfig.java > ...

```java

@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Bean
    @Order(1)
    public SecurityFilterChain authorizationServerSecurityFilterChain(HttpSecurity http)
            throws Exception {
        // @fold:on
        OAuth2AuthorizationServerConfiguration.applyDefaultSecurity(http);
        http.getConfigurer(OAuth2AuthorizationServerConfigurer.class)
            .oidc(Customizer.withDefaults());   // Enable OpenID Connect 1.0
        // @formatter:off
        http
            // Redirect to the login page when not authenticated from the
            // authorization endpoint
            .exceptionHandling((exceptions) -> exceptions
                .defaultAuthenticationEntryPointFor(
                    new LoginUrlAuthenticationEntryPoint("/login"),
                    new MediaTypeRequestMatcher(MediaType.TEXT_HTML)
                )
            )
            // Accept access tokens for User Info and/or Client Registration
            .oauth2ResourceServer((oauth2) -> oauth2.jwt(Customizer.withDefaults()));
        // @formatter:on

        // @fold:off
        return http.cors(Customizer.withDefaults()).build();
    }
```



Implement AuthenticationConverter

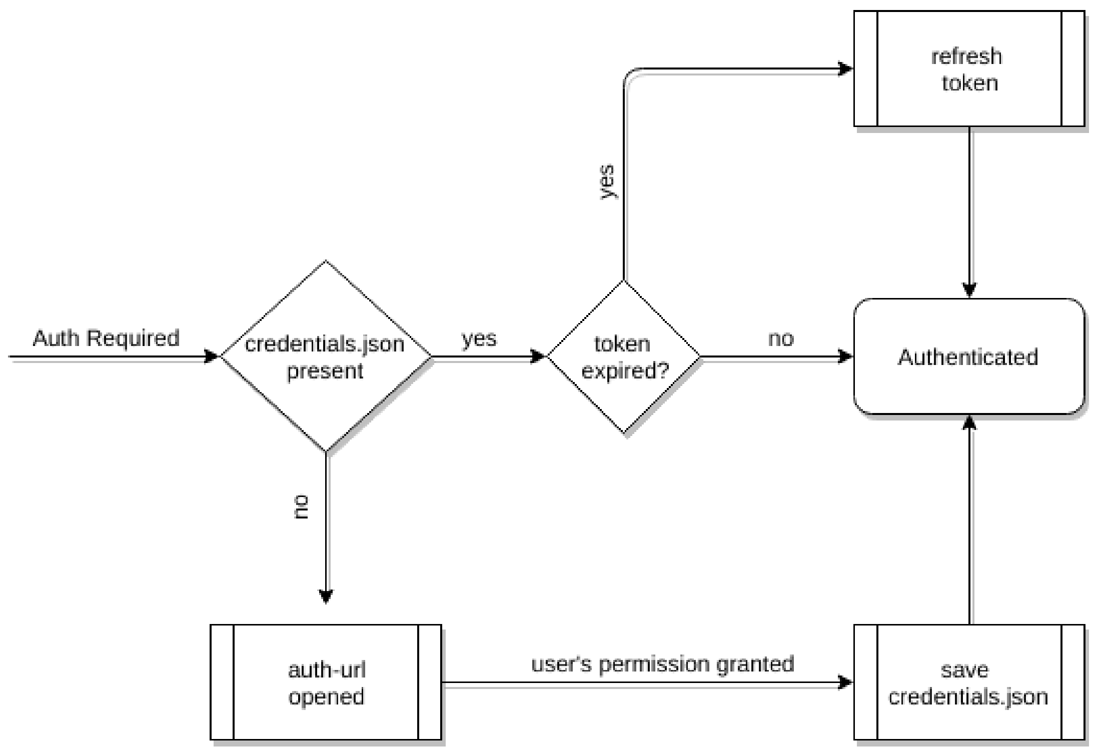
Implement AuthenticationProvider

Configure OAuth2 Token Endpoint

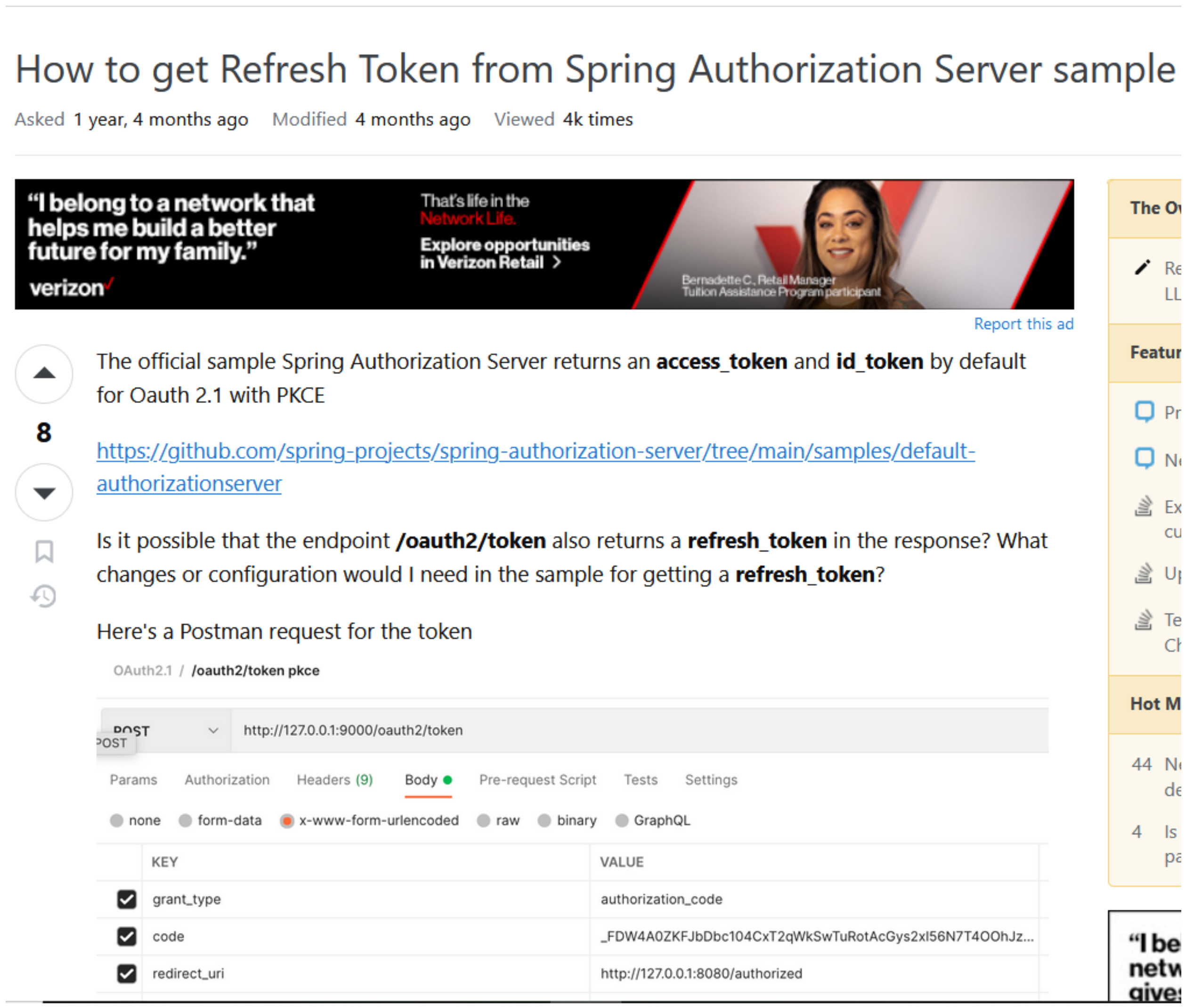
Request the Access Token

This are the files with name to be modified , with the order so that we can add 1 functionality here .
How to see files which are in the same folder

Auth-Flow



Sample workflow not exact workflow to explain progress



How to get Refresh Token from Spring Authorization Server sample

Asked 1 year, 4 months ago Modified 4 months ago Viewed 4k times

"I belong to a network that helps me build a better future for my family."
verizon
That's life in the Network Life.
Explore opportunities in Verizon Retail >
Bernadette C., Retail Manager Tuition Assistance Program participant

Report this ad

8

The official sample Spring Authorization Server returns an **access_token** and **id_token** by default for Oauth 2.1 with PKCE

https://github.com/spring-projects/spring-authorization-server/tree/main/samples/default-authorizationserver

Is it possible that the endpoint **/oauth2/token** also returns a **refresh_token** in the response? What changes or configuration would I need in the sample for getting a **refresh_token**?

Here's a Postman request for the token

OAuth2.1 / **/oauth2/token pkce**

POST http://127.0.0.1:9000/oauth2/token

Params Authorization Headers (9) Body ● Pre-request Script Tests Settings

none form-data ● x-www-form-urlencoded raw binary GraphQL

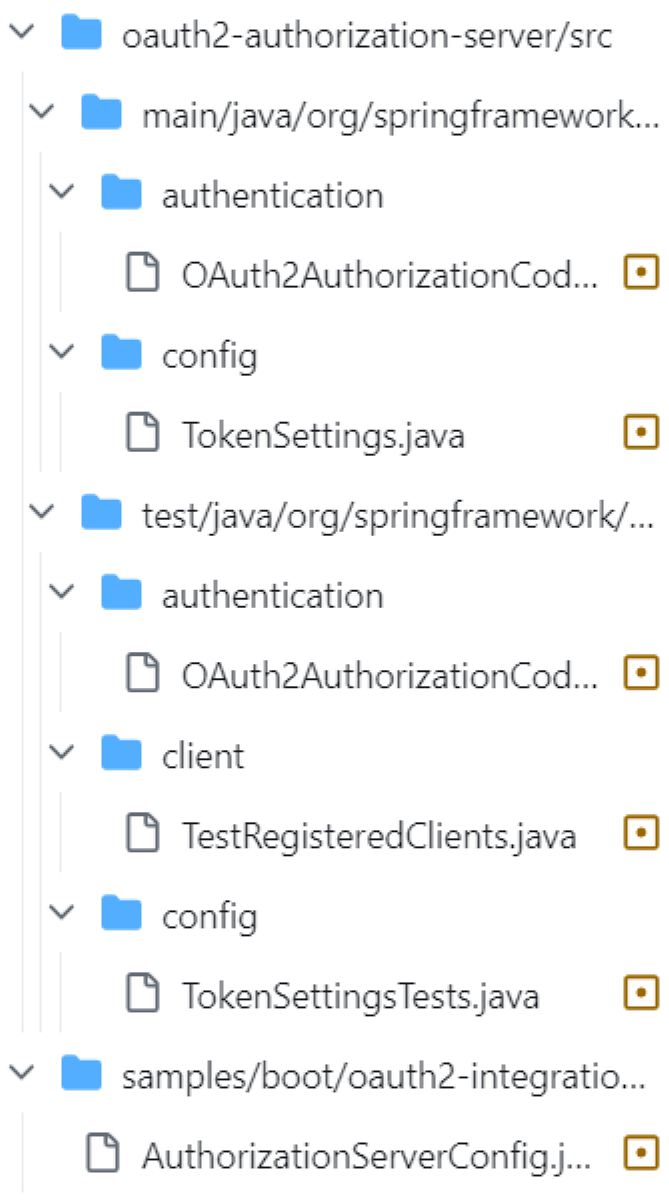
	KEY	VALUE
✓	grant_type	authorization_code
✓	code	_FDW4A0ZKFJbDbc104CxT2qWkSwTuRotAcGys2xl56N7T4OOhJz...
✓	redirect_uri	http://127.0.0.1:8080/authorized

Now we can understand , how this message and implementation really works.

This is StackOverflow .. Here you will find the details about this particular stage of workflow of things.

Also will find how much of the particular state in the workflow connects with each other.

Find the files required to it


oauth2-authorization-server/src
main/java/org/springframework...
authentication
OAuth2AuthorizationCod...
config
TokenSettings.java
test/java/org/springframework/...
authentication
OAuth2AuthorizationCod...
client
TestRegisteredClients.java
config
TokenSettingsTests.java
samples/boot/oauth2-integratio...
AuthorizationServerConfig.j...

These
This is the directory structure

The Bug fixes
oauth2-authorization-server/src/main/java/org/springframework/security/oauth2/server/authorization/authentication/OAuth2AuthorizationCodeAuthenticationProvider.java

```
    if (isConfiguredForRefreshToken(registeredClient)) {
    if (registeredClient.getAuthorizationGrantTypes()
            .contains(AuthorizationGrantType.REFRESH_TOKEN)) {
```

First statement shows registeredClient as a model , isConfiguredForRefreshToken and getAuthorizationGrantType as methods and contains as an utility method

We find that isConfiguredForRefreshToken being configured in another file and we keep the second conditon

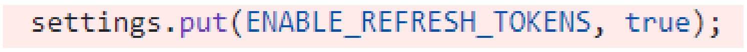
```
private boolean isConfiguredForRefreshToken(RegisteredClient registeredClient) {
    return registeredClient.getTokenSettings().enableRefreshTokens() &&
            registeredClient.getAuthorizationGrantTypes().contains(AuthorizationGrantType.REFRESH_TOKEN)
}
```

oauth2-authorization-server/src/main/java/org/springframework/security/oauth2/server/authorization/config/TokenSettings.java

```
public static final String ENABLE_REFRESH_TOKENS = TOKEN_SETTING_BASE.concat("enable-refresh-tokens");
```

We define the string ENABLE_REFRESH_TOKENS and use it in different functions
First we have overloading of constructors
Then in second function we return the settings while in the first function we return if they are enabled or not.

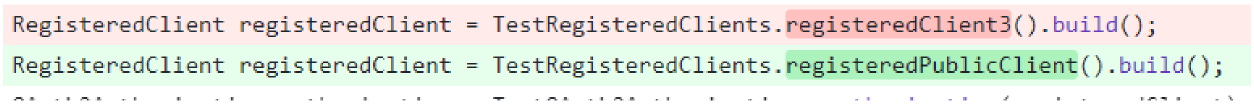
```
/**
 * Returns {@code true} if refresh tokens are enabled. The default is {@code true}.
 *
 * @return {@code true} if refresh tokens are enabled, {@code false} otherwise
 */
public boolean enableRefreshTokens() {
    return setting(ENABLE_REFRESH_TOKENS);
}

/**
 * Set to {@code true} to enable refresh tokens.
 *
 * @param enableRefreshTokens {@code true} to enable refresh tokens, {@code false} otherwise
 * @return the {@link TokenSettings}
 */
public TokenSettings enableRefreshTokens(boolean enableRefreshTokens) {
    setting(ENABLE_REFRESH_TOKENS, enableRefreshTokens);
    return this;
}
```



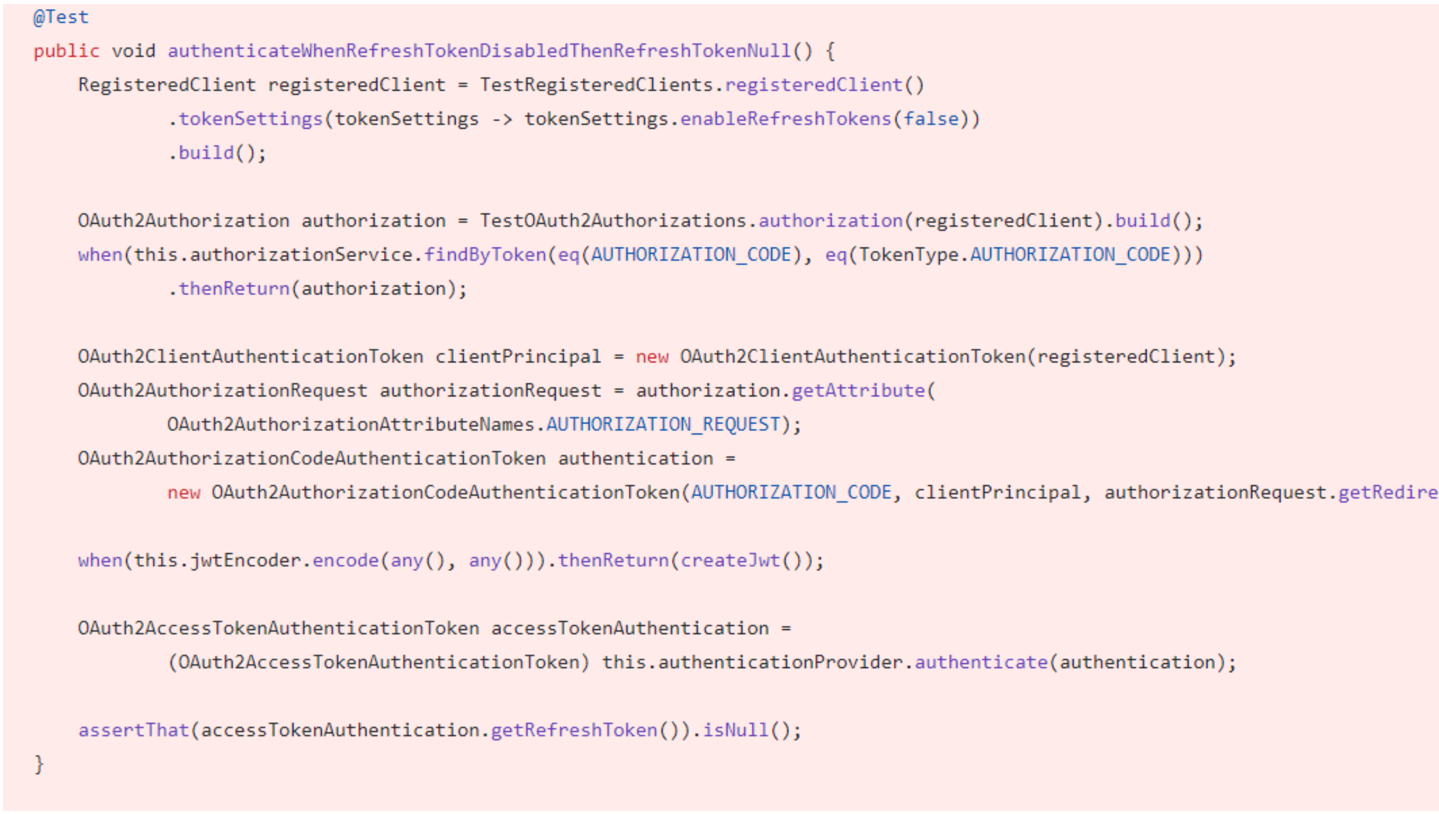
```java
settings.put(ENABLE_REFRESH_TOKENS, true);
```

He had pushed the variables to the settings variable.

oauth2-authorization-server/src/test/java/org/springframework/security/oauth2/server/authorization/authentication/OAuth2AuthorizationCodeAuthenticationProviderTests.java



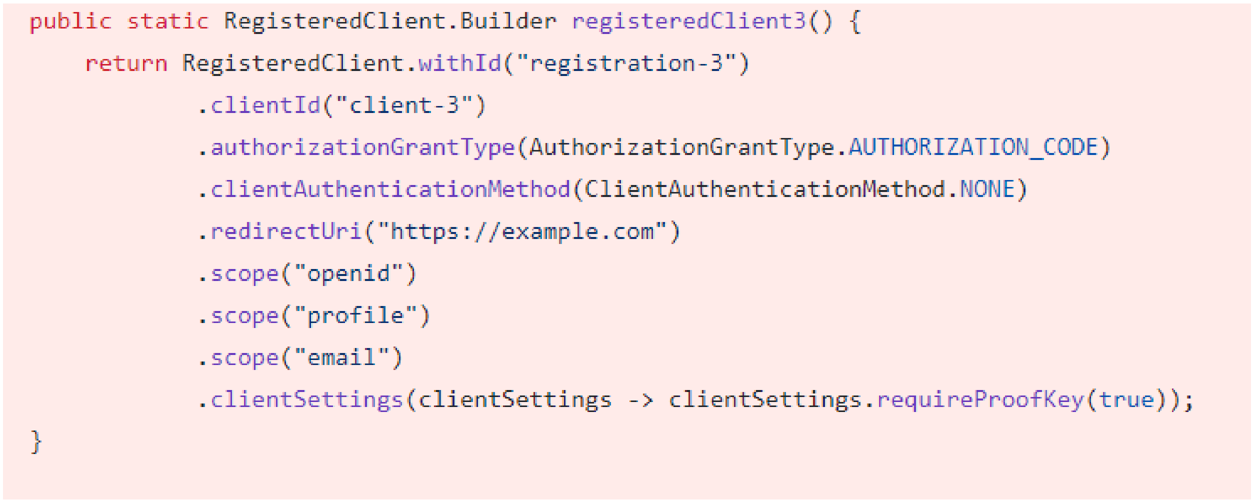
```java
RegisteredClient registeredClient = TestRegisteredClients.registeredClient3().build();
RegisteredClient registeredClient = TestRegisteredClients.registeredPublicClient().build();
```



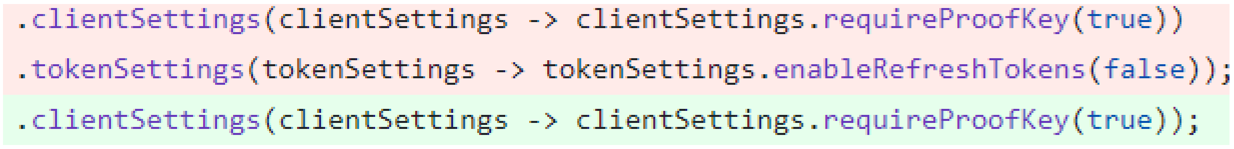
```java
@Test
public void authenticateWhenRefreshTokenDisabledThenRefreshTokenNull() {
    RegisteredClient registeredClient = TestRegisteredClients.registeredClient()
            .tokenSettings(tokenSettings -> tokenSettings.enableRefreshTokens(false))
            .build();

    OAuth2Authorization authorization = TestOAuth2Authorizations.authorization(registeredClient).build();
    when(this.authorizationService.findByToken(eq(AUTHORIZATION_CODE), eq(TokenType.AUTHORIZATION_CODE)))
            .thenReturn(authorization);

    OAuth2ClientAuthenticationToken clientPrincipal = new OAuth2ClientAuthenticationToken(registeredClient);
    OAuth2AuthorizationRequest authorizationRequest = authorization.getAttribute(
            OAuth2AuthorizationAttributeNames.AUTHORIZATION_REQUEST);
    OAuth2AuthorizationCodeAuthenticationToken authentication =
            new OAuth2AuthorizationCodeAuthenticationToken(AUTHORIZATION_CODE, clientPrincipal, authorizationRequest.getRedire

    when(this.jwtEncoder.encode(any(), any())).thenReturn(createJwt());

    OAuth2AccessTokenAuthenticationToken accessTokenAuthentication =
            (OAuth2AccessTokenAuthenticationToken) this.authenticationProvider.authenticate(authentication);

    assertThat(accessTokenAuthentication.getRefreshToken()).isNull();
}
```

This is a test class. The test name should be the what is happening in the test.
While all those functions and methods are important . they should be concentrated while understanding this particular scenario. This will help to breakdown workflow.

oauth2-authorization-server/src/test/java/org/springframework/security/oauth2/server/authorization/client/TestRegisteredClients.java



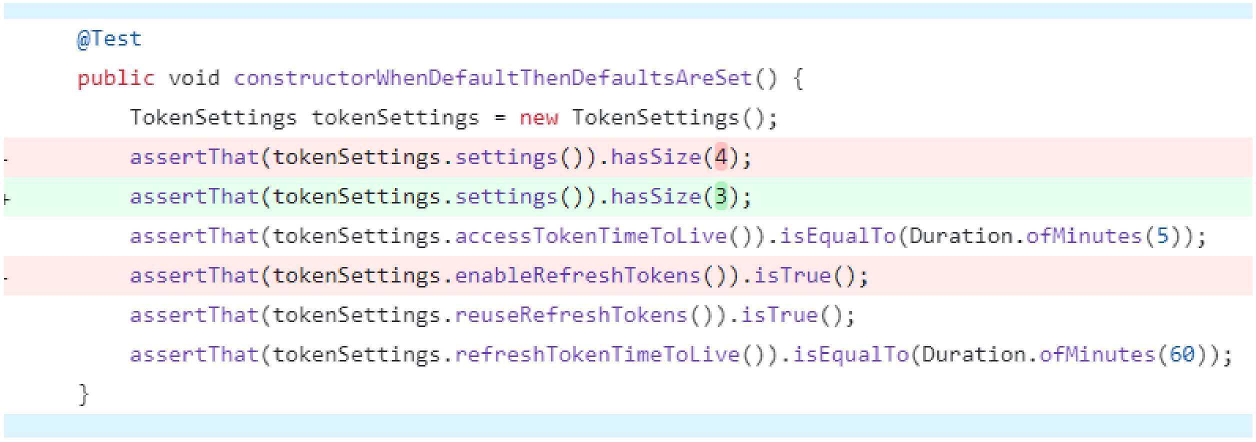
```
public static RegisteredClient.Builder registeredClient3() {
    return RegisteredClient.withId("registration-3")
            .clientId("client-3")
            .authorizationGrantType(AuthorizationGrantType.AUTHORIZATION_CODE)
            .clientAuthenticationMethod(ClientAuthenticationMethod.NONE)
            .redirectUri("https://example.com")
            .scope("openid")
            .scope("profile")
            .scope("email")
            .clientSettings(clientSettings -> clientSettings.requireProofKey(true));
}
```



```
.clientSettings(clientSettings -> clientSettings.requireProofKey(true))
.tokenSettings(tokenSettings -> tokenSettings.enableRefreshTokens(false));
.clientSettings(clientSettings -> clientSettings.requireProofKey(true));
```

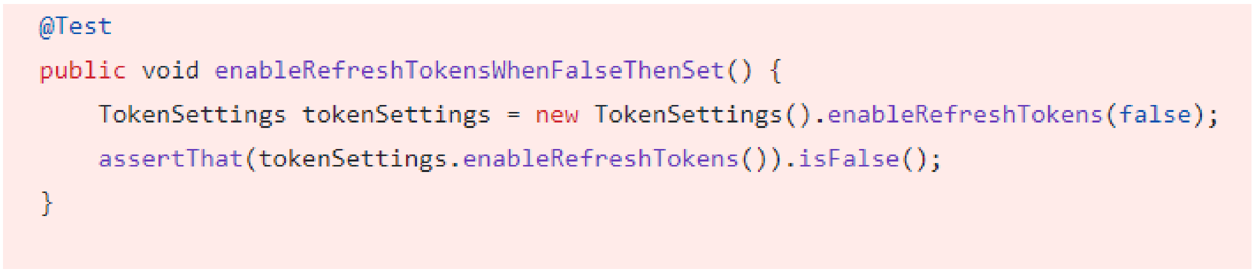
WHAT ????
tokenSettings RefreshToken Settings is getting falsed
Why??
When The client is getting new request , only the client side should be refreshed and not the entire token

oauth2-authorization-server/src/test/java/org/springframework/security/oauth2/server/authorization/config/TokenSettingsTests.java



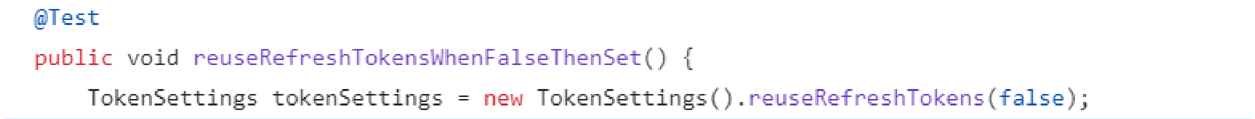
```
    @Test
    public void constructorWhenDefaultThenDefaultsAreSet() {
        TokenSettings tokenSettings = new TokenSettings();
-       assertThat(tokenSettings.settings()).hasSize(4);
+       assertThat(tokenSettings.settings()).hasSize(3);
        assertThat(tokenSettings.accessTokenTimeToLive()).isEqualTo(Duration.ofMinutes(5));
-       assertThat(tokenSettings.enableRefreshTokens()).isTrue();
        assertThat(tokenSettings.reuseRefreshTokens()).isTrue();
        assertThat(tokenSettings.refreshTokenTimeToLive()).isEqualTo(Duration.ofMinutes(60));
    }
```

This is how we use enableRefreshTokens() on tokenSettings ….

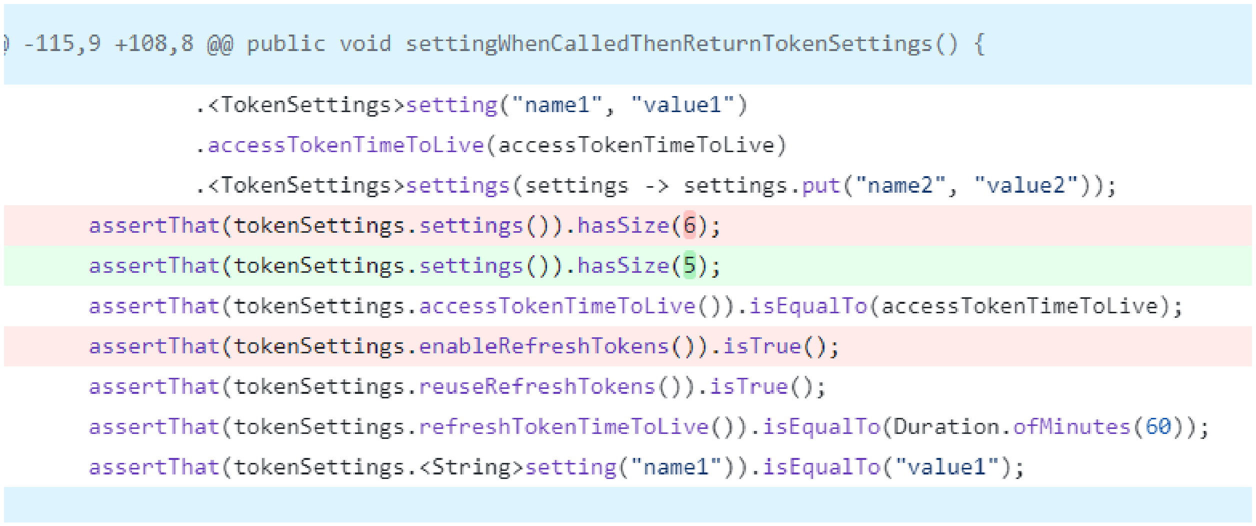


```java
@Test
public void enableRefreshTokensWhenFalseThenSet() {
    TokenSettings tokenSettings = new TokenSettings().enableRefreshTokens(false);
    assertThat(tokenSettings.enableRefreshTokens()).isFalse();
}
```

Using encapsulation here , enableTokenSettings takes two values with false as parameter and then not any parameter and we call one function and then in it call other function.



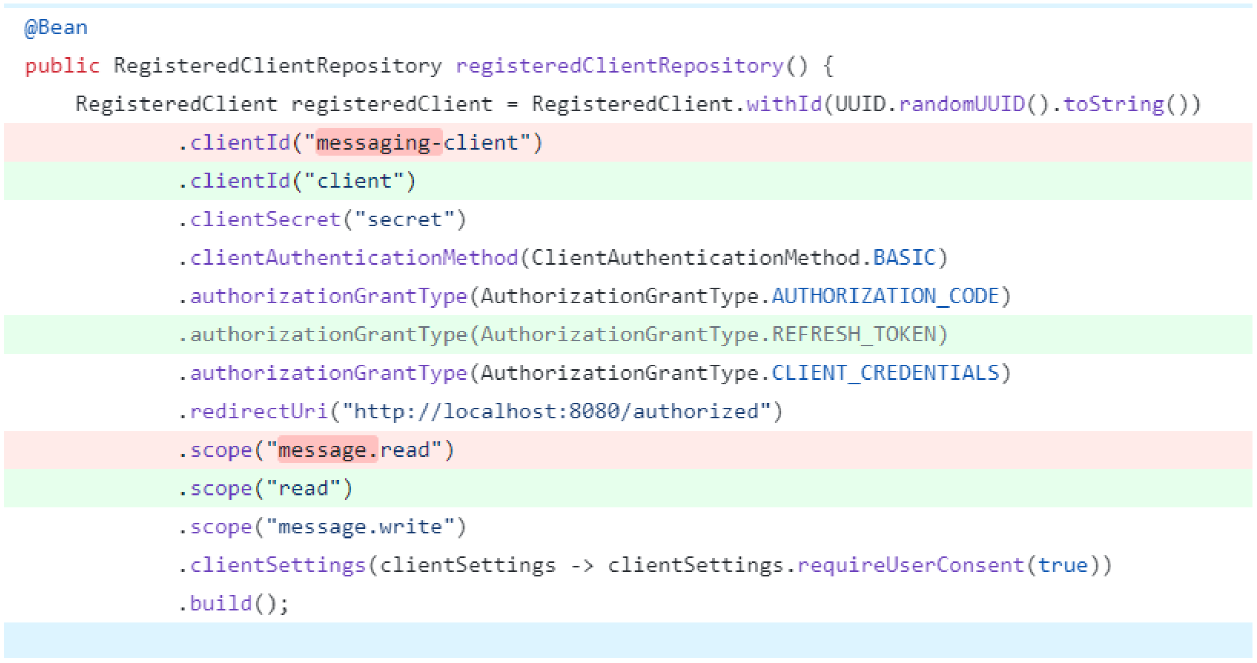
```java
@Test
public void reuseRefreshTokensWhenFalseThenSet() {
    TokenSettings tokenSettings = new TokenSettings().reuseRefreshTokens(false);
```



```java
 -115,9 +108,8 @@ public void settingWhenCalledThenReturnTokenSettings() {
                .<TokenSettings>setting("name1", "value1")
                .accessTokenTimeToLive(accessTokenTimeToLive)
                .<TokenSettings>settings(settings -> settings.put("name2", "value2"));
        assertThat(tokenSettings.settings()).hasSize(6);
        assertThat(tokenSettings.settings()).hasSize(5);
        assertThat(tokenSettings.accessTokenTimeToLive()).isEqualTo(accessTokenTimeToLive);
        assertThat(tokenSettings.enableRefreshTokens()).isTrue();
        assertThat(tokenSettings.reuseRefreshTokens()).isTrue();
        assertThat(tokenSettings.refreshTokenTimeToLive()).isEqualTo(Duration.ofMinutes(60));
        assertThat(tokenSettings.<String>setting("name1")).isEqualTo("value1");
```

Remove enablerefreshTokens() function used here is removed while the no of settings has decreased to 5.

samples/boot/oauth2-integration/authorizationserver/src/main/java/sample/config/AuthorizationServerConfig.java



```
@Bean
public RegisteredClientRepository registeredClientRepository() {
    RegisteredClient registeredClient = RegisteredClient.withId(UUID.randomUUID().toString())
            .clientId("messaging-client")
            .clientId("client")
            .clientSecret("secret")
            .clientAuthenticationMethod(ClientAuthenticationMethod.BASIC)
            .authorizationGrantType(AuthorizationGrantType.AUTHORIZATION_CODE)
            .authorizationGrantType(AuthorizationGrantType.REFRESH_TOKEN)
            .authorizationGrantType(AuthorizationGrantType.CLIENT_CREDENTIALS)
            .redirectUri("http://localhost:8080/authorized")
            .scope("message.read")
            .scope("read")
            .scope("message.write")
            .clientSettings(clientSettings -> clientSettings.requireUserConsent(true))
            .build();
```

Change the settings of clientId and scope

Refrences

https://docs.spring.io/spring-authorization-server/reference/overview.html

https://www.youtube.com/playlist?list=PLEocw3gLFc8UNX_0du8e-up2k2wKYNLcj

https://github.com/spring-projects/spring-authorization-server/issues/155