Lab 7: Implement OAuth2:

* As a first step, update pom.xml with the OAuth2 dependency, as follows:
* <dependency>  
   <groupId>org.springframework.security.oauth</groupId>  
   <artifactId>spring-security-oauth2</artifactId>  
   <version>2.0.9.RELEASE</version>  
  </dependency>
* Next, add two new annotations, @EnableAuthorizationServer and @EnableResourceServer, to the Application.java file. The @EnableAuthorizationServer annotation creates an authorization server with an in-memory repository to store client tokens and provide clients with a username, password, client ID, and secret. The @EnableResourceServer annotation is used to access the tokens. This enables a spring security filter that is authenticated via an incoming OAuth2 token.
* In our example, both the authorization server and resource server are the same. However, in practice, these two will run separately. Take a look at the following code:
* @EnableResourceServer  
  @EnableAuthorizationServer  
  @SpringBootApplication  
  public class Application {
* Add the following properties to the application.properties file:
* security.user.name=guest  
  security.user.password=guest123  
  security.oauth2.client.clientId: trustedclient  
  security.oauth2.client.clientSecret: trustedclient123  
  security.oauth2.client.authorized-grant-types: authorization\_code,refresh\_token,password  
  security.oauth2.client.scope: openid
* Then, add another test case to test OAuth2, as follows:
* @Test  
   public void testOAuthService() {  
   ResourceOwnerPasswordResourceDetails resource = new ResourceOwnerPasswordResourceDetails();  
   resource.setUsername("guest");  
   resource.setPassword("guest123");  
   resource.setAccessTokenUri("http://localhost:8080/oauth/token");  
   resource.setClientId("trustedclient");  
   resource.setClientSecret("trustedclient123");  
   resource.setGrantType("password");  
     
   DefaultOAuth2ClientContext clientContext = new DefaultOAuth2ClientContext();  
   OAuth2RestTemplate restTemplate = new OAuth2RestTemplate(resource, clientContext);  
     
   Greet greet = restTemplate.getForObject("http://localhost:8080", Greet.class);  
    
   Assert.assertEquals("Hello World!", greet.getMessage());  
   }
* As shown in the preceding code, a special REST template, OAuth2RestTemplate, is created by passing the resource details encapsulated in a resource details object. This REST template handles the OAuth2 processes underneath. The access token URI is the endpoint for the token access.
* Rerun the application using mvn install. The first two test cases will fail, and the new one will succeed. This is because the server only accepts OAuth2-enabled requests.

These are quick configurations provided by Spring Boot out of the box but are not good enough to be production grade. We may need to customize ResourceServerConfigurer and AuthorizationServerConfigurer to make them production-ready. This notwithstanding, the approach remains the same.