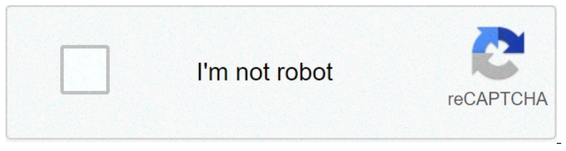

Continue

Spring boot oauth tutorial

Oauth 2.0 spring boot tutorial. Spring boot oauth jwt tutorial. Spring boot oauth tutorialspoint. Spring boot security oauth tutorial. Oauth 2.0 tutorial java spring boot.

Starting with this chapter?Clone the application repository and check the build-api branch:git clone git@github.com:auth0-blog/menu-api-spring-boot-java.git \ menu-api \ --branch build-apiMake the project folder the current directory:cd menu-api dependencies project using Gradle:./gradlew --refresh-dependenciesFinally, update the application.properties file in src/main/resources as follows:server.port=7000Run the project by running the following command:./gradlew bootRunLearn how to protect an API with the world's most widely used Java framework and Auth So far, you've built an API that allows anyone to read and write data. It is time to strengthen security, so that only users with the role of menu administrator can create, update and delete menu items.To know what a user can do, you need to know who the user is. This is known as authentication. It is often done by asking for a set of credentials, such as username and password. Once verified, the client receives information about the user's identity and access.To easily implement these Identity and Access Management (IAM) activities, you can use OAuth 2.0, an authorization framework, and OpenID Connect (OIDC), a simple identity layer.OAuth encapsulates access information into an access token. In turn, OpenID Connect encapsulates the identity information into a token ID. The authentication server can send these two tokens to the client application that starts the process. When the user requests a protected API endpoint, they must send the access token along with the request.You don't have to worry about implementing OAuth, OpenID Connect, or an authentication server. Instead, you'll use Auth0. Auth0 is a flexible, drop-down solution for adding authentication and authorization services to applications. Your team and organization can avoid the costs, time and risks associated with building your solution. In addition, there are tons of documents and SDKs to get started and easily integrate Auth0 into your stack.Auth0 is a flexible, drop-down solution to add authentication and authorization services to your applications. Your team and organization can avoid the costs, time, and risks associated with creating your solution to authenticate and authorize users. Auth0 offers tons of guidance and SDKs to get started and integrate Auth0 into your stack with ease.To get started, you need to create a free Auth0 account if you don't already have one.Try the most powerful authentication platform for free.Start after you create your account, you'll create an Auth0 Tenant account, which is a container that Auth0 uses to store your identity service configuration and your users in isolation".No other Auth0 client can peek or access your tenant. It's like I'm a condo tenant. takes care of the building, while the apartment is all yours to live and personalize. However, each apartment is completely insulated (no windows, soundproof walls, etc.) so that neighbors can not intrude After creating your tenant, you need to create an API log with Auth0, which is an API you define within your Auth0 tenant and which you can use from your applications to process authentication and authorization requests.After creating your account, go to the API section of the Auth0 Dashboard and press the Create API button.Then, as Auth0 shows:Add a name to your API account: API Menu.Set its Identifier to the signature algorithm as RS256 as it is the best option from a security point of view.Identifiers are unique strings that help Auth0 differentiate between different APIs. The use of URLs is considered a good practice, as they are predictable and easy to read. Don't worry, Auth0 will not invoke them or will never call them.With these values, press the Create button.Your API needs some configuration variables to identify with Auth0: an Audience value and a Domain value. Open the application.properties file in src/main/resources and update it:server.port=7000 auth0.audience= auth0.domain= spring.security.oauth2.resourceserver.jwt.issuer-uri=https://${auth0.domain}/Back to the Auth0 API page and follow these steps to get the Pub Auth0 Public:Click on the "Settings" tab.Find the "Identify" field and copy its value.Paste the value "Identifier" as the value of auth0.audience in application.properties.Now follow these steps to get the Auth0 value domain:Click on the "Test" tab.Click on the section called "Ask Auth0 tokens from my application".Click on the cURL tab to view a requestOST.Copy your Auth0 domain, which is part of the --url parameter: tenant-name.region.auth0.com.Paste the value of the Auth0 domain as the value of auth0.domain application.properties.Tips to get the Auth0 domainThe Auth0 domain is the substring between the protocol, https:// and the /oauth path/token.The Auth0 domain follows this pattern: tenant-name.region.auth0.com.The subdomain of the region (au, us or eu) is optional. Some Auth0 domains don't have it.Please click on the image above if you have any doubts about how to get the Auth0 domain value.Restart the server so that Spring Boot can recognize changes made to application.properties. Stop the running process and run ./gradlew bootRun again.Remember the Identity and Access Management (IAM) stream:Users will start with authentication with a username and password managed by Auth0.Once authenticated, the client will receive a JWT representing an access token.The client will include the access token in the authorization header of each request to a secure endpoint. server will validate the access token and determine if it has the right permissions, using the information contained in the token.To protect your API, first add New dependencies in your Build.Gradle: Dependency {Implementation Implementation implementation 'org.springframework.security:spring-security-oauth2-resource-server' implementation 'org.springframework.security:spring-security-oauth2-jose' }spring-boot-starter-security provides the main security entities you need to build an app.spring-security-oaut h2-resource-server contains support for OAuth 2.0 resource servers, which is primarily used to protect APIs via OAuth 2.0 Bearer Tokens. Finally, the spring-security-oauth2-jose provides you with the JOSE (Javascript Object Signing and Encryption) framework, built from a collection of specifications you'll need, such as JWT & JWK. Sync Gradle and then create a security package under the com.example. menu package. Under the new security package, create a class called SecurityConfig: package com.example.menu.security; import org.springframework.http.HtpMethod; import org.springframework.security.config.annotation.web.builders.HtpSecurfigureity; import org.springframework.security.config.annotation.web.configuration. WebSecurityConfigurerAdapter { @Override protected void configure (HttpSecurity http) throws Exception { http.authorizeRequests ().mvcMatchers (HttpMethod.GET, "/api/menu/items/**).permitAll ().anyRequest ().authenticated ().and ().oauth2ResourceServer ().jwt (); }Let's unpack. The @EnableWebSecurity annotation tells Spring to apply the web security configuration declared by the class. The class extends WebSecurityConfigurerAdapter, which provides a convenient basis for customization. You can override the configuration method to ensure that GET requests can be processed without authentication. Other requests require a JWT, which will be verified using the sender-uri from the application.Property file. HtttpSecurity is a builders class and provides numerous convenience methods that can be chained. Under the hood, each method adds a filter the HTTP request must go through. For added security, you will also want to control the audience. To do this, you need a custom validator. In the same security package, create a class called AudienceValidator: package com.example.menu.security; import org.springframework.security.oauth2.core.OAuth2Error; import org.springframework.security.oauth2.core.OAuth2ErrorCodes; import org.springframework.security2. List; import java.util. Objects; class Audience The validator implements OAuth2TokenValidator {final private study public; AudienceValidator(Supplement) { Assert.hasText(audience, "audience is null or empty"); this.audience = audience; } public OAuth2TokenValidatorResult validate(Jwt jwt) { Recipient list = jwt.getAudience (); if (audiesistacontains) err = new OAuth2Error (OAuth2ErrorCodes.INVALID_TOKEN); return OAuth2TokenValidatorResult.failure (err); } } }OAuth2TokenValidator interface and its validation method provide means to verify custom attributes OAuth 2.0 Token. With the class above, make sure only tokens containing the specified audience, or aud claiming to be accurate, are valid. To apply the custom validator, you need to upgrade the SecurityConfig class: package com.example.menu.security; import org.springframework.beans.factory.annotation. Value; import org.springframework.http.HttpMethod; import org.springframework.security.config.annotation.web.builders.HtpSecurity; import org.springframe.security.config.annotation.web.configuration.EnableWebSecurity; import org.springframe.security.config.annotation.web WebSecurityConfigurerAdapter { @Value ("${auth0.audience}") public private string; @Value("${spring.security.oauth2.resourceserver.jwt.issuer-uri}") private String issuer; @Override blank protected configure (HttpSecurity http) throws the exception { http.authorizeRequests() (Execution) OAuth2TokenValidator validator = new DelegateOAuth2TokenValidator (with Audience, withIssuer); NimbusJwtDecoder jwtDecoder = (NimbusJwtDecoder) JwtDecoders.fromOidcIssuerLocation (problemr); jwtDecoder.setJwtValidator (validator); return jwtDecoder; } The @Value annotation on an instance variable is the spring way to assign a property value from the property file application. In the jwtDecoder method, you make sure that both the public complaint (aud) and the issuer's complaint (iss) are validated. The authentication process is now complete. Use the Gradle command and try it:curl -X POST -H 'Content-Type: application/json' -d '{ "name": "Salad", "price": 499, "description": "Fresh", "image": " }' -iYou will get an unauthorized 401 response. However, the end point GET /api/menu/items works:curl -iTo test the authentication feature of your application, you need valid access tokens. A client, such as a Single-Page application (SPA), would gain access tokens by logging in and then passing the access token in an authorization header to the API. This is where the COSABYTE Dashboard comes in. You need a client application to simulate an interaction with the end userAPI and see its security in action. To make the simulation more fun and engaging, you'll use the WHATABYTE Dashboard, a demo client application that lets you manage items for a restaurant menu. You will create a user with Auth0, login and login pages that make requests to your API endpoints under the hood.To enable interaction with the end user, you will need to create a Single Page Applications Register with Auth0. This register provides the configuration values needed to connect the demo client application with Auth0, i.e. the Auth0 Domain and Auth0 Client ID. Once configured, the client application can communicate with the Auth0 authentication server and get access tokens for users who have logged in.The process of creating an Auth0 Single-Page application log is simple:Open the Auth0 Applications section of the Auth0 Dashboard.Click the Create Application button.Provide a Name value like WHATABYTE Demo Client.Choose Single Page Web Applications as the application type.Click on the Create button.A new page is loaded with details on the Auth0 application register. Click on its Settings tab to access its configuration values.Next, visit to open the WHATABYTE Dashboard demo client application.If you are not on the Auth0 Demo Settings page, click the "Settings" tab in the left navigation bar and then click the "Edit" button.Enable the authentication features of the demo application. Then use the configuration values on the "Settings" page of the Auth0 application to fill in the Auth0 Domain and Auth0 Client ID values in the demo settings module:For the Auth0 API Audience value, use , which is the identifier of the MENU API you registered with Auth 0 previously in the tutorial.For the value of Auth0 Callback URL . See how Auth0 uses this callback value in the next section.Click the Save button below the form. The WHATABYTE Dashboard is a client for your API server. To test this connection, click the Menu tab and see how it populates with the menu items you've defined in your API store.Connecting a client application with Auth0Back to the Settings tab of the Auth0 application registration page and update the following fields:Allowed Call URLsUse the value of the Auth0 Call URL from the demo settings module, a genuine user, Auth0 0 calls only any of the URLs listed in this field. You can specify multiple valid URLs by separating them by commas (typically to handle different environments such as QA or testing). Be sure to specify the protocol, http:// or https://; otherwise, the may fail in some cases. Web sources allow use of the client application will make requests under the hood at an url auth0 to manage authentication requests. as such, it is necessary to add the URL of origin of the application to avoidResource sharing issues (CORS).Allowed Logout URLSUSE . This field contains a set of URLs that Auth0 can redirect after a user registers from your application. The default Demo client configuration uses the value provided for redirecting users. With these values in place, you can scroll to the end of the "Settings" page and click the Save Changes button. Button. COURSE IN THE PRINCIPLE You used the @crossorigin annotation to enable cranti for theController element. In this section, configure the corrupt in your SecurityConfig class.Open Your SecurityConfig class from the security package and replace its content with the following: Com.example.menu.security package; import org.springframework.beans. Factory.ANNOTATION.Value; import org.springframework.http.method; import org.springframework.security.config.annotation.web.builders.httpsecurity; import org.springwork.security.config.annotation.web.configuration.neableWebSecurity; import org.springframework.security.config.annotation.web.configuration.websecurityconfigureadapter; import org.springframework.security.oauth2.core.delegawoauth2tokenwork import org.springframework.web.cors.corsconfigurationsource; import org.springframework.web.cors.urlbasedcorsconfigurationsource; import java.util.list; @EnableWebSecurity Public Class SecurityConfig extends WebSecurityConfigureaDapter {@Value ("$ {Auth0.audience}") Private Public Strings; @Value ("$ {Spring.security.oauth2.resourceserver.jwt.issuer-uri}") Private string issuer; @Override Protected Void Configuration (HTTPSecurity HTTP) Gesta Exception {http.authorizerequests () .mvcmatchers (httpmethod.get "/ API / Menu / Articles /**"). PermetAll () .Anyrequest () .Authenticato (). E () .cors () .configurationsource (Corsonfigurationsource ()). e () .oauth2resourceserver () .jwt () .decoder (jwtdecoder ()); } CorsonfigurationSource CorsonfigurationsOurce () {Corsconfiguration Configuration = New CorsConfiguration (); configuration.sewallowedmethods (list.of (httpmethod.get.name (), httpmethod.put.name (), httpmethod.post.name (), httpmethod.delete.name ()); Urlbasedcorsconfigurationsource source = new urlbasedcorsconfigurationsource (); source. RegisterCorsConfiguration ("/*", configuration. ApplyPermitDefaultValues ()); return source; Jwtdecoder jwtdecoder () {oauth2tokenvalidator withaudience = new audiencevalidator (public); Oauth2tokenvalidator conissuer = jwtvalidators.createdefaultwithissuer (issuer); Oauth2tokenvalidator validator = new delegatengoauth2tokenvalidator < > (withaudience, witissuer); Nimbusjwtdecoder jwtdecoder = (nimbusjwtdecoder)(Manufacturer); jwtdecoder.setjwtvalidator (validator); return jwtdecoder; }} You can also delete the following row from= " ") Stop the running server and run ./gradlew BootRRUN one more time to make these effective changes. In the Demo client, click the Sign In button. The client redirects you to the Auth0 Universal Login page to log in or register. Since this might be the first user you're adding to Auth0, go ahead and click the registration link at the bottom of the form. Then, provide an email and password to register a new user. The user logs in, the user interface of the DEMO client changes: The Log in button becomes a registration button. You can find a user card under the registration button. In the User tab to view a profile page with your name or email as title and your profile picture – If you are logged in with Google: the Demo client is aimed at three types of users: Non-authenticated visitors: any visitor who has not registered In â" ¬"Some literature may refer to this type of user as a "guest" or "anonymous ". Automatic users: any visitor who logs in successfully. Users: Any user authenticated with the role of the menu-adminator. This tutorial is to use the menu-administrator role and its associated permissions as access control artifacts. The plan is to allow only admin users to create, update, and delete menu items in the whabyte dashboard. In the Access Control (RBAC) section based on this tutorial of this tutorial, you will create the menu-admin role, associate associated permissions and assign it to a new user that you will create via the Auth0 Dashboard.However, you Let's start protecting your API write endpoints against unauthenticated visitors. Experiment with the Demo client: add items by clicking the Add Item button located in the upper right corner of the "Menu." Click Elements and try editing or deleting them. You can do any reading or writing task right now. Security Exercise: Try your Protectionlog endpoint outside the demo application. Click the Settings tab on the left-hand navigation bar of the DEMO client. Then, click the Edit button. The "Auth0 Demo Settings" page is loaded. Disable authentication features: Click the Save button. It can be reloaded of the demo application, click again on the MENU tab. You will notice that the Add Item button is now visible. In this mode, the demo client allows you to access the UI elements making requests to your endpoints write API as an unauthenticated visitor. As such, such requests will not include an access token. If your API security is working properly, you should refuse such requests. Click the Add Item button to open a form and click the Save button. You will have a mistake, no authorization tokens were found: success! The spring start API server is actually protecting your writing endpoints against unauthorized requests. In this context, only authenticated users are allowed to access API writing endpoints. Click the Cancel button on the "Add Item menu" page. The menu jobs charge again. Click on the "Burger" element and try to modify or delete it. Can be two actionsEven fail: you have tested that Spring Boot is protecting your creation, update and eliminate endpoints correctly, concluding this short exercise. To continue with the rest of this tutorial, reactivate the demo client authentication functionalities. Click the Settings tab and click the Edit button. The "AUTH0 DEMO SETTINGS" page charge. Enable authentication features, fill out the necessary value and click the Save button. Configuring Role-Based Access Control (RBAC) Any request with a valid access token can use the API to read and write the data. But not all users are the same: some only need to read the data, while others may want to add, delete or change data in the store. It is necessary to further develop the authorization strategy to check if a user who makes a request can perform a certain operation. Manage access with AUTH0A simple way to implement this level of authorization is through the access control based on the role (RBAC). Assign permissions to users based on their roles. A Menu-Admin role, for example, could have all the necessary permissions to create, update, and delete menu items. When users will successfully access, access token Auth0 has information on any permissions that users have based on their assigned roles. Since AUTH0 emits AKEN access as a JSON Web Token (JWT), which access information is added to the token as a complaint called permissions. JWT statements are essentially coded key-value pairs as a JSON object. The server application can check the access token and compare the values in its permissions claim with the permissions required by the EPI endpoint. If the server can fully correspond to the permissions required by the endpoint, the client request is authorized. Implement RBAC is easily done through the Auth0 dashboard. Here is the plan of what you will do: Create permissions for the menu API you created previously. Create a role called Menu-Admin. Assign permissions from the menu API to the Menu-Admin role. Create a new user and assign it to the Menu-Admin role. Let's start. Open the API page from the Auth0 dashboard and select the menu API you created previously. On the menu API page, click the Permissions tab and create three permissions by filling out each row as follows (+ Add button adds a new line): Create: Items: Create the elements of theUpdate: Items: Update menu items: Delete The elements of the MenuAvanti must be configured AUTH0 to apply the access control permission based on the role (RBAC) for the menu API. Click the Settings tab and scroll down until the section appears RBAC. Use the activation button next to Enable RBAC to turn it on, which enforces Auth0 to evaluate RBAC authorization policies during a user login transaction. Next, enable Add Permissions in the Access Token to add an access token permission property created by Auth0 when a user logs in. The property of permits is a key value pair known as a token claim. The presence of this complaint is fundamental to the implementation of RBAC in your APIenable these options, make sure to click the Save button. Creating roles Open the Roles page from the Auth0 Dashboard and click the Create Role button. Fill out the pop-up form as follows:Name: menu-adminDescription: Create, update, and delete menu items. Once done, click the Create button to complete the role creation. Now, you need to associate the permissions you created with this role, mapping it to the resources of your API. Click the Permissions tab on the role page. Once there, click the Add Permissions button. In the dialog that appears, select the menu API from the drop-down box and select all the boxes in the Scopes section. Once done, click the Add Permissions button. You are back to the menu-admin role page, which now lists all its associated permissions. A purpose is a term used by the OAuth 2.0 protocol to define limits on the amount of access that can be granted to an access token. Essentially, permissions define the scope of an access token. Get User Roles Auth0 attaches the menu-admin role permissions as a claim to the login token, but not the role itself. The demo client application needs this information as it makes the UI conditionally based on the user's role. To include the user's role as a complaint in the tokens that Auth0 sends to the client, you can use the Auth0 rules. When a user successfully logs in to your application, the Auth0 Authorization Server sends two tokens to the client: After a user authenticates and authorizes access, the client application receives an access token from the Auth0 authentication server. The client passes the access token as credentials when calling a secure endpoint of the target API. This token informs the server that the client is allowed to access the API. Through its claim permissions, the token access tells the server what actions the client can take on which resources. The ID Token is a JSON Web Token (JWT) that contains claims representing user profile attributes such as name or email, which are values that clients typically use to customize the user interface. Using Auth0 rules, you can add to each of these tokens a new claim, representing the roles assigned to a user. What are the rules of Auth0? Auth0 rules are JavaScript functions that run when a user accesses your application. They are run once the authentication process is complete, and you can use them to customize and extend the capabilities of Auth0. For security, the Rules code runs in a sandbox, isolated from the code of other Auth0 tenants. You can easily create Auth0 Rules using the Auth0 Dashboard. Follow these steps create a rule that adds user roles to the tokens: Open the Rules page from the Dashboard Auth0. Click the Create Rule button. Click the Empty Rule option. Provide a name to your rule, such as "Add User Roles to Tokens". Subsequently, replace the content of the Script section with the following function:function(user, context, callback) { const namespace = « »; if&& context.authorization.les) {CONST ASSIGNDRALI = CONTEXT.AUTHORIZATION.LOLES; IF (context.idtoken) {CONST IDTOKENCLAIMS = context.idtoken; idoTokenclamas [`$ {namespace} / roles] = assigned; context.idtoken = idoTokenclaims; } If (context.accesstoken) {CONST ACCESSTOKENCLAIMS = CONTEXT.ACCESSTOKEN; AccessOKENCLAIMS [`$ {namespace} / Roles] = Assigned; context.accesstoken = accessTokenclaims; }} Callback (NULL, user, context); } Click the Save Changes button. What is this rule doing? When the user successfully authentic, this rule function is performed, receiving three parameters: User: An object returned by the identity provider (such as AUTH0 or Google) representing the logged logger.Context: an object that stores contextual information on 'Current authentication transaction, such as the IP address or user position. Callback: a function to send modified tokens or an error for AUTH0. You need to call this function to prevent script.function timeout (user, context, callback) {} to keep your requests customized by the collision with confidential or external claims, you need to provide them with a unique name globally using a namPaciant format . By default, AUTH0 always applies the drive of the NAMESIMA and silently excludes from the tokens any custom complaints with non-named identifiers. PDESPACES are arbitrary identifications, so technically, you can call your namespace whatever you want. For comfort, the value of the namespace is the value of the public API set in the demo demo dashboard demo of whabyte.function (user, context, callback) {CONST namespace = ' '; } Then check if the contextual object has an authorization property and, in turn, if this property has a property roles: function (user, context, callback) {Const namespace = 'https: //menu-api.example. com; if (context.authorization && context.authorization.loles) {}} context.authorization is an object containing information about the authorization transaction, such as roles.context.authorization.loles is a series of strings containing the names of the assigned roles To a user.next, the role array is assigned to the assignment constant and check if a token ID is present or access to the token in the context object: function (user, context, callback) {Const namespace = 'https : // menu -api.example.com '; If (context.authorization && context.authorization.loles) {CONST ASSIGNDROLES = CONTEXT.AUTHORIZATION.OLES; If (context.idtoken) {} if (context.accesstoken) {}}} If one of these tokens is present, adds to the token object to / properties roles with the role array, assigned, Like its value, effectively create a custom complaint on the token that represents user roles: function (user, context, callback) {Const namespace = se (context.Authorization && context.authorization.Loles) {CONST ASSIGNADROLES = context.Authorization.oles; IF (context.idtoken) {const idtokenclaims = context.idtoken; IdtokenClaims [`$ {Namespace} / ruoli`] = = context.idtoken = IdtokenClaims; } if (context.accesstoken) {CONST AccessTokenClaims = context.accesstoken; AccessTokenClaims ['$ {namespace} / ruoli'] = assigned; context.accesstoken = AccessTokenClaims; }}}} Finally, the callback function is invoked to send potentially modified tokens to Auth0, which in turn sends them to the client: function (user, context, callback) {callback (null, user, context); } This is all you need to create an Auth0 rule that adds user roles to the tokens. What is left to do is for you to create a user who has the role of menu-admin.Before doing so, check the way the UI restricts access to certain UI items and views when a user does not have the role of menu admin. Back to the Demo client.Next, click the "Settings" tab from the left navigation bar and click the "Edit" button to change the demo settings. The settings "Auth0 Demo Settings" is loaded. Enables role-based access control (RBAC), which reveals the scope of the user's role. Populate that field with the following value: menu-admin.itce you have set that value, leave any other field as it is. Then, click the Save button. You're back to the application, log in. WARNING How the Add Item button is no longer visible in the "Menu Items" page. If you click on a menu item, the Edit or Delete buttons will not be displayed. You need to grant yourself or any other user you create admin access! Create a useropen admin The users page from the Auth0 Dashboard and click Create User. Fill in the form that opens with the following: Email: admin@example.compassword and repeat password: any password of your choice: username-password-authenticationClick on the Create button. The admin@example.com user page loads. On this page, click the "Roles" tab and then click the Assign Roles button. From the drop-down menus, select the role of the menu-adminator you created previously and click the Assign button. Verify that the user has the permissions by clicking on the "Permissions" tab. In that case, your admin user is all set up and ready for use.Alternatively, you can assign the menu-administrator role to the existing user who was used to access the demo application. Return to the Demo client and exit. Click the Login button again and, this time, log in as admin@example.com user or as any user you have granted the menu-admin role. This time around, the UI unlocks the administration functionality. Open the "Menu" page and notice the "Add Item" button is back in the top right corner. Click on a menu item and notice how you can now edit or delete At this time, non-administrative users could evade the client-side path protections to unlock UI's administration functionalities. Furthermore, they could extract the access token sent by AUTH0 using the browser developer tools and make requests directly to the server writing endpoints using the terminal, for example. Your server must implement the access control based on roles to mitigate these attacks attacks Role-based access control In the Boota JWT spring released by an authorization server usually has an attribute of the scope, listing the permissions granted. Spring calls them to the authorities granted. Instead, AUTH0 uses a personalized complaint called permissions to specify them. The JWT useful load is similar to this: {"Scope": "Openid e-mail profile", "Authorizations": ["Creation: Elements", "Delete: Elements", "Read: Elements", "Update: Elements "]} The spring provides a default JWTautannicationConverter instance that the authorities granted in a SCP or claims. To use the permissions instead, update your SecurityConfig class to the final form: pack com.example.menu.security; Import org.springframework.beans.factory.annotation.value; Import org.springframework.http.httpmethod; Import org.springframework.security.config.annotation.method.configuration.enableGlobalMethodsecurity; Import org.springframework.security.config.annotation.web.builders.httpsecurity; Import org.springframework.security.config.annotation.web.configuration.neakwebsecurity; Import org.springframework.security.config.annotation.web.configuration.websecurityConfigureadapter; Import org.springframework.security.oauth2.core.delegawoauth2tokenValidator; Import org.springframework.security.oauth2.core.oauth2tokenValidator; Import org.springframework.security.oauth2.jwt.jwt; Import org.springframework.security.oauth2.jwt.jwtdecoder; Import org.springframework.security.oauth2.jwt.jwtdecoder; Import org.springframework.security.oauth2.jwt.jwtvalidators; Import org.springframework.security.oauth2.jwt.nimbusjwtdecoder; Import org.springframework.security.oauth2.server.resource.authentication.jwtauthenticationConverter; Import org.springframework.security.oauth2.server.resource.authentication.jwtgrantauthieritiesconverter; Import org.springframework.web.cors.corsconfiguration; Import org.springframework.web.cors.corsconfigurationSource; Import org.springframework.web.Cors.UrlBasedCorsConfigurationSource; Import Java.util.list; @Enablewebsecurity @enableglobalmethodsecurity (prepostenablebled = true) public class securityconfig extends websecurityconfigureadapter {@value ("$ {auth0.audience}") private public; @Value ("$ {spring.security.oauth2.resourceerver.jwt.issuer-uri}") private string issuer; @Override protected void configuration (httpsecurity http) Exception {http.authorizerequests () .mvcatchers (httpmethod.get "/ bees / menus / articles / **"). Permetall () .anyrequest () .Autheticato (). And () .cors () .ConfigurationSource (corsconfigurationsource ()). E () ..oauth2resourceserver () .jwt () .decoder (jwtdecoder ()) .jwtauthentiationconverter (jwtautannicionconverter ()); } CorsonfigurationSource CorsonfigurationSource () {corsconfiguration configuration = new corsconfiguration (); configuration.sewallowedmethods (list.of (httpmethod.get.name (), (), httpmethod.post.name (), httpmethod.delete.name ())); URLBASEDCORSCONFIGURATIONSOURCE SOURCE = NEW URLBASEDCORSCONFIGURATIONSOURCE (); (); configuration. applyPermitDefaultValues()); return source; JwtDecoder jwtDecoder() {OAuth2TokenValidator_Jwt> with Audience = new AudienceValidator(audience); OAuth2TokenValidator conIssuer = JwtValidators.createDefaultWithIssuer(problemr); OAuth2TokenValidator validator = new D stylishOAuth2TokenValidator (with Audience, withIssuer); JwtDecoder jwtDecoder = (NimbusJwtDecoder) JwtDecoders. fromOidcIssuerLocation(problem); jwtDecoder.setJwtValidator(validator); jwtDecoder; jwtConverter return; } }Finally, add an @PreAuthorize ad to the relevant methods in the ItemController, update that class to its final module FieldError; import org.springframework.validation. ObjectError; import org.springframework.web.bind.MethodArgumentNotValidframeception; import org.springframework.web.bind.annotation. ServletUriComponentsBuilder; import javax.validation. Valid; import java.net. URI; import java.util. HashMap; import java.util. List; import java.util. Map; import java.util. @Requests/Requests Since you have ensured that they will be read by the authorization claim, this is the final step of the authorization process. Make sure to run the boot Gradle Run the command to make the changes effective:. /gradlew boot RunSign out and come back asadministrator user in the demo client. try to add a new element. the "Add Item" page has a preloaded form with some data to make this process easier for you. if you have already created the salad element, try to create a coffee product with these data:name: coffee price: 299 description: glove image: on that newly created entry and notice that you can edit Try both operations. safety exercise: remove the rolelog admin from the demo application. Click the Settings tab on the left navigation bar of the demo client. then, click the edit button. the page "auth0 demo settings" charge. delete the oer role value, leave blank, and then click the save button. now, or:(a) signs as a non-admin user, or(b) remove the menu-admin role from your current user in the auth0 dashboard and access as such user. you will have access to the elements of the admin user interface. Click the tea article and try to delete it. you will receive an error message, insufficient application field: This error message is telling you that you don't have enough permission to do that. If you inspect the network card or browser development tool console, you will notice that the api string boot server responded with a 403 error (forbidden.) you will get the same type of error if you try to add or change an item. you have confirmed that your api string boot server is effectively protecting your writing endpoints from unenacted users and authenticated users who do not have permissions to access them. Click the Settings tab on the left navigation and click the Edit button. restore the oer role value to the menu-admin and save the changes. if you have removed the menu-admin role from a user, go back to the auth0 tab and return the role to the user. what is nextthis concludes the Spring Authorization tutorial. you have implemented permission to control the resources your users can access. you have learned to implement different levels of access: access based on the authentication status. If you have logged in, you are authorized to access resources. access based on permissions. If you have logged in and have the required permissions, you are authorized to access resources. This tutorial covered the most common usage cases of authorization for a spring start bee server. However, auth0 is an extended and flexible platform that can help you achieve even more. If you have a more complex case of use, check the [Autth0 architecture scenario](to learn more about typical architecture scenariosWe identified when we work with customers on Auth0 implementation. What other chapters should be added? This is what I have in mind for the future: distribution of a Spring Boot application to AWS. Connecting a Spring Boot application to a MongoDB or PostgreSQL store. Using the QL or GRPC graph with spring spring I know what you think in the feedback section, and thanks for reading! I have feedback or I had a problem