(/)

Accessing Keycloak Endpoints Using Postman

FEATURED VIDEOS

Last updated: May 11, 2024

> Written by: baeldung (https://www.baeldung.com/author/baeldung)

> Reviewed by: Eric Martin (https://www.baeldung.com/editor/eric-editor)

Security (https://www.baeldung.com/category/security)

Testing (https://www.baeldung.com/category/testing)

Keycloak (https://www.baeldung.com/tag/keycloak)

OAuth (https://www.baeldung.com/tag/oauth)

Postman (https://www.baeldung.com/tag/postman)

1. Introduction

In this tutorial, we'll start with a quick review of OAuth 2.0, OpenID, and Keycloak. Then we'll learn about the Keycloak REST APIs and how to call them in Postman.

2. OAuth 2.0

OAuth 2.0 (https://tools.ietf.org/html/rfc6749) is an authorization framework that lets an authenticated user grant access to third parties via tokens. A token is usually limited to some scopes with a limited lifetime. Therefore, it's a safe alternative to the user's credentials.

OAuth 2.0 comes with four main components:

- **Resource Owner** – the end-user or system that owns a protected resource or data
- **Resource Server** – the service exposes a protected resource, usually through an HTTP-based API
- **Client** – calls the protected resource on behalf of the resource owner
- **Authorization Server** – issues an OAuth 2.0 token and delivers it to the client after authenticating the resource owner

OAuth 2.0 is a protocol with some standard flows (https://auth0.com/docs/protocols/protocol-oauth2), but we're especially interested in the authorization server component here.

3. OpenID Connect

OpenID Connect 1.0 (https://openid.net/connect/) (OIDC) is built on top of OAuth 2.0 to add an identity management layer to the protocol. Hence, it allows clients to verify the end user's identity and access basic profile

information via a standard OAuth 2.0 flow. OIDC has introduced a few standard scopes to OAuth 2.0 (/spring-security-openid-connect), like *openid*, *profile*, and *email*.

4. Keycloak as Authorization Server

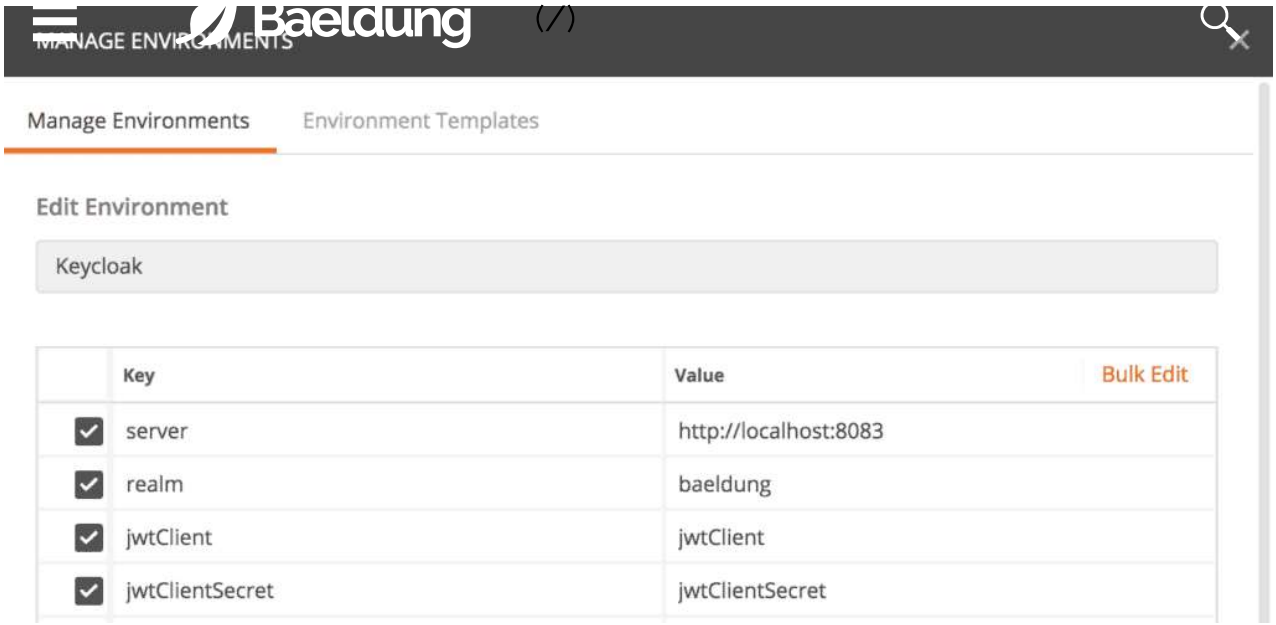
JBoss has developed Keycloak (http://www.keycloak.org/) as a Java-based open-source Identity and Access Management solution. Besides the support of both OAuth 2.0 and OIDC, it also offers features like identity brokering, user federation, and SSO.

We can use Keycloak as a standalone server with an admin console (/spring-boot-keycloak)or embed it in a Spring application (/keycloak-embedded-in-spring-boot-app). Once we have our Keycloak running in either of these ways, we can try the endpoints.

5. Keycloak Endpoints

Keycloak exposes a variety of REST endpoints for OAuth 2.0 flows.

To use these endpoints with Postman (https://www.postman.com/), we'll start by creating an Environment called "*Keycloak.*" Then we'll add some key/value entries for the Keycloak authorization server URL, the realm, OAuth 2.0 client id, and client password:



(/wp-content/uploads/2020/10/Screen-Shot-2020-10-22-at-6.25.07-PM.png)

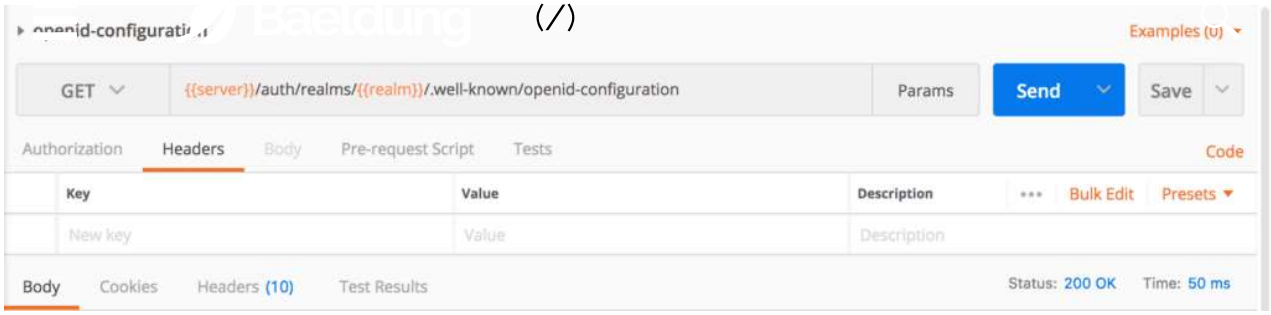
Finally, we'll create a collection where we can organize our Keycloak tests. Now we're ready to explore available endpoints.

5.1. OpenID Configuration Endpoint

The configuration endpoint is like the root directory. It returns all other available endpoints, supported scopes and claims, and signing algorithms (/spring-security-openid-connect).

Let's create a request in Postman: ***{{server}}/*auth/realms/*{{realm}}/*.well-known/openid-configuration.** Postman sets the values of *{{server}}* and *{{realm}}* from the selected environment during runtime:

(/)



(/wp-content/uploads/2020/10/postman-endpoint-openidconfig.png)
Then we'll execute the request, and if everything goes well, we'll have a response:

```
{
    "issuer": "http://localhost:8083/auth/realms/baeldung",
    "authorization_endpoint":
"http://localhost:8083/auth/realms/baeldung/protocol/openid-
connect/auth",
    "token_endpoint":
"http://localhost:8083/auth/realms/baeldung/protocol/openid-
connect/token",
    "token_introspection_endpoint":
"http://localhost:8083/auth/realms/baeldung/protocol/openid-
connect/token/introspect",
    "userinfo_endpoint":
"http://localhost:8083/auth/realms/baeldung/protocol/openid-
connect/userinfo",
    "end_session_endpoint":
"http://localhost:8083/auth/realms/baeldung/protocol/openid-
connect/logout",
    "jwks_uri":
"http://localhost:8083/auth/realms/baeldung/protocol/openid-
connect/certs",
    "check_session_iframe":
"http://localhost:8083/auth/realms/baeldung/protocol/openid-
connect/login-status-iframe.html",
    "grant_types_supported": [...],
    ...
    "registration_endpoint":
"http://localhost:8083/auth/realms/baeldung/clients-
registrations/openid-connect",
    ...
    "introspection_endpoint":
"http://localhost:8083/auth/realms/baeldung/protocol/openid-
connect/token/introspect"
}
```

As mentioned before, we can see all the available endpoints in the response, such as "*authorization_endpoint,*" "*token_endpoint,*" and so on.

Moreover, there are other useful attributes in the response. For example, we can figure out all the supported grant types from "*grant_types_supported*" or all the supported scopes from "*scopes_supported*."

5.2. Authorize Endpoint

Let's continue our journey with the authorize endpoint responsible for OAuth 2.0 Authorization Code Flow (https://auth0.com/docs/flows/authorization-code-flow). It's available as *"authorization_endpoint"* in the OpenID configuration response.

The endpoint is:

***{{server}}*/auth/realms/*{{realm}}*/protocol/openid-connect/auth?response_type=code&client_id=jwtClient**

Moreover, this endpoint accepts *scope* and *redirect_uri* as optional parameters.

We won't use this endpoint in Postman. Instead, we usually initiate the authorization code flow via a browser. Then Keycloak redirects the user to a login page if no active login cookie is available. Finally, the authorization code is delivered to the redirect URL.

Next we'll see how to obtain an access token.

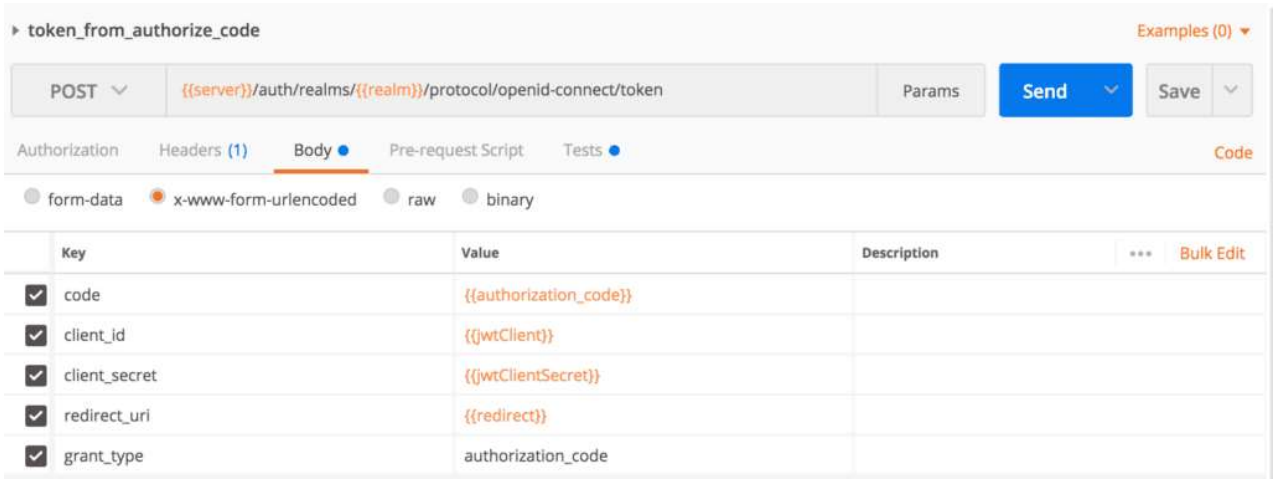
5.3. Token Endpoint

The token endpoint allows us to retrieve an access token, refresh token, or id token. OAuth 2.0 supports different grant types, like *authorization_code*, *refresh_token*, or *password*.

The token endpoint is:
***{{server}}*/auth/realms/*{{realm}}*/protocol/openid-connect/token**

However, each grant type needs some dedicated form parameters.

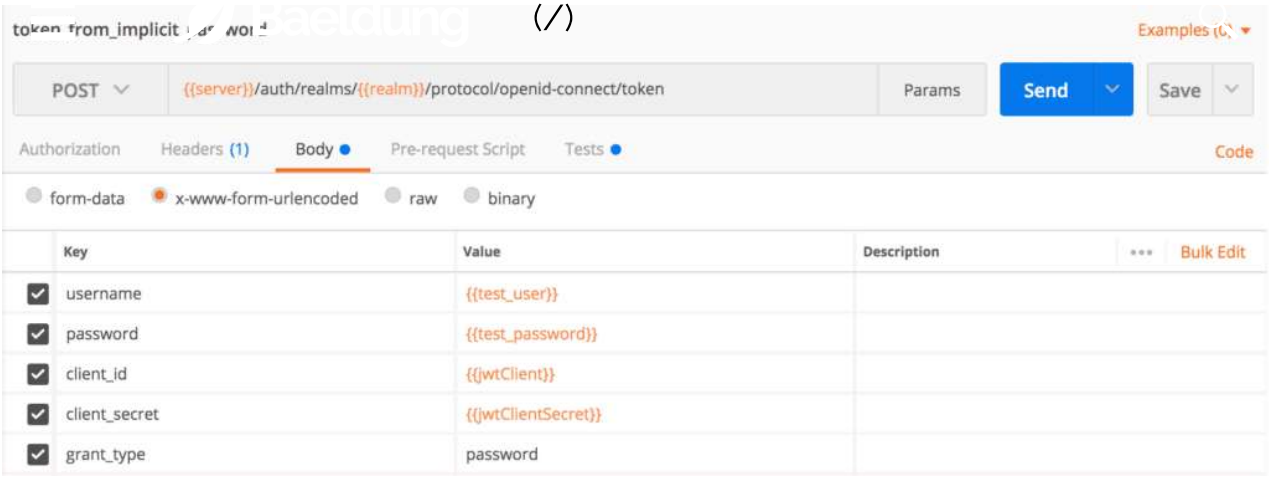
We'll first test our token endpoint to obtain an access token for our authorize code. We'll have to pass these form parameters in the request body: *client_id*, *client_secret*, *grant_type*, *code*, and *redirect_uri*. The token endpoint also accepts *scope* as an optional parameter:



(/wp-content/uploads/2020/10/postman-token-authcode.png)
If we want to bypass the authorization code flow, the *password* grant type is our choice. Here we'll need user credentials, so we can use this flow when we have a built-in login page on our website or application.

Let's create a Postman request and pass the form parameters *client_id*, *client_secret*, *grant_type*, *username*, and *password* in the body:

(/)



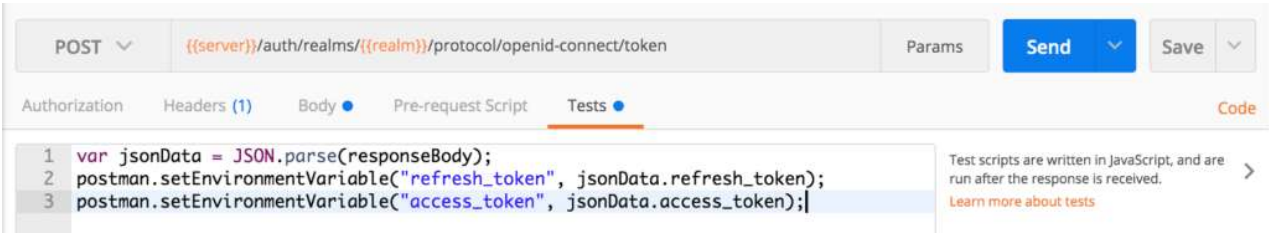
(/wp-content/uploads/2020/10/postamn-token-password.png)

Before executing this request, we have to add the *username* and *password* variables to Postman's environment key/value pairs.

Another useful grant type is *refresh_token*. We can use this when we have a valid refresh token from a previous call to the token endpoint. The refresh token flow requires the parameters *client_id*, *client_secret*, *grant_type*, and *refresh_token*.

We need the response *access_token* to test other endpoints. To speed up our testing with Postman, we can write a script in the *Tests* section of our token endpoint requests:

```
var jsonData = JSON.parse(responseBody);
postman.setEnvironmentVariable("refresh_token",
jsonData.refresh_token);
postman.setEnvironmentVariable("access_token", jsonData.access_token);
```



(/wp-content/uploads/2020/10/postman-test-script.png)

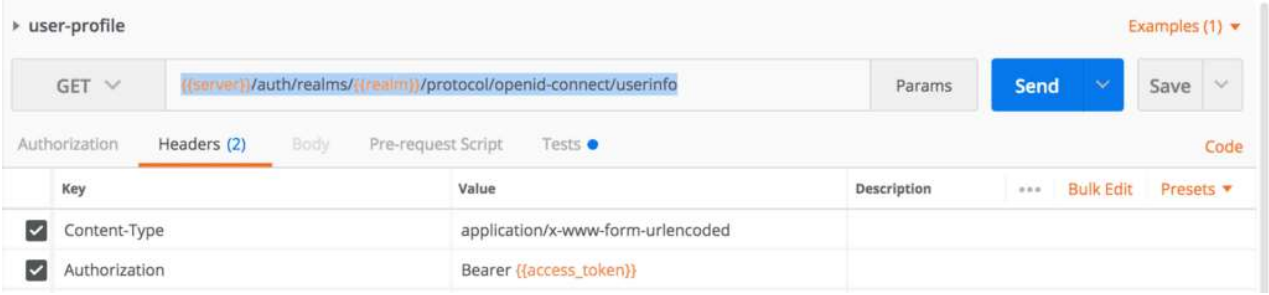
5.4. User Information Endpoint

We can retrieve user profile data from the user information endpoint when we have a valid access token.

(/)

The user information endpoint is available at:
***{{server}}*/auth/realms/*{{realm}}*/protocol/openid-connect/userinfo**

Now we'll create a Postman request for it, and pass the access token in the *Authorization* header:



(/wp-content/uploads/2020/10/postamn-userinfo.png)
Then we'll execute the request. Here's the successful response:

```
{
    "sub": "a5461470-33eb-4b2d-82d4-b0484e96ad7f",
    "preferred_username": "john@test.com",
    "DOB": "1984-07-01",
    "organization": "baeldung"
}
```

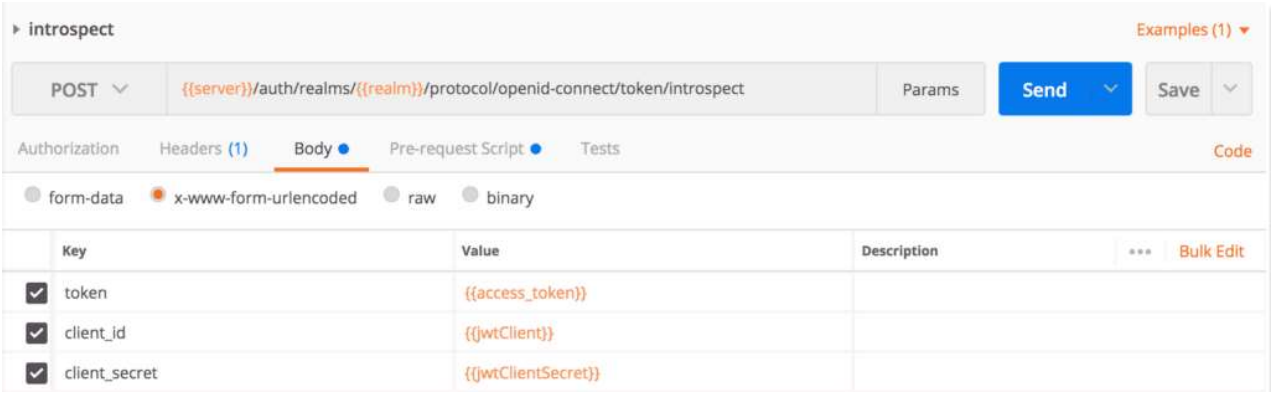
5.5. Token Introspect Endpoint

If a resource server needs to verify that an access token is active or wants more metadata about it, especially for opaque access tokens (https://auth0.com/docs/tokens/access-tokens), then the token introspect

endpoint is the answer. In this case, the resource server integrates the introspect process with the security configuration (/spring-security-oauth-resource-server).

We'll call Keycloak's introspect endpoint: *{{server}}*/auth/realms/*{{realm}}*/protocol/openid-connect/token/introspect

Then we'll create an introspect request in Postman, and pass *client_id*, *client_secret*, and *token* as form parameters:



(/wp-content/uploads/2020/10/postman-introspect.png)
If the *access_token* is valid, then we'll have our response:

```
{
    "exp": 1601824811,
    "iat": 1601824511,
    "jti": "d5a4831d-7236-4686-a17b-784cd8b5805d",
    "iss": "http://localhost:8083/auth/realms/baeldung",
    "sub": "a5461470-33eb-4b2d-82d4-b0484e96ad7f",
    "typ": "Bearer",
    "azp": "jwtClient",
    "session_state": "96030af2-1e48-4243-ba0b-dd4980c6e8fd",
    "preferred_username": "john@test.com",
    "email_verified": false,
    "acr": "1",
    "scope": "profile email read",
    "DOB": "1984-07-01",
    "organization": "baeldung",
    "client_id": "jwtClient",
    "username": "john@test.com",
    "active": true
}
```

However, if we use an invalid access token, then the response will be:

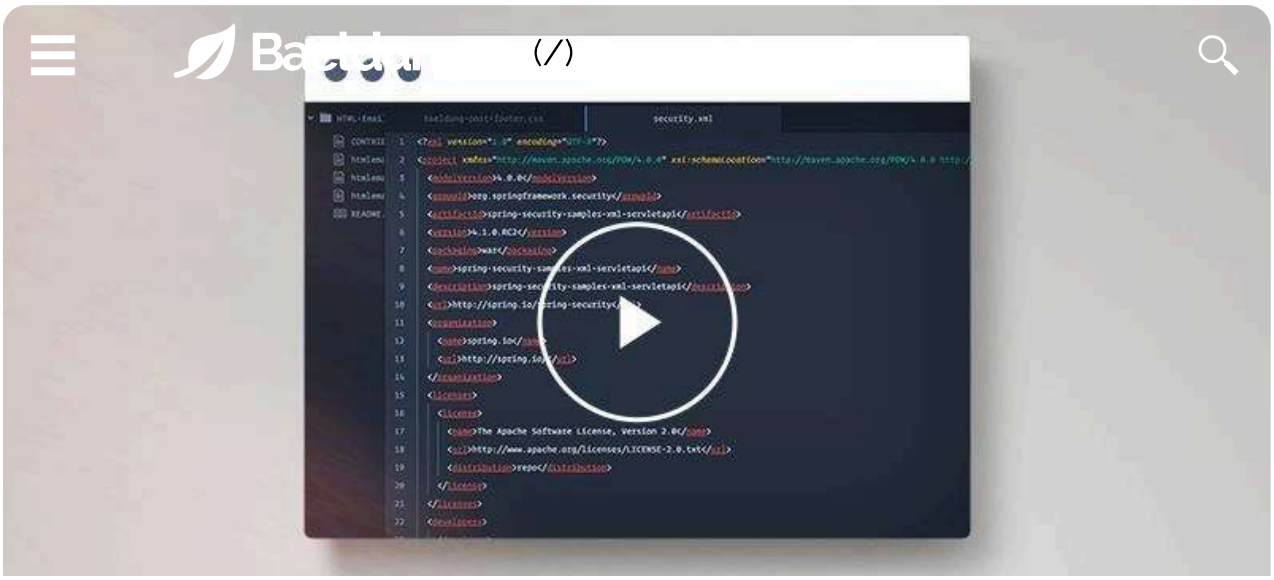
(/)

```
{
    "active": false
}
```

6. Conclusion

In this article, with a running Keycloak Server, we created Postman requests for the authorization, token, user information, and introspect endpoints.

As always, the complete examples of the Postman requests are available over on GitHub (https://github.com/Baeldung/spring-security-oauth/tree/master/oauth-jwt/jwt-auth-server).



(/)

Learn the basics of securing a REST API with Spring

Get access to the video lesson (/security-video-guide)

2 COMMENTS

Oldest

View Comments

(/)

COURSES

ALL COURSES (/COURSES/ALL-COURSES)
ALL BULK COURSES (/COURSES/ALL-BULK-COURSES)
ALL BULK TEAM COURSES (/COURSES/ALL-BULK-TEAM-COURSES)
THE COURSES PLATFORM (HTTPS://COURSES.BAELDUNG.COM)

SERIES

JAVA "BACK TO BASICS" TUTORIAL (/JAVA-TUTORIAL)
JACKSON JSON TUTORIAL (/JACKSON)
APACHE HTTPCLIENT TUTORIAL (/HTTPCLIENT-GUIDE)
REST WITH SPRING TUTORIAL (/REST-WITH-SPRING-SERIES)
SPRING PERSISTENCE TUTORIAL (/PERSISTENCE-WITH-SPRING-SERIES)
SECURITY WITH SPRING (/SECURITY-SPRING)
SPRING REACTIVE TUTORIALS (/SPRING-REACTIVE-GUIDE)
JAVA ARRAY (HTTPS://WWW.BAELDUNG.COM/CATEGORY/JAVA/JAVA-ARRAY)

ABOUT

ABOUT BAELDUNG (/ABOUT)
THE FULL ARCHIVE (/FULL_ARCHIVE)
EDITORS (/EDITORS)