

[Open in app](#)

Tech Geek

153 Followers

[About](#)[Follow](#)

Easily secure your Spring Boot applications with Keycloak



Tech Geek Nov 22, 2018 · 6 min read

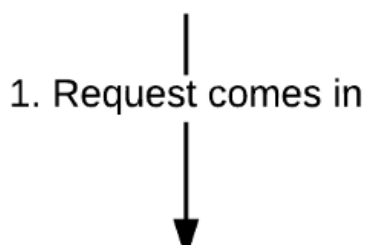


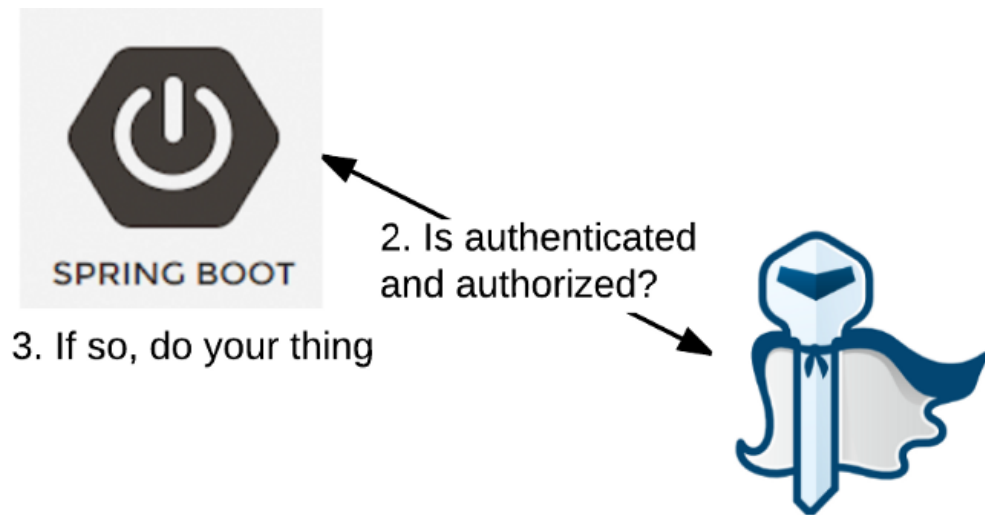
What is keycloak ?

Security is a crucial aspect of any application, its implementation can be difficult. Keycloak is open source authentication server (as per the official docs of the keycloak).

Keycloak is more than just an authentication server, it also provides a complete Identity Management system.

Spring Boot and Keycloak





Keycloak provides adapters for an application that needs to interact with a Keycloak instance. There are adapters for WildFly/EAP, NodeJS, Javascript and of course for Spring Boot.

Setting up a Keycloak server:

You have different options to set up a Keycloak server but the easiest one is probably to grab a standalone distribution, unzip it.

Downloads

4.6.0.Final - [Release notes](#)

For a list of community maintained extensions check out the [Extensions](#) page.

Server

Server	Standalone server distribution	ZIP (sha1)	TAR.GZ (sha1)
Overlay *DEPRECATED*	Server add-on for WildFly. Not recommended in production. *DEPRECATED*	ZIP (sha1)	TAR.GZ (sha1)

Gatekeeper

Linux	TAR.GZ (sha1)
Windows	TAR.GZ (sha1)
Darwin	TAR.GZ (sha1)

<https://www.keycloak.org/downloads.html>

You have different options to set up a Keycloak server but the easiest one is probably to grab a standalone distribution, unzip it.

Open a terminal and go to your unzipped Keycloak server and from the bin directory simply run:

```
./standalone.sh
```

```
chirag@chirag-Lenovo-Ideapad-330-15IKB: ~/practise/keycloak/server/keycloak-4.6.0.Final/bin
chirag@chirag-Lenovo-Ideapad-330-15IKB: ~/practise/keycloak/product-app/src/main/resources$ sudo ./standalone.shbat
[sudo] password for chirag:
chirag@chirag-Lenovo-Ideapad-330-15IKB: ~/practise/keycloak/server/keycloak-4.6.0.Final/bin$ sudo ./standalone.sh
=====
JBoss Bootstrap Environment

JBoss_HOME: /home/chirag/practise/keycloak/server/keycloak-4.6.0.Final

JAVA: /usr/lib/jvm/java-8-oracle/bin/java

JAVA_OPTS: -server -Xms64m -Xmx512m -XX:MetaspaceSize=96M -XX:MaxMetaspaceSize=256m -Djava.net.preferIPv4Stack=true -Djboss.modules.system.pkgs=org.jboss.byteman -Djava.awt.headless=true
=====
14:30:23,232 INFO [org.jboss.modules] (main) JBoss Modules version 1.8.6.Final
14:30:24,112 INFO [org.jboss.msc] (main) JBoss MSC version 1.4.3.Final
14:30:24,120 INFO [org.jboss.threads] (main) JBoss Threads version 2.3.2.Final
14:30:24,226 INFO [org.jboss.as] (MSC service thread 1-2) WFLYSRV0049: Keycloak 4.6.0.Final (WildFly Core 6.0.2.Final) starting
14:30:24,794 INFO [org.wildfly.security] (ServerService Thread Pool -- 17) ELY00001: WildFly Elytron version 1.6.0.Final
14:30:25,190 INFO [org.jboss.as.controller.management-deprecated] (Controller Boot Thread) WFLYCTL0028: Attribute 'security-realm' in the resource at address '/core-service=management/management-interface=http-interface' is deprecated, and may be removed in a future version. See the attribute description in the output of the read-resource-description operation to learn more about the deprecation.
14:30:25,218 INFO [org.jboss.as.controller.management-deprecated] (ServerService Thread Pool -- 5) WFLYCTL0028: Attribute 'security-realm' in the resource at address '/subsystem=undertow/server=default-server/https-listener=https' is deprecated, and may be removed in a future version. See the attribute description in the output of the read-resource-description operation to learn more about the deprecation.
14:30:25,284 INFO [org.jboss.as.server] (Controller Boot Thread) WFLYSRV0039: Creating http management service using socket-binding (management-http)
14:30:25,299 INFO [org.xnio] (MSC service thread 1-7) XNIO version 3.6.5.Final
14:30:25,304 INFO [org.xnio.nio] (MSC service thread 1-7) XNIO NIO Implementation Version 3.6.5.Final
14:30:25,357 INFO [org.jboss.as.clustering.infinispan] (ServerService Thread Pool -- 33) WFLYCLINF0001: Activating Infinispan subsystem.
14:30:25,360 INFO [org.jboss.as.naming] (ServerService Thread Pool -- 39) WFLYNAM0001: Activating Naming Subsystem
14:30:25,361 WARN [org.jboss.as.txn] (ServerService Thread Pool -- 45) WFLYTX0013: The node-identifier attribute on the /subsystem=transactions is set to the default value. This is a danger for environments running multiple servers. Please make sure the attribute value is unique.
14:30:25,368 INFO [org.jboss.as.jaxrs] (ServerService Thread Pool -- 34) WFLYRS0016: RESTEasy version 3.6.1.Final
14:30:25,373 INFO [org.jboss.as.security] (ServerService Thread Pool -- 44) WFLYSEC0002: Activating Security Subsystem
14:30:25,376 INFO [org.jboss.as.mail.extension] (MSC service thread 1-6) WFLYMAIL0002: Unbound mail session [java:jboss/mail/Default]
14:30:25,377 INFO [org.jboss.as.security] (MSC service thread 1-6) WFLYSEC0001: Current PicketBox version=5.0.3.Final
14:30:25,397 INFO [org.wildfly.extension.undertow] (MSC service thread 1-3) WFLYUT0003: Undertow 2.0.13.Final starting
14:30:25,399 INFO [org.jboss.as.connector] (MSC service thread 1-6) WFLYJCA0009: Starting JCA Subsystem (WildFly/IronJacamar 1.4.11.Final)
```

Then open a browser and go to <http://localhost:8080/auth>.



Welcome to **Keycloak**



Administration Console

Please create an initial admin user to get started.

Username

Password

Password confirmation

Create



Documentation >

User Guide, Admin REST API and Javadocs



Keycloak Project >



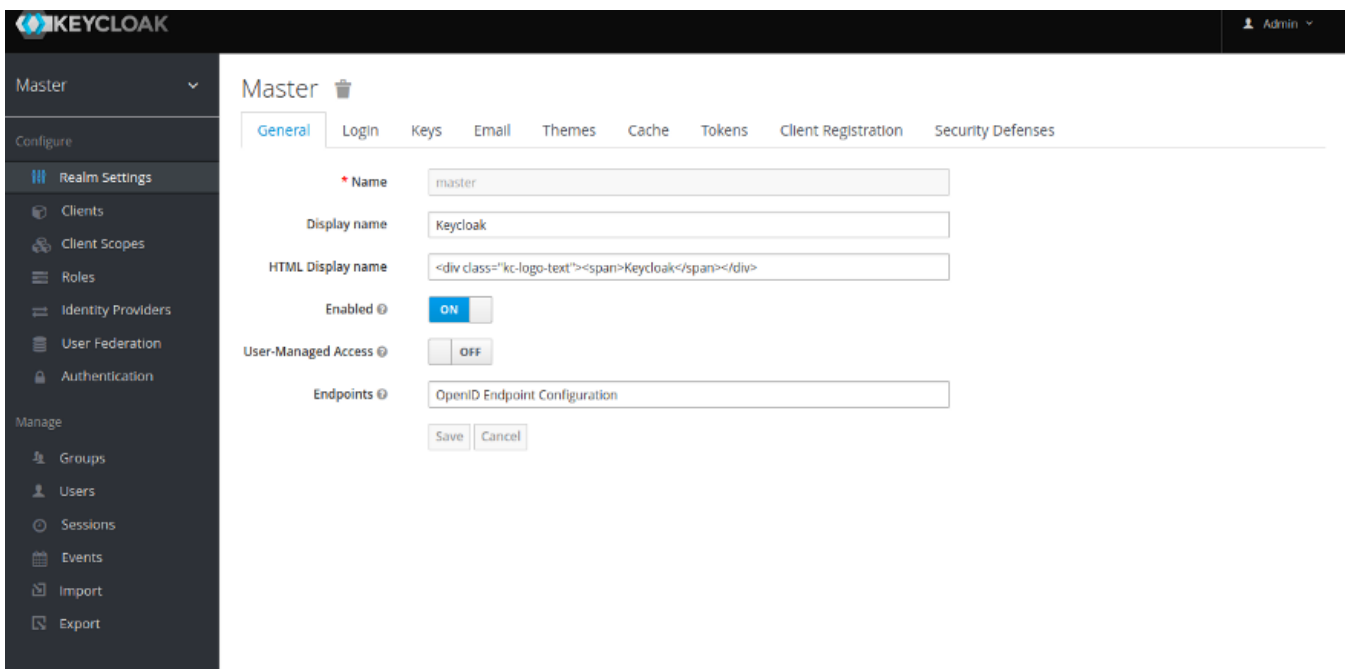
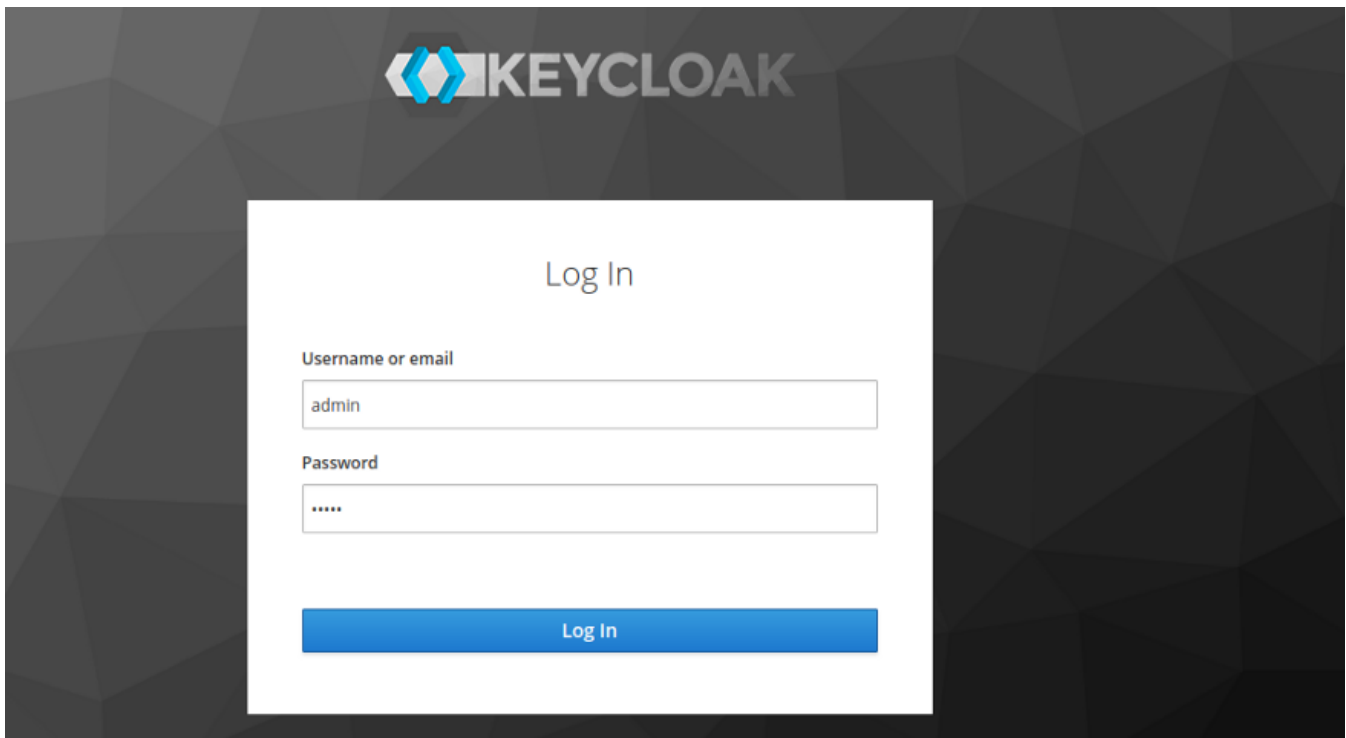
Mailing List >



Report an Issue >

Since it's the first time that the server runs you will have to create an admin user, so let's create an admin user with admin as username and admin for the password.

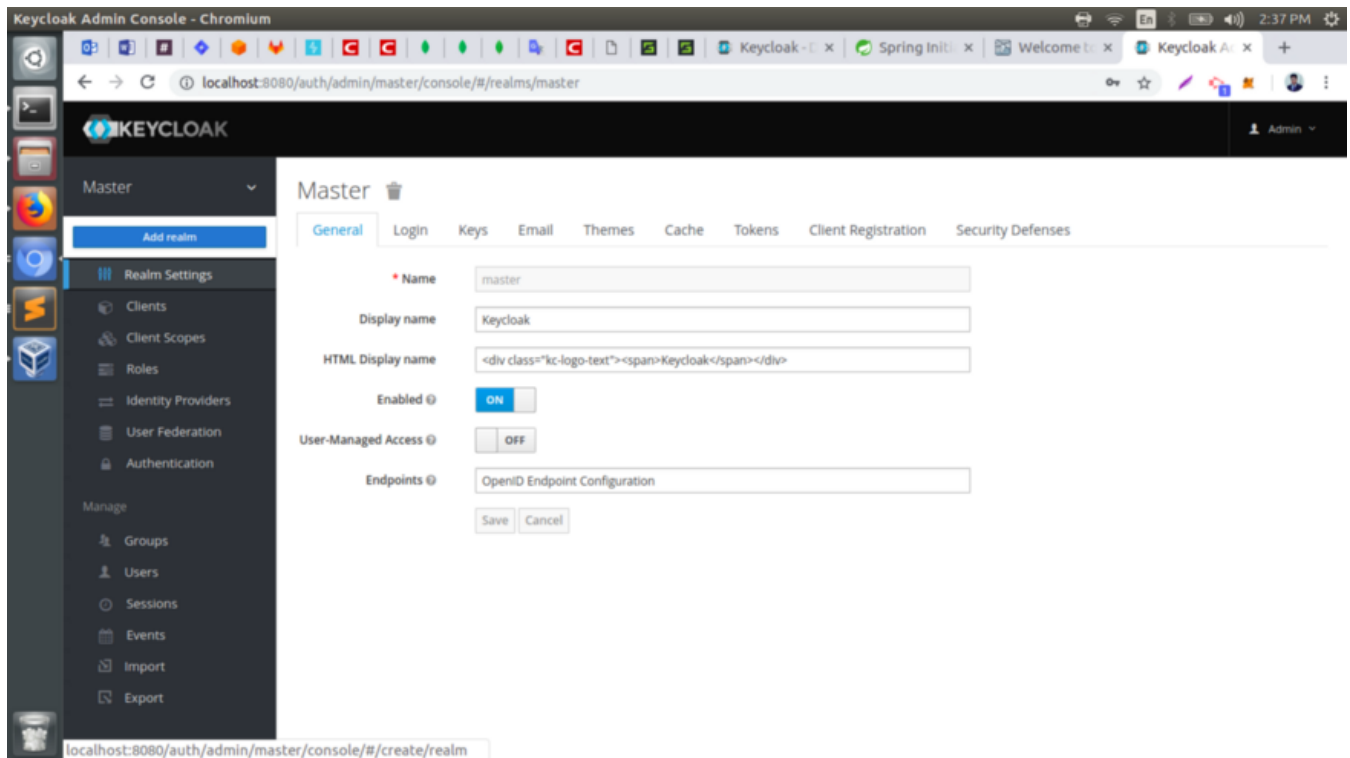
Now you can log in into your administration console and start configuring Keycloak.



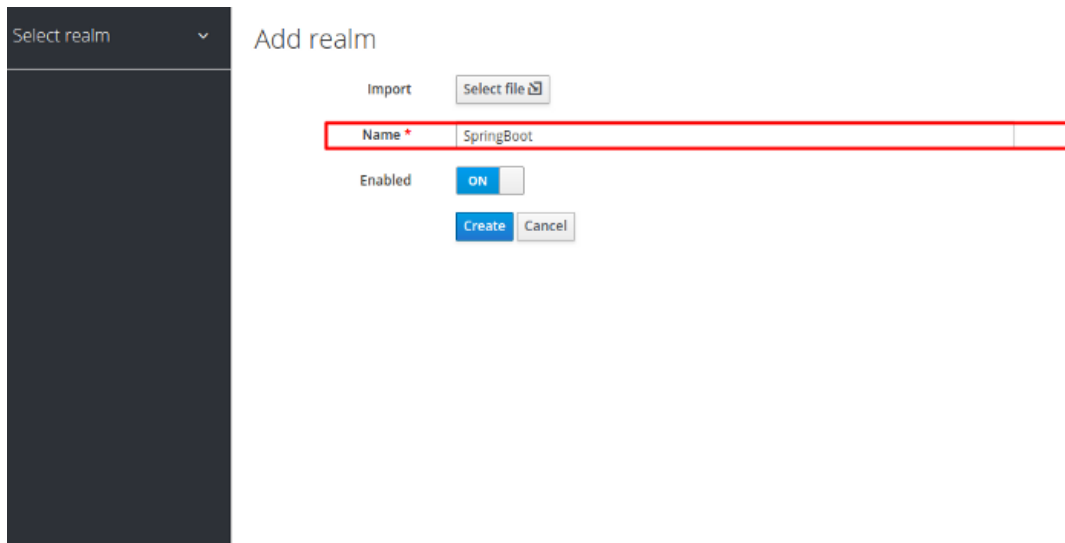
Creating a new Realm

Keycloak defines the concept of a realm in which you will define your clients, which in Keycloak terminology means an application that will be secured by Keycloak, it can be a Web App, a Java EE backend, a Spring Boot etc.

So let's create a new realm by simply clicking the “Add realm” button:

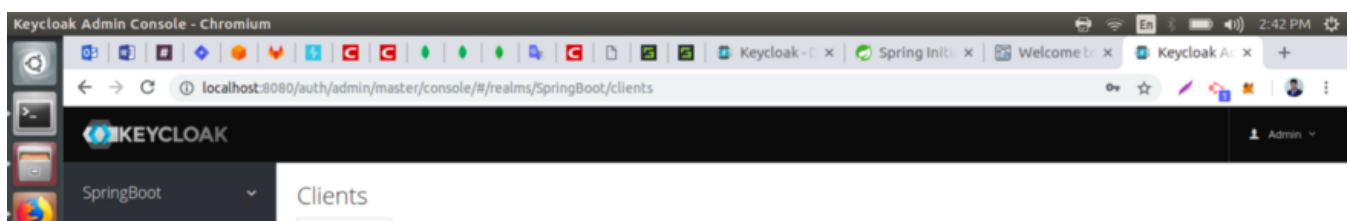


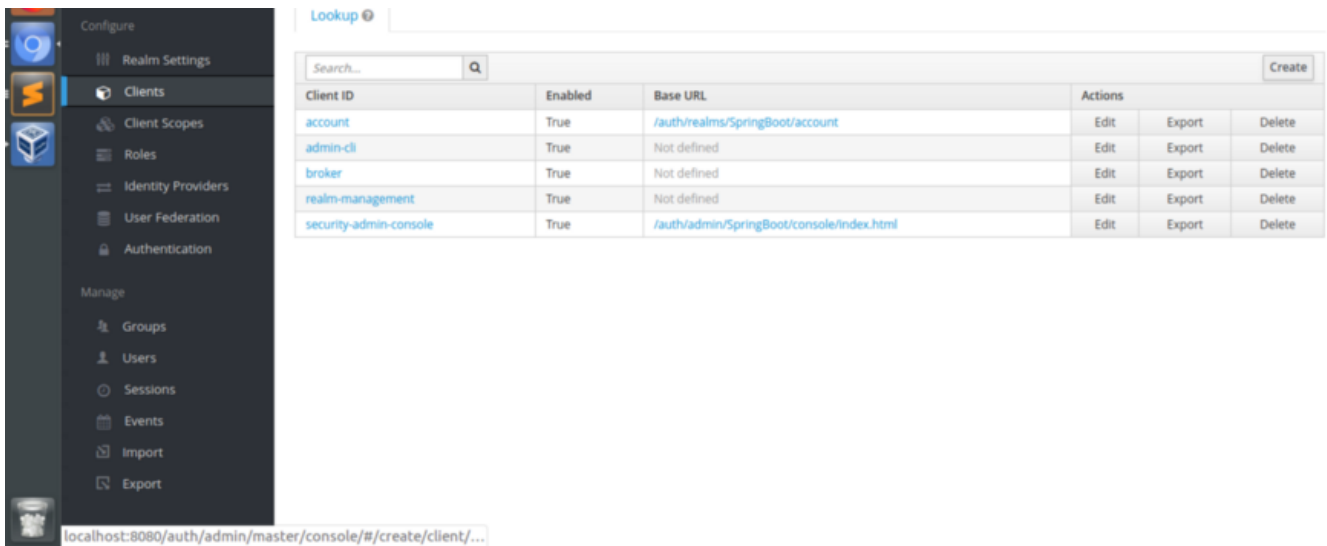
Let's call it “SpringBoot”.



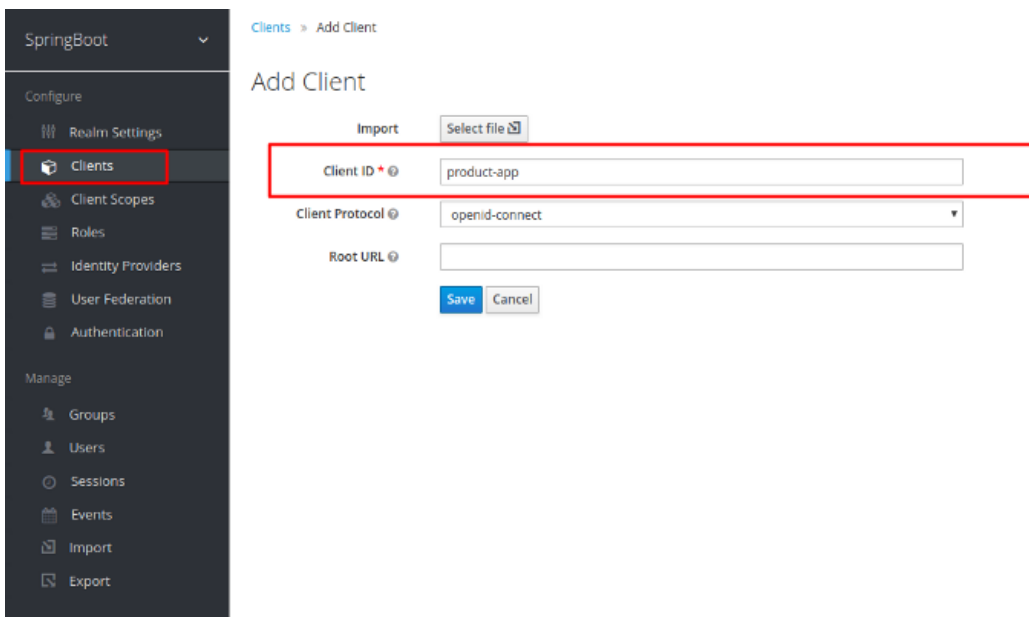
Creating the client, the role, and the user

Now we need to define a client, which will be our Spring Boot app. Go to the “Clients” section and click the “create” button. We will call our client “product-app”:





Client ID	Enabled	Base URL	Actions
account	True	/auth/realms/SpringBoot/account	Edit Export Delete
admin-cli	True	Not defined	Edit Export Delete
broker	True	Not defined	Edit Export Delete
realm-management	True	Not defined	Edit Export Delete
security-admin-console	True	/auth/admin/SpringBoot/console/index.html	Edit Export Delete



SpringBoot

Configure

- Realm Settings
- Clients**
- Client Scopes
- Roles
- Identity Providers
- User Federation
- Authentication

Manage

- Groups
- Users
- Sessions
- Events
- Import
- Export

Clients > Add Client

Add Client

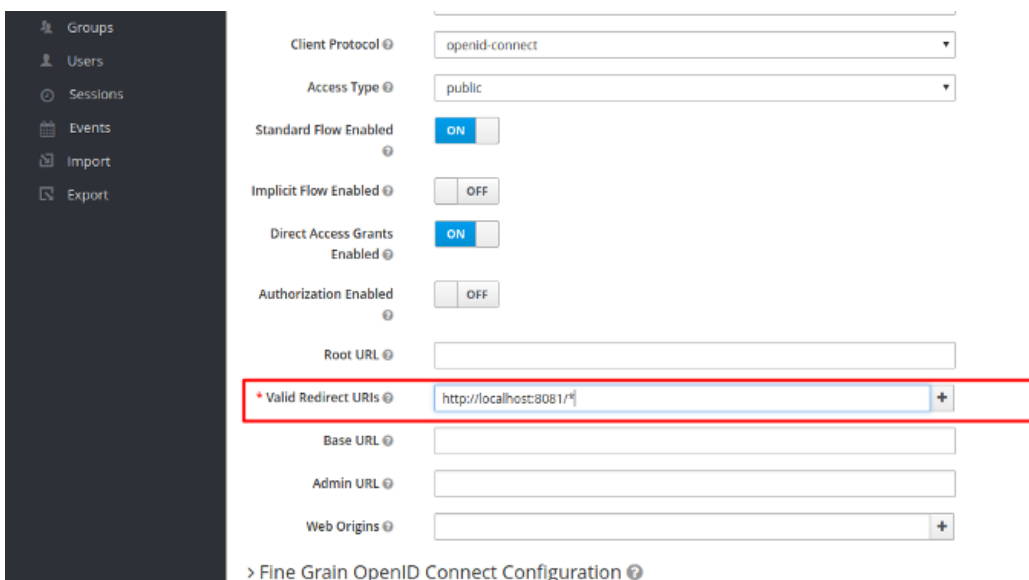
Import

Client ID *

Client Protocol

Root URL

On the next screen, we can keep the defaults settings but just need to enter a valid redirect URL that Keycloak will use once the user is authenticated. Put as value: “http://localhost:8081/*”



Groups

Users

Sessions

Events

Import

Export

Client Protocol

Access Type

Standard Flow Enabled ☒

Implicit Flow Enabled ☐

Direct Access Grants Enabled ☒

Authorization Enabled ☐

Root URL

* Valid Redirect URIs

Base URL

Admin URL

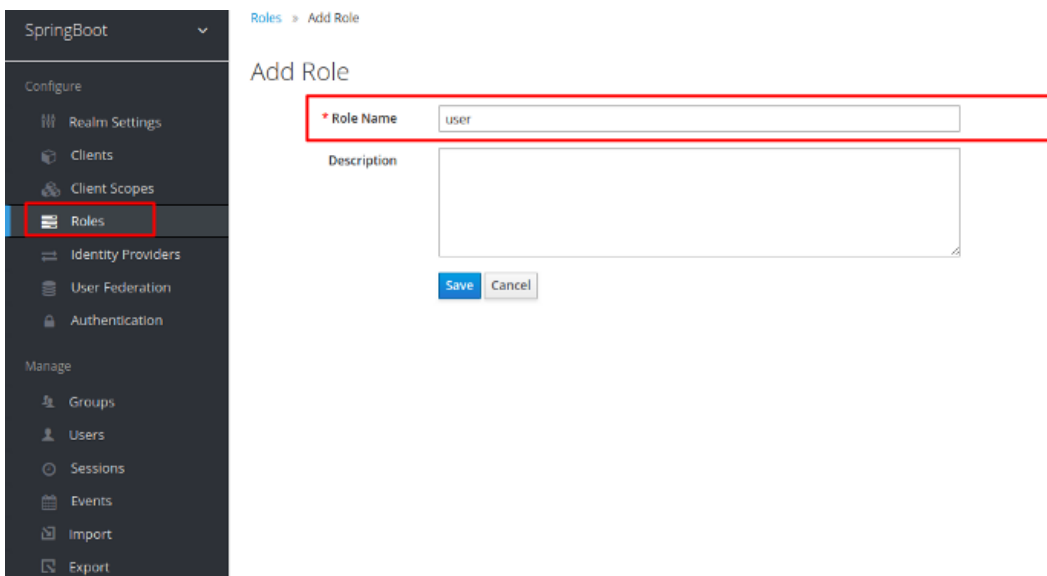
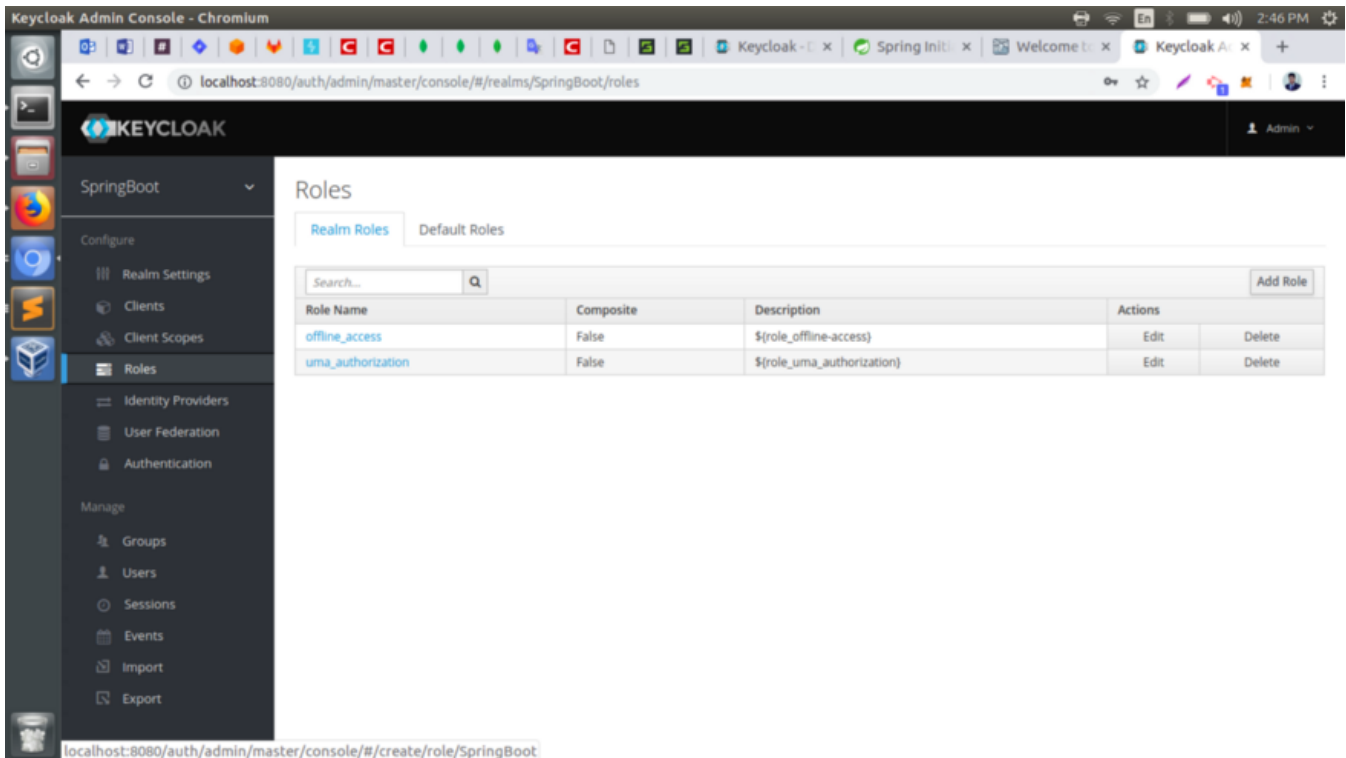
Web Origins

> Fine Grain OpenID Connect Configuration

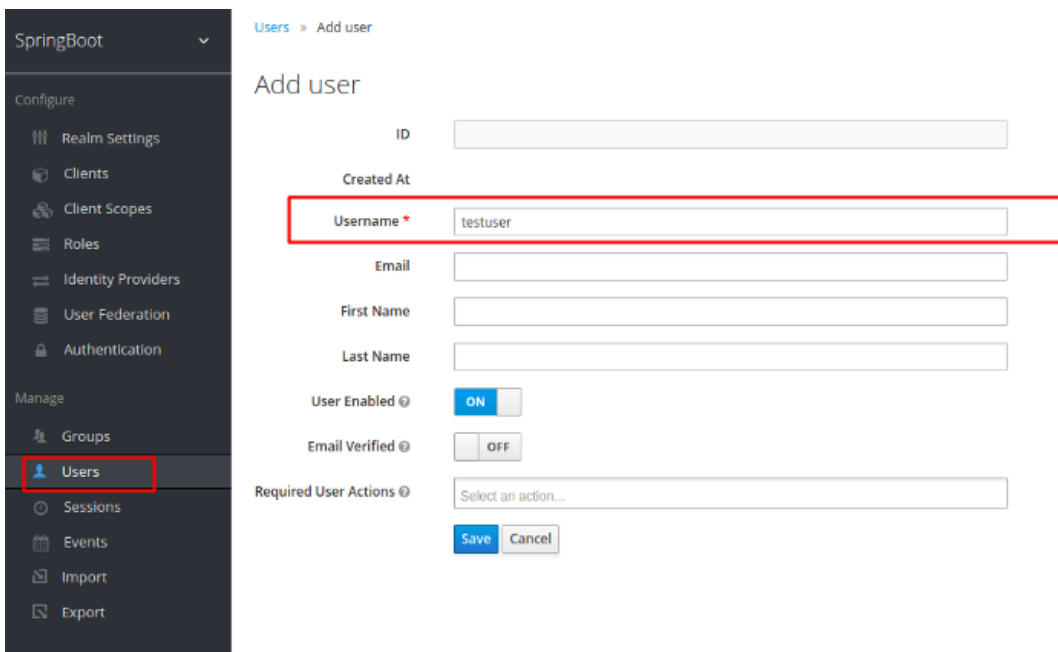
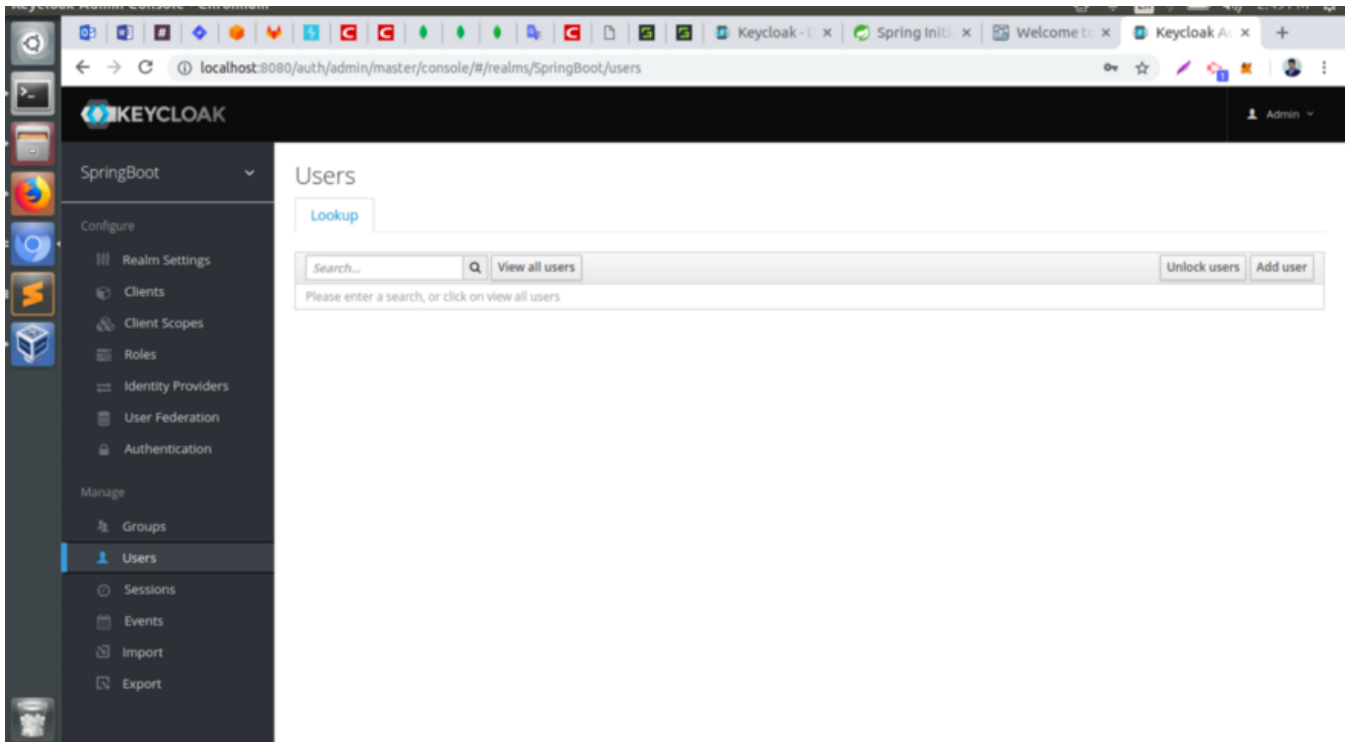
[> OpenID Connect Compatibility Modes ?](#)[> Advanced Settings ?](#)

Don't forget to Save!

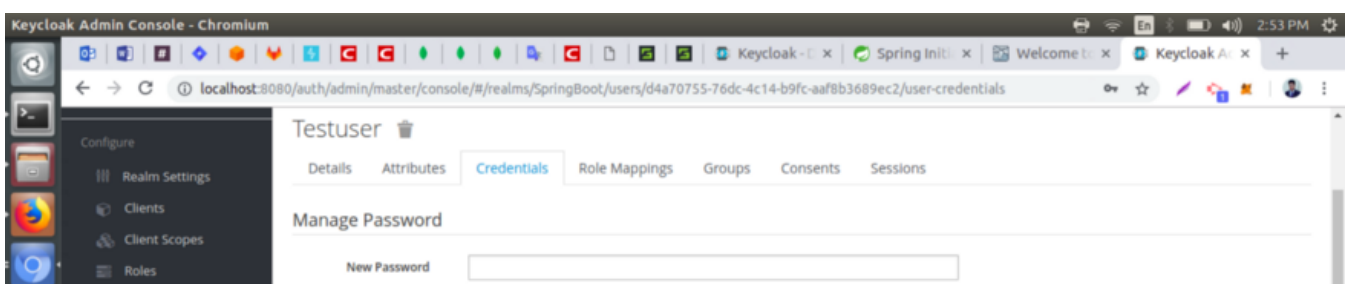
Now, we will define a role that will be assigned to our users, let's create a simple role called "user":

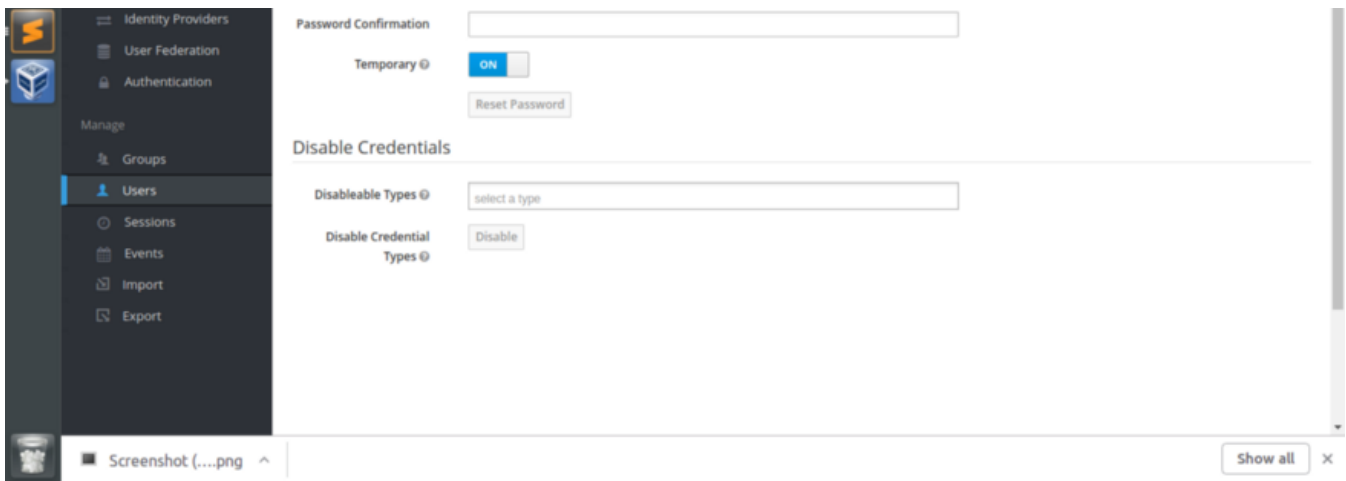


And at last but not least let's create a user, only the username property is needed, let's call him "testuser":

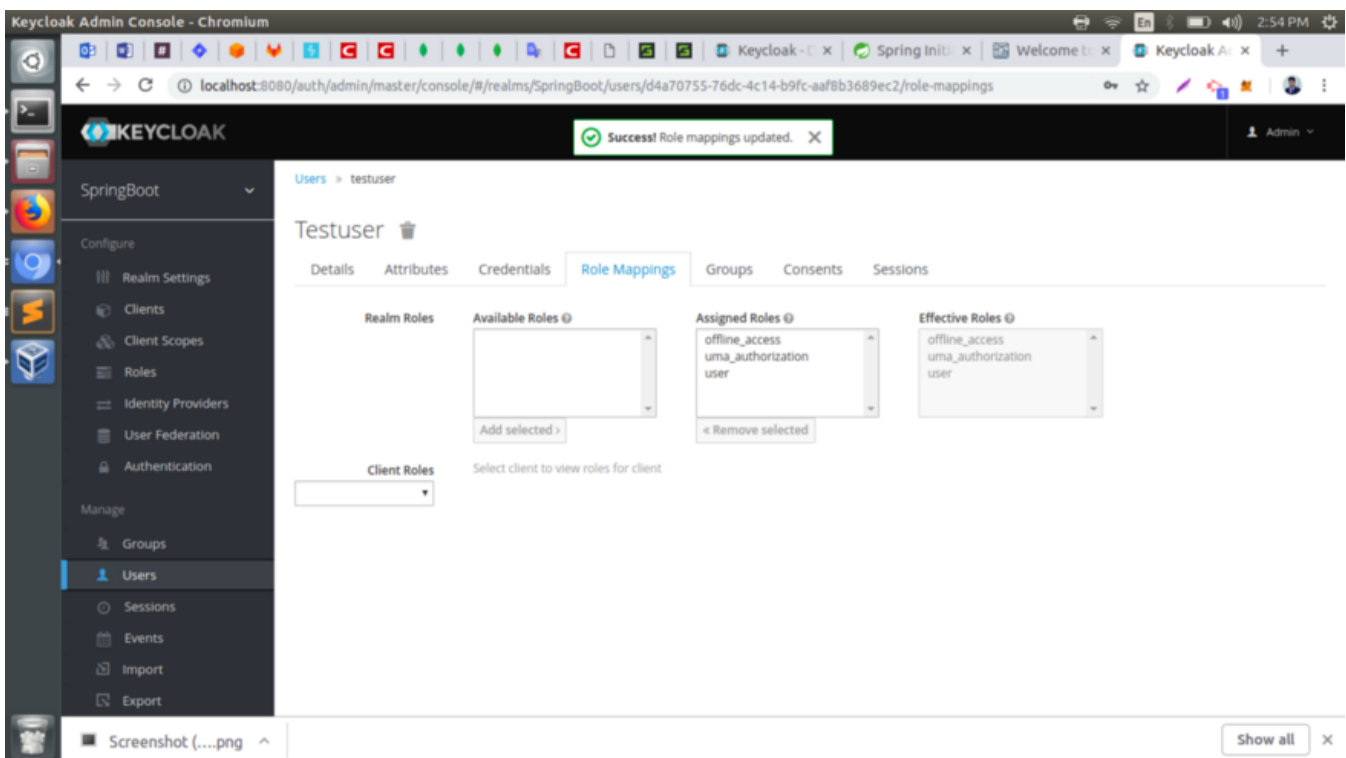


And finally, we need to set his credentials, so go to the credentials tab of your user and choose a password, I will be using “password”, make sure to turn off the “Temporary” flag unless you want the user to have to change his password the first time he authenticates.





Now proceed to the “Role Mappings” tab and assign the role “user”:



We are done for now with the Keycloak server configuration and we can start building our Spring Boot App!

Creating a simple app

Let's create a simple Spring Boot application, you might want to use the Spring Initializr and choose the following options:

Web

Freemarker

Keycloak

Name your app “product-app” and download the generated project:

Generate a Maven Project with Java and Spring Boot 1.5.17

Project Metadata
Artifact coordinates
Group

Artifact

Dependencies
Add Spring Boot Starters and dependencies to your application
Search for dependencies

Selected Dependencies
Web Freemarker Keycloak

[Generate Project alt + ⌘](#)

Don't know what to look for? Want more options? [Switch to the full version.](#)

<https://start.spring.io/>

<https://start.spring.io/>

Import the application in your favorite IDE/editor, I will be using Sublime.

Our app will be simple and will contain only 2 pages:

- 1 An index.html which will be the landing page containing just a link to the product page.
- 2 Products.ftl which will be our product page template and will be only accessible for authenticated user.

Let's start by creating in simple index.html file in “/src/resources/static”:

```
<html>
<head>
  <title>My awesome landing page</title>
</head>
<body>
  <h1>Landing page</h1>
  <a href="/products">My products</a>
</body>
</html>
```

Now we need a controller:

```
@Controller
class ProductController {

  @Autowired ProductService productService;
```

```
@GetMapping(path = "/products")
public String getProducts(Model model) {
    model.addAttribute("products", productService.getProducts());
    return "products";
}

@GetMapping(path = "/logout")
public String logout(HttpServletRequest request) throws
ServletException {
    request.logout();
    return "/";
}
}
```

As you can see, it's simple; we define a mapping for the product page and one for the logout action. You will also notice that we are calling a "**ProductService**" that will return a list of strings that will put in our Spring MVC Model object, so let's create that service:

```
@Component
class ProductService {
    public List<String> getProducts() {
        return Arrays.asList("iPad", "iPod", "iPhone");
    }
}
```

We also need to create the **products.ftl** template, create this file in "**src/resources/templates**":

```
<#import "/spring.ftl" as spring>
<html>
<h1>My products</h1>
<ul>
<#list products as product>
    <li>${product}</li>
</#list>
</ul>
<p>
    <a href="/logout">Logout</a>
</p>
</html>
```

Here we simply iterate through the list of products that are in our Spring MVC Model object and we add a link to log out from our application.

All that is left is adding some keycloak properties in our **application.properties**.

Defining Keycloak's configuration

Some properties are mandatory:

```
keycloak.auth-server-url=http://localhost:8080/auth
keycloak.realm=springboot
keycloak.public-client=true
keycloak.resource=product-app
```

Then we need to define some Security constraints as you will do with a Java EE app in your web.xml:

```
keycloak.security-constraints[0].authRoles[0]=user
keycloak.security-
constraints[0].securityCollections[0].patterns[0]=/products/*
```

Here, we simply define that every request to /products/* should be done with an authenticated user and that this user should have the role "user".

One last property is to make sure our application will be running on port 8081:

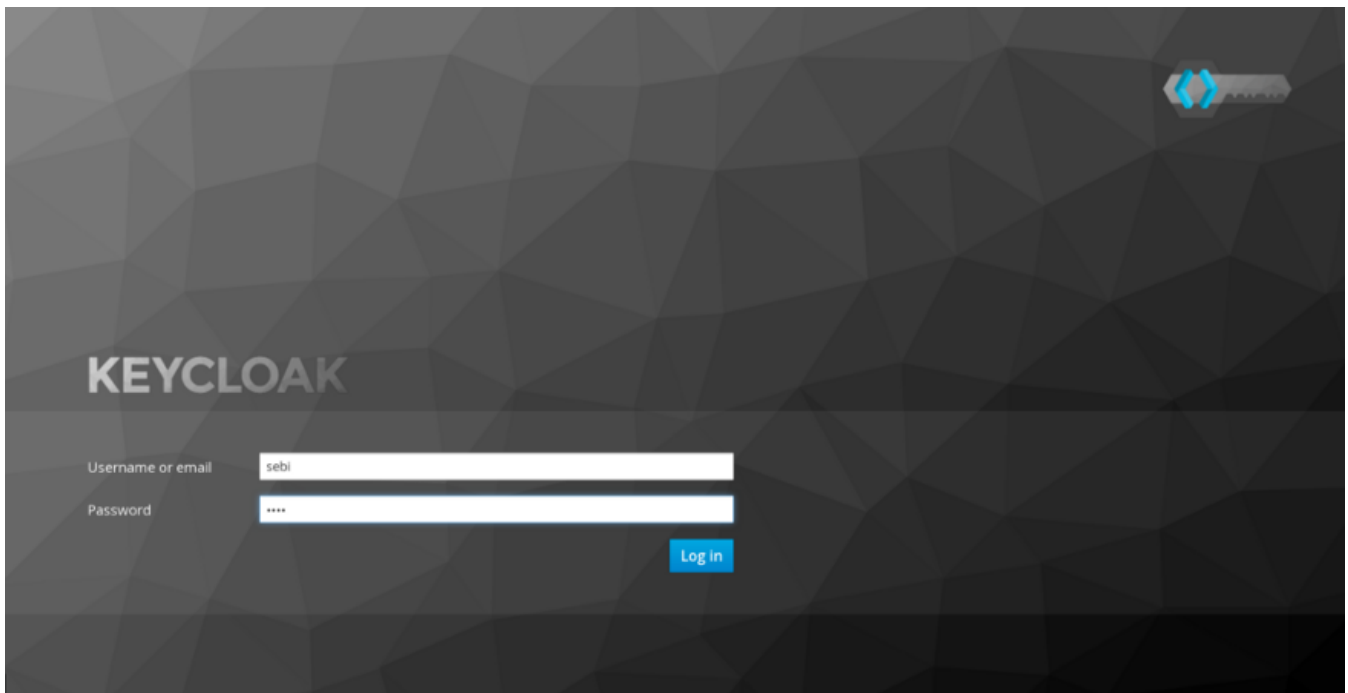
```
server.port=8081
```

We are all set and we can run our app!

You have several options to run your Spring Boot application, with Maven you can simply do:

```
mvn clean spring-boot:run
```

Now browse to “<http://localhost:8080>” and you should see the landing page, click the “products” links and you will be redirected to the Keycloak login page:



Login with our user “testuser/password” and should be redirected back to your product page:



Congratulations! You have secured your first Spring Boot app with Keycloak.

Conclusion

We saw in this article how to deploy and configure a Keycloak Server and then secure a Spring Boot app.

[Spring Boot](#) [Java](#) [Keycloak](#) [Security](#)

[About](#) [Write](#) [Help](#) [Legal](#)

Get the Medium app

