



[New Contributor Contest] Gana un iPad Mini, un Amazon E...

[Leer los detalles del concurso](#)

Mejores prácticas recomendadas de la API REST

por Vladimir Pecanac MVB · 31 de enero, 18 · Zona de integración

Modernice sus arquitecturas de aplicaciones con microservicios y API con las mejores prácticas de esta serie gratuita de cumbres virtuales. Presentado en asociación con CA Technologies .

Muchos gigantes como Facebook, Google, GitHub, Netflix, Amazon y Twitter tienen sus propias API REST (ful) a las que puedes acceder para obtener o incluso escribir datos.

¿Pero por qué toda la necesidad de REST?

¿Es tan bueno y por qué es tan frecuente?

Seguramente no es la única forma de transmitir mensajes?

¿Cuál es la diferencia entre REST y HTTP?

Bueno, resulta que **REST es bastante flexible y compatible con HTTP** (que es el protocolo principal en el que se basa Internet). Como es un estilo arquitectónico y no el estándar, **proporciona mucha libertad para implementar diversas mejores prácticas de diseño** . ¿Mencioné que es independiente del idioma?

En esta publicación de blog, mi objetivo será explicar REST de la manera más clara posible para que pueda comprender claramente cuándo y cómo usarlo, así como también lo que es en esencia.

Analizaremos algunos conceptos básicos y definiciones, además de mostrar algunas de **las mejores prácticas de REST API** . Esto debería darle todo el conocimiento que necesita para implementar API REST en cualquier idioma en el que prefiera codificar.

Si no está familiarizado con HTTP, le recomiendo leer nuestra serie HTTP , o al menos parte 1 , para que pueda digerir este material más fácilmente.

Entonces, ¿qué es REST esencialmente?

REST (Representational State Transfer) es un estilo arquitectónico fundado por Roy Fielding en su Ph.D. disertación " Estilos arquitectónicos y el diseño de arquitecturas de software basadas en red " en UC Irvine. Lo desarrolló en paralelo con HTTP 1.1 (sin presión).

Usamos REST principalmente como una forma **de comunicarnos entre sistemas informáticos en la World Wide Web** .

¿Está REST enlazado a HTTP?

Por definición, no lo es. Aunque puede utilizar algún otro protocolo de aplicación con REST, HTTP ha seguido siendo el campeón indiscutido entre los protocolos de aplicación cuando se trata de la implementación de REST.

¿Qué significa RESTful API?

"RESTful" implica algunas características:

- **Arquitectura cliente-servidor** : el servicio completo está compuesto por el "Cliente" que es el front-end y el "Servidor" que es la parte de back-end de todo el sistema.
- **Sin estado**: el servidor no debe guardar ningún estado entre diferentes solicitudes. El estado de la sesión se deja exclusivamente a la

responsabilidad del cliente. Según la definición de REST: ***todas las interacciones REST son sin estado. Es decir, cada solicitud contiene toda la información necesaria para que un conector comprenda la solicitud, independientemente de cualquier solicitud que pueda haberle precedido. (Disertación de Roy, capítulo 5.2.2)***

- **Cacheable** : el cliente debe poder almacenar las respuestas en un caché para un mayor rendimiento.

Entonces, la API RESTful es un servicio que sigue estas reglas (con suerte) y usa métodos HTTP para manipular el conjunto de recursos.

¿Pero por qué necesitamos o usamos API RESTful?

Porque nos brindan una manera fácil, flexible y escalable de realizar aplicaciones distribuidas que se comunican a través de Internet.

¿Podemos tener demasiado RESTO?

Sí, lo has adivinado. Si podemos.

Incluso hay una frase para las personas que siguen REST fanáticamente, como lo define Mike Schinkel.

Un RESTifarian es un ferviente defensor del estilo arquitectónico del software REST , tal como lo definió Roy T. Fielding en el Capítulo 5 de su Ph.D. disertación en UCIrvine . Puede encontrar RESTifarians en la naturaleza en la lista de correo de REST-discuss . Pero tenga cuidado, los RESTifarians pueden ser extremadamente meticulosos al discutir los puntos más delicados de REST, como aprendí recientemente

mientras participaba en la lista.

Demasiado de todo puede ser malo.

Necesitamos un poco de **pragmatismo** para hacer buenas aplicaciones y servicios. Es importante conocer y comprender una teoría, pero la implementación de esa teoría es lo que diferencia las aplicaciones malas de las buenas frente a las excelentes. Así que sé inteligente, ten en cuenta al usuario final.

Así que veamos algunos puntos importantes que hacen que las API "brillen" y que la vida de los usuarios sea mucho más fácil.

Resumen vs API concretas

Cuando desarrollamos software, a menudo usamos abstracción y polimorfismo para obtener la mayoría de nuestras aplicaciones. Queremos reutilizar la mayor cantidad de código posible.

Entonces, ¿deberíamos escribir nuestras API de esa manera también?

Bueno, ese no es exactamente el caso con las API. Para las API REST, el **concreto es mejor que abstracto**. ¿Puedes adivinar por qué?

Déjame mostrarte algunos ejemplos.

Veamos dos versiones de API. ¿Es mejor tener una API que tenga una `/entities` o una API que tenga `/owners`, `/blogs` y, por `/blogposts` separado?

¿Cuál te parece más descriptivo como desarrollador?
¿Qué API preferirías usar?

Yo siempre elegiría el segundo.

Formato URI (sustantivos, no verbos): buena URL vs malos ejemplos de URL

Aquí hay otra práctica recomendada REST API.
¿Cómo debe formatear sus endpoints?

Si usa el enfoque de desarrollo de software, terminará con algo como esto:

`/ getAllBlogPosts`

`/ updateBlogPost / 12`

`/ deleteBlogPost / 12`

`/ getAuthorById / 3`

`/ deleteAuthor / 3`

`/ updateAuthor / 3`

Entiende el punto ... Habrá una tonelada de puntos finales, cada uno haciendo otra cosa. Hay un mejor sistema para resolver este desastre.

Trate el recurso como un sustantivo, y el método HTTP como un verbo. Si lo haces así, terminarás con algo como esto:

`GET /blogposts` - obtiene todas las publicaciones del blog.

`GET /blogposts/12` - obtiene la publicación del blog con la id 12.

`POST /blogposts` - agrega una nueva publicación de blog y devuelve los detalles.

`DELETE /blogposts/12` - Elimina la publicación de blog con la identificación 12.

`GET /authors/3/blogposts` - obtiene todas las publicaciones del blog del autor con id 3.

Esta es una manera más limpia y más precisa de usar la API. Está claro para el usuario final, y hay un método para la locura.

Puede hacerlo aún más limpio utilizando singulares en lugar de plurales para los nombres de recursos. Ese depende de usted.

Manejo de errores

Este es otro aspecto importante de la construcción de

Este es otro aspecto importante de la construcción de API. Hay algunas buenas formas de manejar los errores.

Veamos cómo lo hacen los mejores perros.

Gorjeo:

- Solicitud: GET
https://api.twitter.com/1.1/account/settings.json
- Respuesta: código de estado 400

```
1 { "errors" : [{ "code" : 215 , "message" : "Da
```

Twitter le proporciona el código de estado y el código de error con una breve descripción de la naturaleza del error que ocurrió. Dependen de usted buscar los códigos en su página de códigos de respuesta .

Facebook:

- Solicitud: GET
https://graph.facebook.com/me/photos
- Respuesta: código de estado 400

```
1 {
2   "error" : {
3     "mensaje" : "Se debe usar un token de acc
4     "tipo" : "OAuthException" ,
5     "código" : 2500 ,
6     "fbtrace_id" : "DzkTMkgIA7V"
7   }
8 }
```

Facebook te da un mensaje de error más descriptivo.

Twilio:

- Solicitud:
- GET https://api.twilio.com/2010-04-01/Accounts/1234/IncomingPhoneNumbers/1234
- Respuesta: código de estado 404

```
1 <? xml version = '1.0' encoding = 'UTF-8'?>
2 < TwilioResponse >
3   < RestException >
4     < Code > 20404 </ Code >
```

```
4      < Mensaje > No se encontró el recurso s
5      < MoreInfo > https://www.twilio.com/doc
6      < Estado > 404 </ Estado >
7      </ RestException >
8      </ TwilioResponse >
```

Twilio le proporciona una respuesta XML por defecto y el enlace a la documentación donde puede encontrar los detalles del error.

Como puede ver, los enfoques para el manejo de errores difieren de la implementación a la implementación.

Lo importante es **no dejar al usuario de la API REST "colgado"**, es decir, sin saber lo que sucedió ni vagar sin rumbo por los desechos de StackOverflow en busca de la explicación.

Códigos de estado

Al diseñar una API REST, se comunica con el usuario de API mediante el uso de códigos de estado HTTP . Hay muchos códigos de estado que describen múltiples respuestas posibles.

Pero, ¿cuántos debe usar? **¿Debería tener un código de estado estricto para cada situación?**

Al igual que con muchas cosas en la vida, el principio KISS se aplica aquí también. Hay más de 70 códigos de estado por ahí. ¿Los conoces de memoria? ¿El potencial usuario de la API los conocerá a todos, o volverá a generar contenido en Google?

La mayoría de los desarrolladores están familiarizados con los códigos de estado más comunes:

- 200 OK
- 400 Bad Request
- 500 Internal Server Error

Al comenzar con estos tres, puede cubrir la mayoría de las funcionalidades de su API REST.

Otros códigos comúnmente vistos incluyen:

- 201 Created
- 204 No Content
- 401 Unauthorized
- 403 Forbidden
- 404 Not Found

Puede usar estos para ayudar al usuario a descubrir rápidamente cuál fue el resultado. Probablemente deberías incluir algún tipo de mensaje si crees que el código de estado no es lo suficientemente descriptivo como discutimos en la sección de manejo de errores. Una vez más, sea pragmático, ayude al usuario mediante el uso de una **cantidad limitada de códigos** y mensajes descriptivos.

Puede encontrar la lista de códigos de estado HTTP completa, así como otras cosas útiles de HTTP aquí .

Seguridad

No hay mucho que decir sobre la seguridad de la API **REST** porque **REST no se ocupa de la seguridad** . Se basa en mecanismos HTTP estándar como la autenticación básica o digestiva .

Toda solicitud debe hacerse a **través de HTTPS** .

Existen muchos trucos para mejorar la seguridad de su API REST, pero debe tener cuidado al implementarlos, debido a la naturaleza apátrida de REST. Recordar el estado de la última solicitud sale de la ventana, y el **cliente está donde se debe almacenar y verificar el estado**.

Las solicitudes de **sellado y registro de tiempo también** pueden ayudar.

Hay mucho más que decir sobre este tema, pero está fuera del alcance de esta publicación. Tenemos una buena publicación sobre **Seguridad HTTP** si desea obtener más información al respecto.

RESE API Versioning

Ya ha escrito su API REST y ha tenido mucho éxito y muchas personas la han utilizado y están contentas con ella. Pero tienes esa nueva y jugosa funcionalidad

que rompe otras partes del sistema. El cambio de ruptura.

¡No temas, hay una solución para eso!

Antes de comenzar a crear su API, puede versionar su API prefijando los puntos finales por la versión de la API:

<https://api.example.com/v1/authors/2/blogposts/13>

De esta forma, siempre puede incrementar el número de versión de su API (por ejemplo, v2, v3 ...) siempre que haya cambios importantes en su API. Esto también les indica a los usuarios que algo drástico ha cambiado y deben tener cuidado al usar la nueva versión.

Importancia de la documentación

Este es un obvio. Podrías ser el mejor diseñador de API en el mundo, pero **sin documentación, tu API está muerta.**

La documentación adecuada es esencial para cada producto de software y servicio web por igual.

Puede ayudar al usuario siendo consistente y usando una sintaxis clara y descriptiva, claro. Pero no hay un reemplazo real para las páginas de documentación buenas.

Estos son algunos de los mejores ejemplos:

- <https://www.twilio.com/docs/api/rest/>
- <https://developers.facebook.com/docs/>
- <https://developers.google.com/maps/documentation/>

Hay muchas herramientas que pueden ayudarlo a documentar su API, pero no olvide agregar el toque humano, solo un ser humano puede entender correctamente a otro. Por ahora al menos (mirándote a ti).

Conclusión

Revisamos muchos conceptos del desarrollo de la API REST y cubrimos algunas de las mejores prácticas recomendadas de REST API. Estos pueden parecer un poco extraños o abrumadores cuando se sirven a la vez, pero intente crear su propia API REST. Y trate de implementar algunas de las mejores prácticas de REST API que aprendió aquí.

Haga la API más pequeña posible y vea cómo se ve. Se sorprenderá de lo bien que puede resultar simplemente siguiendo estas pocas prácticas.

Tenemos una serie continua en la que demostraremos algunas de estas prácticas. Además, si es un desarrollador de C #, consulte nuestro artículo sobre cómo consumir API RESTful de diferentes maneras .

Y con esto dicho, **DESCANSO mi caso ...** kmhnh ... ya sabes, como en la corte ... oh, no importa :)

¿Hay algo importante que me haya perdido? Si lo hice, iházmelo saber en la sección de comentarios!

La Zona de Integración está orgullosamente patrocinada por CA Technologies . Aprenda de microservicios expertos y presentaciones API en Modernizing Application Architectures Virtual Summit Series .

Temas: INTEGRACIÓN, API DE
DESCANSO, DOCUMENTACIÓN DE LA API DE REPOSO,
MANEJO DE ERRORES

Publicado en DZone con permiso de Vladimir Pecanac, DZone MVB . [Vea el artículo original aquí.](#)



Las opiniones expresadas por los contribuidores de DZone son suyas.

Recursos para socios de integración

Modernizar arquitecturas de aplicaciones con microservicios y API

CA Technologies



Construye plataformas, no solo productos: usa los 4 superhéroes de la economía de red [eBook]

Elementos de la nube



EBook de estrategia y arquitectura API

CA Technologies



La guía definitiva para las integraciones de API: explore las integraciones de API debajo de la superficie [eBook]

Elementos de la nube



Kanban Conceptos erróneos y mitos

por Anna Majowska MVB · 31 de enero de 18 · Zona Ágil

Vea cómo tres soluciones trabajan juntas para ayudar a sus equipos a tener las herramientas que necesitan para entregar un software de calidad rápidamente . Presentado en asociación con CA Technologies .

"Kanban no es realmente un método de flujo de trabajo. Es una herramienta de mejora de procesos".

A menudo se dice que Kanban es una herramienta de construcción de conocimiento, a diferencia de un método de gestión de flujo de trabajo. Sí, Kanban agrega nuevas formas de ver qué implica su proceso e informa a todos los involucrados sobre las etapas actuales de los artículos. Y sí, se puede decir, que Kanban no es un método de gestión del flujo de trabajo, ya que no nombra procedimientos específicos de cómo trabajar, sino que es una adición visual a todo lo que ya está haciendo. Agregar Kanban a su método actual tiene un gran impacto en la forma de trabajar y, por esa razón, a menudo se lo denomina un método en sí mismo. También hay una sorprendente cantidad de equipos que no tienen un

metodo de trabajo prescrito, pero que usan Kanban para ayudar en sus esfuerzos. ¿Es un error decir que *usan el método Kanban* ? ¿Es un mito? Podría argumentarse, que si Kanban es un método o no es solo un **argumento técnico** , que tiene poco impacto real sobre cómo trabajan las personas.

"Kanban escoge los pasos del proceso que nos gustan y nos permite omitir los que nosotros no"

That is not correct, to say the least. What you want is to name all known steps in the process, and then regularly review them. Do other stages exist that were not included at first? Do you find that there are stages that are unnecessary, cause waste, and should be removed? Kanban is meant to facilitate making these changes as they occur — always — without you ever reaching a point of saying "*This is it: our process from now until forever after.*" In that sense, Kanban is indeed a knowledge making tool. Knowledge of your process, its state, strengths, faults and — this is the key word here — **improvement opportunities**.

"There Is No Breathing Room in Kanban"

Some approaches allow slack time for workers, mainly because of the processes asynchronous nature. While team members wait for items to reach them from previous steps, time is gained to take a moment to relax, learn or be creative. It may then seem, that if we visualize where each item is in the process — and can therefore see what each individual worker is doing, there is no way of finding any *breathing room*. It doesn't have to be this way — for as long as teams are made aware that they can still take time to relax and learn new things, not having an *in-progress* item on the board is not the end of the world. It's more a **matter of internal policy** than Kanban itself.

"Kanban Only Supports Linear Processes"

The fact that a Kanban board presents workflow

stages as linear, doesn't mean it necessarily puts any limitations on how the work is done. The board is there to visualize current stages of each item, not to limit the way they are being worked on. Again, it's a visualization tool, not an oracle.

"WIP Limits Remove Flexibility and Ability to Respond Quickly"

The misunderstanding here is that having reached the top of one's WIP limits stops you from being able to respond to urgent requests. It doesn't have to. The idea of WIP limit is to control the number of things done **simultaneously**. It doesn't have to mean you can't switch the permitted no. of tasks from 2 regular to 2 urgent if need be.

"We Can't Use Time-Boxes in Kanban"

Lack of time-boxes is typically named as one of the distinctions between Kanban and Scrum. Even if you don't apply a strict time-frame to projects, you can still easily learn of how long they took to complete, from looking at Lead & Cycle Time and Cumulative Flow diagrams.

Though it's not a time-box, the information of when you should expect a certain type of project to be done can be based on past speed. It's therefore a benefit, if the assumption behind a time-box is to control time expenditure, but not necessarily to put distinct pressure on the team.

Consider these before you make your mind up about the Kanban method — or Kanban non-method — whichever side you're on. The easiest way to learn the value of Kanban is to try it. We encourage everyone to test the approach for themselves with online Kanban Tool.

Discover how TDM Is Essential To Achieving Quality At Speed For Agile, DevOps, And Continuous Delivery. Brought to you in partnership with CA Technologies.

Published at DZone with permission of Anna Majowska, DZone MVB. [See the original article here.](#)



Opinions expressed by DZone contributors are their own.

Build Authentication the Easy Way with Spring Security 5.0 and OIDC

by Matt Raible MVB · Feb 02, 18 · Java Zone

“I love writing authentication and authorization code.”
~ No Developer Ever. Try Okta instead

Spring Security is not just a powerful and highly customizable authentication and access-control framework, it is also the de-facto standard for securing Spring-based applications. Once upon a time Spring Security required reams of XML to configure everything, but those days are long past. These days, Spring Security offers much simpler configuration via Spring's JavaConfig. If you look at the SecurityConfiguration.java class from the JHipster OIDC example I wrote about recently, you'll see it's less than 100 lines of code!

Spring Security 5.0 resolves 400+ tickets, and has a plethora of new features:

- OAuth 2.0 Login
- Reactive
Support: `@EnableWebFluxSecurity`, `@EnableReactiveMethodSecurity`, and WebFlux Testing Support
- Modernized Password Encoding

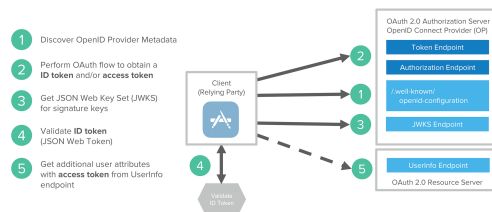
Today, I'll be showing you how to utilize the OAuth 2.0 Login support with Okta. I'll also show you to retrieve a user's information via OpenID Connect (OIDC).

You know that Okta offers free developer accounts with up to 7,000 active monthly users, right? That should be enough to get your killer app off the ground.

Spring Security makes authentication with OAuth 2.0 pretty darn easy. It also provides the ability to fetch a user's information via OIDC. Follow the steps below to learn more!

What is OIDC? If you're not familiar with OAuth or OIDC, I recommend you read What the Heck is OAuth. An Open ID Connect flow involves the following steps:

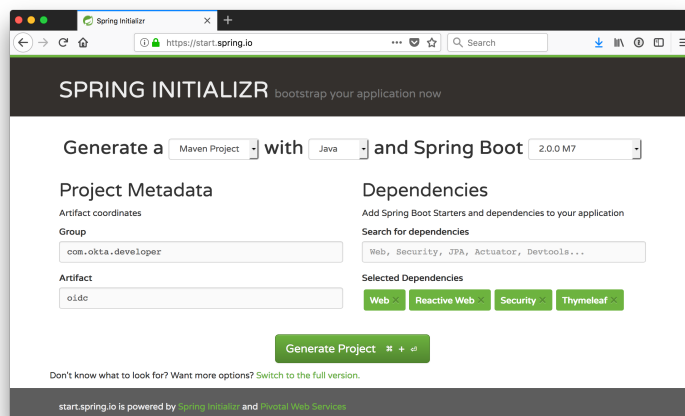
- 1. Discover OIDC metadata**
- 2. Perform OAuth flow to obtain ID token and access tokens**
- 3. Get JWT signature keys and optionally dynamically register the Client application**
- 4. Validate JWT ID token locally based on built-in dates and signature**
- 5. Get additional user attributes as needed with access token**



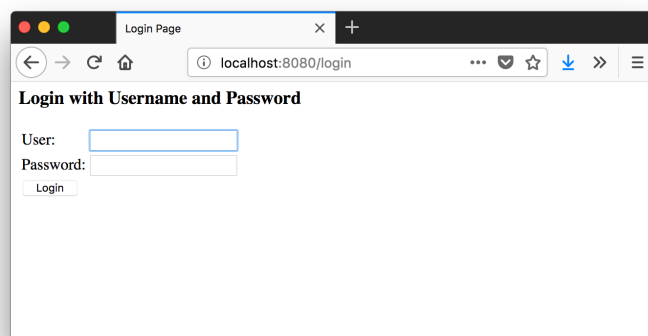
Create a Spring Boot App

Open start.spring.io in your browser. Spring Initializr is a site that allows you to create new Spring Boot applications quickly and easily. Set the Spring Boot version (in the top right corner) to `2.0.0.M7`. Type in a group and artifact name. As you can see from the screenshot below, I

chose `com.okta.developer` and `oidc`. For dependencies, select **Web**, **Reactive Web**, **Security**, and **Thymeleaf**.



Click **Generate Project**, download the zip, expand it on your hard drive, and open the project in your favorite IDE. Run the app with `./mvnw spring-boot:run`, and you'll be prompted to log in.



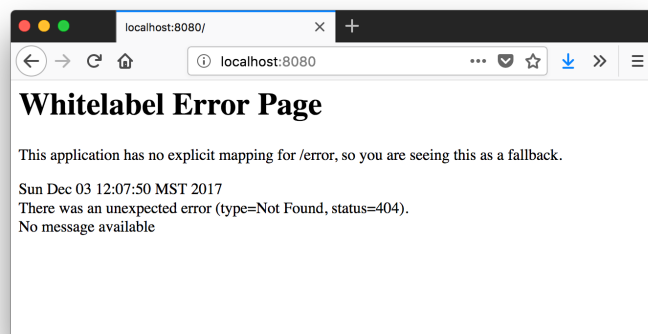
Spring Security 4.x prompts you with basic authentication rather than with a login form, so this is one thing that's different with Spring Security 5.

The Spring Security starter creates a default user with username "user" and a password that changes every time you start the application. You can find this password in your terminal, similar to the one below.

```
Using default security password: 103c55b4-2760-
```


1

In the form, enter “user” for the User and the generated password for Password. The next screen will be a 404 since your app doesn’t have a default route configured for the / path.



In Spring Boot 1.x, you could change the user’s password, so it’s the same every time by adding the following

to `src/main/resources/application.properties`.

1

```
security.user.password=spring security is ph@!
```

However, this is a deprecated feature in Spring Boot 2.0. The good news is this change will likely be reverted before a GA release.

In the meantime, you can copy the password that’s printed to your console and use it with HTTPie.

1

```
$ http --auth user:'bf91316f-f894-453a-9268-482
```

2

```
HTTP/1.1 404
```

3

```
Cache-Control: no-cache, no-store, max-age=0, n
```

4

```
Content-Type: application/json;charset=UTF-8
```

5

```
Date: Sun, 03 Dec 2017 19:11:50 GMT
```

6

```
Expires: 0
```

7

```
Pragma: no-cache
```

8

```
Set-Cookie: JSESSIONID=65283FCBDB9E6EF1C0679296
```

9

```
Transfer-Encoding: chunked
```

10

```
X-Content-Type-Options: nosniff
```

11

```
X-Frame-Options: DENY
```

```
12 X-XSS-Protection: 1; mode=block
```

The response will be a 404 as well.

```
1 {
2   "error": "Not Found",
3   "message": "No message available",
4   "path": "/",
5   "status": 404,
6   "timestamp": "2017-12-03T19:11:50.846+0000"
7 }
```

You can get rid of the 404 by creating a `MainController.java` in the same directory as `OidcApplication.java` (`src/main/java/com/okta/developer/oidc`). Create a `home()` method that maps to `/` and returns the user's name.

```
1 package com.okta.developer.oidc;
2
3 import org.springframework.web.bind.annotation.*;
4 import org.springframework.web.bind.annotation.*;
5
6 import java.security.Principal;
7
8 @RestController
9 public class MainController {
10
11     @GetMapping("/")
12     String home(Principal user) {
13         return "Hello " + user.getName();
14     }
15 }
```

Restart your server, log in with `user` and the generated password and you should see `Hello user` .

```
1 $ http --auth user:'d7c4138d-a1cc-4cc9-8975-97f
2 HTTP/1.1 200
3 Cache-Control: no-cache, no-store, max-age=0, n
4 Content-Length: 10
```

```

4 Content-Length: 10
5 Content-Type: text/plain; charset=UTF-8
6 Date: Sun, 03 Dec 2017 19:26:54 GMT
7 Expires: 0
8 Pragma: no-cache
9 Set-Cookie: JSESSIONID=22A5A91051B7AFBA1DC8BD30
10 X-Content-Type-Options: nosniff
11 X-Frame-Options: DENY
12 X-XSS-Protection: 1; mode=block
13
14 Hello user

```

Add Authentication With Okta

In a previous tutorial, I showed you how to use Spring Security OAuth to provide SSO to your apps. You can do the same thing in Spring Security 5, but you can also specify multiple providers now, which you couldn't do previously. Spring Security 5 has a OAuth 2.0 Login sample, and documentation on how everything works.

Create an OpenID Connect App

To integrate with Okta, you'll need to sign up for an account on developer.okta.com. After confirming your email and logging in, navigate

to **Applications > Add Application**.

Click **Web** and then click **Next**. Give the app a name you'll remember, specify `http://localhost:8080` as a Base URI, as well

as `http://localhost:8080/login/oauth2/code/okta` for a Login redirect URI.

Rename `src/main/resources/application.properties` to `src/main/resources/application.yml` and populate it with the following.

```

1 spring:
2   thymeleaf:
3     cache: false
4   security:
5     oauth2:
6       client:
7         registration:

```

```

7
8         okta:
9             client-id: {clientId}
10            client-secret: {clientSecret}
11        provider:
12            okta:
13                authorization-uri: https://{yourOktaDomain}/oauth2/authorize
14                token-uri: https://{yourOktaDomain}/oauth2/token
15                user-info-uri: https://{yourOktaDomain}/oauth2/userinfo
16                jwk-set-uri: https://{yourOktaDomain}/oauth2/keys

```

Copy the client ID and secret from your OIDC app into your `application.yml` file. For `{yourOktaDomain}`, use the value of your domain (e.g. `dev-158606.oktapreview.com`). Make sure it does *not* include `-admin` in it.

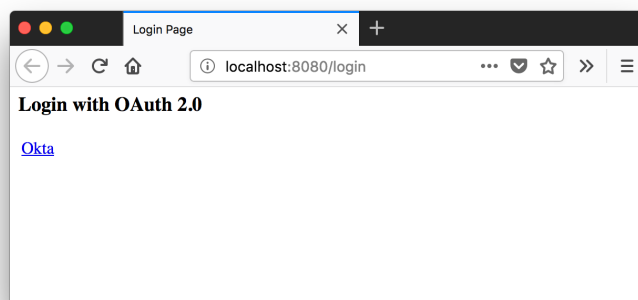
You'll need to add some dependencies to your `pom.xml` for Spring Security 5's OAuth configuration to initialize correctly.

```

1  <dependency>
2      <groupId>org.springframework.security</groupId>
3      <artifactId>spring-security-config</artifactId>
4  </dependency>
5  <dependency>
6      <groupId>org.springframework.security</groupId>
7      <artifactId>spring-security-oauth2-client</artifactId>
8  </dependency>
9  <dependency>
10     <groupId>org.springframework.security</groupId>
11     <artifactId>spring-security-oauth2-jose</artifactId>
12 </dependency>
13 <dependency>
14     <groupId>org.thymeleaf.extras</groupId>
15     <artifactId>thymeleaf-extras-springsecurity5</artifactId>
16 </dependency>

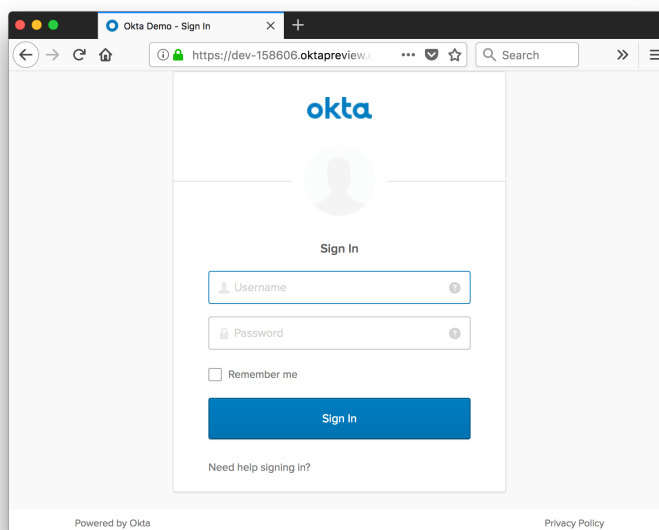
```

Restart your app and navigate to `http://localhost:8080` again. You'll see a link to click on to log in with Okta.

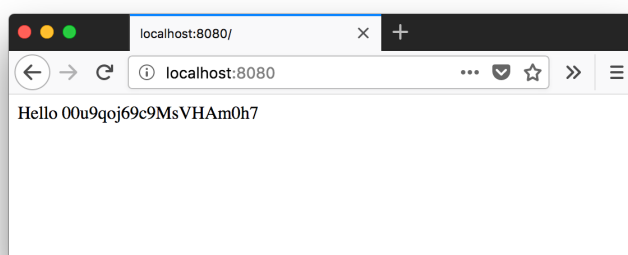


NOTE: If you'd like to learn how to customize the login screen that Spring Security displays, see its OAuth 2.0 Login Page documentation.

After clicking on the link, you should see a login screen.



Enter the credentials you used to create your account, and you should see a screen like the following after logging in.



NOTE: It's possible to change things

so `Principal#getName()` returns a different value.

However, there is a bug in Spring Boot 2.0.0.M7 that prevents the configuration property from working.

Get User Information With OIDC

Change your `MainController.java` to have the code below. This code adds a `/userinfo` mapping that uses Spring WebFlux's `WebClient` to get the user's information from the user info endpoint. I copied the code below from Spring Security 5's OAuth 2.0 Login sample.

```

1  /*
2   * Copyright 2002-2017 the original author or authors
3   *
4   * Licensed under the Apache License, Version 2.0
5   * you may not use this file except in compliance
6   * with the License. You may obtain a copy of the License at
7   *
8   *     http://www.apache.org/licenses/LICENSE-2.0
9   *
10  * Unless required by applicable law or agreed to in writing,
11  * software distributed under the License is distributed
12  * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND,
13  * either express or implied. See the License for the specific
14  * language governing permissions and limitations under the License.
15  */
16  package com.okta.developer.oidc;
17
18  import org.springframework.beans.factory.annotation.Value;
19  import org.springframework.http.HttpHeaders;
20  import org.springframework.security.oauth2.client.OAuth2ClientRegistration;
21  import org.springframework.security.oauth2.client.OAuth2AuthorizedClient;
22  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientManager;
23  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProvider;
24  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
25  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
26  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
27  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
28  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
29  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
30  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
31  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
32  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
33  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
34  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
35  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
36  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
37  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
38  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
39  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
40  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
41  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
42  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
43  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
44  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
45  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
46  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
47  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
48  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
49  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
50  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
51  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
52  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
53  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
54  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
55  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
56  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
57  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
58  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
59  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
60  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
61  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
62  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
63  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
64  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
65  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
66  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
67  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
68  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
69  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
70  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
71  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
72  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
73  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
74  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
75  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
76  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
77  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
78  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
79  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
80  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
81  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
82  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
83  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
84  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
85  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
86  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
87  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
88  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
89  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
90  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
91  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
92  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
93  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
94  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
95  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
96  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
97  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
98  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
99  import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
100 import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;

```

```

import org.springframework.stereotype.Controller;
23
import org.springframework.ui.Model;
import org.springframework.util.StringUtils;
25
import org.springframework.web.bind.annotation.*;
26
import org.springframework.web.reactive.function.*;
27
import org.springframework.web.reactive.function.*;
28
import org.springframework.web.reactive.function.*;
29
import reactor.core.publisher.Mono;
30
import java.util.Collections;
import java.util.Map;
33
/**
35 * @author Joe Grandja
37 */
@Controller
38 public class MainController {
39
40     private final OAuth2AuthorizedClientService
41
42     public MainController(OAuth2AuthorizedClient
43
44         this.authorizedClientService = authorize
45     }
46
47     @RequestMapping("/")
48     public String index(Model model, OAuth2Auth
49
50         OAuth2AuthorizedClient authorizedClient
51
52         model.addAttribute("userName", authentic
53
54         model.addAttribute("clientName", authori
55
56         return "index";
57     }
58
59     @RequestMapping("/userinfo")
60     public String userinfo(Model model, OAuth2Au
61
62         OAuth2AuthorizedClient authorizedClient
63

```

```

58     map userAttributes = Collections.emptyMap()
59     String userInfoEndpointUri = authorizedClientService.loadClient(
60         .getProviderDetails().getUserInfoEndpointUri()
61         if (!StringUtils.isEmpty(userInfoEndpointUri)
62             userAttributes = WebClient.builder()
63                 .filter(oauth2Credentials(authentication.getAuthorizedClient(
64                     .get()).uri(userInfoEndpointUri)
65                     .retrieve()
66                     .bodyToMono(Map.class)).block())
67         }
68         model.addAttribute("userAttributes", userAttributes)
69         return "userinfo";
70     }
71
72     private OAuth2AuthorizedClient getAuthorizedClient() {
73         return this.authorizedClientService.loadClient(
74             authentication.getAuthorizedClientName()
75         }
76
77     private ExchangeFilterFunction oauth2Credentials() {
78         return ExchangeFilterFunction.ofRequestHandler(
79             clientRequest -> {
80                 ClientRequest authorizedRequest =
81                     .header(HttpHeaders.AUTHORIZATION,
82                         "Bearer " +
83                         .build());
84                 return Mono.just(authorizedRequest);
85             });
86     }
87 }

```


at `src/main/resources/templates/index.html` . You can use Thymeleaf's support for Spring Security to show/hide different parts of the page based on the user's authenticated status.

```

1  <!DOCTYPE html>
   <html xmlns="http://www.w3.org/1999/xhtml" xmlns:th="http://www.thymeleaf.org/thymeleaf"
2  <
   xmlns:sec="http://www.thymeleaf.org/thymeleaf"
3  <
   <head>
4     <title>Spring Security - OAuth 2.0 Login</title>
5     <meta charset="utf-8" />
6   </head>
7   <body>
8     <div style="float: right" th:fragment="logout">
9       <div style="float: left">
10        <span style="font-weight: bold">User: </span>
11      </div>
12      <div style="float: none"> </div>
13      <div style="float: right">
14        <form action="#" th:action="@{/logout}">
15          <input type="submit" value="Logout">
16        </form>
17      </div>
18    </div>
19    <h1>OAuth 2.0 Login with Spring Security</h1>
20    <div>
21      You are successfully logged in <span style="font-weight: bold">
22      via the OAuth 2.0 Client <span style="font-weight: bold">
23    </div>
24    <div> </div>
25    <div>
26      <a href="/userinfo" th:href="@{/userinfo}">
27    </div>
28  </body>
29  </html>
30

```

Create another template

at `src/main/resources/templates/userinfo.html` to display the

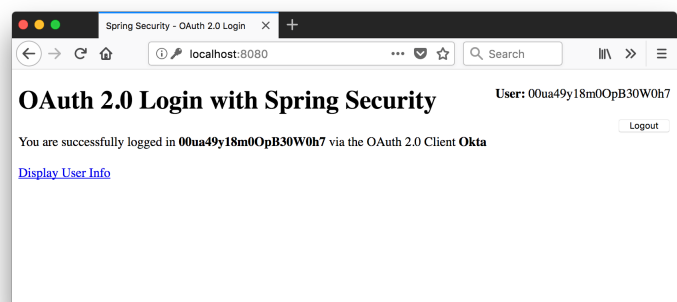
at `src/main/resources/userinfo.html` to display the user's attributes.

```

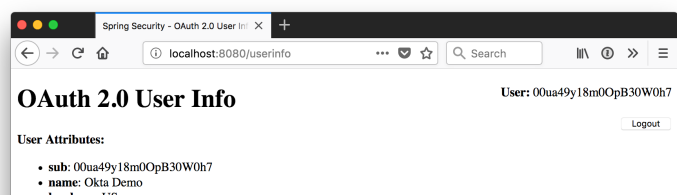
1  <!DOCTYPE html>
   <html xmlns="http://www.w3.org/1999/xhtml" xmlr
2  <
3  <head>
   <title>Spring Security - OAuth 2.0 User Info
4  <
5   <meta charset="utf-8" />
6  </head>
7  <body>
8  <div th:replace="index::logout"></div>
9  <h1>OAuth 2.0 User Info</h1>
10 <div>
   <span style="font-weight:bold">User Attribut
11 <
12   <ul>
       <li th:each="userAttribute : ${userAttri
13 <
       <span style="font-weight:bold" th:te
14 <
       </li>
15   </ul>
16 </div>
17 </body>
18 </html>
19

```

Now, when you're logged in, you'll see a link to display user info.



Click on the link, and you'll see the contents of the ID Token that's retrieved from the user info endpoint.



```
• email: matt.raible@okta.com
• preferred_username: demo@okta.com
• given_name: Okta
• family_name: Demo
• zoneinfo: America/Los_Angeles
• updated_at: 1511970736
• email_verified: true
• groups: [Everyone, ROLE_USER]
```

Learn More About Spring Security and OIDC

This article showed you how to implement login with OAuth 2.0 and Spring Security 5. I also showed you how to use OIDC to retrieve a user's information. The source code for the application developed in this article can be found on GitHub.

These resources provide additional information about Okta and OIDC:


- [Okta Developer Documentation and its OpenID Connect API](#)
- [Identity, Claims, & Tokens – An OpenID Connect Primer, Part 1 of 3](#)
- [OIDC in Action – An OpenID Connect Primer, Part 2 of 3](#)
- [What's in a Token? – An OpenID Connect Primer, Part 3 of 3](#)
- [Add Role-Based Access Control to Your App with Spring Security and Thymeleaf](#)

If you have any questions about this post, please leave a comment below. You can also post to Stack Overflow with the `okta` tag or use our developer forums.

Get Started with Spring Security 5.0 and OIDC was originally published on the Okta developer blog on December 18, 2017.

Tired of building the same login screens over and over? Try the Okta API for hosted authentication, authorization, and multi-factor auth.

Topics: JAVA, AUTHENTICATION, OIDC, SPRING SECURITY 5, SPRING APP, TUTORIAL

Published at DZone with permission of Matt Raible,
DZone MVB. [See the original article here.](#) 
Opinions expressed by DZone contributors are their
own.