Partner ORACLE UNIVERSITY

Certifica TIC Beyond Certification

INICIO

Desarrollo de Aplicaciones Empresariales con Spring Framework Core 5

ISC. Ivan Venor García Baños

ATC Advanced Training Consultants

SCRUMstudy Authorized Training Partner

Agenda

1. Presentación
2. Objetivos
3. Contenido
4. Despedida

3. Contenido

i. Introducción a Spring Framework
ii. Spring Core
iii. Spring AOP
iv. Spring JDBC – Transaction
v. Spring ORM – Hibernate 5
vi. Spring Data JPA
vii. Fundamentos Spring MVC y Spring REST
viii. Fundamentos Spring Security
ix. Seguridad en Servicios REST
x. Introducción Spring Boot

ix. Seguridad en Servicios REST

ix. Seguridad en Servicios REST (a)

ix.i Introducción

a. Servicios Stateless vs Stateful

b. Autenticación HTTP por Sesión vs Token

ix. Seguridad en Servicios REST (b)

ix.ii Autenticación HTTP con Spring Security

a. Autenticación Básica

Práctica 33. Implementación de Autenticación Básica de Servicios REST

b. Autenticación Digest (resumen)

Práctica 34. Implementación de Autenticación Digest de Servicios REST

c. Autenticación Bearer con JSON Web Token (JWT)

Práctica 35. Implementación de Autenticación Bearer con JWT de Servicios REST

ix.i Introducción

Objetivos de la lección

ix.i Introducción

- Comprender que son los servicios REST con y sin estado (Stateful y Stateless).
- Comprender el funcionamiento sin estado del protocolo HTTP.
- Analizar las diferencias entre autenticación por Sesión y autenticación por Token.

ix.i Introducción

a. Servicios Stateless vs Stateful

b. Autenticación HTTP por Sesión vs Token

ix.i Servicios Stateless vs Stateful (a)

- Servicios Stateless vs Stateful

- Los conceptos Stateless y Stateful aplican a la definición de servicios REST (APIs REST) y a aplicaciones informáticas principalmente basadas en la web (aplicaciones web).

ix.i Servicios Stateless vs Stateful (b)

- Servicios Stateful

- Stateful, se refiere a un servicio o aplicación que mantiene estado, es decir, hace uso de sesiones u algún otro mecanismo de almacenamiento para identificar al cliente que realiza la petición durante una conversación de múltiples solicitudes entre el cliente y el servidor.

ix.i Servicios Stateless vs Stateful (c)

- Servicios Stateful

- Stateful se refiere al almacenamiento de información del cliente del lado del servidor, lo que ocasiona que el servidor “mantega estado”.

- Los servicios y/o aplicaciones con estado o Stateful, son difíciles de escalar.

ix.i Servicios Stateless vs Stateful (d)

- Servicios Stateful

- Servicios Stateful (con estado) típicamente utilizan sesiones/cookies para mantener el estado entre peticiones.

- Aplicaciones con estado son típicamente bases de datos o repositorios de datos.

- Servicios Stateful involucran transaccionabilidad entre peticiones, tal como aplicaciones bancarias (conversación entre peticiones).

ix.i Servicios Stateless vs Stateful (e)

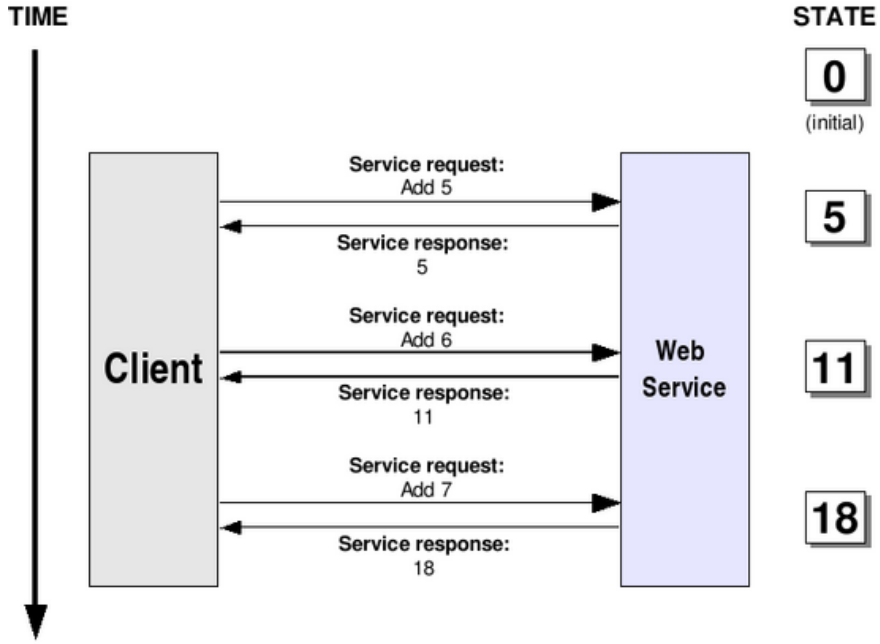
- Servicios Stateful

- Servicios o aplicaciones Stateful procesan solicitudes basándose en la información almacenada en solicitudes previas, lo que significa que los servidores deban mantener la información generada previamente (en memoria o algún otro medio persistente) para procesar la solicitud entrante.

- Un mismo servidor debe procesar las solicitudes Stateful para que pueda procesar las nuevas solicitudes dependientes de las anteriores o utilizar un mecanismo que comparta el estado entre múltiples servidores.

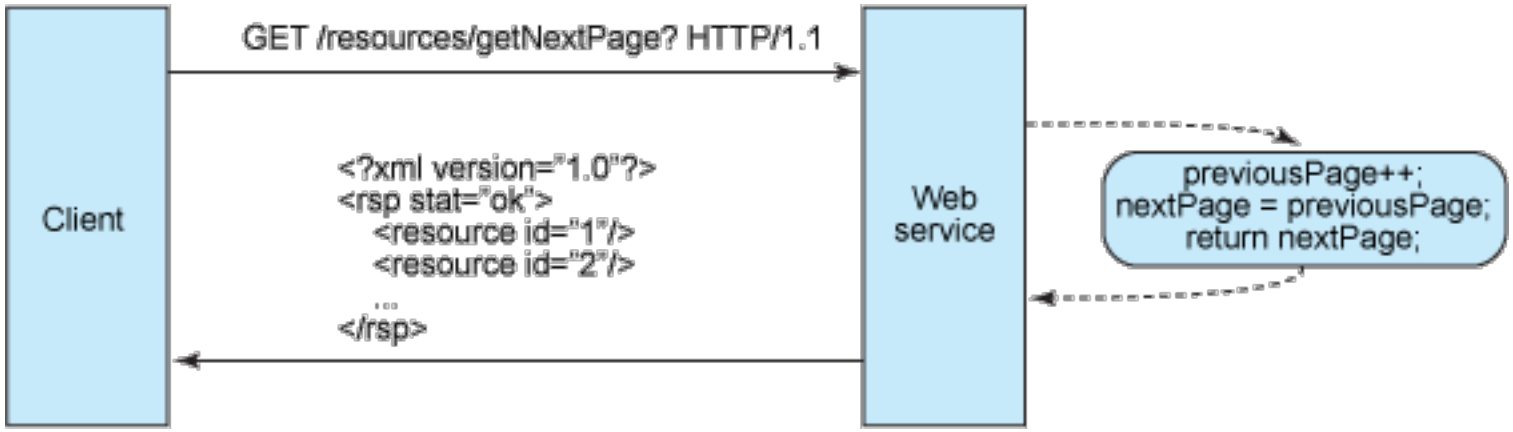
ix.i Servicios Stateless vs Stateful (f)

- Servicios Stateful



ix.i Servicios Stateless vs Stateful (g)

- Servicios Stateful



ix.i Servicios Stateless vs Stateful (h)

- Servicios Stateless

- Stateless se refiere a que no existe almacenamiento de información del cliente del lado del servidor, lo que ocasiona que el servidor necesite recibir toda la información que requiera, desde el cliente, para procesar la solicitud.

- Los servicios Stateless, no mantienen estado.

- Los servicios y/o aplicaciones sin estado o Stateless, son más fáciles de escalar.

ix.i Servicios Stateless vs Stateful (i)

- Servicios Stateless

- Los servicios y/o aplicaciones Stateless implementan una única función o servicio.

- Los servicios Stateless requieren recibir toda la información necesaria para poder procesar la solicitud y no dependen de peticiones previas para procesar la solicitud entrante.

- Las aplicaciones o servicios Stateless no requieren mantener estado entre peticiones.

ix.i Servicios Stateless vs Stateful (j)

- Servicios Stateless

- Diferentes solicitudes de un mismo cliente (browser/user-agent) pueden ser atendidas por diferentes servidores o instancias del servicio.

- El hecho de que un servicio sea Stateless (sin estado) no significa que no pueda almacenar estado en algún otro servicio back-end con estado como una base de datos.

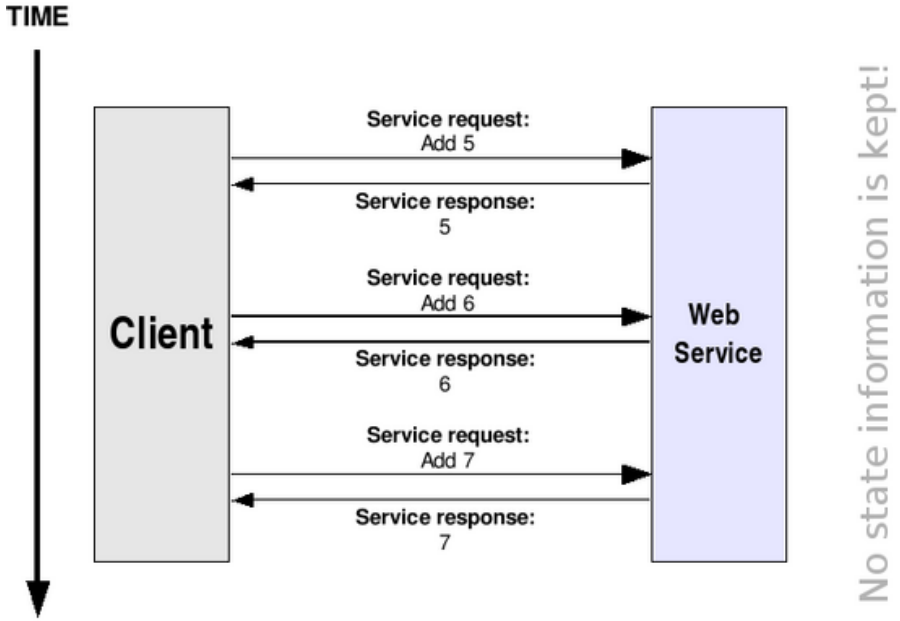
ix.i Servicios Stateless vs Stateful (k)

- Servicios Stateless

- Los servicios Stateless que no mantienen estado y recuperan el estado de la aplicación desde otro servicio back-end para poder ejecutar la lógica requerida del servicio ofrecen “resiliencia” (resistencia / confianza), elasticidad y alta disponibilidad del servicio mediante replicación.

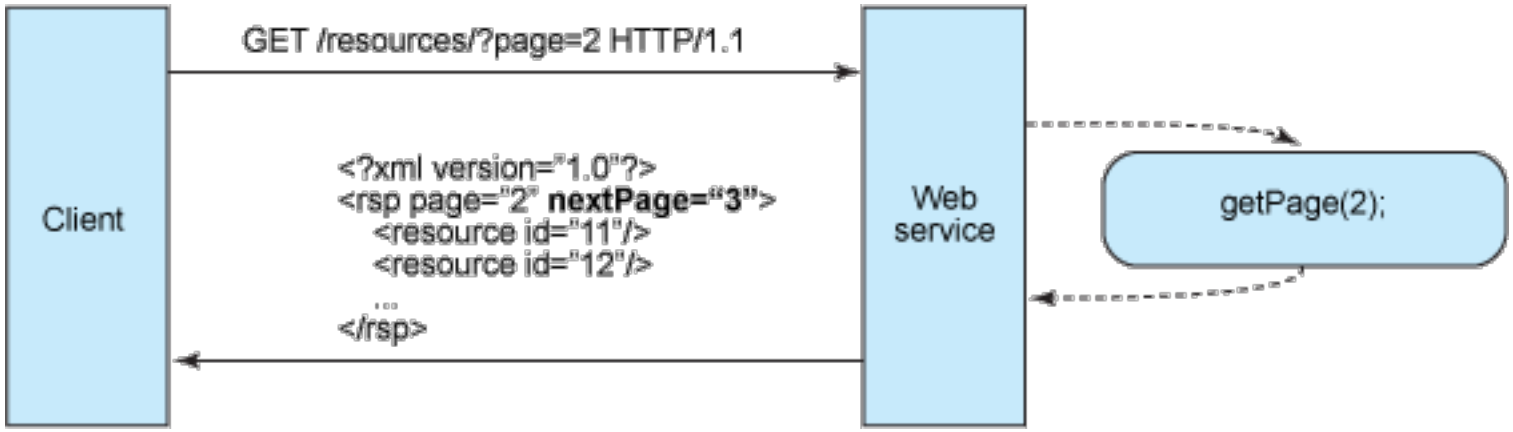
ix.i Servicios Stateless vs Stateful (I)

- Servicios Stateless



ix.i Servicios Stateless vs Stateful (m)

- Servicios Stateless



ix.i Introducción

a. Servicios Stateless vs Stateful

b. Autenticación HTTP por Sesión vs Token

ix.i Autenticación HTTP por Sesión vs Token (a)

- HTTP, por sus siglas en inglés: "Hypertext Transfer Protocol", es el protocolo el cual nos permite realizar una petición de datos y recursos a un servidor web, tal como pueden ser documentos HTML.

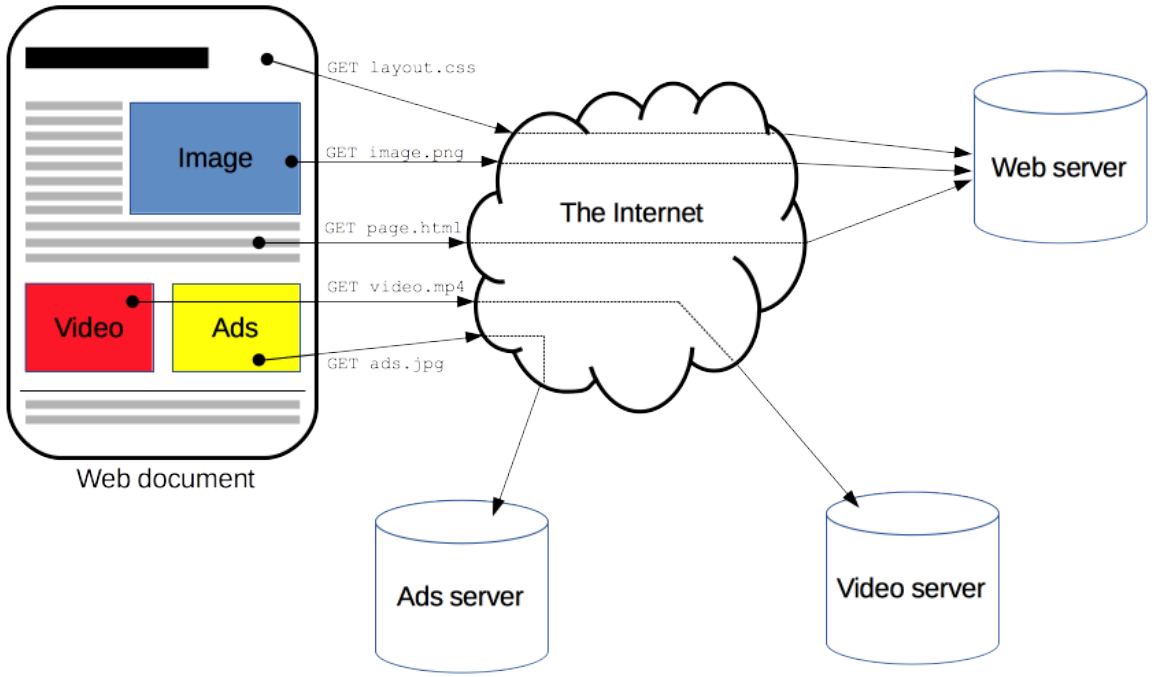
- HTTP es la base de cualquier intercambio de datos en la Web, y un protocolo que define la estructura en la comunicación cliente-servidor. Esto quiere decir que una petición de datos es iniciada por el elemento que recibirá los datos (el cliente, normalmente un navegador Web).

ix.i Autenticación HTTP por Sesión vs Token (b)

- A través de HTTP es el medio por el cual un servidor web, puede enviar una página web completa, origen del resultado de la unión de distintos sub-documentos/recursos recibidos, como, por ejemplo:
  - Un documento que especifique el estilo de maquetación de la página web u hoja de estilos CSS.
  - El texto de la página web
  - Las imágenes, vídeos, scripts y otros recursos de la página web.

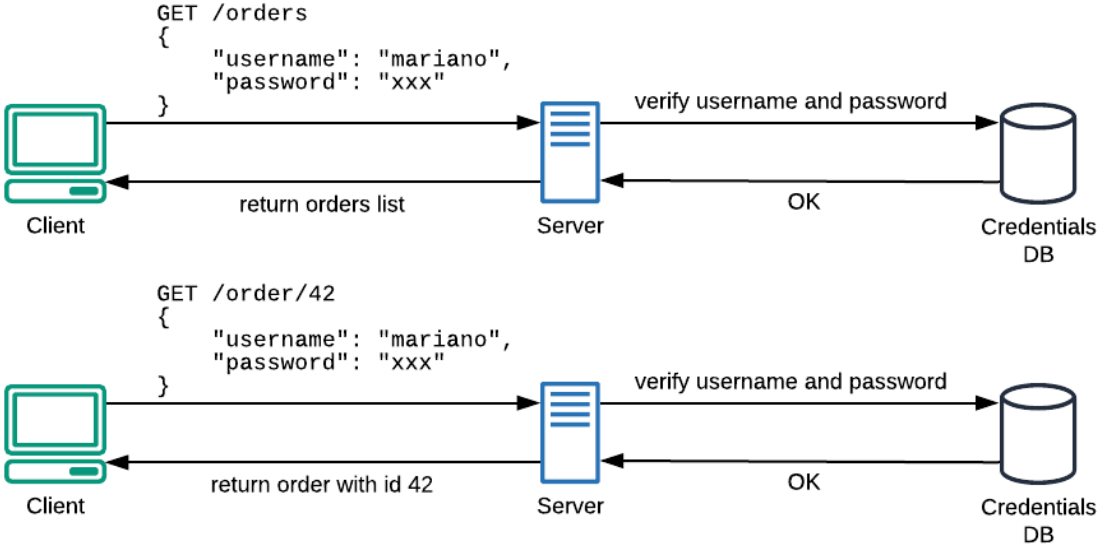
ix.i Autenticación HTTP por Sesión vs Token (c)

- HTTP



ix.i Autenticación HTTP por Sesión vs Token (d)

- HTTP
- La especificación HTTP es Stateless, es decir, el protocolo HTTP no mantiene estado.



ix.i Autenticación HTTP por Sesión vs Token (e)

- Autenticación.

- La autenticación es el proceso en el cual un componente de software, o medio electrónico, determina si el usuario que está intentando acceder a un sistema, o recurso protegido, tiene permiso para acceder a el o no.

- La autenticación no determina qué tareas puede hacer el usuario, ni qué archivos puede ver o acceder, pero identifica y verifica quién es el usuario o sistema que se autentica.

- La autenticación es la validación de que el usuario o sistema es quién dice ser.

ix.i Autenticación HTTP por Sesión vs Token (f)

- Autorización.

- La autorización determina qué tareas puede hacer el usuario, qué archivos puede ver o acceder.

- Para identificar el nivel de autorización de un usuario o sistema, primeramente es necesario autenticarlo.

ix.i Autenticación HTTP por Sesión vs Token (g)

- HTTP es el protocolo de comunicación entre un cliente y un servidor y, a su vez, HTTP es sin estado (Stateless).

- Que el protocolo HTTP sea Stateless, significa que cada solicitud entrante a un servidor no debe depender de solicitudes previas, por tanto, si un usuario de alguna aplicación se autentica (*loggea*/inicia sesión) en una aplicación web y navega, el sistema no puede determinar si el usuario que está navegando se ha autenticado previamente.

ix.i Autenticación HTTP por Sesión vs Token (h)

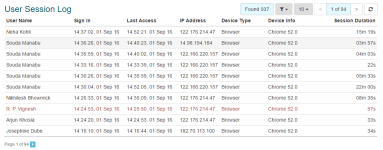
- Para que un sistema pueda determinar si el usuario que está navegando en una web, se ha autenticado previamente, el sistema debe implementar un mecanismo para reconocer al usuario autenticado.

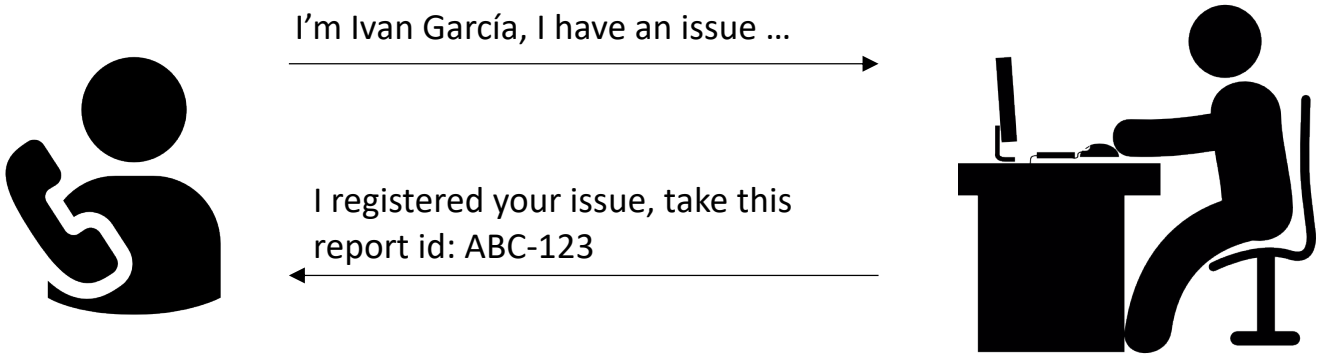
- En la actualidad, los sistemas basados en la web implementan el uso de autenticación por sesión o por token.

ix.i Autenticación HTTP por Sesión vs Token (i)

- Analogía autenticación por sesión.

Ticket Log

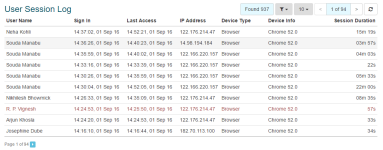


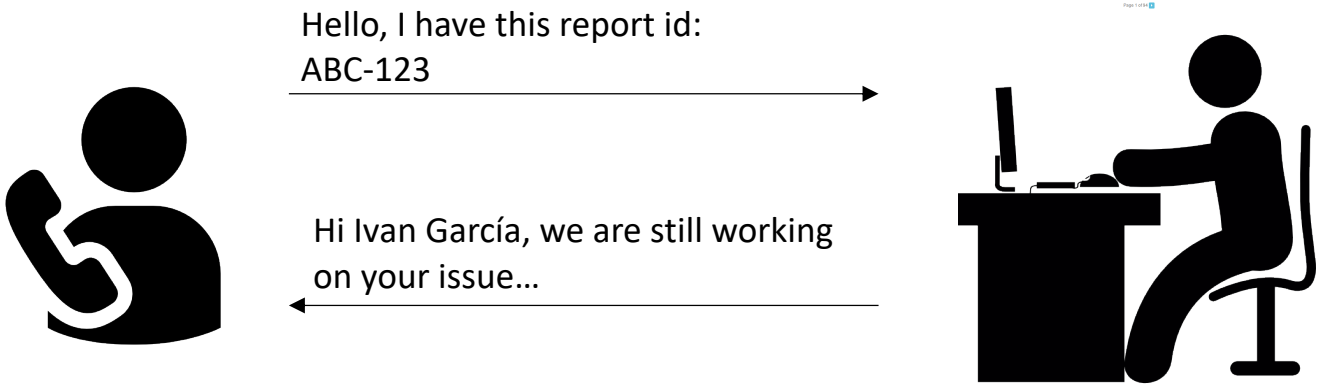


ix.i Autenticación HTTP por Sesión vs Token (j)

- Analogía autenticación por sesión.

Ticket Log

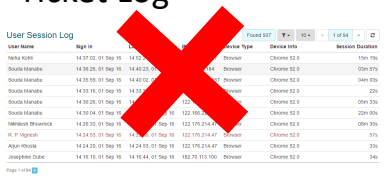


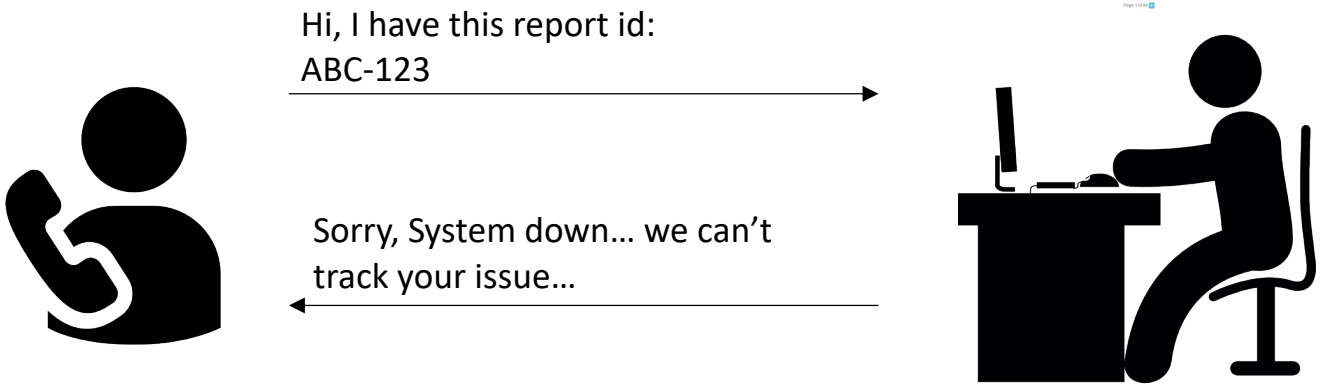


ix.i Autenticación HTTP por Sesión vs Token (k)

- Analogía autenticación por sesión.

Ticket Log





ix.i Autenticación HTTP por Sesión vs Token (I)

- Autenticación HTTP por Sesión

- En la autenticación HTTP por sesión, el servidor es responsable de crear una sesión (session id) para un usuario en particular el cual se ha autenticado en el sistema.

- Una vez que el servidor ha generado el id de sesión, el sistema debe almacenar la información relacionada al usuario en memoria o algún medio persistente, utilizando el id de sesión como llave de acceso a dicha información.

ix.i Autenticación HTTP por Sesión vs Token (m)

- Autenticación HTTP por Sesión

- Por último, el servidor debe enviar el id de sesión al cliente HTTP (navegador web) en formato de cookie.

- Para cualquier solicitud subsecuente, el cliente HTTP (navegador web) debe enviar el id de sesión, para que el servidor pueda identificar al usuario, comparando el id de sesión enviado (a través de la cookie) con el directorio de sesiones que el servidor tiene almacenado.

ix.i Autenticación HTTP por Sesión vs Token (n)

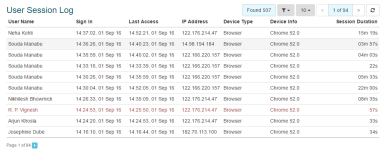
- Autenticación HTTP por Sesión

- Las solicitudes HTTP requieren el id de sesión para funcionar, para identificar al usuario, por tanto mantienen estado.

ix.i Autenticación HTTP por Sesión vs Token (ñ)

- Autenticación HTTP por Sesión

Sessions



username “Ivan García” &
password “123”

SESSION ID

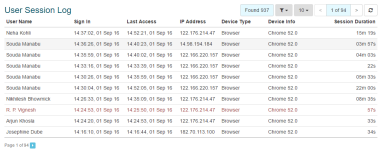
ix.i Autenticación HTTP por Sesión vs Token (o)

- Autenticación HTTP por Sesión

SESSION ID
get accounts

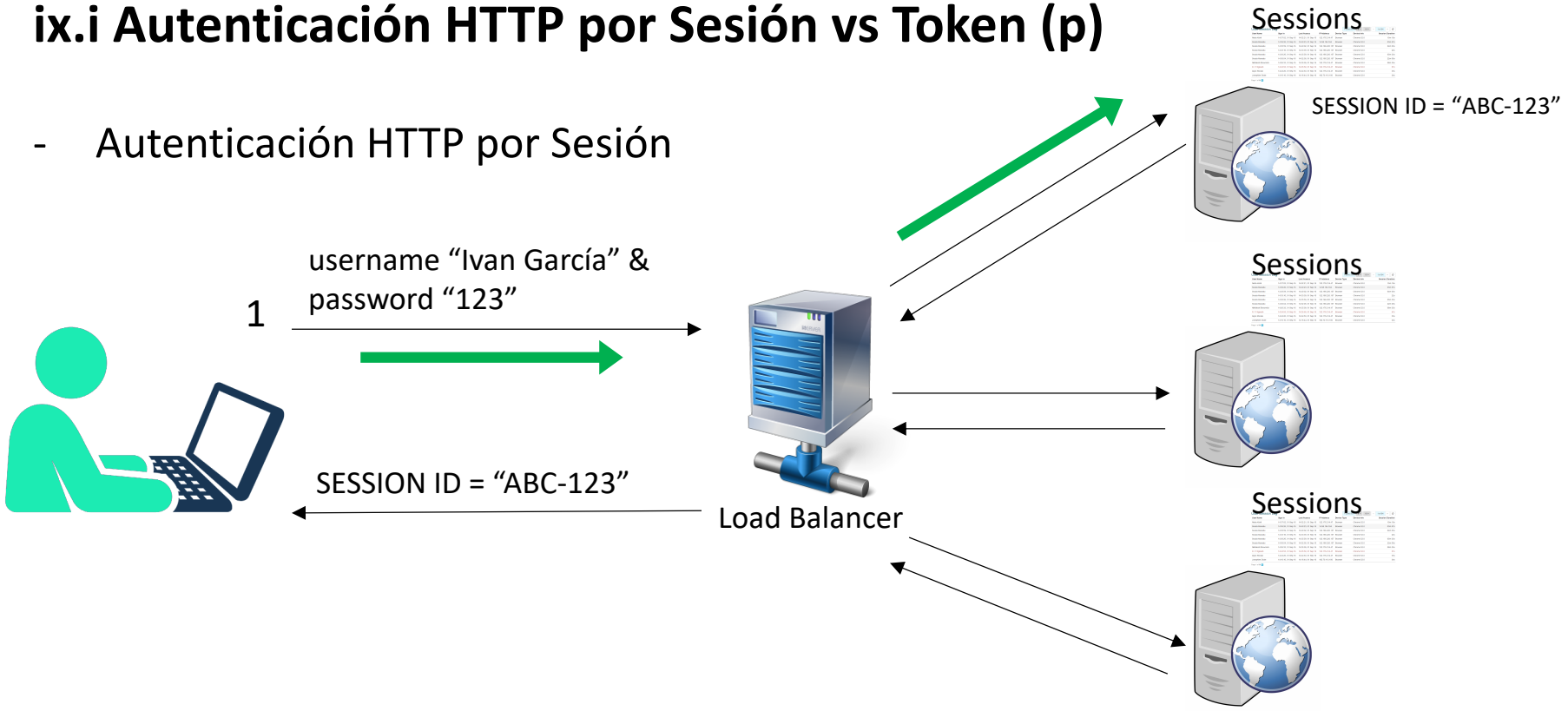
Ivan García accounts

Sessions



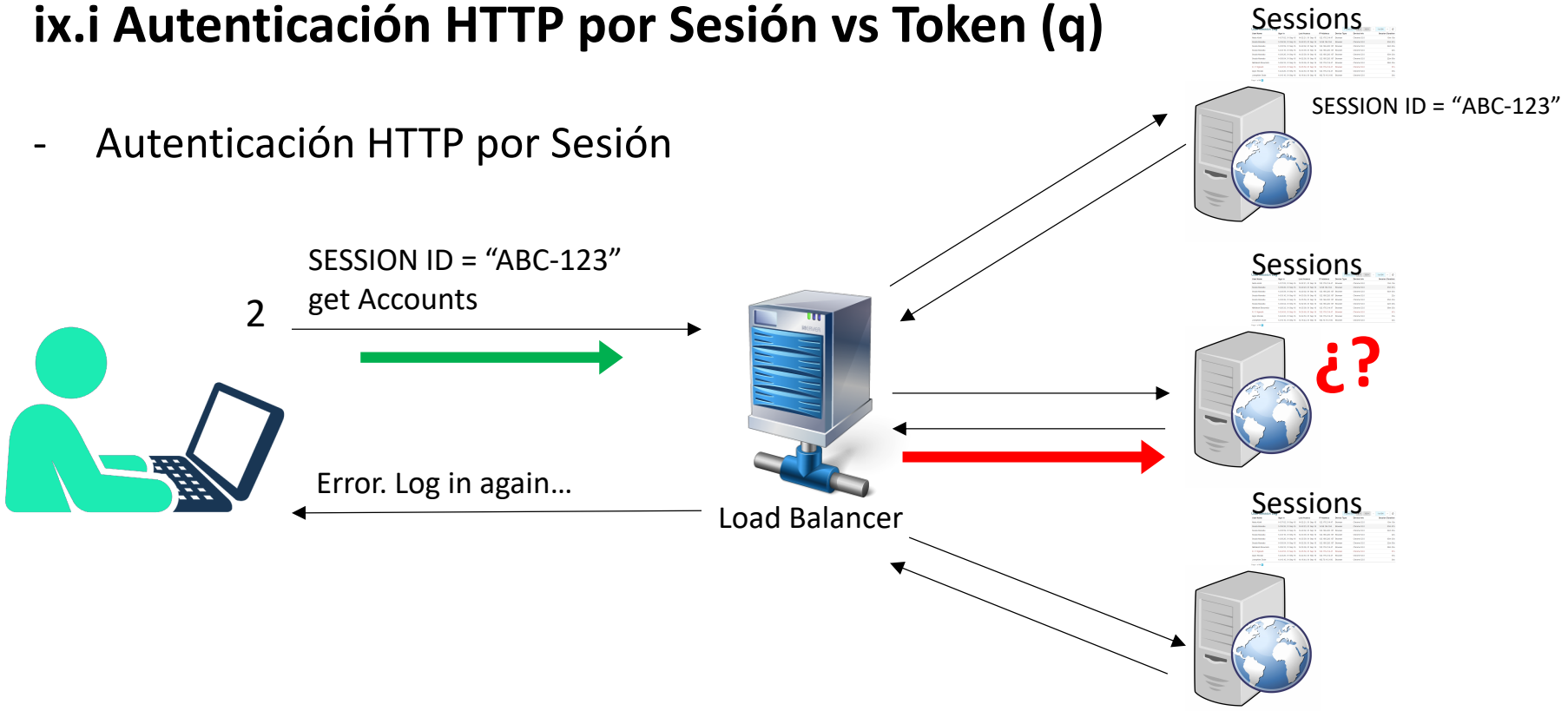
ix.i Autenticación HTTP por Sesión vs Token (p)

- Autenticación HTTP por Sesión



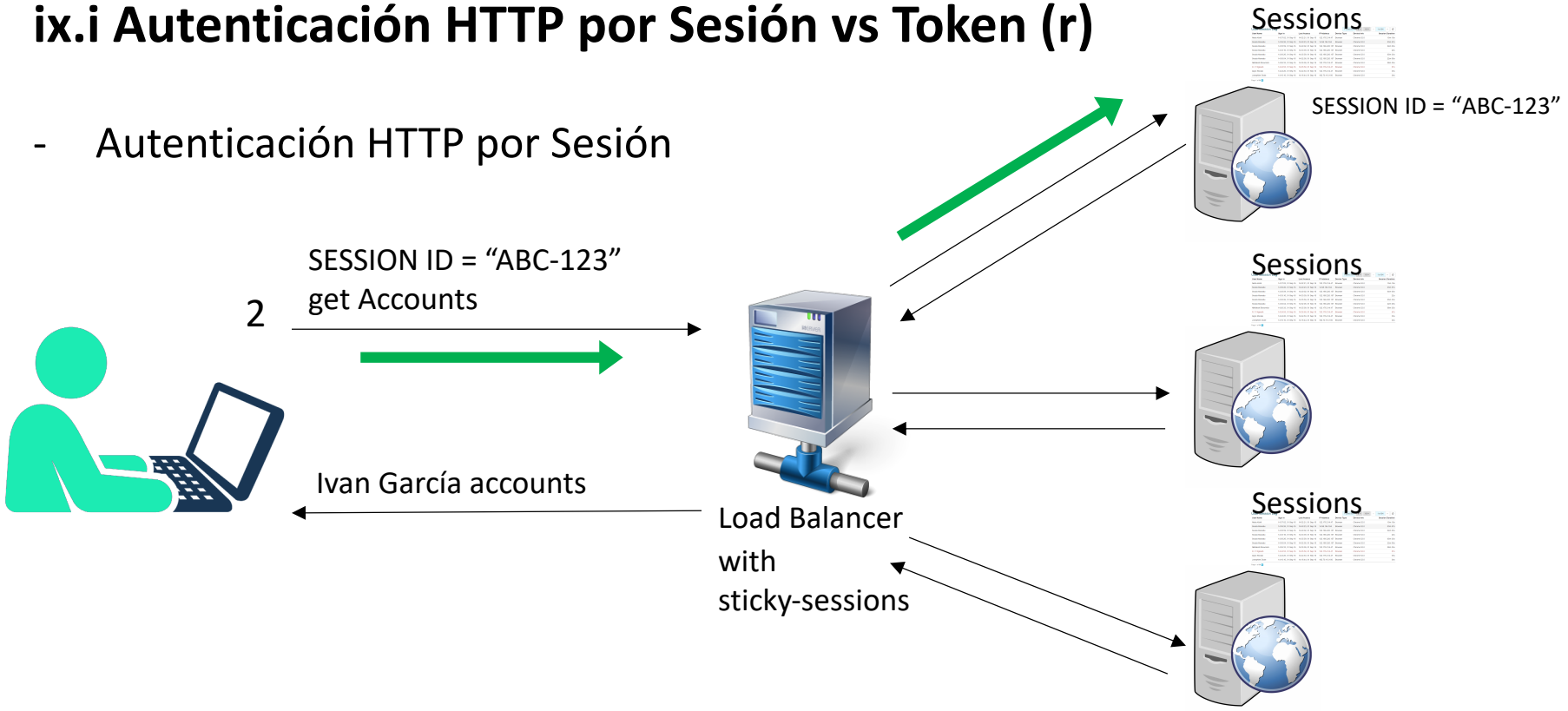
ix.i Autenticación HTTP por Sesión vs Token (q)

- Autenticación HTTP por Sesión



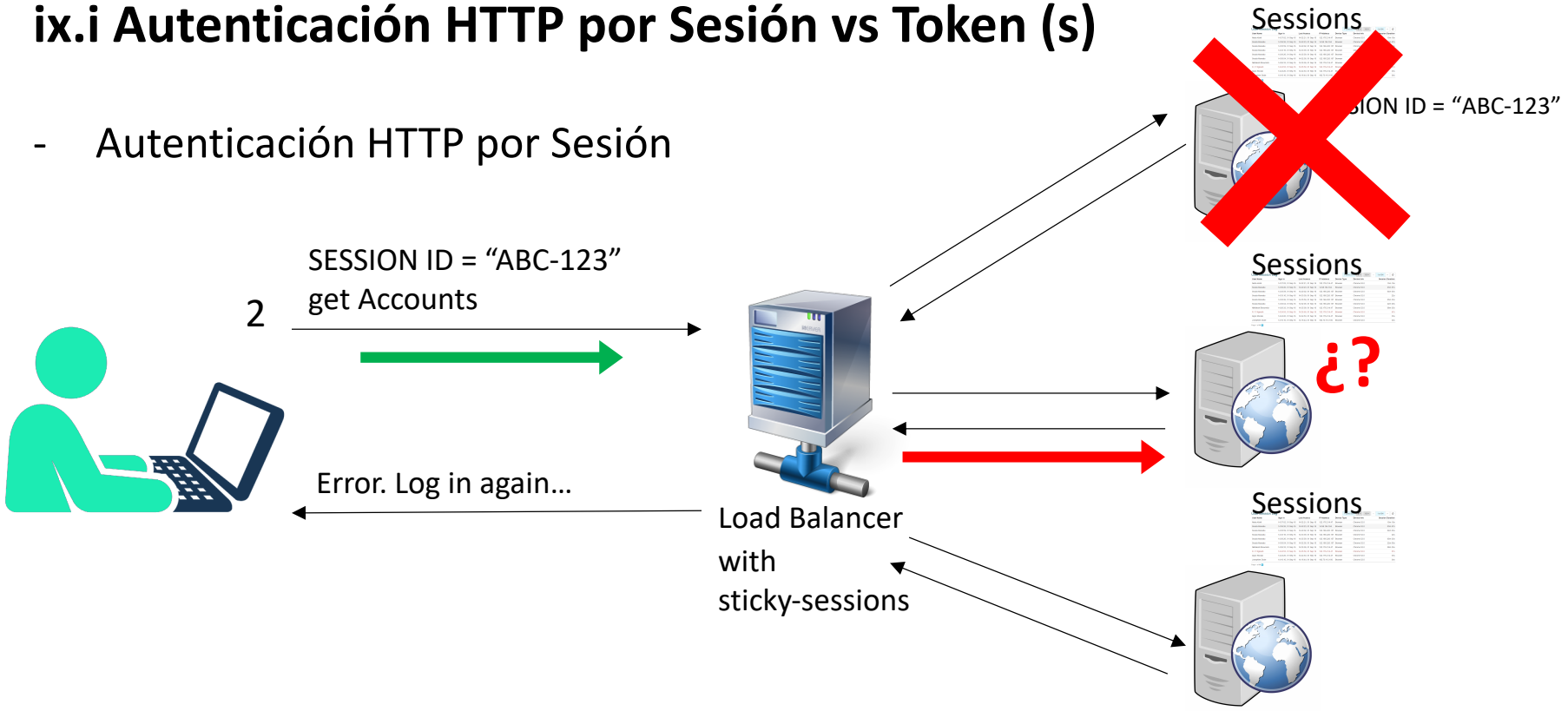
ix.i Autenticación HTTP por Sesión vs Token (r)

- Autenticación HTTP por Sesión



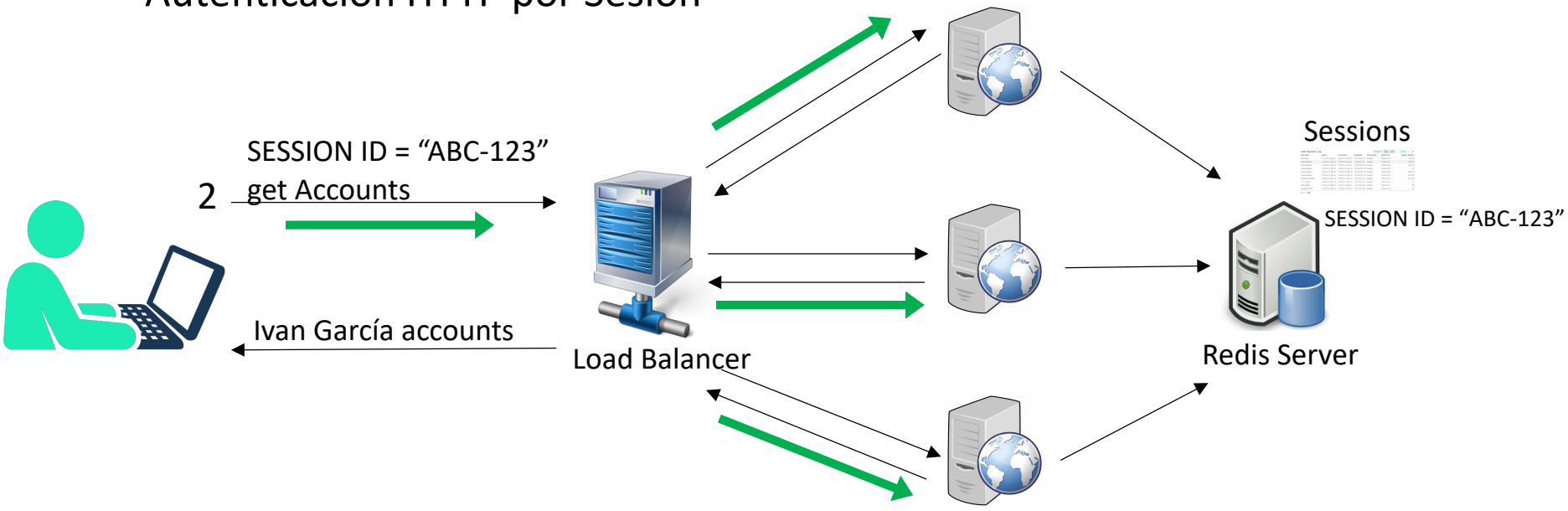
ix.i Autenticación HTTP por Sesión vs Token (s)

- Autenticación HTTP por Sesión



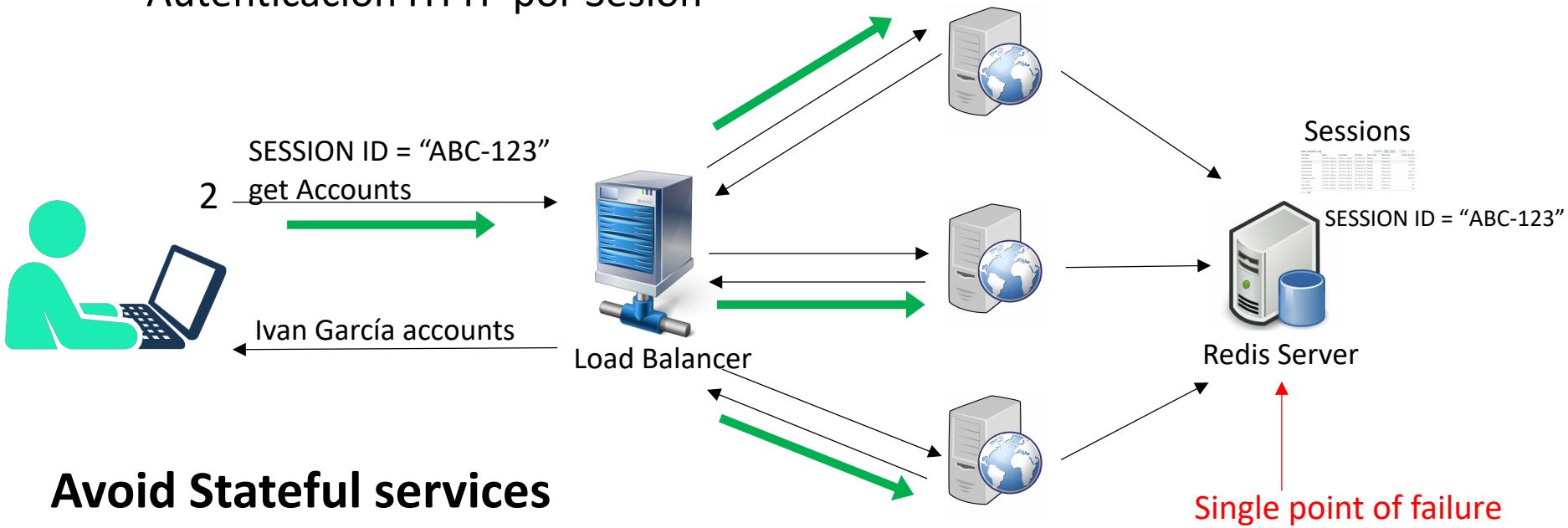
ix.i Autenticación HTTP por Sesión vs Token (t)

- Autenticación HTTP por Sesión



ix.i Autenticación HTTP por Sesión vs Token (u)

- Autenticación HTTP por Sesión



Avoid Stateful services

ix.i Autenticación HTTP por Sesión vs Token (v)

- Autenticación HTTP por Token

- En la autenticación HTTP por token, el servidor crea un Token (por lo regular un JSON Web Token/JWT), al momento de la autenticación del usuario y es retornado al cliente.

- El token contiene información no sensible relacionada al usuario y, el token, es almacenado en el cliente HTTP en el almacenamiento local (*local storage*) o en formato de cookie.

ix.i Autenticación HTTP por Sesión vs Token (w)

- Autenticación HTTP por Token

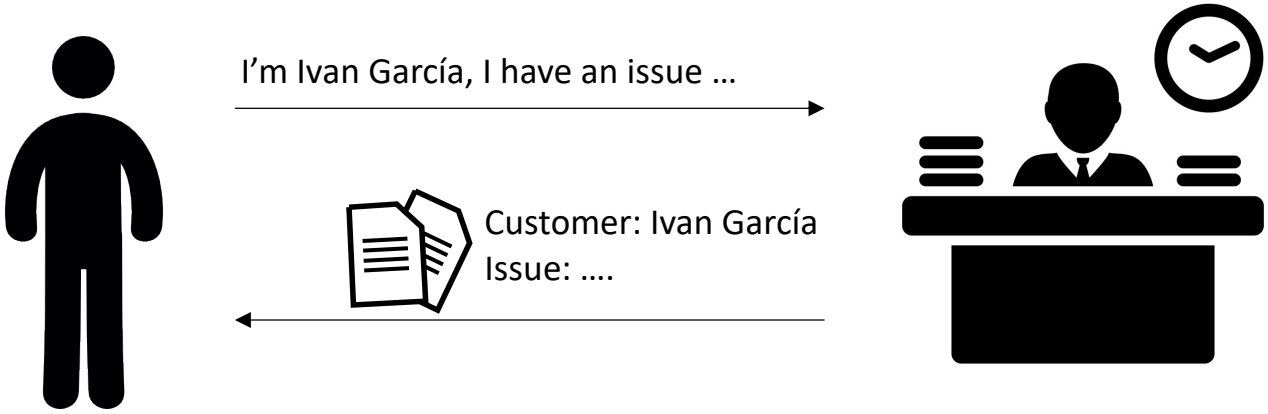
- Para cualquier solicitud subsecuente, el cliente HTTP (navegador web) debe enviar el token, para que el servidor pueda identificar al usuario.

- El servidor al recibir el token, debe validarlo, comprobar su autenticidad para así verificar la identidad del cliente.

- El token, que contiene la información relacionada al usuario, es quien mantiene el estado y no el servidor.

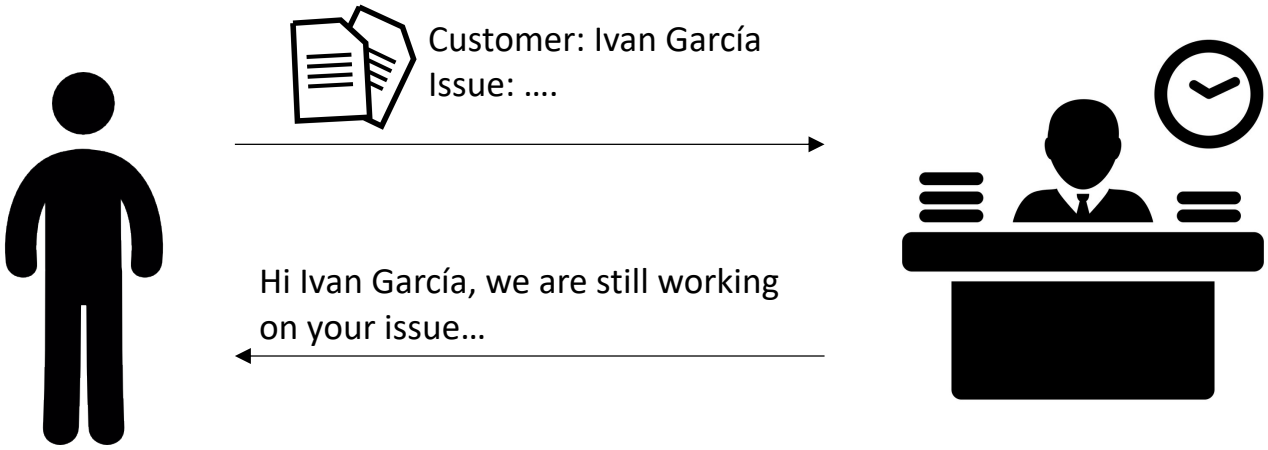
ix.i Autenticación HTTP por Sesión vs Token (x)

- Analogía autenticación por Token.



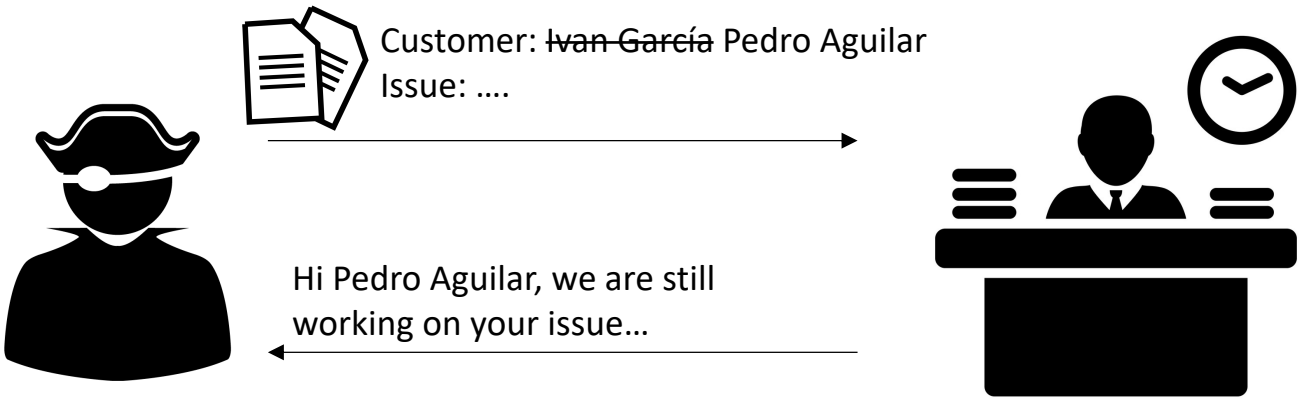
ix.i Autenticación HTTP por Sesión vs Token (y)

- Analogía autenticación por Token.



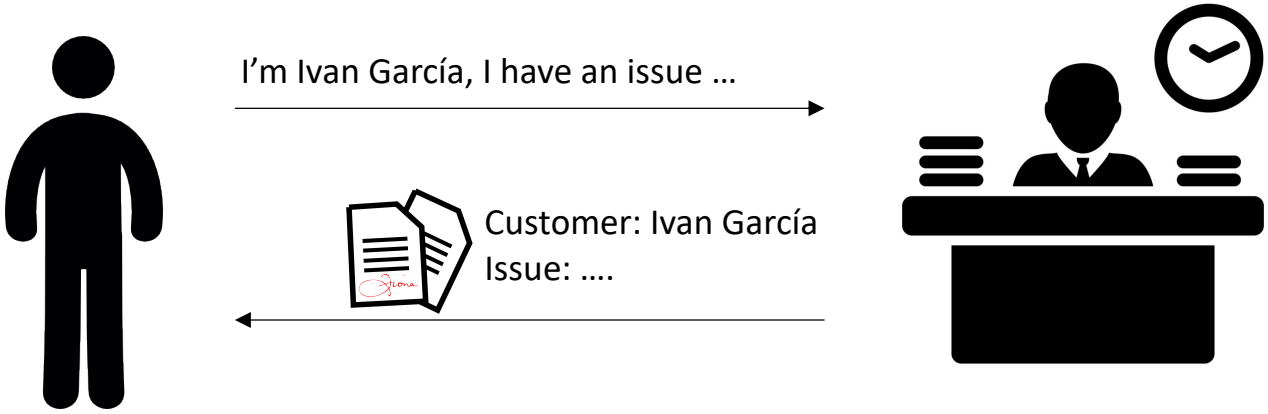
ix.i Autenticación HTTP por Sesión vs Token (z)

- Analogía autenticación por Token.



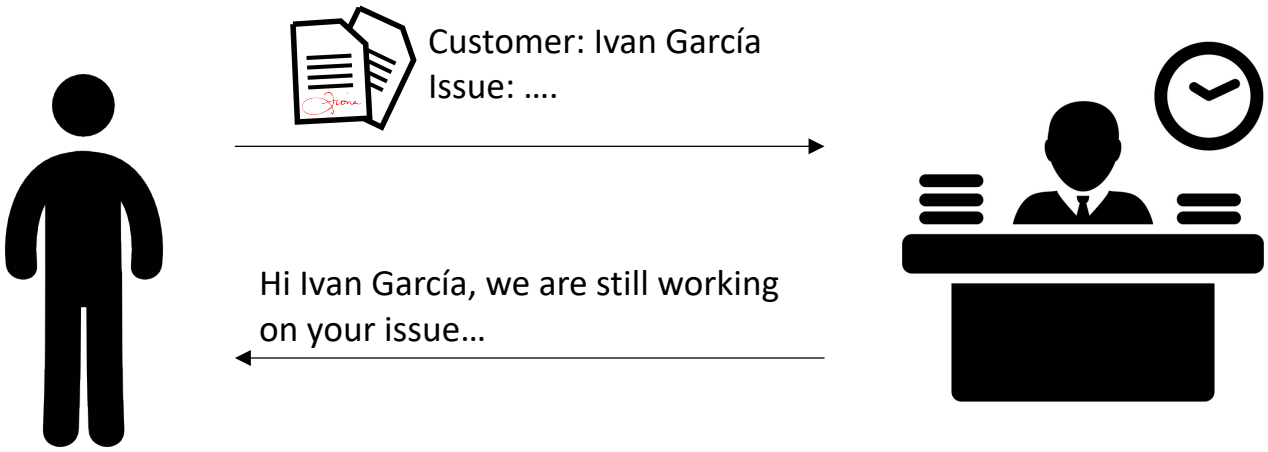
ix.i Autenticación HTTP por Sesión vs Token (a')

- Analogía autenticación por Token.



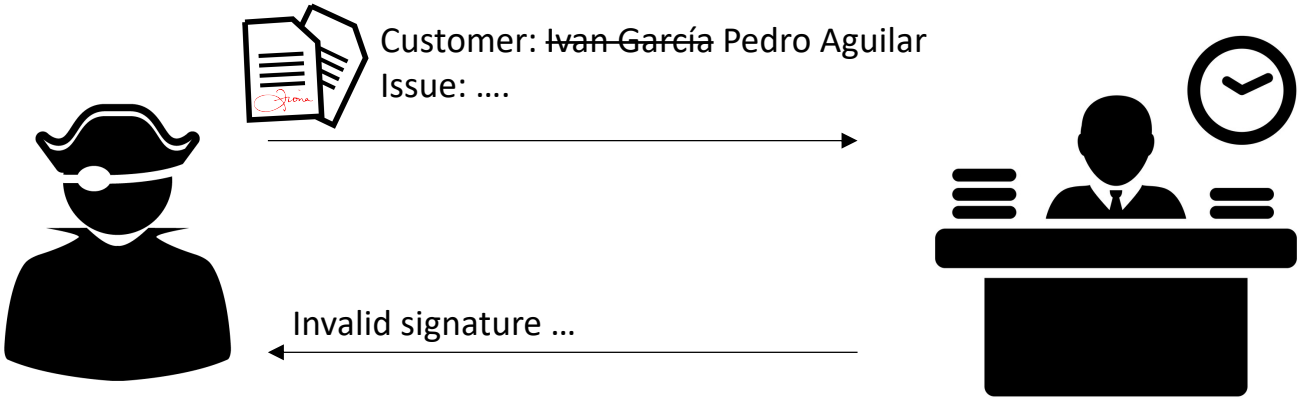
ix.i Autenticación HTTP por Sesión vs Token (b')

- Analogía autenticación por Token.



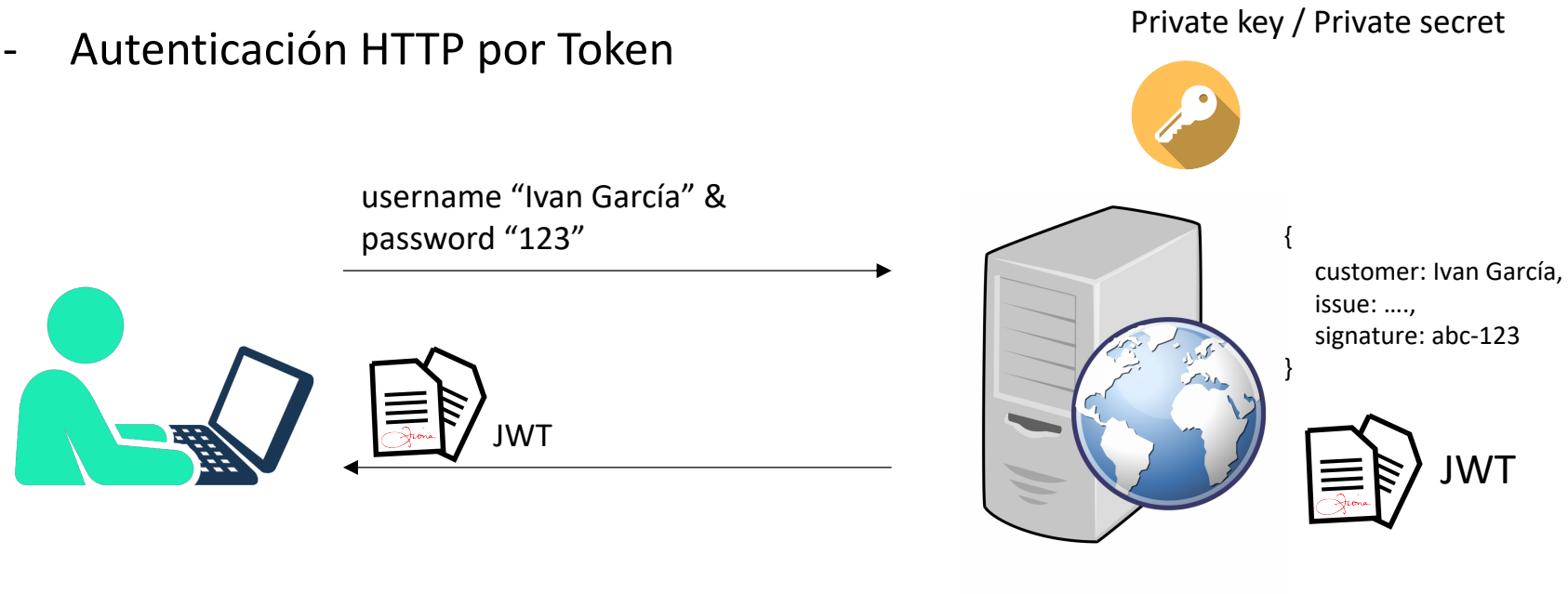
ix.i Autenticación HTTP por Sesión vs Token (c')

- Analogía autenticación por Token.



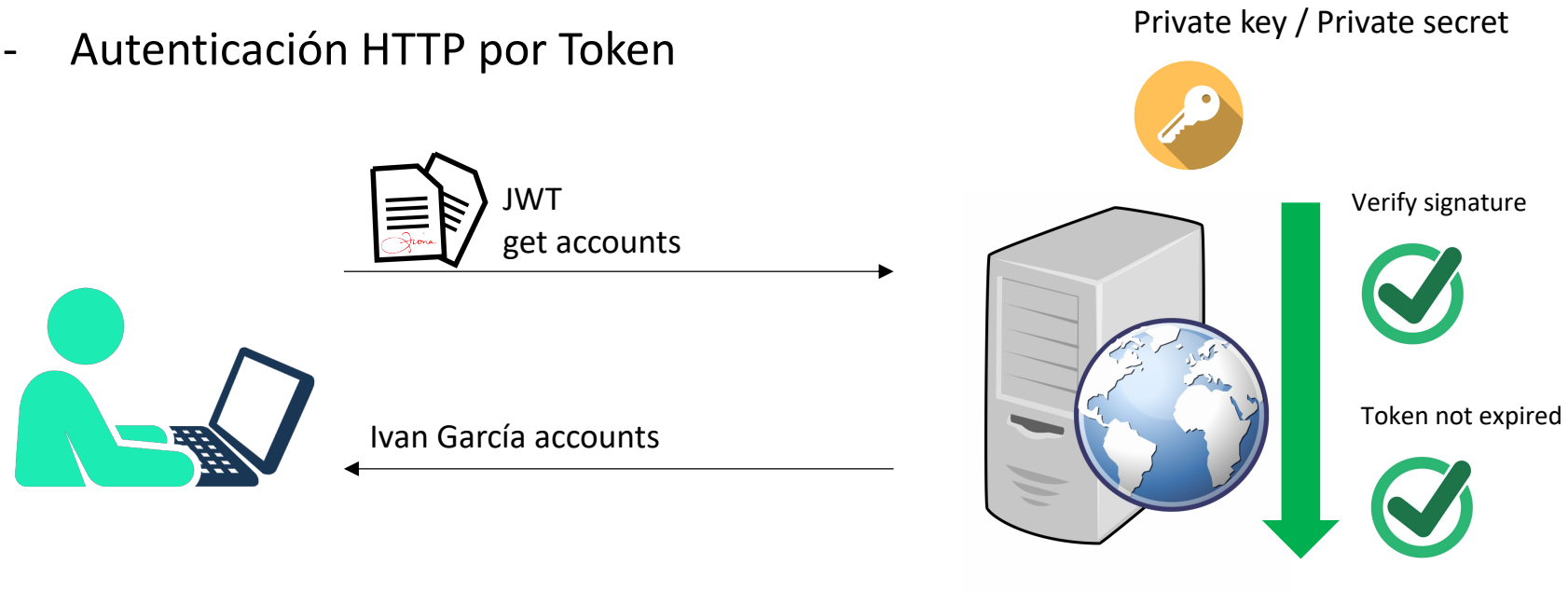
ix.i Autenticación HTTP por Sesión vs Token (d')

- Autenticación HTTP por Token



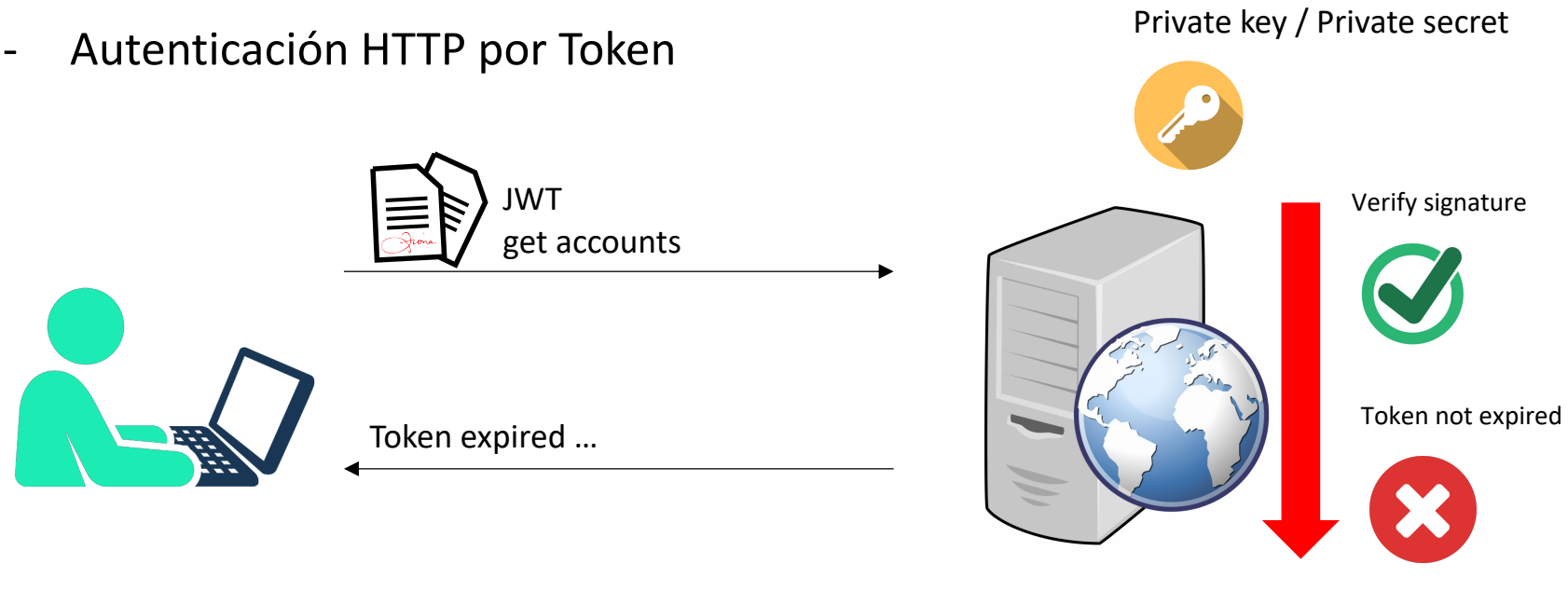
ix.i Autenticación HTTP por Sesión vs Token (e')

- Autenticación HTTP por Token



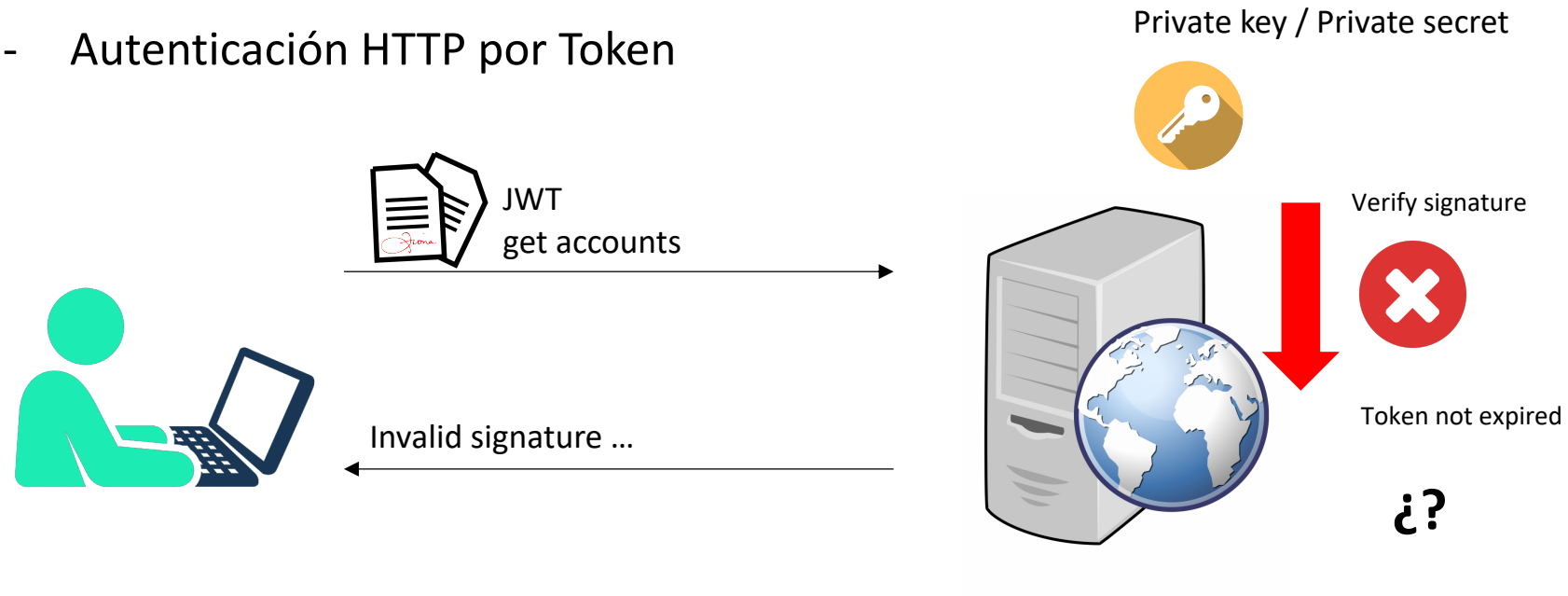
ix.i Autenticación HTTP por Sesión vs Token (f')

- Autenticación HTTP por Token



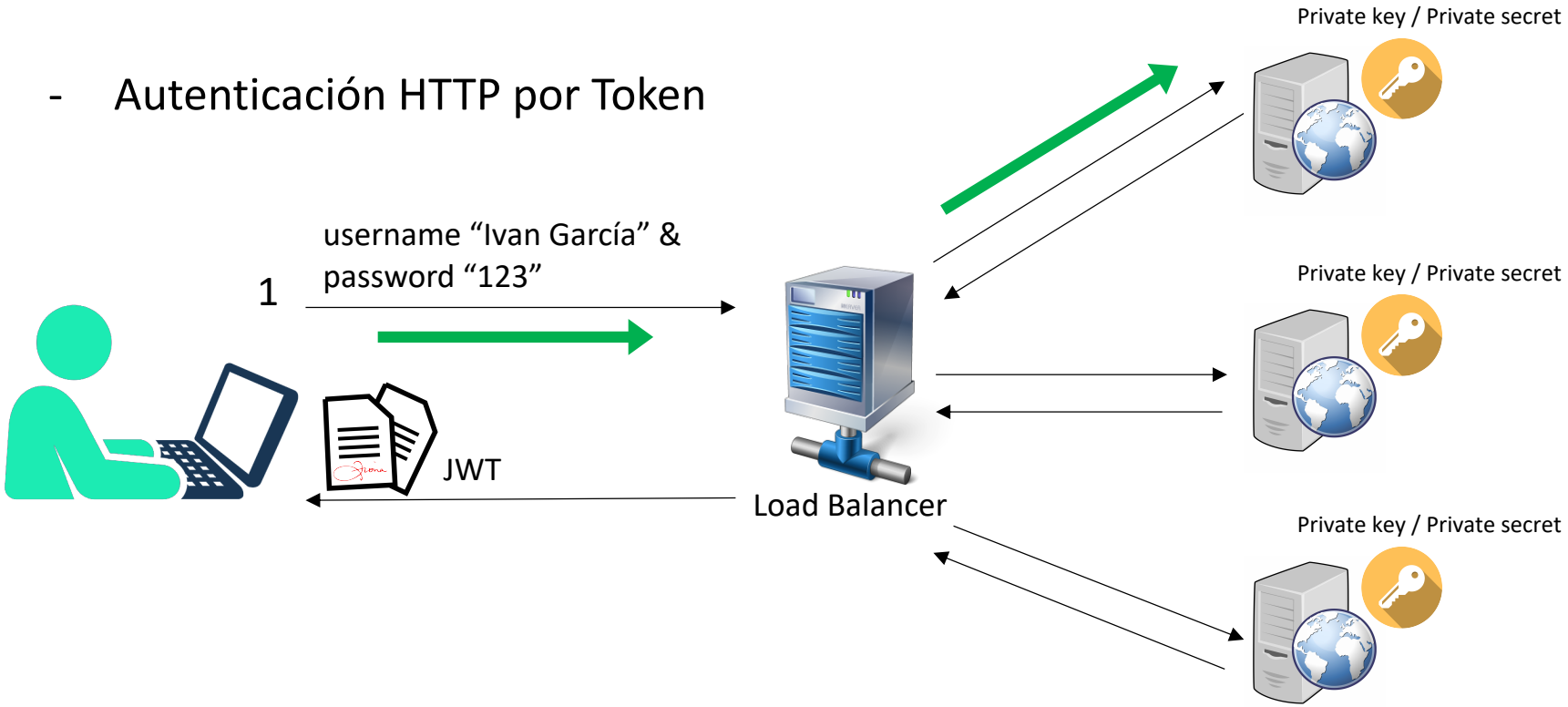
ix.i Autenticación HTTP por Sesión vs Token (g')

- Autenticación HTTP por Token



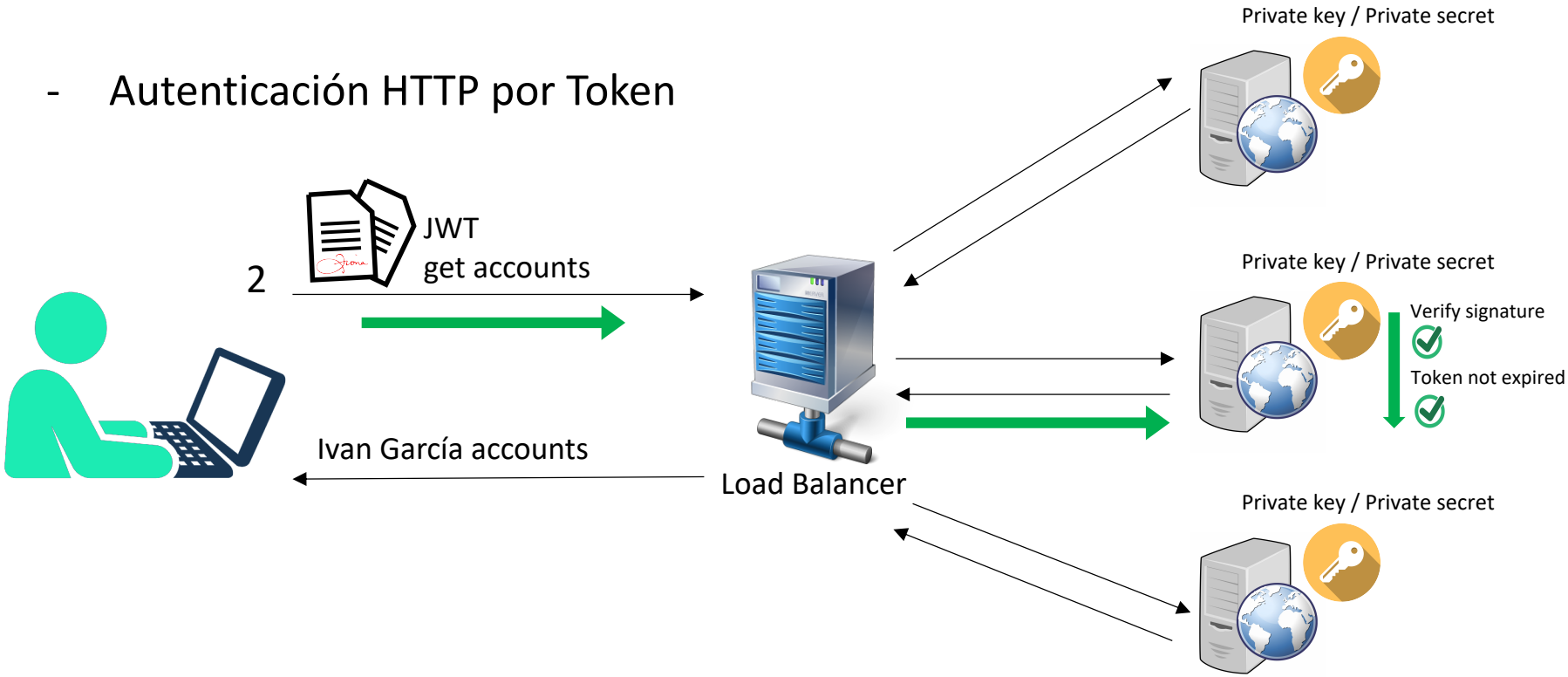
ix.i Autenticación HTTP por Sesión vs Token (h')

- Autenticación HTTP por Token



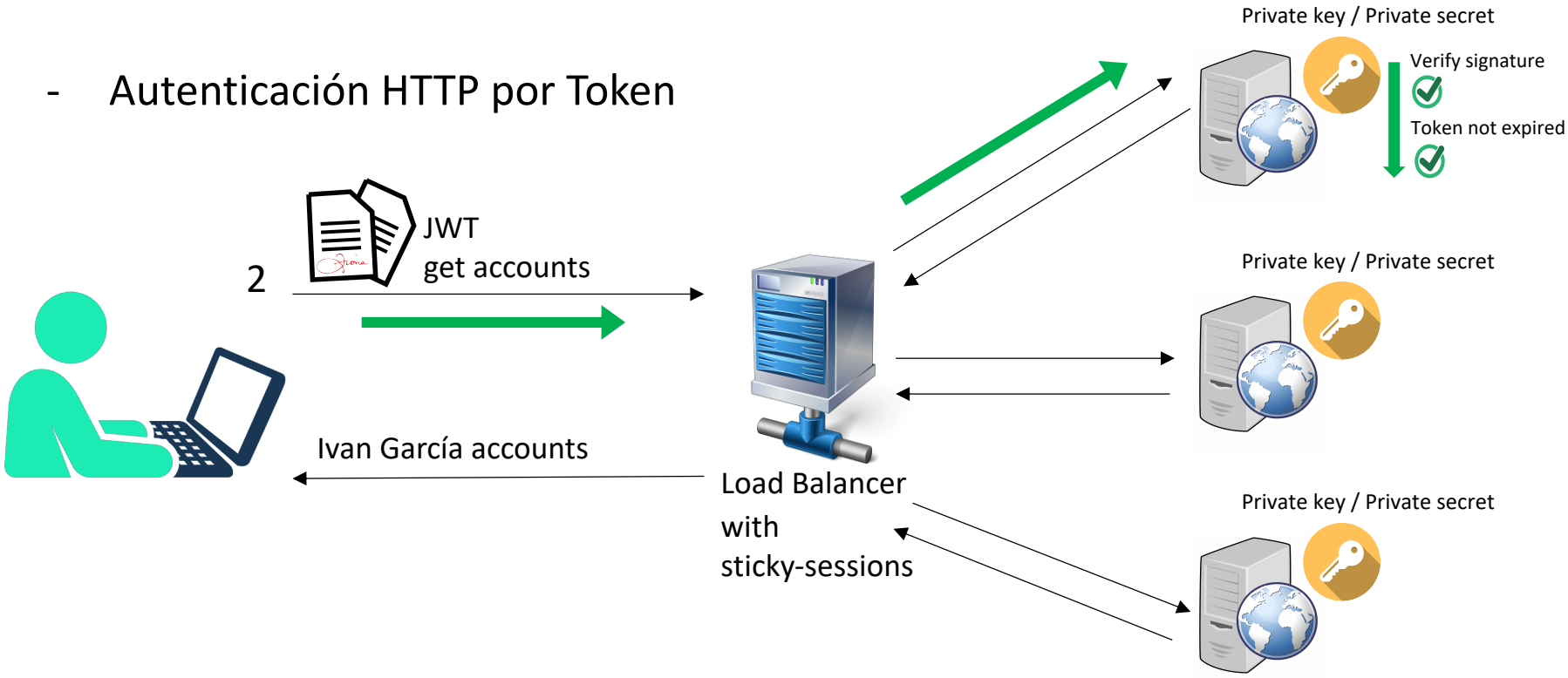
ix.i Autenticación HTTP por Sesión vs Token (i')

- Autenticación HTTP por Token



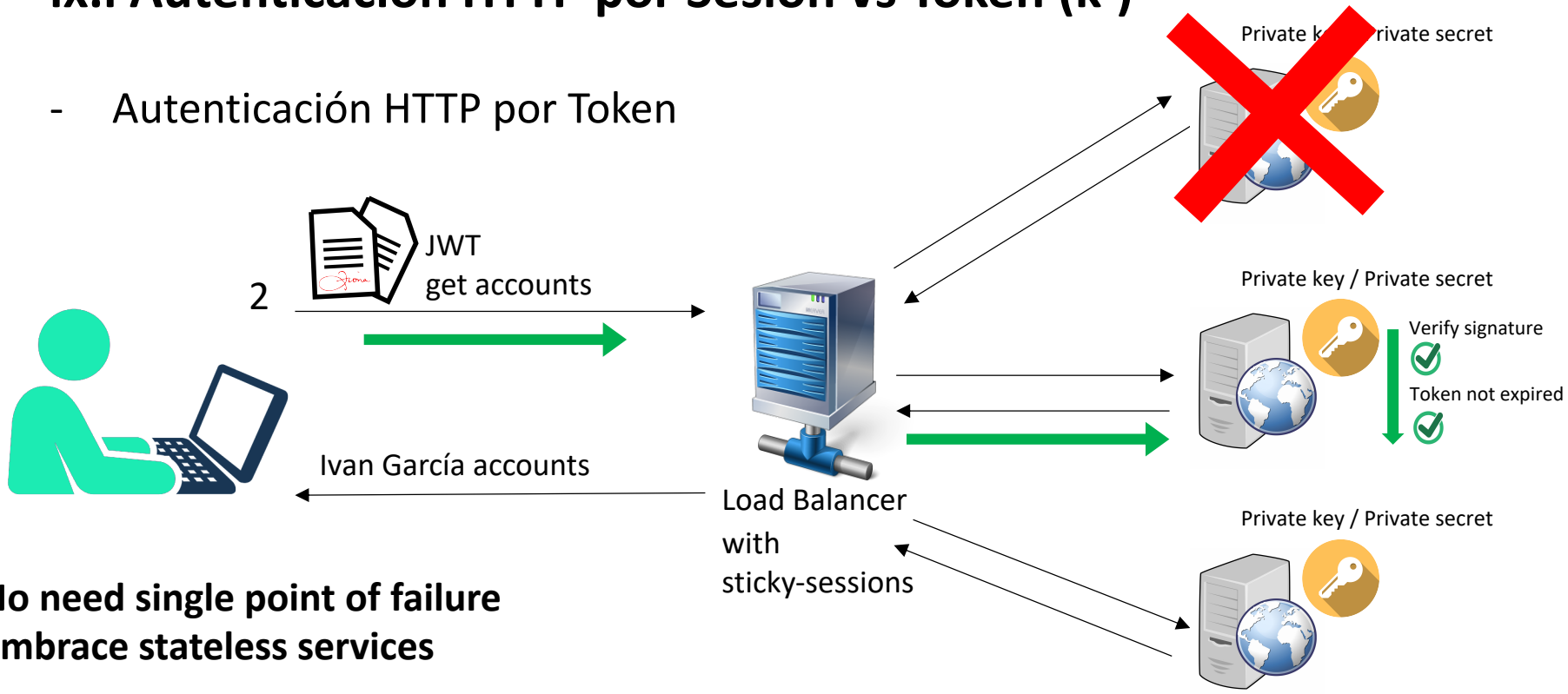
ix.i Autenticación HTTP por Sesión vs Token (j')

- Autenticación HTTP por Token



ix.i Autenticación HTTP por Sesión vs Token (k')

- Autenticación HTTP por Token



No need single point of failure
Embrace stateless services

ix.i Autenticación HTTP por Sesión vs Token (I')

- Escalabilidad en autenticación HTTP por Token

- Generalmente las aplicaciones que implementan autenticación HTTP basada en tokens, comúnmente utilizando JSON Web Token, son más fáciles de escalar debido a que los servicios y la aplicación web en si, no mantiene estado, es decir, son Stateless.

- La autenticación HTTP basada en tokens evita la necesidad de almacenar información del cliente del lado del servidor.

ix.i Autenticación HTTP por Sesión vs Token (m')

- Seguridad en autenticación HTTP por Token

- Los tokens generados por el servidor, usualmente son almacenados en el *local storage* del navegador o a través de cookies y mediante JavaScript es posible acceder a estos recursos.

- Debido a lo anterior, los tokens son vulnerables a ataques XSS (Cross-site Scripting), por ello no se recomienda almacenar información sensible en los tokens.

ix.i Autenticación HTTP por Sesión vs Token (n')

- Seguridad en autenticación HTTP por Token

- Las cookies son vulnerables a ataques CSRF (Cross-site Request Forgery). Una forma de evitar los ataques CSRF es asegurar que las cookies son unicamente accesibles por el dominio de la aplicación web (utilizar atributo Secure y HttpOnly).

- La recomendación para evitar el daño que puede causar un atacante al posible robo de un token es utilizar siempre un protocolo seguro de comunicación como HTTPS/TLS y, agregar y validar periodos de vigencia cortos del token.

Resumen de la lección

ix.i Introducción

- Analizamos lo que son los servicios REST con y sin estado (Stateful y Stateless).
- Verificamos las diferencias entre servicios y aplicaciones con y sin estado.
- Comprendimos el funcionamiento sin estado del protocolo HTTP.
- Analizamos la autenticación HTTP por Sesión.
- Analizamos la autenticación HTTP por Token.
- Verificamos las diferencias entre autenticación HTTP por Sesión y autenticación HTTP por Token.

Esta página fue intencionalmente dejada en blanco.

ix. Seguridad en Servicios REST (b)

ix.ii Autenticación HTTP con Spring Security

a. Autenticación Básica

Práctica 33. Implementación de Autenticación Básica de Servicios REST

b. Autenticación Digest (resumen)

Práctica 34. Implementación de Autenticación Digest de Servicios REST

c. Autenticación Bearer con JSON Web Token (JWT)

Práctica 35. Implementación de Autenticación Bearer con JWT de Servicios REST

ix.ii Autenticación HTTP con Spring Security

Objetivos de la lección

ix.ii Autenticación HTTP con Spring Security

- Implementar autenticación Básica de servicios REST mediante Spring Security a través de configuración por XML y JavaConfig.
- Implementar autenticación Digest de servicios REST mediante Spring Security a través de configuración por XML y JavaConfig.
- Implementar autenticación por token JWT de servicios REST mediante Spring Security a través de configuración JavaConfig.

ix.ii Autenticación HTTP con Spring Security

a. Autenticación Básica

Práctica 33. Implementación de Autenticación Básica de Servicios REST

b. Autenticación Digest (resumen)

Práctica 34. Implementación de Autenticación Digest de Servicios REST

c. Autenticación Bearer con JSON Web Token (JWT)

Práctica 35. Implementación de Autenticación Bearer con JWT de Servicios REST

ix.ii Autenticación Básica (a)

- HTTP Basic Authentication.

- La autenticación Básica o *Basic HTTP Authentication* está definido en el RFC 7617 el cual transmite las credenciales del usuario como un par usuario:contraseña codificado en base64.

- Al utilizar la autenticación Básica se envian las credenciales en texto plano, por lo tanto únicamente se recomienda utilizarlo a través de protocolo seguro HTTPS/TLS.

ix.ii Autenticación Básica (b)

- Cuando un cliente HTTP trata de acceder a un recurso protegido con Autenticación Básica, el servidor debe enviar un código de estatus HTTP **401 Unauthorized** y el encabezado **WWW-Authenticate** con el formato: **WWW-Authenticate: Basic realm="some-domain"**.

- El navegador o cliente HTTP al recibir el encabezado de respuesta **WWW-Authenticate: Basic realm="some-domain"** debe autenticarse a través de autenticación básica.

ix.ii Autenticación Básica (c)

- Para ejecutar la autenticación básica, el cliente HTTP debe utilizar el encabezado **Authorization: Basic <base64(user:password)>** para enviar e informar las credenciales al servidor.

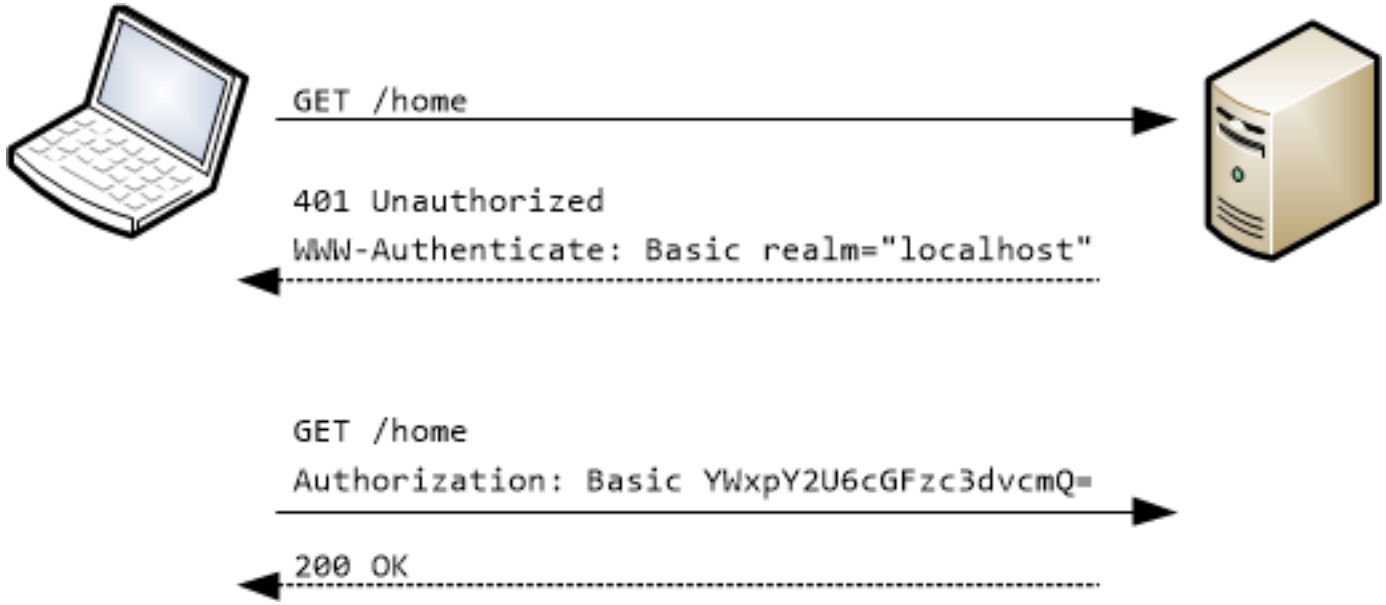
- En caso de que las credenciales sean válidas y se obtenga acceso al recurso solicitado, el servidor debe devolver un código de estatus **HTTP 200 OK**.

ix.ii Autenticación Básica (d)

- En caso de que las credenciales sean válidas, pero la autorización al acceso a dicho recurso sea denegado, el servidor debe devolver un código de estatus **HTTP 403 Forbidden**.

ix.ii Autenticación Básica (e)

- HTTP Basic Authentication.



ix.ii Autenticación Básica (f)

- Implementación de Autenticación Básica con Spring Security.

- Para la implementación de autenticación Básica con Spring Security en una aplicación Spring MVC, es necesario agregar las siguientes dependencias:
  - spring-security-core-<version>.jar
  - spring-security-config-<version>.jar
  - spring-security-web-<version>.jar
  - spring-security-taglibs-<version>.jar
  - spring-security-aspects-<version>.jar
  - spring-security-acl-<version>.jar
  - spring-security-ldap-<version>.jar
  - spring-security-openid-<version>.jar
  - spring-security-cas-<version>.jar
  - spring-security-crypto-<version>.jar
  - spring-security-remoting-<version>.jar

ix.ii Autenticación Básica (g)

- Implementación de Autenticación Básica con Spring Security, configuración por XML y web.xml.

- Configuración habitual del dispatcherServlet por web.xml (a).

```
<servlet>
    <servlet-name>dispatcherServlet</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/spring/mvc/dispatcherServlet-servlet.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>
```

ix.ii Autenticación Básica (h)

- Configuración habitual del dispatcherServlet por web.xml (b).

```
<servlet-mapping>
    <servlet-name>dispatcherServlet</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>
```

ix.ii Autenticación Básica (i)

- Configuración habitual del ContextLoaderListener por web.xml, el cual levanta el Root Application Context y el Security Context.

```
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/spring/root-application-context.xml,
        /WEB-INF/spring/security/spring-security-application-context.xml</param-value>
</context-param>

<listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
```

ix.ii Autenticación Básica (j)

- Configuración habitual del filtro springSecurityFilterChain el cual filtra todas las peticiones entrantes a la aplicación.

```
<filter>
    <filter-name>springSecurityFilterChain</filter-name>
    <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
</filter>

<filter-mapping>
    <filter-name>springSecurityFilterChain</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

ix.ii Autenticación Básica (k)

- Debido a que el requerimiento es implementar autenticación básica a servicios REST, se asume que no habrán vistas ni recursos css/js/imágenes que Spring MVC tenga que servir a los clientes HTTP.

- Por lo anterior, únicamente es necesario habilitar configuración por anotaciones de Spring MVC para habilitar el escaneo de controladores @RestController en el dispatcherServlet-servlet.xml.

```
<context:component-scan base-package=”controller.base.package" />

<mvc:annotation-driven />
```

ix.ii Autenticación Básica (I)

- En caso de implementar configuración por Java Config, aplicar @EnableWebMvc y el escaneo de controladores mediante @ComponentScan en una clase de configuración @Configuration.

- De igual forma, se asume que no habrán vistas ni recursos css/js/imágenes que Spring MVC tenga que servir a los clientes HTTP.

```
@Configuration
@ComponentScan(basePackages = "controller.base.package")
@EnableWebMvc
```

ix.ii Autenticación Básica (m)

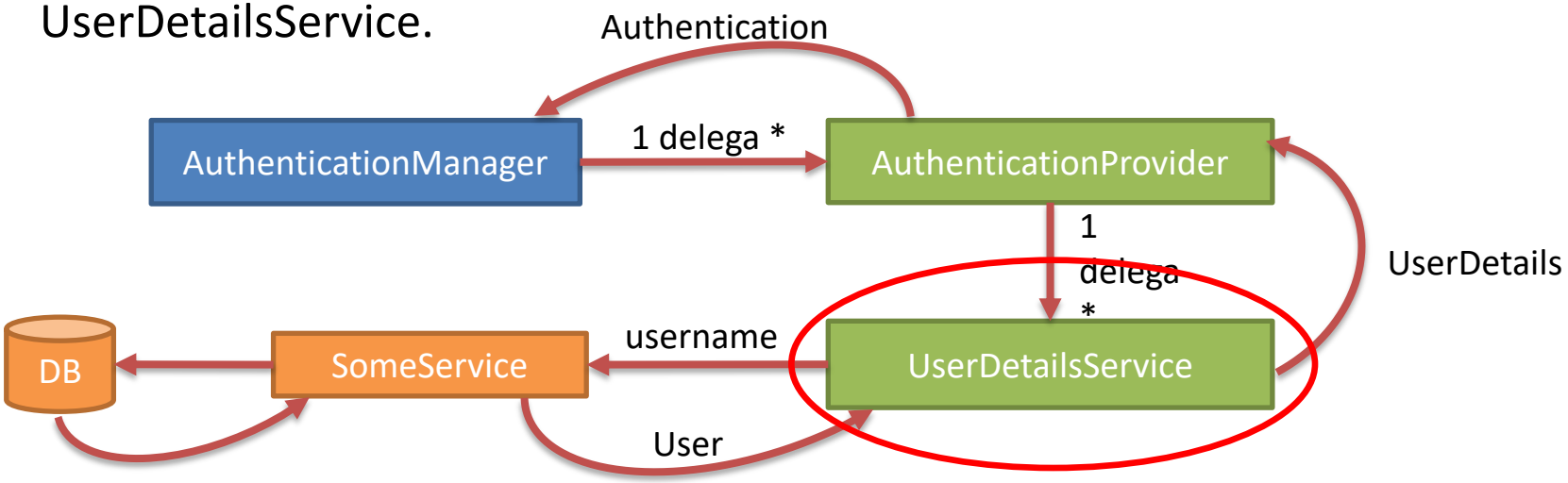
- Configuración del AuthenticationManager mediante autenticación en memoria (útil para fase de desarrollo y pruebas) en el spring-security-application-context.xml.

```
<security:authentication-manager>
    <security:authentication-provider>
        <security:user-service>
            <security:user name="admin" password="admin“ authorities="ROLE_ADMIN" />
            <security:user name="xvanhalenx" password="123123“
                                                    authorities="ROLE_ROOT,ROLE_ADMIN" />
            <security:user name="user" password="user“ authorities="ROLE_USER" />
        </security:user-service>
    </security:authentication-provider>
</security:authentication-manager>
```

In Memory UserService Authentication

ix.ii Autenticación Básica (n)

- Par ambientes productivo, es posible configurar el Authentication Manager a través de delegar la obtención de usuarios registrados en la aplicación mediante la implementación de la interface UserDetailsService.



ix.ii Autenticación Básica (ñ)

- Configuración del AuthenticationManager mediante UserDetailsService (a).

```
<security:authentication-manager>
    <security:authentication-provider user-service-ref="userDetails">
    </security:authentication-provider>
</security:authentication-manager>

<bean id="userDetails" class="CustomUserDetailsService">
```

ix.ii Autenticación Básica (o)

- Configuración del AuthenticationManager mediante UserDetailsService (b).

```
public interface UserDetailsService {
   UserDetails loadUserByUsername(String username) throws UsernameNotFoundException;
}

public class CustomUserDetailsService implements UserDetailsService {
    @Override
    public UserDetails loadUserByUsername(String username)
                                                    throws UsernameNotFoundException {
        User user = retrieveFromSomeRepository(username);
        return user;
    }
}
```

ix.ii Autenticación Básica (p)

- Configuración de autorización a nivel de URL.

Evita crear Cookie de Sesión (JSESSIONID)

```
<security:http use-expressions="true“ create-session="stateless">
    <security:intercept-url pattern="/welcome“ access="isAuthenticated()" />
    <security:intercept-url pattern="/login“ access="permitAll" />
    <security:intercept-url pattern="/root*/**“
                                        access="hasAnyRole('ROLE_ROOT','ROLE_ADMIN')" />
    <security:intercept-url pattern="/admin*/**“ access="hasAnyRole('ROLE_ADMIN')" />
    <security:intercept-url pattern="/user*/**“ access="hasRole('ROLE_USER')" />

    <security:http-basic />

</security:http>
```

Agrega soporte para autenticación básica

ix.ii Autenticación Básica (q)

- Configuración de autorización básica.

- La propiedad **create-session** "**stateless**", de la etiqueta <security:http> evita la generación de Cookie de sesión **JSESSIONID**.

- La etiqueta **<security:http-basic />** habilita soporte para autenticación HTTP básica, incorporada en Spring Security.

```
<security:http use-expressions="true“ create-session="stateless">
    <security:http-basic />
</security:http>
```

ix.ii Autenticación Básica (r)

- Configuración de autorización básica.

- Spring Security, por default, envía el encabezado **WWW-Authenticate** en la respuesta al cliente HTTP, cuando se trata de acceder a un recurso sin la autenticación básica correspondiente.

(HTTP Response)
HTTP/1.1 401
WWW-Authenticate: Basic realm="Spring Security Application"
Content-Type: text/html;charset=utf-8
…

ix.ii Autenticación Básica (s)

- Configuración de autorización básica.

- Para configurar el encabezado de respuesta **WWW-Authenticate**, para cambiar el valor del "**realm**", es necesario definir y configurar el bean **BasicAuthenticationEntryPoint** y asignarlo como "**entry-point-ref**" en la etiqueta **<security:http-basic />**.

```
<security:http use-expressions="true“ create-session="stateless">
    <security:http-basic entry-point-ref="basicAuthenticationEntryPoint" />
</security:http>
```

ix.ii Autenticación Básica (t)

- Configuración de autorización básica.

- Definición del bean **BasicAuthenticationEntryPoint**.

```
<bean id="basicAuthenticationEntryPoint"
    class="org.springframework.security.web.authentication.www.BasicAuthenticationEntryPoint">
    <property name="realmName" value="My application realm" />
</bean>
```

ix.ii Autenticación Básica (u)

- HTTP Response cuando se trata de acceder a un recurso sin la autenticación básica correspondiente, configurando el bean **BasicAuthenticationEntryPoint**.

(HTTP Response)
HTTP/1.1 401
WWW-Authenticate: Basic realm="My application realm"
Content-Type: text/html;charset=utf-8
Content-Language: es
Content-Length: 1057
Date: Tue, 24 Dec 2020 20:10:11 GMT

ix.ii Autenticación Básica (v)

- Para realizar la petición de un recurso protegido con autenticación básica, es necesario agregar el encabezado **Authorization** con el valor "**Basic base64(username:password)**".

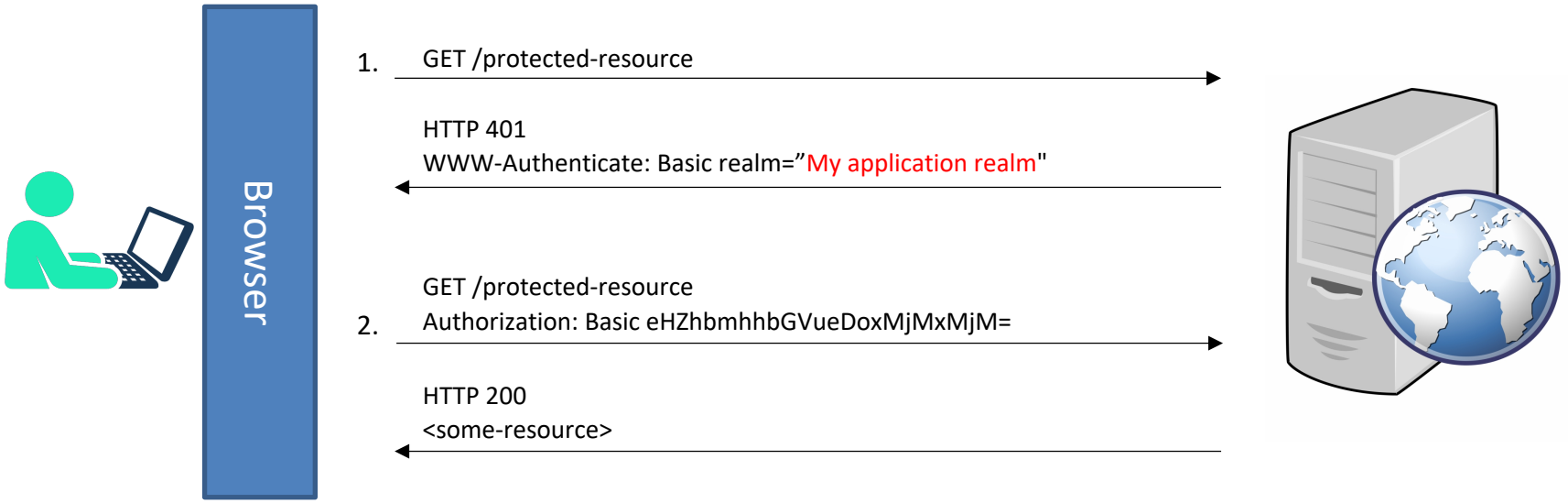
(HTTP Request)
GET /some-protected-resource HTTP/1.1
Host: localhost:8080
Authorization: Basic eHZhbmhhbGVueDoxMjMxMjM=
User-Agent: curl/7.64.1
Accept: */*

(HTTP Response)
HTTP/1.1 200
Content-Type: text/plain;charset=ISO-8859-1
Content-Length: 21
Date: Tue, 24 Dec 2020 20:13:37 GMT

Protected resource...

ix.ii Autenticación Básica (w)

- Autenticación Básica



Bonus. Spring Security Java Config (a)

- Spring Security se permite configurar en base a Java Config, a través de la anotación **@EnableWebSecurity** sobre una clase de configuración **@Configuration** la cual debe heredar de la clase **WebSecurityConfigurerAdapter**.

```
@Configuration
@EnableWebSecurity
public class SecurityContextConfiguration extends WebSecurityConfigurerAdapter {
    ...
}
```

Bonus. Spring Security Java Config (b)

- Para la configuración del AuthenticationManager a través de Java config, es necesario sobre-escribir el método **configure(AuthenticationManagerBuilder auth)** de la clase definida que hereda de **WebSecurityConfigurerAdapter**.

- El método **configure(AuthenticationManagerBuilder auth)** recibe un objeto **AuthenticationManagerBuilder**, el cual permite configurar el autentication manager mediante *method chaining.*

- **AuthenticationManagerBuilder** permite configurar un **inMemoryAuthentication** o definiendo un bean **UserDetailsService.**

ix. Seguridad en Servicios REST - ix.ii Autenticación HTTP con Spring Security

Bonus. Spring Security Java Config (c)

- Configuración de **inMemoryAuthentication** mediante **AuthenticationManagerBuilder**.

```
@Configuration
@EnableWebSecurity
public class SecurityContextConfiguration extends WebSecurityConfigurerAdapter {
    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.inMemoryAuthentication()
            .withUser("admin").password("admin").roles("ADMIN").and()
            .withUser("xvanhalenx").password("123123").roles("ROOT", "ADMIN").and()
            .withUser("user").password("user").roles("USER");
    }
}
```

ix. Seguridad en Servicios REST - ix.ii Autenticación HTTP con Spring Security

Bonus. Spring Security Java Config (d)

- Configuración de **UserDetailsService** mediante **AuthenticationManagerBuilder**.

```
@Configuration
@EnableWebSecurity
public class SecurityContextConfiguration extends WebSecurityConfigurerAdapter {
    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.userDetailsService(customUserDetailsService());
    }
    @Bean
    public UserDetailsService customUserDetailsService() {
        return new CustomUserDetailsService();
    }
}
```

ix. Seguridad en Servicios REST - ix.ii Autenticación HTTP con Spring Security

Bonus. Spring Security Java Config (e)

- Para configurar la autorización a nivel de URL, definir login form así como todas las configuraciónes que se realizan sobre <security:http>, es necesario sobre-escribir el método **configure(HttpSecurity http)** de la clase definida que hereda de **WebSecurityConfigurerAdapter**.

- El método **configure(HttpSecurity http)** recibe un objeto HttpSecurity, el cual permite configurar las autorizaciones y demás configuraciones (como el soporte de autenticación básica) de Spring security mediante *method chaining.*

Bonus. Spring Security Java Config (f)

- Configuración de autorización a nivel de URL y soporte de autenticación básica a través de HttpSecurity.

```
@Configuration
@EnableWebSecurity
public class SecurityContextConfiguration extends WebSecurityConfigurerAdapter {
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.csrf().disable()
            .authorizeRequests()
                .antMatchers("/welcome").authenticated()
                .antMatchers("/").permitAll()...
            .and().httpBasic();
    }
}
```

ix. Seguridad en Servicios REST - ix.ii Autenticación HTTP con Spring Security

Bonus. Spring Security Java Config (g)

- Configuración para evitar la creación de Cookie de sesión **JSESSION** y perzonalización de ”**realm**” sobre encabezado **WWW-Authenticate**.

```
http.csrf().disable()
    .authorizeRequests().
        .antMatchers("/welcome").authenticated()…
    .and()
    .httpBasic().realmName(“My Application Realm”)
    .and()
    .sessionManagement()
    .sessionCreationPolicy(SessionCreationPolicy.STATELESS);
```

ix.ii Autenticación HTTP con Spring Security

a. Autenticación Básica

Práctica 33. Implementación de Autenticación Básica de Servicios REST

b. Autenticación Digest (resumen)

Práctica 34. Implementación de Autenticación Digest de Servicios REST

c. Autenticación Bearer con JSON Web Token (JWT)

Práctica 35. Implementación de Autenticación Bearer con JWT de Servicios REST

Bonus. No web.xml deployment (a)

- A partir de la especificación Servlet 3.0+, es posible evitar definir el descriptor de despliegue web.xml en aplicaciónes Java Web.

- Primeramente es encesario agregar la dependencia correspondientes en el pom.xml de maven.

```
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>3.0.1</version>
    <scope>provided</scope>
</dependency>
```

Bonus. No web.xml deployment (b)

- También, es requerido agregar el plugin maven-war-plugin de maven para evitar que la construcción y despliegue del aplicativo WAR falle al no detectar el archivo /WEB-INF/web.xml.

```
<plugins>
    <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-war-plugin</artifactId>
        <version>2.6</version>
        <configuration>
            <failOnMissingWebXml>false</failOnMissingWebXml>
        </configuration>
    </plugin>
</plugins>
```

Bonus. No web.xml deployment (c)

- Debido a que ya no será requerido definir el descriptor de despliegue web.xml:

  ¿Cómo definimos el Web Application Context del DispatcherServlet?

  ¿Cómo definimos el mapeo del servlet DispatcherServlet?

  ¿Cómo definimos el Root Application Context a través del ContexLoaderListener?

  ¿Cómo definimos y mapeamos el DelegatingFilterProxy springSecurityFilterChain para que Spring Security filtre y aplique la seguridad?

Bonus. No web.xml deployment (d)

- Para definir el **Root Application Context**, el **Web Application Context**, registrar el **DispatcherServlet** y definir el "**servlet-mapping**" del **DispatcherServlet** es necesario definir una clase que herede de **AbstractAnnotationConfigDispatcherServletInitializer**.

- La clase heredada de **AbstractAnnotationConfigDispatcherServletInitializer**, definirá la configuración Java Config para el **Root** y el **Web Application Context** a través de clases **@Configuration**. 100% Java Config.

Bonus. No web.xml deployment (e)

- Definición de **AbstractAnnotationConfigDispatcherServletInitializer**.

```
public class SpringWebMvcDispatcherServletInitializer extends AbstractAnnotationConfigDispatcherServletInitializer {
    @Override
    protected Class<?>[] getRootConfigClasses() {
        return new Class[] { RootContextConfiguration.class, SecurityContextConfiguration.class };
    }
    @Override
    protected Class<?>[] getServletConfigClasses() {
        return new Class[] { ServletContextConfiguration.class };
    }
    @Override
    protected String[] getServletMappings() {
        return new String[] { "/" };
    }
}
```

Bonus. No web.xml deployment (f)

- Clases de configuración **@Configuration**.

```
@Configuration
@EnableWebSecurity
public class SecurityContextConfiguration
                            extends WebSecurityConfigurerAdapter {
    @Override
    protected void configure(
        AuthenticationManagerBuilder auth) throws Exception {
        auth.userDetailsService(customUserDetailsService());
    }
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.csrf().disable()
        .authorizeRequests()
        .antMatchers("/welcome").authenticated()
    }
    // @Bean definitions
}
```

```
@Configuration
public class RootContextConfiguration {
    // @Bean definitions
}
```

```
@Configuration
@EnableWebMvc
@ComponentScan(basePackages = "controller.base.package")
public class ServletContextConfiguration {
    // @Bean definitions
}
```

Bonus. No web.xml deployment (g)

- Debido a que ya no será requerido definir el descriptor de despliegue web.xml:

¿Cómo definimos el Web Application Context del DispatcherServlet?

¿Cómo definimos el mapeo del servlet DispatcherServlet?

¿Cómo definimos el Root Application Context a través del ContexLoaderListener?

¿Cómo definimos y mapeamos el DelegatingFilterProxy springSecurityFilterChain para que Spring Security filtre y aplique la seguridad?

Bonus. No web.xml deployment (h)

- Para definir el **DelegatingFilterProxy** springSecurityFilterChain, filtro el cual da entrada a Spring Security para filtrar los requests entrantes a la aplicación web, es encesario definir una clase que herede de **AbstractSecurityWebApplicationInitializer**.

- La clase heredada de **AbstractSecurityWebApplicationInitializer**, filtra por default los requests entrantes mediante el mapeo "**/***" es decir, el filtro springSecurityFilterChain filtra todos los requests entrantes a la aplicación web.

Bonus. No web.xml deployment (i)

- Definición de **AbstractSecurityWebApplicationInitializer**.

```
public class SpringSecurityFilterChainInitializer
                                    extends AbstractSecurityWebApplicationInitializer {
    // Nothing to override
}
```

Bonus. No web.xml deployment (j)

- Debido a que ya no será requerido definir el descriptor de despliegue web.xml:

- ¿Cómo definimos el Web Application Context del DispatcherServlet?
- ¿Cómo definimos el mapeo del servlet DispatcherServlet?
- ¿Cómo definimos el Root Application Context a través del ContexLoaderListener?
- ¿Cómo definimos y mapeamos el DelegatingFilterProxy springSecurityFilterChain para que Spring Security filtre y aplique la seguridad?

ix.ii Autenticación Básica. Práctica 33. (a)

- Práctica 33. Implementación de Autenticación Básica de Servicios REST

- Configurar Spring Security para securizar servicios REST en una aplicación Spring en contexto web *(WebApplicationContext)*.
- Implementar configuración de Spring Security con XML y descriptor web.xml.
- Implementar configuración de Spring Security con @Anotaciones y sin descriptor web.xml.
- Implementar autenticación Básica mediante Spring Security.

ix.ii Autenticación HTTP con Spring Security

a. Autenticación Básica

Práctica 33. Implementación de Autenticación Básica de Servicios REST

b. Autenticación Digest (resumen)

Práctica 34. Implementación de Autenticación Digest de Servicios REST

c. Autenticación Bearer con JSON Web Token (JWT)

Práctica 35. Implementación de Autenticación Bearer con JWT de Servicios REST

ix.ii Autenticación Digest (resumen) (a)

- HTTP Digest Authentication.

- La autenticación Digest (hash/resumen) o Digest HTTP Authentication está definido en el **RFC 2617** el cual es una actualización del **RFC 2069**.

- Digest HTTP Authentication presenta un mecanismo para implementar autenticación mediante encabezados HTTP sin enviar la contraseña en texto plano, contrarestando la vulnerabilidad de la autenticación básica. La autenticación Digest es preferible a la autenticación Básica en conexiónes que no son cifradas, es decir, que no implementan HTTPS/TLS.

ix.ii Autenticación Digest (resumen) (b)

- La autenticación Digest utiliza la función *hash* **MD5**, para digerir ("*hashear*") la contraseña y, de esta forma, evitar enviar la contraseña en texto plano a través de la red.

- El algoritmo de autenticación Digest implementa un mecanismo para evitar ataques de repetición a través de un "***nonce***".

- ***Nonce*** en computación significa "un número que puede utilizarse una única vez".

ix.ii Autenticación Digest (resumen) (c)

- La autenticación Digest implementa un mecanismo para que, a través del intercambio de variables, tanto el cliente como el servidor puedan ambos calcular un *hash* o "resumen" para de esta forma, asegurar que el cliente es quien dice ser, en otras palabras, autenticarlo.

ix.ii Autenticación Digest (resumen) (d)

- Autenticación Digest RFC 2069.

- Para implementar la autenticaicón Digest, el cliente HTTP, utiliza el password y otras variables (bits de información) para calcular un *hash* el cual, al recibir el servidor dichas variables de información, pueda éste calcular el mismo *hash* y de esta forma identificar que el cliente es quien dice ser.

ix.ii Autenticación Digest (resumen) (e)

- Autenticación Digest RFC 2069.

- El servidor calcula dos numeros (hexadecimales) aleatorios, un número ***nonce*** y otro número ***opaque***. El número ***nonce*** debe ser único e irrepetible.

- Tanto el ***nonce*** como el ***opaque*** deben ser enviados al cliente a través del encabezado **WWW-Authenticate**.

```
WWW-Authenticate: Digest realm=”My application realm",
nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
opaque="5ccc069c403ebaf9f0171e9517f40e41"
```

ix.ii Autenticación Digest (resumen) (f)

- Autenticación Digest RFC 2069.

- El número ***opaque*** que calcula y envia el servidor al cliente HTTP, se espera sea enviado por el cliente HTTP al servidor sin modificar; éste número sirve para mantener estado entre peticiones.

- El cliente HTTP, al recibir los valores ***nonce***, ***opaque*** y ***realm*** en el encabezado **WWW-Authenticate**, los utiliza para computar un *hash* y así enviar las “credenciales” para su autenticación.

ix.ii Autenticación Digest (resumen) (g)

- Autenticación Digest RFC 2069.

- El cliente HTTP calcula el siguiente resumen, *hash* o ***digest***, para el intercambio de "credenciales" y lo envía a través del encabezado **Authorization** para su autenticación.

  hash1 = md5(username:realm:password)
  hash2 = md5(requestMethod:requestURI)
  response = md5(hash1:nonce:hash2)

  Authorization: Digest username="**xvanhalenx**",
  realm="**My application realm**", nonce="*<nonce>*", opaque="*<opaque>*",
  uri="**/some-uri**", response="*<response>*"

ix.ii Autenticación Digest (resumen) (h)

- Autenticación Digest RFC 2069.

- El servidor, al recibir el encabezado **Authorization**, obtiene los valores ***username***, ***realm***, ***nonce***, ***opaque***, ***uri*** y ***response*** y, realiza el cálculo del hash correspondiente.

  hash1 = md5(username:realm:password)
  hash2 = md5(requestMethod:requestURI)
  response = md5(hash1:nonce:hash2)

- Para poder calcular *hash1*, debe consultar en base de datos el password del usuario dado por ***username*** y validar que el valor ***realm***, sea el que el servidor envio previamente.

ix.ii Autenticación Digest (resumen) (i)

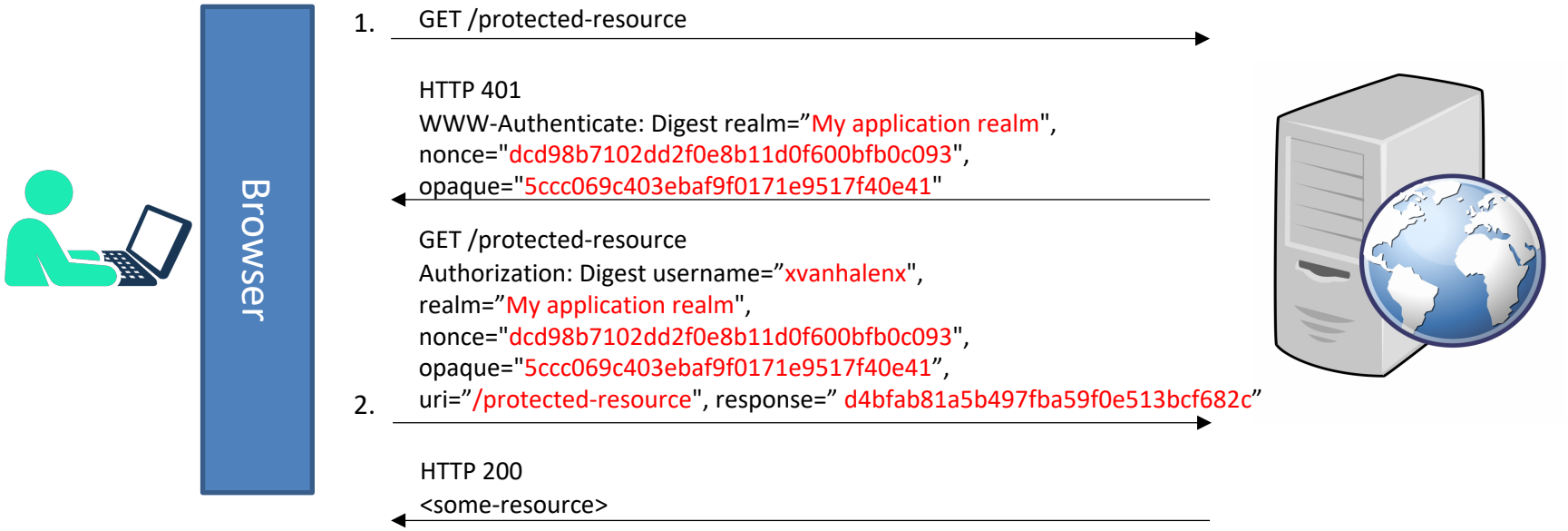
- Autenticación Digest RFC 2069.

- El servidor verifica el valor del ***response*** obtenido en el encabezado **Authorization**, con el calculado a través de ***hash1*** y ***hash2*** y, si son iguales, da la autenticación como satisfactoria.

- De este modo se autentica un cliente mediante autenticación Digest sin intercambiar la contraseña en texto claro (ni en Base 64).

ix.ii Autenticación Digest (resumen) (j)

- Autenticación Digest RFC 2069.



ix.ii Autenticación Digest (resumen) (k)

- Autenticación Digest RFC 2617.

- La autenticación Digest definida en el RFC 2617 agrega las variables ***qop*** (*quality of protection*), ***cnonce*** (*client nonce*) y ***nc*** (nounce count).

- ***qop*** (*quality of protection*): Se especifica en el encabezado **WWW-Authenticate** y puede tener el valor "**auth**" o "**auth-int**". Si no se especifica ningun valor, por defecto es "**auth**". El valor de ***qop*** "**auth**" define que únicamente es necesario autenticar al usuario. El valor de ***qop*** "**auth-int**" define que es necesario autenticar al usuario y verificar la integridad de la respuesta (no soportada por Spring Security).

ix.ii Autenticación Digest (resumen) (I)

- Autenticación Digest RFC 2617.

- ***cnonce*** (*client nonce*): El valor ***cnonce*** es similar a ***nonce*** pero es generado por el cliente y enviado a través del encabzeado **Authorization**. Se utiliza el valor ***cnonce*** para calcular el resumen, *hash* o ***digest*** de la autenticación y comparar el ***response*** enviado con el calculado. Este valor proporciona integridad en la respuesta y autenticación mutua. Cuando la directiva ***qop*** se envía por parte del servidor, el cliente debe incluir un valor ***cnonce***.

ix.ii Autenticación Digest (resumen) (m)

- Autenticación Digest RFC 2617.

- ***nc*** (*nounce count*): El valor ***nc*** se refiere a una cuenta hexadecimal del número de requests que el cliente ha enviado con el valor de un mismo ***nonce***. De esta forma el servidor puede protegerse frente ataques de repetición.

ix.ii Autenticación Digest (resumen) (n)

- Autenticación Digest RFC 2617.

- El cliente HTTP calcula el siguiente resumen, *hash* o ***digest***, para el intercambio de "credenciales" y lo envía a través del encabezado **Authorization** para su autenticación.

  hash1 = md5(username:realm:password)
  hash2 = md5(requestMethod:requestURI)
  response = md5(hash1:nonce:nc:cnonce:qop:hash2)

  Authorization: Digest username="**xvanhalenx**",
  realm="**My application realm**", nonce="*<nonce>*", uri="**/some-uri**",
  cnonce="*<cnonce>*", nc="*<nc>*", qop="*<qop>*", response="*<response>*"

ix.ii Autenticación Digest (resumen) (ñ)

- Autenticación Digest RFC 2617.

- El servidor, al recibir el encabezado **Authorization**, obtiene los valores ***username***, ***realm***, ***nonce***, ***cnonce***, ***nc***, ***qop***, ***uri*** y ***response*** y, realiza el cálculo del hash correspondiente.

  hash1 = md5(username:realm:password)
  hash2 = md5(requestMethod:requestURI)
  response = md5(hash1:nonce:nc:cnonce:qop:hash2)

- Para poder calcular *hash1*, debe consultar en base de datos el password del usuario dado por ***username*** y validar que el valor ***realm***, sea el que el servidor envio previamente.

ix.ii Autenticación Digest (resumen) (o)

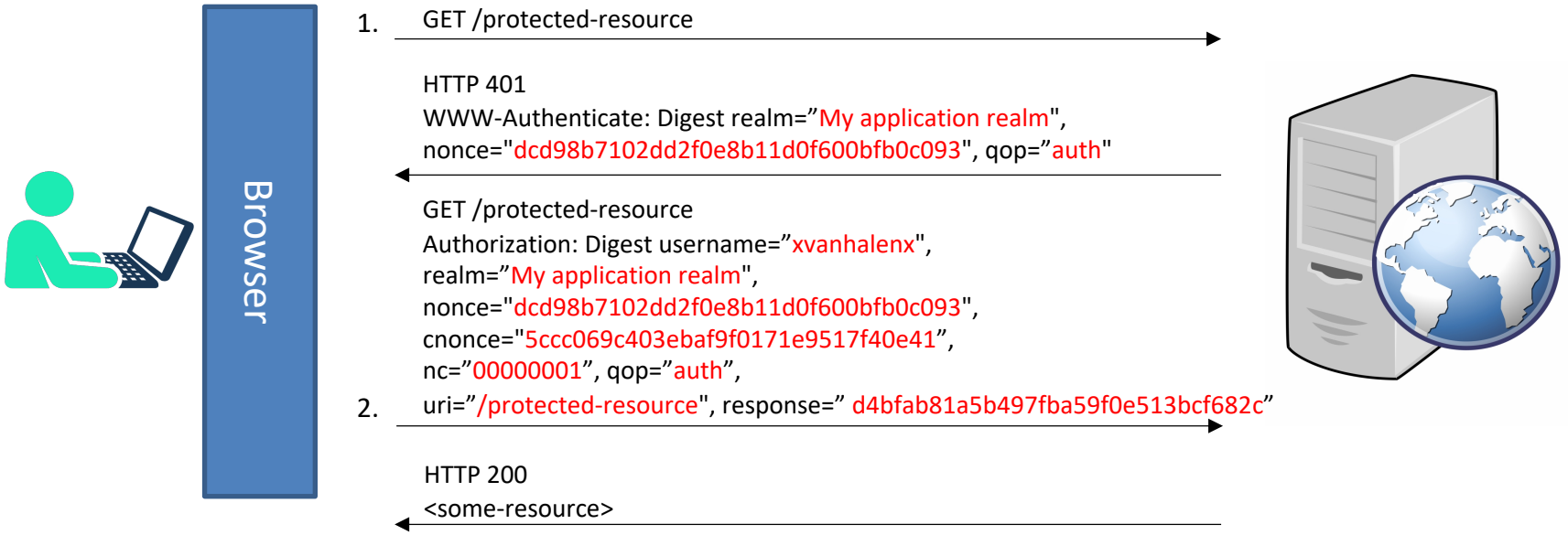
- Autenticación Digest RFC 2617.

- El servidor verifica el valor del ***response*** obtenido en el encabezado **Authorization**, con el calculado a través de ***hash1*** y ***hash2*** y, si son iguales, da la autenticación como satisfactoria.

- De este modo se autentica un cliente mediante autenticación Digest sin intercambiar la contraseña en texto claro (ni en Base 64).

ix.ii Autenticación Digest (resumen) (p)

- Autenticación Digest RFC 2617.



ix.ii Autenticación Digest (resumen) (q)

- La comunidad de Spring Security no recomienda la integración de autenticación Digest porque no se considera un mecanismo seguro debido a que:

- Es necesario almacenar las contraseñas en el repositorio de datos en **texto plano**, **encryptadas** o ***hasheadas*** con *MD5*. Cualquiera de estos formatos de almacenamiento se considera inseguro.

- Se recomienda almacenar las contraseñas ***hasheadas*** con "*salt*" mediante la aplicación de algoritmos adaptativos (one-way) como ***BCrypt***, ***PBKDF2***, ***SCrypt***, entre otros.

ix.ii Autenticación Digest (resumen) (r)

- A pesar de que la autenticación Digest no se considera un mecanismo seguro debido al modo de almacenamiento de las contraseñas, es preferible utilizar autenticación Digest en lugar de autenticación basada en formularios (*form-login*) o autenticación Básica cuando se utilice protocolos de comunicación no cifrados como HTTP.

- Utilice autenticación Digest siempre que no sea posible utilizar protocolo de comunicación cifrados como HTTPS/TLS.

- Si el protocolo de comunicación es cifrado, a través de HTTPS/TLS, la autenticación básica o por formulario no es vulnerable.

ix.ii Autenticación Digest (resumen) (s)

- Implementación de Autenticación Digest con Spring Security.

- Al igual que la autenticación Básica, para implementar autenticación Digest con Spring Security en una aplicación Spring MVC, es necesario agregar las siguientes dependencias:
  - spring-security-core-<version>.jar
  - spring-security-config-<version>.jar
  - spring-security-web-<version>.jar
  - spring-security-taglibs-<version>.jar
  - spring-security-aspects-<version>.jar
  - spring-security-acl-<version>.jar
  - spring-security-ldap-<version>.jar
  - spring-security-openid-<version>.jar
  - spring-security-cas-<version>.jar
  - spring-security-crypto-<version>.jar
  - spring-security-remoting-<version>.jar

ix.ii Autenticación Digest (resumen) (t)

- Implementación de Autenticación Digest con Spring Security, configuración por XML y web.xml.

- Configuración habitual del **dispatcherServlet** por web.xml a través del registro del servlet ***dispatcherServlet*** utilizando la clase **org.springframework.web.servlet.DispatcherServlet** (con su **<servlet-mapping>** correspondiente) e inicializando el **<init-param> contextConfigLocation** con la ruta al archivo **dispatcherServlet-servlet.xml** que configura el **Web Application Context** deseado (analizar láminas 77 y 78).

ix.ii Autenticación Digest (resumen) (u)

- Implementación de Autenticación Digest con Spring Security, configuración por XML y web.xml.

- Configuración habitual del **<context-param> contextConfigLocation** con **<param-value>** asignando la ruta de los archivos de configuración **Root Application Context** y/o del **Security Context** (configuración de Spring Security).

- Registrar el listener **org.springframework.web.context.ContextLoaderListener** para que carge la configuración definida en el **<context-param> contextConfigLocation**.

ix.ii Autenticación Digest (resumen) (v)

- Implementación de Autenticación Digest con Spring Security, configuración por XML y web.xml.

- Configuración habitual del Filtro **springSecurityFilterChain** registrando el **<filter-class> org.springframework.web.filter.DelegatingFilterProxy**.

- Definir la URL del mapeo del Filtro **springSecurityFilterChain** con valor "**/***" (para que filtre todos los requests entrantes) mediante la etiqueta **<url-pattern>** (etiqueta interna de la definición de **<filter-mapping>**) (analizar lámina 80).

ix.ii Autenticación Digest (resumen) (w)

- De forma análoga, es posible configurar la Autenticación Digest con Spring Security, mediante configuración por Java Config y sin archivo descriptor web.xml.

- Primeramente es encesario agregar la dependencia correspondientes en el pom.xml de maven.

```
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>3.0.1</version>
    <scope>provided</scope>
</dependency>
```

ix.ii Autenticación Digest (resumen) (x)

- Configurar el plugin maven-war-plugin para para evitar que la construcción y despliegue del aplicativo WAR falle al no detectar el archivo /WEB-INF/web.xml.

```
<plugins>
    <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-war-plugin</artifactId>
        <version>2.6</version>
        <configuration>
            <failOnMissingWebXml>false</failOnMissingWebXml>
        </configuration>
    </plugin>
</plugins>
```

ix.ii Autenticación Digest (resumen) (y)

- Definir una clase que herede de **AbstractAnnotationConfigDispatcherServletInitializer**, en donde se defina el **Root Application Context**, el **Web Application Contex** y, registrar y definir el "**servlet-mapping**" del **DispatcherServlet** de Spring MVC.

ix.ii Autenticación Digest (resumen) (z)

- Definición de **AbstractAnnotationConfigDispatcherServletInitializer**.

```
public class SpringWebMvcDispatcherServletInitializer extends AbstractAnnotationConfigDispatcherServletInitializer {
    @Override
    protected Class<?>[] getRootConfigClasses() {
        return new Class[] { RootContextConfiguration.class, SecurityContextConfiguration.class };
    }
    @Override
    protected Class<?>[] getServletConfigClasses() {
        return new Class[] { ServletContextConfiguration.class };
    }
    @Override
    protected String[] getServletMappings() {
        return new String[] { "/" };
    }
}
```

ix.ii Autenticación Digest (resumen) (a')

- Definir una clase que herede de **AbstractSecurityWebApplicationInitializer**, la cual se encargará de registrar el **springSecurityFilterChain** mediante la clase **org.springframework.web.filter.DelegatingFilterProxy** (por default filtra todas las peticiones entrantes mediante el url-pattern "**/***").

```
public class SpringSecurityFilterChainInitializer
                                    extends AbstractSecurityWebApplicationInitializer {
    // Nothing to override
}
```

ix.ii Autenticación Digest (resumen) (b')

- Una vez registrado el **DispatcherServlet** y el **springSecurityFilterChain**, ya sea por descriptor web.xml o mediante configuración por medio de clases Java. Habilitar configuración por anotaciones de Spring MVC y habilitar el escaneo de controladores mediante configuración por XML o Java Config.

```
<context:component-scan base-package="controller.base.package" />

<mvc:annotation-driven />
```

```
@Configuration
@ComponentScan(basePackages = "controller.base.package")
@EnableWebMvc
```

ix.ii Autenticación Digest (resumen) (c')

- Configurar el **AuthenticationManager** mediante configuración por XML o Java Config (heredando de la clase **WebSecurityConfigurerAdapter** y sobre-escribiendo el método **configure(AuthenticationManagerBuilder auth)**).

- Configuración del **AuthenticationManager** mediante XML:

```
<security:authentication-manager>
    <security:authentication-provider user-service-ref="userDetails">
    </security:authentication-provider>
</security:authentication-manager>

<bean id="userDetails" class="CustomUserDetailsService">
```

ix.ii Autenticación Digest (resumen) (d')

- Configuración del **AuthenticationManager** mediante Java Config:

```
@Configuration
@EnableWebSecurity
public class SecurityContextConfiguration extends WebSecurityConfigurerAdapter {
    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.userDetailsService(customUserDetailsService());
    }
    @Bean
    public UserDetailsService customUserDetailsService() {
        return new CustomUserDetailsService();
    }
}
```

ix.ii Autenticación Digest (resumen) (e')

- Clase CustomDetailsService:

```
public interface UserDetailsService {
   UserDetails loadUserByUsername(String username) throws UsernameNotFoundException;
}

public class CustomUserDetailsService implements UserDetailsService {
    @Override
    public UserDetails loadUserByUsername(String username)
                                                        throws UsernameNotFoundException {
        User user = retrieveFromSomeRepository(username);
        return user;
    }
}
```

ix.ii Autenticación Digest (resumen) (f')

- Configuración de autorización a nivel de URL por XML.

Evita crear Cookie de Sesión (JSESSIONID)

```
<security:http use-expressions="true“ create-session="stateless">
    <security:intercept-url pattern="/welcome“ access="isAuthenticated()" />
    <security:intercept-url pattern="/login“ access="permitAll" />
    <security:intercept-url pattern="/root*/**“
                                        access="hasAnyRole('ROLE_ROOT','ROLE_ADMIN')" />
    <security:intercept-url pattern="/admin*/**“ access="hasAnyRole('ROLE_ADMIN')" />
    <security:intercept-url pattern="/user*/**“ access="hasRole('ROLE_USER')" />
</security:http>
```

ix.ii Autenticación Digest (resumen) (g')

- Configuración de autorización a nivel de URL por Java Config.

```
@Configuration
@EnableWebSecurity
public class SecurityContextConfiguration extends WebSecurityConfigurerAdapter {
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.csrf().disable()
            .authorizeRequests()
                .antMatchers("/welcome").authenticated()
                .antMatchers("/").permitAll()...
            .and()
            .sessionManagement()
                .sessionCreationPolicy(SessionCreationPolicy.STATELESS);
    }
}
```

Evita crear Cookie de Sesión (JSESSIONID)

ix.ii Autenticación Digest (resumen) (h')

- Filtro y EntryPoint para configuración de autorización digest.

- La autorización digest no tiene soporte por default en Spring Security, para ello es requerido definir un bean de la clase **DigestAuthenticationFilter** y registrarlo dentro del **springSecurityFilterChain** depués del filtro **BasicAuthenticationFilter** (el orden es importante).

- Para registrar el filtro **DigestAuthenticationFilter** es necesario definir un bean de la clase **DigestAuthenticationEntryPoint** donde se configura el ***realm***, el ***key*** (con el que se calcula el ***nonce***) y la validez del nonce (en segundos).

ix.ii Autenticación Digest (resumen) (i')

- Definición de beans **DigestAuthenticatioFilter** y **DigestAuthenticationEntryPoint** por XML.

```
<bean id="digestFilter"
    class="org.springframework.security.web.authentication.www.DigestAuthenticationFilter">
    <property name="userDetailsService" ref="userDetails" />
    <property name="authenticationEntryPoint" ref="digestEntryPoint" />
</bean>

<bean id="digestEntryPoint"
    class="org.springframework.security.web.authentication.www.DigestAuthenticationEntryPoint">
    <property name="realmName" value="My application realm" />
    <property name="key" value="my-secret-key" />
    <property name="nonceValiditySeconds" value="10" />
</bean>
```

ix.ii Autenticación Digest (resumen) (j')

- Definición de beans **DigestAuthenticatioFilter** y **DigestAuthenticationEntryPoint** por Java Config (a).

```
@Configuration
@EnableWebSecurity
public class SecurityContextConfiguration extends WebSecurityConfigurerAdapter {
    ...
    @Bean
    public DigestAuthenticationFilter digestAuthFilter() {
        DigestAuthenticationFilter daf = new DigestAuthenticationFilter();
        daf.setUserDetailsService(customUserDetailsService());
        daf.setAuthenticationEntryPoint(digestAuthEntryPoint());
        return daf;
    }
    ...
```

ix.ii Autenticación Digest (resumen) (k')

- Definición de beans **DigestAuthenticatioFilter** y **DigestAuthenticationEntryPoint** por Java Config (b).

```
@Configuration
@EnableWebSecurity
public class SecurityContextConfiguration extends WebSecurityConfigurerAdapter {
    ...
    @Bean
    public DigestAuthenticationEntryPoint digestAuthEntryPoint() {
        DigestAuthenticationEntryPoint daep = new DigestAuthenticationEntryPoint();
        daep.setKey("secret-key");
        daep.setRealmName("My application realm");
        daep.setNonceValiditySeconds(10);
        return daep;
    }
}
```

ix.ii Autenticación Digest (resumen) (I')

- Configuración de **DigestAuthenticatioFilter** y **DigestAuthenticationEntryPoint** dentro del **springSecurityFilterChain** por XML.

```
<security:http use-expressions="true“ create-session="stateless”
        entry-point-ref="digestEntryPoint">

    <security:custom-filter after="BASIC_AUTH_FILTER" ref="digestFilter" />

    <security:intercept-url
        pattern="/welcome“ access="isAuthenticated()" />
    ...
</security:http>
```

Debido a que no autenticamos ni por form, ni por autenticación básica, es necesario definir un EntryPoint

El registro del filtro DigestAuthenticationFilter debe ser después del BasicAuthenticationFilter

ix.ii Autenticación Digest (resumen) (m')

- Configuración de **DigestAuthenticatioFilter** y **DigestAuthenticationEntryPoint** dentro del **springSecurityFilterChain** por Java Config (**WebSecurityConfigurerAdapter**).

```
http
    .exceptionHandling().authenticationEntryPoint(digestAuthEntryPoint())
    .and()
    .addFilterAfter(digestAuthFilter(), BasicAuthenticationFilter.class)
    .csrf().disable()
    .authorizeRequests()
        .antMatchers("/welcome").authenticated()
        .antMatchers("/").permitAll()...
    .and()
        .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);
```

Debido a que no autenticamos ni por form, ni por autenticación básica, es necesario definir un EntryPoint

El registro del filtro DigestAuthenticationFilter debe ser después del BasicAuthenticationFilter

ix.ii Autenticación Digest (resumen) (n')

- Definir y configurar los controladores **@RestController** de forma habitual, ya sea por configuración por XML y/o JavaConfig mediante el escaneo de paquetes.

- Implementar los *endpoints* / servicios REST correspondientes a los definidos en la configuración del **Security Context**.

- *Global method security* no esta disponible cuando se utiliza Autenticación Básica o Digest.

ix.ii Autenticación HTTP con Spring Security

a. Autenticación Básica

Práctica 33. Implementación de Autenticación Básica de Servicios REST

b. Autenticación Digest (resumen)

Práctica 34. Implementación de Autenticación Digest de Servicios REST

c. Autenticación Bearer con JSON Web Token (JWT)

Práctica 35. Implementación de Autenticación Bearer con JWT de Servicios REST

ix.ii Autenticación Digest (resumen). Práctica 34. (a)

- Práctica 34. Implementación de Autenticación Digest de Servicios REST

- Configurar Spring Security para securizar servicios REST en una aplicación Spring en contexto web *(WebApplicationContext)*.
- Implementar configuración de Spring Security con XML y descriptor web.xml.
- Implementar configuración de Spring Security con @Anotaciones y sin descriptor web.xml.
- Implementar autenticación Digest mediante Spring Security.

ix.ii Autenticación HTTP con Spring Security

a. Autenticación Básica

Práctica 33. Implementación de Autenticación Básica de Servicios REST

b. Autenticación Digest (resumen)

Práctica 34. Implementación de Autenticación Digest de Servicios REST

c. Autenticación Bearer con JSON Web Token (JWT)

Práctica 35. Implementación de Autenticación Bearer con JWT de Servicios REST

ix.ii Autenticación Bearer con JSON Web Token (JWT) (a)

- HTTP Bearer Authentication.

- La especificación **RFC 2617**, *HTTP Authentication: Basic and Digest Access Authentication*, define dos esquemas de autenticación **Basic** y **Digest**.

- El esquema de autenticación **Bearer** o "portador" se aplica para autenticación por token, donde el cliente no se autentica para cada petición de un recurso, sino más bien, presenta un token que le valida su autorización.

ix.ii Autenticación Bearer con JSON Web Token (JWT) (b)

- HTTP Bearer Authentication.

- El esquema de autenticación Bearer se presenta en el **RFC 6750**, **The OAuth 2.0 Authorization Framework: Bearer Token Usage**, donde se especifica el uso del esquema de autorización Bearer para el transporte del token de authorización.

ix.ii Autenticación Bearer con JSON Web Token (JWT) (c)

- HTTP Bearer Authentication.

- Cuando autenticamos a un usuario mediante el modelo de autenticación Bearer, se separa la autenticación de la autorización en dos pasos:

  1. Autenticación del cliente, el cual puede darse de diferentes formas ya sea mediante formulario (form-based authentication) e inclusive a través de autenticación Básica, sin embargo, la autenticación debe devolver un token de autorización para que el cliente lo envie en el encabezado **Authentication** (con esquema Bearer) hacia el servidor al solicitar el recurso protegido.

ix.ii Autenticación Bearer con JSON Web Token (JWT) (d)

- HTTP Bearer Authentication.

- Cuando autenticamos a un usuario mediante el modelo de autenticación Bearer, se separa la autenticación de la autorización en dos pasos:

  2. Autorización del cliente, donde el servidor verifica la validez del token y extrae la información del mismo, concluyendo así el nivel de autorización que tiene el protador del token para acceder al recurso. El cliente o portador del token puede utilizar el mismo token para acceder a los recursos que tenga permitido durante un periodo de vigencia que, por seguridad, debe ser corto.

ix.ii Autenticación Bearer con JSON Web Token (JWT) (e)

- JWT

- La especificación **RFC 7519 JSON Web Token (JWT)** define al **JWT** como un formato estándar, compacto y seguro de trasmitir identidad y propiedades o "***claims***" a través de un token firmado criptográficamente entre sistemas.

- Los ***claims*** son propiedades, afirmaciones o en general, información referente al usuario o sistema autenticado al cual le pertenece la identidad y propiedades del token.

ix.ii Autenticación Bearer con JSON Web Token (JWT) (f)

- JWT

- Entre los ***claims*** o propiedades que, por estándar, pueden viajar en el token JWT se encuentran (a):
  - ***iss*** (issuer): Emisor del token JWT.
  - ***sub*** (subject): El identificador o nombre del sujeto, usuario o sistema autenticado al cual pertenecen los ***claims*** o propiedades contenidas en el JWT.
  - ***aud*** (audience): El identificador de la audiencia o receptores que autorizarán al protador del JWT para el acceso a un recurso protegido.

ix.ii Autenticación Bearer con JSON Web Token (JWT) (g)

- JWT

- Entre los ***claims*** o propiedades que, por estándar, pueden viajar en el token JWT se encuentran (b):
  - ***exp*** (expiration time): Denota el tiempo en el cual el token expirará, fecha en la cual el token debe ser rechazado y no aceptado para procesar solicitud alguna.
  - ***nbf*** (not before): Denota el tiempo en el cual el token no debe ser aceptado antes de la fecha descrita.
  - ***iat*** (issued at): Denota el tiempo en el cual el token fue emitido.
  - ***jti*** (jwt id): Identificador único para el token JWT presentado.

ix.ii Autenticación Bearer con JSON Web Token (JWT) (h)

- JWT

- El formato del JWT se basa en 3 partes (a):
  1. Encabezado o *header*, Utilizando el estándar **JOSE Header** (**JSON Object Signing and Encryption**, **RFC 7515**) define el tipo de token y el algoritmo utilizado para establecer la firma criptográfica del JWT. Utiliza formato **JSON** para definir sus valores.

  Ejemplo: {"**typ**": "JWT", "**alg**": "HS256"}

  HS256 indica que token está firmado utilizando **HMAC-SHA256**.

ix.ii Autenticación Bearer con JSON Web Token (JWT) (i)

- JWT

- El formato del JWT se basa en 3 partes:

  2. Contenido o *payload*, El *payload* del token JWT contiene los *claims* o propiedades del portador del token. Utiliza formato **JSON** para definir sus valores.

  Ejemplo:
  { "**sub**": "1234567890", "**name**": "John Doe", "**iat**": 1516239022 }

  El ***claim*** *name* no es estándar de JWT, sin embargo, el *payload* puede contener cualquier información (**Public/Private claim names**).

ix.ii Autenticación Bearer con JSON Web Token (JWT) (j)

- JWT

- El formato del JWT se basa en 3 partes:

  3. Firma o *signature*, La firma digital se calcula codificando el *header* y el *payload* en **base64url**, concatenándose ambas partes con un punto "**.**" (*period*) como separador. Se utiliza la clave privada o *secret key* del servidor y el algoritmo definido en el atributo "**alg**" para aplicar la función *hash* especificada, la cual resulta en la firma digital aplicada al token JWT.

  secret-key = 'my-secret-key'
  unsignedToken = encodeBase64Url(*header*) + '.' + encodeBase64Url(*payload*)
  signature = HMAC-SHA256(**secret-key**, **unsignedToken**)

ix.ii Autenticación Bearer con JSON Web Token (JWT) (k)

- JWT

- Los tokens JWT se encuentran firmados criptográficamente a través de una clave secreta o "***secret key***" que el servidor o proveedor de identidad utiliza para firmar el token para después, verificar la autenticidad del mismo verificando la firma del token.

- La firma de JWT soporta algoritmos asimétricos como RSA Signature with SHA ó algoritmos simétricos HMAC with SHA (soporta 256, 384 y 512 bits).

ix.ii Autenticación Bearer con JSON Web Token (JWT) (I)

- JWT

ix.ii Autenticación Bearer con JSON Web Token (JWT) (m)

- JWT

1 Header

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

2 Payload

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

3 Signature

```
HMACSHA256(
BASE64URL(header)
.
BASE64URL(payload),
secret)
```

ix.ii Autenticación Bearer con JSON Web Token (JWT) (n)

- Autenticación Bearer JWT.

- Una vez que el cliente se ha autenticado, mediante cualquier mecanismo de auntenticación, el servidor debe devolver el token **JWT** generado a través de algún mecanismo conocido tal como:

- Mediante el encabezado **Authorization** en la respuesta de la autenticacíon.
  Authorization: Bearer <token-jwt>

- A través del *response body* en la respuesta de la autenticacíon, utilizando algún formato conocido tal como **JSON**.

ix. Seguridad en Servicios REST - ix.ii Autenticación HTTP con Spring Security

ix.ii Autenticación Bearer con JSON Web Token (JWT) (ñ)

- Recepción de token JWT.

- Mediante el encabezado **Authorization** en la respuesta de la autenticacíon.

(HTTP Request)
POST /api/authenticate HTTP/1.1
Host: localhost:8080
Accept: */*
Content-Length: 35
Content-Type: application/x-www-form-urlencoded

username=xvanhalenx&password=123123

(HTTP Response)
HTTP/1.1 200
X-Content-Type-Options: nosniff
…
Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzUxMiJ9.eyJpc3MiOi
I2LXNwcmluZy1tdmMt…hAt5rHMtNBvNziIJvEaA5gl=
Content-Type: application/json;charset=ISO-8859-1
Date: Fri, 27 Dec 2020 05:34:29 GMT

ix.ii Autenticación Bearer con JSON Web Token (JWT) (o)

- Recepción de token JWT.

- A través del *response body* en la respuesta de la autenticacíon (**JSON**).

(HTTP Request)
POST /api/authenticate HTTP/1.1
Host: localhost:8080
Accept: */*
Content-Length: 35
Content-Type: application/x-www-form-urlencoded

username=xvanhalenx&password=123123

(HTTP Response)
HTTP/1.1 200
…
Content-Type: application/json;charset=ISO-8859-1
Transfer-Encoding: chunked

{"jwt":"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzUxMiJ9.eyJpc3MiOiI2LXNwcmluZy1tdmMtc2VjdXJpdHktand0LWphdmFjb25ma…S1hcHAiLCJzdWIiOiJ4dmFuaGFsZW54PVCJdfQ.LQJskAJv9mzCDy_EhtYaV_uGhnTHiFx2lhAt5raVu6gHMtNspBvNziIJvEaA5glHfImutxbt55GtFP="}

ix. Seguridad en Servicios REST - ix.ii Autenticación HTTP con Spring Security

ix.ii Autenticación Bearer con JSON Web Token (JWT) (p)

- Solicitud de recurso protegido de token JWT.

- Una vez que el cliente se ha autenticado y ha recibido un token JWT, debe enviarlo en encabezado **Authorization** (con esquema **Bearer**) al momento de solicitar un recurso protegido.

(HTTP Request)
GET /secure-resource HTTP/1.1
…
Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzUxMiJ9.eyJpc3
MiOiI2LXNwcmlu…mMtc2VjdiOlsiUk9MRV9BRE
1JTiIsIlJPTEVfUk9PVCJdfQ.9CzBm9Q5wlma4yM
cAkuVAVirLu1DzU3lMk1KacpULrWq6LGZyFj_6=

(HTTP Response)
HTTP/1.1 200
X-Content-Type-Options: nosniff
…
Content-Type: text/plain;charset=ISO-8859-1
Transfer-Encoding: chunked
Date: Fri, 27 Dec 2020 05:34:29 GMT

Hello authenticated user !

ix.ii Autenticación Bearer con JSON Web Token (JWT) (q)

- Implementación de Autenticación Bearer JWT con Spring Security.

- Al igual que la autenticación Básica y Digest, para implementar autenticación Bearer JWT con Spring Security en una aplicación Spring MVC, es necesario agregar las siguientes dependencias:
  - spring-security-core-<version>.jar
  - spring-security-config-<version>.jar
  - spring-security-web-<version>.jar
  - spring-security-taglibs-<version>.jar
  - spring-security-aspects-<version>.jar
  - spring-security-acl-<version>.jar
  - spring-security-ldap-<version>.jar
  - spring-security-openid-<version>.jar
  - spring-security-cas-<version>.jar
  - spring-security-crypto-<version>.jar
  - spring-security-remoting-<version>.jar

ix.ii Autenticación Bearer con JSON Web Token (JWT) (r)

- Implementación de Autenticación Bearer JWT con Spring Security, configuración por JavaConfig y sin descriptor de despliegue web.xml.

- Primeramente es encesario agregar la dependencia maven **javax.servlet:javax.servlet-api:3.0.1:provided** al archivo pom.xml.

ix.ii Autenticación Bearer con JSON Web Token (JWT) (s)

- Configurar el plugin maven-war-plugin para para evitar que la construcción y despliegue del aplicativo WAR falle al no detectar el archivo /WEB-INF/web.xml.

```
<plugins>
    <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-war-plugin</artifactId>
        <version>2.6</version>
        <configuration>
            <failOnMissingWebXml>false</failOnMissingWebXml>
        </configuration>
    </plugin>
</plugins>
```

ix.ii Autenticación Bearer con JSON Web Token (JWT) (t)

- Definir una clase que herede de **AbstractAnnotationConfigDispatcherServletInitializer**, en donde se defina el **Root Application Context**, el **Web Application Contex** y, registrar y definir el "**servlet-mapping**" del **DispatcherServlet** de Spring MVC.

- El **Root Application Context** incluye la configuración del **Security Context**.

ix.ii Autenticación Bearer con JSON Web Token (JWT) (u)

- Definición de **AbstractAnnotationConfigDispatcherServletInitializer**.

```
public class SpringWebMvcDispatcherServletInitializer extends AbstractAnnotationConfigDispatcherServletInitializer {
    @Override
    protected Class<?>[] getRootConfigClasses() {
        return new Class[] { RootContextConfiguration.class, SecurityContextConfiguration.class };
    }
    @Override
    protected Class<?>[] getServletConfigClasses() {
        return new Class[] { ServletContextConfiguration.class };
    }
    @Override
    protected String[] getServletMappings() {
        return new String[] { "/" };
    }
}
```

ix.ii Autenticación Bearer con JSON Web Token (JWT) (v)

- Definir una clase que herede de **AbstractSecurityWebApplicationInitializer**, la cual se encargará de registrar el **springSecurityFilterChain** mediante la clase **org.springframework.web.filter.DelegatingFilterProxy** (por default filtra todas las peticiones entrantes mediante el url-pattern "**/***").

```
public class SpringSecurityFilterChainInitializer
                                    extends AbstractSecurityWebApplicationInitializer {
    // Nothing to override
}
```

ix.ii Autenticación Bearer con JSON Web Token (JWT) (w)

- Una vez registrado el **DispatcherServlet** y el **springSecurityFilterChain** a través de heredar de la clase **AbstractSecurityWebApplicationInitializer**, habilitar configuración por anotaciones de Spring MVC y escaneo de controladores Spring MVC.

```
@Configuration
@ComponentScan(basePackages = "controller.base.package")
@EnableWebMvc
public class ServletContextConfiguration {
}
```

ix.ii Autenticación Bearer con JSON Web Token (JWT) (x)

- Configurar el **AuthenticationManager** a través de heredar de la clase **WebSecurityConfigurerAdapter** y sobre-escribir el método **configure(AuthenticationManagerBuilder auth)**.

```
@Configuration
@EnableWebSecurity
public class SecurityContextConfiguration extends WebSecurityConfigurerAdapter {
    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.userDetailsService(customUserDetailsService());
    }
    @Bean
    public UserDetailsService customUserDetailsService() {
        return new CustomUserDetailsService();
    }
}
```

ix.ii Autenticación Bearer con JSON Web Token (JWT) (y)

- Clase CustomDetailsService:

```
public interface UserDetailsService {
   UserDetails loadUserByUsername(String username) throws UsernameNotFoundException;
}

public class CustomUserDetailsService implements UserDetailsService {
    @Override
    public UserDetails loadUserByUsername(String username)
                                                    throws UsernameNotFoundException {
        User user = retrieveFromSomeRepository(username);
        return user;
    }
}
```

ix.ii Autenticación Bearer con JSON Web Token (JWT) (z)

- Configuración de autorización a nivel de URL por Java Config.

```
@Configuration
@EnableWebSecurity
public class SecurityContextConfiguration extends WebSecurityConfigurerAdapter {
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.csrf().disable()
            .authorizeRequests()
                .antMatchers("/welcome").authenticated()
                .antMatchers("/").permitAll()...
            .and()
            .sessionManagement()
                .sessionCreationPolicy(SessionCreationPolicy.STATELESS);
    }
}
```

ix. Seguridad en Servicios REST - ix.ii Autenticación HTTP con Spring Security

ix.ii Autenticación Bearer con JSON Web Token (JWT) (a')

- Al igual que la autenticación Digest, la autorización por cualquier otro mecanismo, como Bearer a traves de tokens (JWT) no tiene soporte por default en Spring Security sin embargo, el framework, es fácil de extender y proveer la funcionalidad correspondiente.

- Para la autenticación Básica o Digest, el usuario se autentica en cada petición a un recurso protegido, dado que en cada petición el cliente HTTP envía usuario y contraseña.

ix.ii Autenticación Bearer con JSON Web Token (JWT) (b')

- Al utilizar autenticación por token, existen dos momentos (a):

- Autenticación:
  - En un primer momento el usuario se autentica y se le expide un token JWT con los *claims* o propiedades del usuario autenticado el cual contiene los roles o permisos de éste.

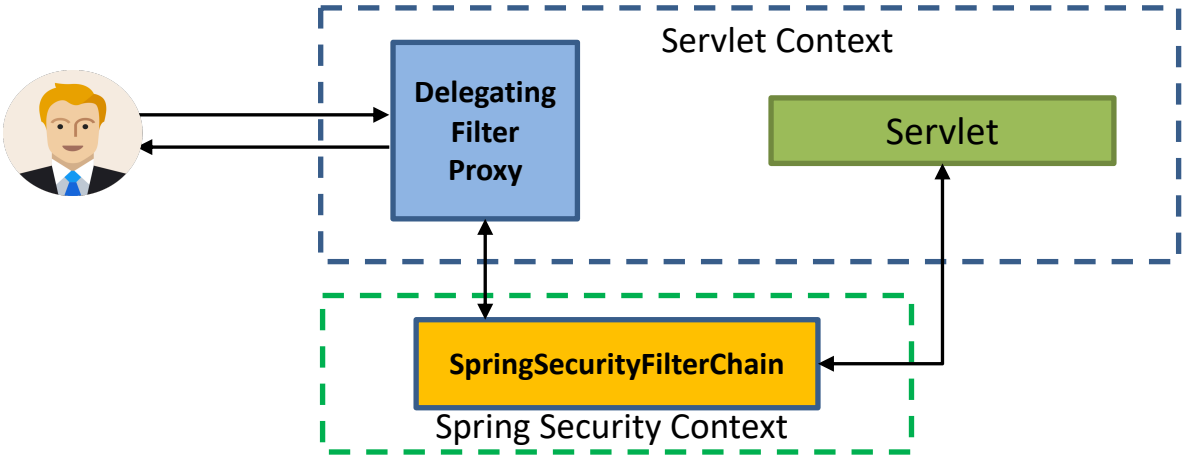
ix.ii Autenticación Bearer con JSON Web Token (JWT) (c')

- Al utilizar autenticación por token, existen dos momentos (b):

- Autorización:
  - En un segundo momento el cliente HTTP envía el token JWT emitido en el encabezado **Authroization** y se verifica la validez del token. En caso de ser un token válido, se extraen los roles o permisos incluidos en los claims del token y, se valida el nivel de autorización que el usuario tiene hacia el recurso que está solicitando. En caso de ser un token inválido, se responde un error HTTP **401 Unauthorized**. En caso de que el usuario no tenga los permisos suficientes para acceder al recurso, se le responde un error HTTP **403 Forbbiden**.

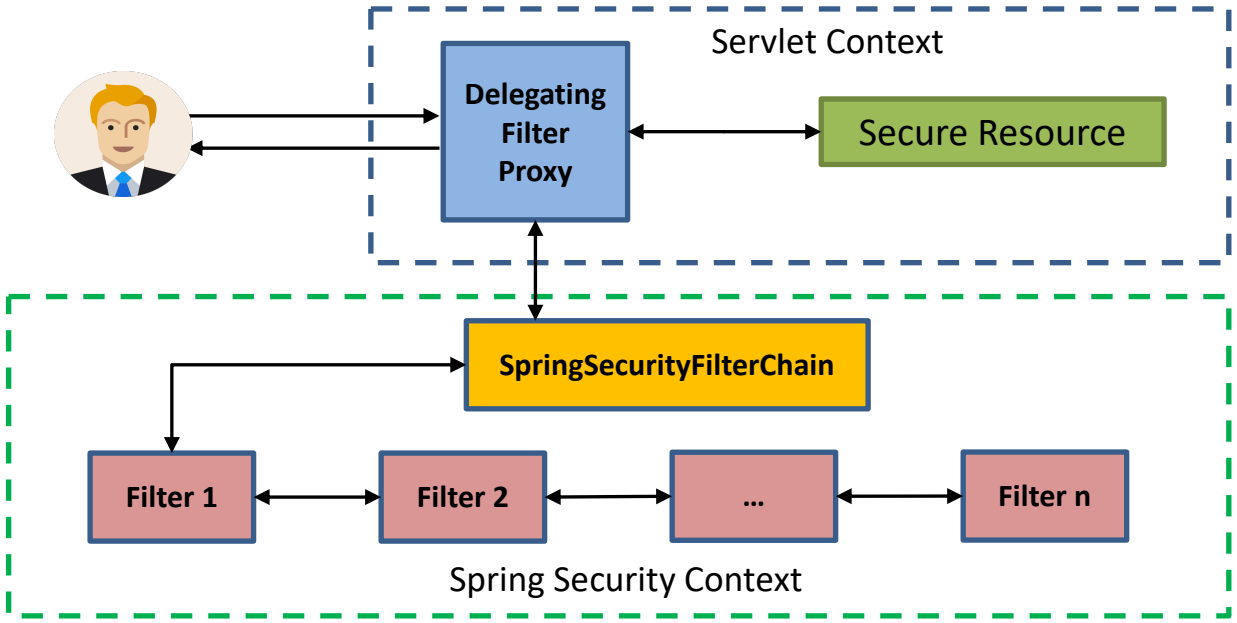
ix.ii Autenticación Bearer con JSON Web Token (JWT) (d')

- Una vez comprendido que, al utilizar autenticación por token se requiere autenticación y autorización por separado, es necesario implementar dos flitros en el **springSecurityFilterChain** que correspondan con implementar estos dos momentos.



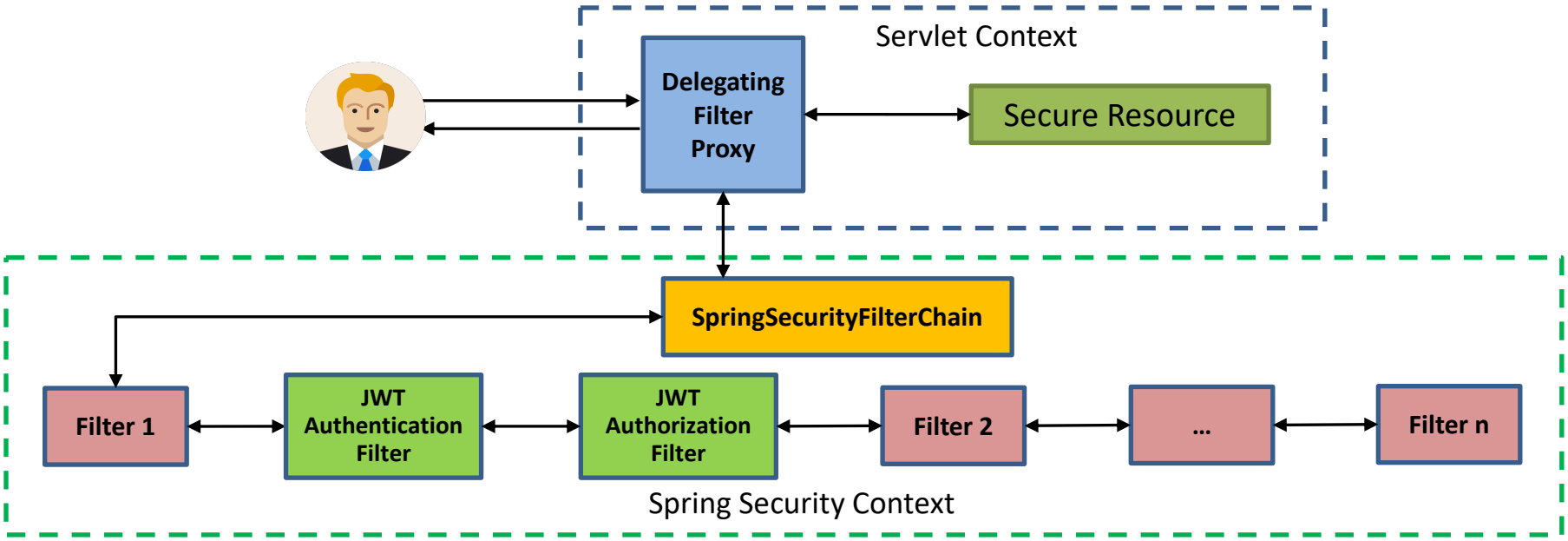
ix.ii Autenticación Bearer con JSON Web Token (JWT) (e')

- Definición de **springSecurityFilterChain**.



ix.ii Autenticación Bearer con JSON Web Token (JWT) (f')

- Filtros de autenticación y autorización por token JWT.



ix.ii Autenticación Bearer con JSON Web Token (JWT) (g')

- Filtro de autenticación por username y password para emisión de JWT.
- El filtro **JwtAuhtenticationFilter** será el filto encargado de autenticar al usuario y de emitir un token JWT en el *response* de la autenticación.
- Es posible implementar múltiples formas de autenticar al usuario, por ejemplo mediante ***form-login*** o similar a autenticación Básica.
- A su vez, es posible implementar múltiples formas de devolver el token **JWT** generado, ya sea a través de encabezado **Authorization** (con esquema *Bearer*) en la respuesta o mediante el *body* del *response*.

ix.ii Autenticación Bearer con JSON Web Token (JWT) (h')

- Filtro de autenticación por username y password para emisión de JWT.

- El filtro **JwtAuhtenticationFilter** deberá ser una clase que herede de la clase **AbstractAuthenticationProcessingFilter** de Spring Security, la cual es una clase abstracta que define implementar el método **Authentication attemptAuthentication(HttpServletRequest request, HttpServletResponse response)**.

- A su vez, la clase **AbstractAuthenticationProcessingFilter**, permite sobre escribir los métodos **successfulAuthentication** y **unsuccessfulAuthentication** para definir el comportamiento adecuado.

ix.ii Autenticación Bearer con JSON Web Token (JWT) (i')

- Filtro de autenticación por username y password para emisión de JWT.

- La clase **AbstractAuthenticationProcessingFilter** requiere que se invoque explicitamente al **AuthenticationManager** para procesar la autenticación del usuario.

- Si el proceso de autenticación es válido, se ejecutará el método **successfulAuthentication** el cual debemos sobre-escribir para generar el token JWT y devolverlo en el *response* al cliente HTTP.

ix.ii Autenticación Bearer con JSON Web Token (JWT) (j')

- Pseudo-código de clase AbstractAuthenticationProcessingFilter (a).

```
public class JwtAuthenticationFilter extends AbstractAuthenticationProcessingFilter {
    @Override
    public Authentication attemptAuthentication(HttpServletRequest request,
                                                HttpServletResponse response) {
        // Obtiene y valida username y password
        // genera objeto Authentication mediante UsernamePasswordAuthenticationToken
        // autentica al usuario a traves del método authenticate del objeto AuthenticationManager.
    }

    ...
```

ix.ii Autenticación Bearer con JSON Web Token (JWT) (k')

- Pseudo-código de clase AbstractAuthenticationProcessingFilter (b).

```
public class JwtAuthenticationFilter extends AbstractAuthenticationProcessingFilter {
    …
    @Override
    protected void successfulAuthentication(HttpServletRequest request, HttpServletResponse
                                            response, FilterChain filterChain, Authentication authentication) {
        // obtiene el username del usuario autenticado del bjeto Authentication.
        // obtiene una lista de String de los roles que tiene el objeto Principal (usuario autenticado).
        // genera un token JWT con subject = username y claims que contenga los roles del usuario
        // autenticado.
        // devuelve token JWT en formato de encabezado de respuesta Authorization Bearer <token jwt>
    }
}
```

ix.ii Autenticación Bearer con JSON Web Token (JWT) (I')

- Registro de filtro JwtAuthenticationFilter.

- Se utiliza el método **configure(HttpSecurity http)** de la clase que hereda de **WebSecurityConfigurerAdapter** para registrar filtros en el **springSecurityFilterChain**.

- El filtro **JwtAuthenticationFilter** debe ser registrado antes del filtro **UsernamePasswordAuthenticationFilter**.

- Este filtro no debe aplicar a todos los *requests*, solo aquellos que *matchean* el path "**/login**" o "**/api/authenticate**" o aquel que sea definido.

ix.ii Autenticación Bearer con JSON Web Token (JWT) (m')

- Registro de filtro JwtAuthenticationFilter.

```
protected void configure(HttpSecurity http) throws Exception {
    http.csrf().disable()
        .authorizeRequests()
            .antMatchers("/welcome").authenticated()
            ...
        .and()
        .addFilterBefore(new JwtAuthenticationFilter("/api/authenticate", authManager),
                UsernamePasswordAuthenticationFilter.class)
        ...
        .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);
}
```

Path que el filtro, filtrará requests. Será el "endpoint" que dará origen a la autenticación.

Requiere inyectar AuthenticationManager para ejecutar la autenticación.

ix.ii Autenticación Bearer con JSON Web Token (JWT) (n')

- Filtro de autorización que construye un objeto Authentication a partir de validar y parsear el token JWT.

- El filtro **JwtAuthroizationFilter** se encargará de:
  - Validar el encabezado **Auhtorization: Bearer <token jwt>**
  - Parsear el token **JWT** y verificar la firma de mismo.
  - Generar un objeto **Authentication** (simulando su autenticación debido a que no hay sesiones) y asignarlo al **SecurityContetHolder**.

- La autorización del usuario basado en sus roles, será manejado por Spring Security tal como funciona por default.

ix.ii Autenticación Bearer con JSON Web Token (JWT) (ñ')

- Filtro de autorización que construye un objeto Authentication a partir de validar y parsear el token JWT.

- El filtro **JwtAuthroizationFilter** deberá ser una clase que herede de la clase **GenericFilterBean** de Spring Core, la cual es una clase base simple para la implementación de filtros que implementan **javax.servlet.Filter**.

ix.ii Autenticación Bearer con JSON Web Token (JWT) (o')

- Pseudo-código de clase JwtAuthroizationFilter (a).

```
public class JwtAuthroizationFilter extends GenericFilterBean {
    @Override
    public void doFilter(ServletRequest req, ServletResponse res, FilterChain chain)
                                                                    throws IOException, ServletException {
        // obtiene el token jwt del encabezado Authroization Bearer.
        // parsea el token jwt, lo valida y extrae el subject (username) y roles que contiene en sus claims.
        // genera un objeto Authentication a través de la clase UsernamePasswordAuthenticationToken
        // tal como si fuera generado a través del método authenticate del AuthenticationManager.
        // asigna el objeto Authentication al SpringSecurityContextHolder.
        // procede con el filerChain.
    }
}
```

ix.ii Autenticación Bearer con JSON Web Token (JWT) (p')

- Registro de filtro JwtAuthroizationFilter.

- Se utiliza el método **configure(HttpSecurity http)** de la clase que hereda de **WebSecurityConfigurerAdapter** para registrar filtros en el **springSecurityFilterChain**.

- El filtro **JwtAuthroizationFilter** debe ser registrado antes del filtro **UsernamePasswordAuthenticationFilter**.

- Este filtro debe aplicar sobre todos los *requests* que accedan a algún recurso protegido.

ix.ii Autenticación Bearer con JSON Web Token (JWT) (q')

- Registro de filtro JwtAuthroizationFilter.

```
protected void configure(HttpSecurity http) throws Exception {
    http.csrf().disable()
        .authorizeRequests()
            .antMatchers("/welcome").authenticated()
            ...
        .and()
        .addFilterBefore(…)
        .addFilterBefore(new JwtAuthroizationFilter(), UsernamePasswordAuthenticationFilter.class)
        ...
        .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);
}
```

ix.ii Autenticación Bearer con JSON Web Token (JWT) (r')

- JwtAuthenticationEntryPoint para configuración de autorización Bearer.

- Los EntryPoints se utilizan en Spring Security para especificar información en el response de la petición HTTP cuando un usuario no autenticado trata de accer a un recurso protegido.

- Cuando se trata de acceder a un recurso protegido sin la autenticación correspondiente, Spring Security lanza una excepción de tipo **AuthenticationException** o **AccessDeniedException**, lo cual dispara la ejecución del método **commence**a del **AuthenticationEntryPoint** configurado.

ix.ii Autenticación Bearer con JSON Web Token (JWT) (s')

- JwtAuthenticationEntryPoint para configuración de autorización Bearer.

- Para informar a un cliente no autenticado que debe enviar el encabezado **Authorization: Bearer <token jwt>** correspondiente al tratar de acceder a un recurso protegido, es necesario implementar la interface **AuthenticationEntryPoint** e implementar el método **commence** con la lógica correspondiente.

- La clase **JwtAuthenticationEntryPoint**, será una implementación similar a las clases **BasicAuthenticationEntryPoint** o **DigestAuthenticationEntryPoint**.

ix.ii Autenticación Bearer con JSON Web Token (JWT) (t')

- JwtAuthenticationEntryPoint para configuración de autorización Bearer.

- La clase **JwtAuthenticationEntryPoint** implementación de la interface **AuthenticationEntryPoint**, será la encargada de enviar en el encabezado **WWW-Authenticate** con el esquema Bearer y el *realm* correspondiente, informando el código de estatus HTTP **401 Unauthorized**.

- La configuración del **JwtAuthenticationEntryPoint** debe realizarse sobre la clase de configuración que hereda de **WebSecurityConfigurerAdapter**.

ix.ii Autenticación Bearer con JSON Web Token (JWT) (u')

- Configuración de JwtAuthenticationEntryPoint para configuración de autorización Bearer.

```
protected void configure(HttpSecurity http) throws Exception {
    http.csrf().disable()
        .exceptionHandling().authenticationEntryPoint(new JwtAuthenticationEntryPoint())
        .authorizeRequests()
            .antMatchers("/welcome").authenticated()
            ...
        .and()
        .addFilterBefore(…)
        ...
        .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);
}
```

ix. Seguridad en Servicios REST - ix.ii Autenticación HTTP con Spring Security

ix.ii Autenticación Bearer con JSON Web Token (JWT) (v')

- Definir y configurar los controladores **@RestController** de forma habitual, ya sea por configuración por XML y/o JavaConfig mediante el escaneo de paquetes.

- Implementar los *endpoints* / servicios REST correspondientes a los definidos en la configuración del **Security Context**.

ix.ii Autenticación HTTP con Spring Security

a. Autenticación Básica

Práctica 33. Implementación de Autenticación Básica de Servicios REST

b. Autenticación Digest (resumen)

Práctica 34. Implementación de Autenticación Digest de Servicios REST

c. Autenticación Bearer con JSON Web Token (JWT)

Práctica 35. Implementación de Autenticación Bearer con JWT de Servicios REST

ix.ii Autenticación Bearer con JSON Web Token (JWT). Práctica 35. (a)

- Práctica 35. Implementación de Autenticación Bearer con JWT de Servicios REST

- Configurar Spring Security para securizar servicios REST en una aplicación Spring en contexto web *(WebApplicationContext)*.
- Implementar configuración de Spring Security con @Anotaciones y sin descriptor web.xml.
- Aplicar autenticación y autorización de servicios REST a través de token mediante JWT.

Resumen de la lección

ix.ii Autenticación HTTP con Spring Security

- A través de Spring Security, implementamos autenticación Básica de servicios REST mediante configuración por XML y JavaConfig.
- A través de Spring Security, implementamos autenticación Digest de servicios REST mediante configuración por XML y JavaConfig.
- A través de Spring Security, implementamos autenticación por token JWT de servicios REST mediante configuración por JavaConfig.

Esta página fue intencionalmente dejada en blanco.