Seguridad en Aplicaciones: Segunda práctica

Máster Interuniversitario en Ciberseguridad

Curso académico 2023/2024

1. Introducción

La segunda práctica de la asignatura consiste en completar el desarrollo de una aplicación web, para añadir las funcionalidades que faltan de autenticación, autorización y control de acceso.

Para ello, se proporciona, por un lado, la aplicación parcialmente terminada y por otro lado, se proporciona un servidor de OAuth 2.1 para dar soporte a este tipo de autenticación y autorización.

El objetivo principal de la práctica es que los distintos tipos de usuarios sólo puedan acceder a las funcionalidades a las que tienen permiso, según el rol que se les ha asignado.

Al igual que en la primera práctica, la aplicación que es necesario modificar está implementada en Java, utiliza *Spring Boot* y está empaquetada dentro de un proyecto *maven*. El proyecto está preparado para usar las funcionalidades de control de acceso de la librería **Spring Security**.

Esta aplicación, contiene un servicio web REST, junto con una interfaz de usuario SPA que accede al servicio web. Ambos subsistemas están prácticamente terminados y sólo falta por añadir algunas de las funcionalidades de autenticación, autorización y control de acceso.

Este servicio web no debe tener estado (debe configurarse como un servicio *stateless* y por lo tanto no debe crear la cookie de sesión JSESSIONID) y para la identificación de los usuarios utilizará *tokens* **JWT**.

La interfaz de usuario es una aplicación SPA, implementada en JavaScript con la librería *React*.

Inicialmente, esta aplicación SPA, debe obtener los *tokens* JWT usando un endpoint del servicio web, que valida el usuario y contraseña contra una base de datos local. Adicionalmente, se añadirá el soporte necesario para obtener los *tokens* de un servicio **OAuth 2.1**. Para ello, se proporciona un servidor de OAuth 2.1 completamente configurado y operativo.

Nota: cabe destacar que, en un entorno empresarial, no suele ser habitual que una aplicación ofrezca autenticación/autorización propia (usando la base de datos de la aplicación) junto con autenticación/autorización OAuth. Una vez que la autenticación/autorización con OAuth está implantada, la autenticación/autorización propia suele desaparecer. No obstante, por simplicidad, en esta práctica se mantendrán ambos métodos.

La entrega consistirá, por un lado, en una nueva versión de la aplicación que debe implementar todas las funcionalidades de autenticación propia, autorización y control de acceso, además de la autenticación/autorización a través de OAuth.

Por otro lado, también es necesario entregar un documento en el que se detallen todas las modificaciones realizadas.

2. Planteamiento

2.1 Visión global

Después del fracaso desarrollando la tienda de comercio electrónico para la filial de *Amazoncillo*, la empresa tecnológica *BonitoyBarato* ha decidido imponer un mayor control sobre el trabajo que realizan los miembros del equipo de desarrollo.

Para ello, ha decidido implementar un sistema de gestión de proyectos, con el objetivo de realizar un seguimiento más detallado del tiempo que dedican cada uno de los desarrolladores a las distintas tareas que tienen asignadas.

Una vez que la aplicación se ponga en funcionamiento, el responsable de cada proyecto será el encargado de darlo de alta y de crear las diferentes tareas que lo forman. Cada tarea tendrá un responsable (un desarrollador) y ese responsable será el encargado de cambiar el estado de la resolución de la tarea, a medida que va trabajando en ella.

Para implementar las distintas funcionalidades de autenticación, autorización y control de acceso, los responsables de *BonitoyBarato* no tienen demasiada confianza en su propio equipo de desarrollo, por lo que, a la vista del buen resultado que han obtenido los alumnos en el Máster Interuniversitario en Ciberseguridad, encontrando las vulnerabilidades de *Amazoncillo*, deciden encargarles a ellos este trabajo.

2.2 Detalles

El servicio web proporcionado en la segunda práctica, ofrece una API REST que permite realizar distintas operaciones sobre los proyectos y las tareas.

En primer lugar, se han definido dos tipos de usuarios: los responsables de cada proyecto (administradores) y los desarrolladores (usuarios normales).

El responsable de cada proyecto tendrá el rol de administrador (**ADMIN**) y podrá crear proyectos, y modificar o borrar los proyectos que él mismo ha creado (ningún otro administrador podrá realizar esta operación).

Cada proyecto está formado por una lista de tareas que sólo podrá dar de alta, modificar y eliminar el administrador del proyecto (ningún otro administrador podrá realizar estas operaciones sobre proyectos que no sean suyos).

Cada tarea se le asignará a un desarrollador (quien tendrá el rol **USER**) y éste será el encargado de realizarla y de modificar el estado de la resolución y el progreso, a media que avanza con el trabajo.

El estado de cada tarea puede ser abierto (OPEN) o cerrado (CLOSED) y sólo el administrador del proyecto puede modificar este estado. Cuando una tarea está cerrada, el desarrollador que la tiene asignada no podrá realizar ninguna modificación sobre ella.

El estado de la resolución de la tarea podrá tener uno los siguientes valores: sin empezar (NEW), trabajando en ella (IN_PROGRESS), finalizada (COMPLETED). Este valor sólo lo puede modificar el desarrollador encargado de realizar la tarea. Además, mientras la tarea está en progreso, el desarrollador que la tiene asignada también podrá modificar el progreso (porcentaje ya realizado de la tarea).

Para facilitar el trabajo en equipo, cualquier usuario autenticado (tanto responsables de proyecto, como desarrolladores) podrá añadir comentarios en cualquier tarea.

Un usuario que no se encuentre autenticado, sólo podrá ver los proyectos y las tareas, pero no podrá realizar ninguna modificación ni añadir comentarios.

El listado de los usuarios sólo lo podrán consultar los administradores.

La URL de la aplicación SPA es: **http://127.0.0.1:8888/tasks-service/dashboard/**

El API REST está documentado con Swagger: **http://127.0.0.1:8888/tasks-service/swagger-ui.html**

Los usuarios desarrolladores dados de alta, tanto en el servicio web como en el servidor de OAuth2 son: **user1**, **user2** y **user3** y los administradores: **admin1** y **admin2**. La contraseña de todos los usuarios es **secret**.

Las funcionalidades que faltan por añadir son las siguientes:

a) Terminar de implementar el *endpoint* de autenticación/autorización propia contra la base de datos local. Este *endpoint* se encuentra en la clase **com.tasks.rest.UserController**, en el método **doLogin**. Este método debe devolver el *token* de acceso en formato JSON y un código de respuesta HTTP 200.
    1. En el código proporcionado hay dos tokens cableados (uno para **user1** y otro para **admin1**).
    2. En su lugar, hay que generar el token JWT dinámicamente mediante el objeto **tokenProvider**.

b) Realizar una primera configuración del control de acceso que no contemple roles. Para ello, debe modificarse (usando Spring Security) la clase **com.tasks.config.WebSecurityConfig** añadiendo:
    1. Reglas para permitir el acceso a los **recursos estáticos**, es decir, a los elementos que no forman parte de la API REST como, por ejemplo, los ficheros HTML, los ficheros JavaScript o las hojas de estilo.
    2. Reglas para permitir el acceso a los **recursos** de la API REST que no requieren autorización (e.g. las operaciones que permiten recuperar los proyectos y las tareas o el *endpoint* de autenticación/autorización).
    3. NOTA: el apartado 4.3.1 muestra un ejemplo significativo (en ese apartado, se usa el rol MANAGER en lugar de ADMIN).

c) Configurar el control de acceso según los **roles** del usuario (e.g. sólo los administradores pueden dar de alta proyectos). Este paso se puede implementar añadiendo reglas de control de acceso a **WebSecurityConfig**. Finalmente, en algunos casos, es necesario adicionalmente comprobar el concepto de "propietario" (e.g. un administrador sólo puede borrar los proyectos que él mismo ha creado). Para este último aspecto, se deberá modificar el código de **ProjectController**, **TaskController**, **ProjectService** y **TaskService**.

d) Implementar la autenticación/autorización con OAuth, usando el flujo **authorization code** (con PCKE) para aplicaciones web SPA:
    1. El fujo se iniciará desde la función **handleSSO** del componente **Login** (**tasks-app/src/main/resources/static/application/frontend/users/Login.js**). Esta función redirigirá el navegador al endpoint HTML de autorización (apartado 4.2.3, diapositiva 55):

   i. La dirección de este endpoint es: **http://127.0.0.1:7777/oauth2/authorize**.
   ii. Como **client_id** se empleará **tasks_app**.
   iii. Como **redirect_uri** se empleará **http://127.0.0.1:8888/tasks-service/dashboard/loginOAuth**.
   iv. NOTA: no se usará el parámetro **scope**.
2. Cuando el servidor de autorización redirija a la URL especificada en **redirect_uri**, se ejecutará el componente **OAuthLogin** (**tasks-app/src/main/resources/static/application/frontend/users/OAuthLogin.js**), que debe realizar la petición al endpoint que permite obtener el token de acceso a partir del código recibido (apartado 4.2.3, diapositiva 56).
3. El código fuente de los componentes **Login** y **OAuthLogin** incluye comentarios para guiar en la implementación del flujo.
4. El formato de los tokens emitidos por el endpoint de autenticación/autorización propia es compatible con el emitido por el servidor de autorización de OAuth (ambos contienen los campos **sub**, **roles** y **exp** en el cuerpo), por lo que las modificaciones hechas en b) y c) también aplicarán a la autenticación/autorización con OAuth.

2.3 Importante

En todas las URLs (tanto las referentes al servidor de OAuth como a la aplicación web) que aparecen en el apartado 2.2 se utiliza el **127.0.0.1** en lugar de **localhost**, y efectivamente, debe emplearse **127.0.0.1**. Esto es debido a una limitación del servidor de OAuth empleado en la práctica.

3. Normativa y evaluación

3.1 Composición de los grupos

La práctica se realizará, de forma preferente, manteniendo los grupos de la práctica 1.

3.2 Formato de entrega

Es necesario presentar la aplicación web terminada, junto con un documento en el que se detallarán todas las modificaciones realizadas sobre la aplicación original.

3.3 Fecha de entrega

La fecha límite para presentar la práctica es el miércoles, 20 de diciembre (inclusive). En la UDC, la defensa será el jueves, 21 de diciembre, en horario de clase. En UVIGO, la defensa será el viernes, 22 de diciembre, también en horario de clase.

3.4 Evaluación

La puntuación sobre 10 de cada uno de los apartados será la siguiente:

1. Terminar de implementar el *endpoint* de autenticación/autorización propia contra la base de datos local. 1 punto.
2. Configurar el control de acceso que no requiere roles. 3 puntos.
3. Configurar el control de acceso que requiere roles. 3 puntos.
4. Implementar la autenticación/autorización con OAuth. 3 puntos.

Una vez realizada la entrega de la práctica, se realizará una defensa de ésta. En la defensa se tendrá en cuenta la documentación presentada, junto con la versión modificada de la aplicación.

La defensa será conjunta con todos los miembros del grupo, pero se realizarán preguntas de forma individualizada a cada alumno. Por lo tanto, en base a las respuestas obtenidas, la nota de cada uno de los miembros del grupo puede ser diferente. Si durante la defensa, un alumno no es capaz de responder de forma correcta a las preguntas, la práctica se considerará suspensa para ese alumno.

Si se detecta que alguna práctica ha sido copiada, ésta se considerará suspensa para todos los grupos involucrados, sin tener en cuenta al grupo que realmente la ha realizado.

4. Referencias

- Java: https://docs.oracle.com/en/java/javase/17/docs/api/index.html
- Spring Security: https://spring.io/projects/spring-security
- Spring Boot: https://spring.io/projects/spring-boot
- Spring MVC: https://docs.spring.io/spring-framework/reference/web/webmvc.html
- Hibernate: http://hibernate.org/
- React: https://react.dev/