SAPP P2

Índice

a) Endpoint de autenticación/autorización local

El la clase `com.tasks.rest.UserController` sustituimos el siguiente bloque del método `doLogin()`...

```
String token = null;
if("user1".equals(credentials.getUsername())) {
    token = "eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ1c2VyMSIsImV4cCI6MjMwOTYwNTA5NSwic
} else if("admin1".equals(credentials.getUsername())) {
    token = "eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJhZG1pbjEiLCJleHAiOjIzMDk2MDUxMzUsI
}
```

... por las siguientes líneas:

```
UserDetails userDetails = userService.loadUserByUsername(credentials.getUsernam
String token = tokenProvider.generateToken(userDetails);
```

De esta forma, se genera el JWT de forma independiente al usuario cada vez que se realiza el inicio de sesión. Además, se reduce el tiempo de expiración del token del día "`Tue 10 March 2043`" al día siguiente del inicio de sesión.

b) Configuración del control de acceso sin roles

En la clase `com.tasks.config.WebSecurityConfig` modificamos `securityFilterChain()`.

Primero establecimos la política no permisiva "deny-by-default":

```
@Bean
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Except

    http.csrf((csrf) -> csrf.disable())
        .sessionManagement((sessionManagement) -> sessionManagement.sessionCrea
        .addFilterBefore(new JwtAuthorizationFilter(tokenProvider, authenticati
        .authorizeHttpRequests((authorize) -> authorize

            /* ... */

            .anyRequest().denyAll());

    return http.build();

}
```

Después permitimos el acceso a las vistas de la aplicación:

```
/* ... */
.requestMatchers(HttpMethod.GET,  "/dashboard").permitAll()
.requestMatchers(HttpMethod.GET,  "/dashboard/").permitAll()
/* ... */
```

Luego añadimos las rutas a los ficheros estáticos:

```
/* ... */
.requestMatchers(HttpMethod.GET,  "/css/*").permitAll()
.requestMatchers(HttpMethod.GET,  "/react-libs/*").permitAll()
.requestMatchers(HttpMethod.GET,  "/javascript-libs/**").permitAll()
.requestMatchers(HttpMethod.GET,  "/webjars/**").permitAll()
.requestMatchers(HttpMethod.GET,  "/application/*").permitAll()
.requestMatchers(HttpMethod.GET,  "/application/common/*").permitAll()
.requestMatchers(HttpMethod.GET,  "/application/backend/**").permitAll()
.requestMatchers(HttpMethod.GET,  "/application/frontend/**").permitAll()
/* ... */
```

Finalmente permitimos el uso de los *endpoints* del API que no requerían autorización:

```
/* ... */
.requestMatchers(HttpMethod.GET,  "/api/projects").permitAll()
.requestMatchers(HttpMethod.GET,  "/api/projects/*").permitAll()
.requestMatchers(HttpMethod.GET,  "/api/projects/*/tasks").permitAll()
.requestMatchers(HttpMethod.GET,  "/api/tasks").permitAll()
.requestMatchers(HttpMethod.GET,  "/api/tasks/*").permitAll()
.requestMatchers(HttpMethod.GET,  "/api/comments/*").permitAll()
.requestMatchers(HttpMethod.POST,  "/api/login").permitAll()
/* ... */
```

c) Configuración del control de acceso según los roles del usuario

Creación de la excepción PermissionException

Creamos la excepción `com.tasks.business.exceptions.PermissionException`:

```
package com.tasks.business.exceptions;

public class PermissionException extends Exception{
    public PermissionException() {
        super("Not allowed");
    }
}
```

Tras su creación, modificamos la clase `com.tasks.rest.ExceptionsHandler` para el manejo de esta excepción:

```
@ExceptionHandler(PermissionException.class)
public final ResponseEntity<ErrorDetailsResponse> handlePermissionException(Per
    logger.warn(ex.getMessage(), ex);
    ErrorDetailsResponse errorDetails = new ErrorDetailsResponse(System.current
    ex.getMessage(), null, request.getContextPath(), 403);
    return new ResponseEntity<>(errorDetails, HttpStatus.FORBIDDEN);
}
```

Actualización de la cadena de filtros

Actualizamos el control de acceso a los *endpoints* de la API según las especificaciones del enunciado.

```
/* ... */
/* REST API for projects */
.requestMatchers(HttpMethod.POST,  "/api/projects").hasRole("ADMIN")
.requestMatchers(HttpMethod.PUT,  "/api/projects/*").hasRole("ADMIN")
.requestMatchers(HttpMethod.DELETE,  "/api/projects/*").hasRole("ADMIN")
/* REST API for tasks */
.requestMatchers(HttpMethod.POST,  "/api/tasks").hasRole("ADMIN")
.requestMatchers(HttpMethod.PUT,  "/api/tasks/*").hasRole("ADMIN")
.requestMatchers(HttpMethod.DELETE,  "/api/tasks/*").hasRole("ADMIN")
.requestMatchers(HttpMethod.POST,  "/api/tasks/*/changeState").hasRole("ADMIN")
.requestMatchers(HttpMethod.POST,  "/api/tasks/*/changeResolution").hasRole("US
.requestMatchers(HttpMethod.POST,  "/api/tasks/*/changeProgress").hasRole("USER
.requestMatchers(HttpMethod.POST,  "/api/comments").hasAnyRole("USER", "ADMIN")
/* REST API for users */
.requestMatchers(HttpMethod.GET,  "/api/users").hasRole("ADMIN")
/* ... */
```

Implementación del concepto de Propietario

Para la implementación de este concepto modificamos las clases `ProjectController`, `TaskController`, `ProjectService` y `TaskService`.

Primero añadimos la excepción `PermissionException` a las cabeceras de las funciones pertinentes.

En las `ProjectController`, `TaskController` añadimos el argumento `Principal principal` al comienzo de las funciones que requerían modificación. Después se utilizamos el método `principal.getName()` para enviar el nombre de usuario a las funciones del los servicios correspondientes. Las funciones modificadas son las siguientes:

- `com.tasks.rest.ProjectController#doRemoveProjectById`
- `com.tasks.rest.ProjectController#doUpdateProject`
- `com.tasks.rest.TaskController#doChangeTaskProgress`
- `com.tasks.rest.TaskController#doChangeTaskResolution`
- `com.tasks.rest.TaskController#doChangeTaskState`
- `com.tasks.rest.TaskController#doCreateTask`
- `com.tasks.rest.TaskController#doRemoveTaskById`
- `com.tasks.rest.TaskController#doUpdateTask`

En las `ProjectController`, `TaskController` añadimos el argumento `String callerName` al comienzo de las funciones que requerían modificación para recibir la información de los controladores correspondientes. Después implementamos la lógica de comparación del

usuario que inicia la acción con el usuario "propietario", lanzando la excepción en caso de no cumplirse la condición. Las funciones modificadas son las siguientes:

- `com.tasks.business.ProjectService#removeById`
- `com.tasks.business.ProjectService#update`
- `com.tasks.business.TasksService#changeProgress`
- `com.tasks.business.TasksService#changeResolution`
- `com.tasks.business.TasksService#changeState`
- `com.tasks.business.TasksService#create`
- `com.tasks.business.TasksService#removeById`
- `com.tasks.business.TasksService#update`

Detalles adicionales

Se han añadido los decoradores necesarios para completar la documentación de Swagger con los códigos HTTP que no estaban contemplados antes de la implementación de `PermissionException`.

```
@ApiResponse(responseCode = "403", description = "Not Allowed",
    content = {@Content(mediaType = "application/json",
        schema = @Schema(implementation = ErrorDetailsResponse.class))}),
```

Se ha refactorizado la excepción `IvalidStateException` como `InvalidStateException`, corrigiendo la errata.

```
@ExceptionHandler(DuplicatedResourceException.class)
public final ResponseEntity<ErrorDetailsResponse> handleDuplicatedResourceExcep
    DuplicatedResourceException ex, WebRequest request) {
    logger.warn(ex.getMessage(), ex);
    ErrorDetailsResponse errorDetails = new ErrorDetailsResponse(System.current
    ex.getMessage(), null, request.getContextPath(), 409);
    return new ResponseEntity<>(errorDetails, HttpStatus.NOT_FOUND);
}
```

d) Autenticación/autorización con OAuth

Añadimos la vista de *login* de OAuth a la cadena de filtros:

```
/* ... */
.requestMatchers(HttpMethod.GET,  "/dashboard/loginOAuth").permitAll() // OAuth
/* ... */
```

Después completamos los ficheros `Login.js` y `OAuthLogin.js` siguiendo las indicaciones del enunciado.

Login.js

```
// FIXME.
// Set the application URI the authorization server must redirect to after the
// authorization server authentication form.
-const redirectUri = '';
+const redirectUri = 'http://127.0.0.1:8888/tasks-service/dashboard/loginOAuth'
```

```
// FIXME.
// Store codeVerifier in sessionStorage.
+sessionStorage.setItem("codeVerifier", codeVerifier);
```

```
// FIXME
// Make the browser to go to the authorization server authentication form.
// Remember to add all necessary parameters in the URL.
-window.location.replace('');
+window.location.replace("http://127.0.0.1:7777/oauth2/authorize" + '?'
+    + "response_type=code" + '&'
+    + "client_id=tasks_app" + '&'
+    + "redirect_uri=" + redirectUri + '&'
+    + "code_challenge=" + codeChallenge + '&'
+    + "code_challenge_method=S256"
+);
```

OAuthLogin.js

```
// FIXME.
// Get codeVerifier from sessionStorage.
-const codeVerifier = '';
+const codeVerifier = sessionStorage.getItem("codeVerifier");
```

```
// FIXME.
// Remove codeVerifier from sessionStorage.
+sessionStorage.removeItem("codeVerifier");
```

```
// FIXME.
// Add rest of parameters to make the request to the token endpoint.
+tokenParams.set("code", code);
+tokenParams.set("redirect_uri", "http://127.0.0.1:8888/tasks-service/dashboard
+tokenParams.set("grant_type", "authorization_code");
+tokenParams.set("client_id", "tasks_app");
+tokenParams.set("code_verifier", codeVerifier);
```