DESARROLLO DE APLICACIONES JEE CON SPRING FRAMEWORK

Dirección Nacional de Educación Continua

13-11-2021 Módulo 5

INTRODUCCIÓN

- La presente tiene como objetivo explicar las capacidades de un entorno de trabajo, sus funciones y capacidades, además explicar en detalle algunos de los módulos del framework de spring, con la intención de preparar al lector para generar soluciones empresariales simples y efectivas, conociendo la arquitectura y con esto ser un aporte al momento de diseñar una aplicación, aplicando los conocimientos de este documento.

RESUMEN

- En el Módulo anterior, hemos revisado lo siguiente:
- Control de acceso.
- Creando un formulario login
- Manejo de roles
- Añadiendo seguridad a la capa vista.
- Seguridad en los controladores

LA INTEROPERABILIDAD ENTRE SISTEMAS

- La interoperabilidad es un concepto definido en el **Vocabulario de Información y Tecnología ISO/ISO/IEC 2382** como “la capacidad de comunicar, ejecutar programas o transferir datos entre varias unidades funcionales de manera que el usuario no tenga que conocer las características únicas de estas unidades”. (ISO, 2000). Otros autores como Lueders en 2004 lo definieron como “la habilidad de los Sistemas TIC y de los procesos de negocios que ellas soportan, de **intercambiar datos y posibilitar compartir información y conocimiento”.**
- Estas definiciones nos recuerdan aquella vieja idea del ser humano de **poder compartir la información de manera universal**, más allá de la tecnología que se encargue de su almacenamiento, su procesamiento o su distribución. Persiguiendo este objetivo se inventó la imprenta en 1440 y en 1969 se creó la red ARPANET, que pronto se desarrollaría en el Internet que hoy conocemos.
- Hoy en día, en plena explosión de datos digitales, **cobran fuerza las tecnologías que facilitan la interoperabilidad de las empresas** y que les permiten aprovecharse de distintos beneficios.

PROTOCOLOS DE INTERCAMBIO DE DATOS

- Un protocolo es un conjunto de reglas: los **protocolos de red** son estándares y políticas formales, conformados por restricciones, procedimientos y formatos que definen el intercambio de **paquetes** de información para lograr la comunicación entre dos **servidores** o más dispositivos a través de una red.

- Los **protocolos de red** incluyen mecanismos para que los dispositivos se identifiquen y establezcan conexiones entre sí, así como reglas de formato que especifican cómo se forman los **paquetes** y los datos en los mensajes enviados y recibidos. Algunos protocolos admiten el reconocimiento de mensajes y la compresión de datos diseñados para una comunicación de red confiable de alto rendimiento.

TIPOS DE PROTOCOLOS DE RED

- Los protocolos para la **transmisión de datos** en internet más importantes son TCP (Protocolo de Control de Transmisión) e IP (**Protocolo de Internet)**. De manera conjunta (TCP/IP) podemos enlazar los dispositivos que acceden a la red, algunos otros **protocolos de comunicación** asociados a internet son POP, SMTP y HTTP.

- Estos los utilizamos prácticamente todos los días, aunque la mayoría de los usuarios no lo sepan ni conozcan su funcionamiento. Estos protocolos permiten la **transmisión de datos** desde nuestros dispositivos para navegar a través de los sitios, enviar correos electrónicos, escuchar música online, etc.

- Existen varios tipos de protocolos de red:

- Protocolos de comunicación de red: protocolos de comunicación de **paquetes** básicos como TCP / IP y HTTP.
- Protocolos de seguridad de red: implementan la seguridad en las comunicaciones de red entre **servidores**, incluye HTTPS, SSL y SFTP.
- Protocolos de gestión de red: proporcionan mantenimiento y gobierno de red, incluyen SNMP e ICMP.

¿QUE ES EL ESTADO REPRESENTACIONAL DE TRANSFERENCIA?

- La transferencia de estado representacional (REST) es un estilo de arquitectura de software que utiliza un subconjunto de HTTP . Se utiliza comúnmente para crear aplicaciones interactivas que utilizan servicios WEB. Un servicio web que sigue estas pautas se llama RESTful. Dicho servicio web debe proporcionar sus recursos WEB en una representación textual y permitir que se lean y modifiquen con un protocolo sin estado y un conjunto predefinido de operaciones. Este enfoque permite la interoperabilidad entre los sistemas informáticos en Internet que prestan estos servicios. REST es una alternativa a, por ejemplo, SOAP como forma de acceder a un servicio web.

LA NOTACIÓN JSON PARA EL TRASPASO DEINFORMACIÓN

- JSON, cuyo nombre corresponde a las siglas JavaScript Object Notation o Notación de Objetos de JavaScript, es un formato ligero de intercambio de datos, que resulta sencillo de leer y escribir para los programadores y simple de interpretar y generar para las máquinas.

- JSON es un formato de texto completamente independiente de lenguaje, pero utiliza convenciones que son ampliamente conocidos por los programadores, entre ellos:

  - C#
  - Java
  - JavaScript
  - Perl
  - Python
  - C
  - C++

CONSUMIENDO UN SERVICIO REST SPRING Y REST TEMPLATE

```java
@RestController
public class BookController {

    @Autowired
    private BookRepository repository;

    // Find
    @GetMapping("/books")
    List<Book> findAll() {
        return repository.findAll();
    }

    // Save
    @PostMapping("/books")
    //return 201 instead of 200
    @ResponseStatus(HttpStatus.CREATED)
    Book newBook(@RequestBody Book newBook) {
        return repository.save(newBook);
    }

    // Find
    @GetMapping("/books/{id}")
    Book findOne(@PathVariable Long id) {
        return repository.findById(id)
                .orElseThrow(() -> new BookNotFoundException(id));
    }

    // Save or update
    @PutMapping("/books/{id}")
    Book saveOrUpdate(@RequestBody Book newBook, @PathVariable Long id) {

        return repository.findById(id)
                .map(x -> {
                    x.setName(newBook.getName());
                    x.setAuthor(newBook.getAuthor());
                    x.setPrice(newBook.getPrice());
                    return repository.save(x);
                })
```

- Como hemos visto en los módulos anteriores, para consumir un servicio rest construido con el IDE STS utilizaremos una herramienta open llamada Postman.
- Primero que todo se debe configurar el archivo POM.xml

```xml
<dependencies>

    <!-- spring mvc, rest -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
```

PRINCIPIO DISEÑO DE API REST

- Al desarrollar una API RESTful de Java, los diseñadores deben considerar dos elementos clave:
  - Patrón de URL
  - Qué método HTTP usar
- El primer principio importante que enfatizamos es que siempre se debe acceder a los recursos a través de una URL que los identifique de manera única.
- Para cualquiera que haya utilizado un navegador web, este es un concepto completamente nuevo. Cuando visitamos una página web o descargamos un archivo PDF basado en la web, dirigimos el navegador a la URL que identifica el recurso. El mismo concepto también se aplica cuando se utilizan los servicios web RESTful Java para acceder a los recursos del lado del servidor. Si el cliente jQuery o Angular necesita manipular recursos, debe haber una URL única que permita que el código JavaScript relevante identifique y ubique el recurso RESTful correspondiente.

PRINCIPIO DISEÑO DE API REST

Regla de diseño RESTful: la llamada GET no puede cambiar el estado del servidor. Para lidiar con los métodos HTTP, debe seguir importantes reglas de diseño RESTful. Si los diseñadores de la API de RESTful Java violan estas reglas, se extraviarán.

El primer principio es, las llamadas GET nunca pueden cambiar el estado de ningún recurso RESTfull en el servidor.

El segundo principio es, los métodos RESTfull PUT y DELETE deben cumplir con el principio de idempotencia

Aunque no es una regla estricta, los métodos PUT y DELETE mapean aproximadamente los conceptos de guardar y eliminar. Si los diseñadores desean eliminar recursos del servidor, deben usar el método HTTP DELETE. Si necesita crear un nuevo recurso o necesita actualizar un recurso existente, debe usar el método PUT.

Los métodos PUT y DELETE son relativamente simples para guardar y eliminar datos. Pero esta es otra trampa que los diseñadores de API RESTful Java encuentran a menudo. Esto conduce a la segunda regla: El método HTTP debe ser idempotente.

PRINCIPIO DISEÑO DE API REST

RESTful API: método POST

Ya sabemos que eliminar los 10 registros más antiguos de la base de datos es un uso incorrecto del método DELETE, y un simple incremento numérico es una mala aplicación del método PUT. ¿Significa esto que no podemos usar API RESTful para lograr estas cosas? por supuesto no.

En cualquier escenario distinto de las reglas anteriores, se puede utilizar el método POST. Por lo tanto, si desea eliminar los 10 registros más antiguos de la base de datos, puede utilizar el método POST.

CREANDO UNA API REST CON SPRINGMVC

```java
@RestController
public class BookController {
    @Autowired
    private BookRepository repository;
    @GetMapping("/books/{id}")
    Book findOne(@PathVariable Long id) {
        return repository.findById(id)
                .orElseThrow(() -> new BookNotFoundException(id));
    }
    @PutMapping("/books/{id}")
    Book saveOrUpdate(@RequestBody Book newBook, @PathVariable Long id) {

        return repository.findById(id)
                .map(x -> {
                    x.setName(newBook.getName());
                    x.setAuthor(newBook.getAuthor());
                    x.setPrice(newBook.getPrice());
                    return repository.save(x);
                })
                .orElseGet(() -> {
                    newBook.setId(id);
                    return repository.save(newBook);
                });
    }
    @DeleteMapping("/books/{id}")
    void deleteBook(@PathVariable Long id) {
        repository.deleteById(id);
    }
    @PostMapping("/books")
    //return 201 instead of 200
    @ResponseStatus(HttpStatus.CREATED)
    Book newBook(@RequestBody Book newBook) {
        return repository.save(newBook);
    }
    @PatchMapping("/books/{id}")
    Book patch(@RequestBody Map<String, String> update, @PathVariable Long id) {
        return repository.findById(id)
                .map(x -> {

                    String author = update.get("author");
                    if (!StringUtils.isEmpty(author)) {
                        x.setAuthor(author);

                        // better create a custom method to update a value = :newValue w
                        return repository.save(x);
                    } else {
                        throw new BookUnSupportedFieldPatchException(update.keySet());
                    }

                })
                .orElseGet(() -> {
                    throw new BookNotFoundException(id);
                });

    }
```

- Como hemos visto en módulos anteriores, para la creación de un api rest, utilizaremos un proyecto construido con STS con las librerías del framework para web.

```xml
<!-- spring mvc, rest -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

CREANDO UNA API REST CON SPRINGMVC

```
<!DOCTYPE HTML>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

    <title>Spring Boot Thymeleaf Hello World Example</title>

    <link rel="stylesheet" th:href="@{webjars/bootstrap/4.2.1/css/bootstrap.min.css}" />
    <link rel="stylesheet" th:href="@{/css/main.css}" />

</head>

<body>

<nav class="navbar navbar-expand-md navbar-dark bg-dark fixed-top">
    <a class="navbar-brand" href="#">AIEP</a>
</nav>

<main role="main" class="container">

    <div class="starter-template">
        <h1>Spring Boot Web Thymeleaf Example</h1>
        <h2>
            <span th:text="'Hello, ' + ${message}"></span>
        </h2>
    </div>

    <ol>
        <li th:each="task : ${tasks}" th:text="${task}"></li>
    </ol>

</main>
<!-- /.container -->

<script type="text/javascript" th:src="@{webjars/bootstrap/4.2.1/js/bootstrap.min.js}"></script>
</body>
</html>
```

- Como también hemos visto en módulos anteriores para la construcción de un template HTML lo realizamos con Thymeleaf

SEGURIZACIÓN DE UNA API REST MEDIANTE JWT

- **¿Qué es JWT?**
- JSON Web Token (abreviado JWT) es un estándar abierto basado en JSON propuesto por IETF (RFC 7519) para la creación de tokens de acceso que permiten la propagación de identidad y privilegios o claims en inglés. Por ejemplo, un servidor podría generar un token indicando que el usuario tiene privilegios de administrador y proporcionarlo a un cliente. El cliente entonces podría utilizar el token para probar que está actuando como un administrador en el cliente o en otro sistema. El token está firmado por la clave del servidor, así que el cliente y el servidor son ambos capaz de verificar que el token es legítimo. Los JSON Web Tokens están diseñados para ser compactos, poder ser enviados en las URLs -URL-safe- y ser utilizados en escenarios de Single Sign-On (SSO). Los privilegios de los JSON Web Tokens pueden ser utilizados para propagar la identidad de usuarios como parte del proceso de autenticación entre un proveedor de identidad y un proveedor de servicio, o cualquiera otro tipo de privilegios requeridos por procesos empresariales.
- El estándar de JWT se basa en otros estándares basados en JSON JSON Web Signature (RFC 7515) y JSON Web Encryption (RFC 7516)

SEGURIZACIÓN DE UNA API REST MEDIANTE JWT

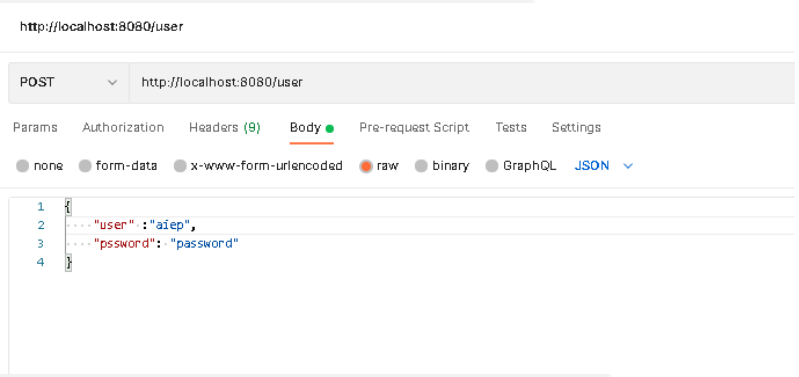
```
<!-- JSON WEB TOKEN -->
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt</artifactId>
    <version>0.9.0</version>
</dependency>
<dependency>
    <groupId>org.json</groupId>
    <artifactId>json</artifactId>
    <version>20160810</version>
</dependency>
<!-- API, java.xml.bind module -->
<dependency>
    <groupId>jakarta.xml.bind</groupId>
    <artifactId>jakarta.xml.bind-api</artifactId>
    <version>2.3.2</version>
</dependency>
<!-- Runtime, com.sun.xml.bind module -->
<dependency>
    <groupId>org.glassfish.jaxb</groupId>
    <artifactId>jaxb-runtime</artifactId>
    <version>2.3.2</version>
</dependency>
```

- Para la segurización de las APIS RESTfull con JWT, utilizaremos el IDE STS y para probar las APIS el programa PostMan.
- Dentro del proyecto, se debe considerar la modificación en los siguientes puntos.
  - POM.xml.
  - Filtro de seguridad visto en el módulo anterior, se agrega el filtro JWT de la librería importada.

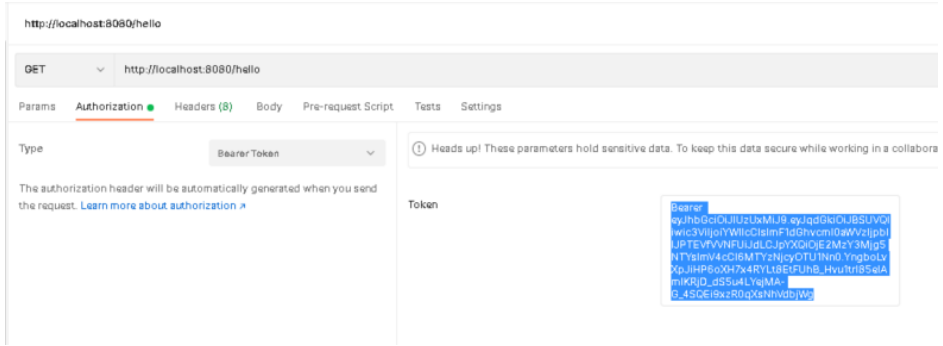
```
@EnableWebSecurity
@Configuration
class WebSecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.csrf().disable()
            .addFilterAfter(new JWTAuthorizationFilter(), UsernamePasswordAuthenticationFilter.class)
            .authorizeRequests()
            .antMatchers(HttpMethod.POST, "/user").permitAll()
            .anyRequest().authenticated();
    }
}
```

SEGURIZACIÓN DE UNA API REST MEDIANTE JWT



- Además se deben construir y modificar servicios.
  - Crear un Servicio de usuario, para obtener el token.
- Luego con Postman se debe invocar el servicio con un usuario y pass valido
  - Te entrega el token Bearer
  - Ese token se debe incluir en la seguridad de los servicios que tengan el



PRUEBA PRÁCTICA NRO5

- El alumno deberá crear un proyecto starter spring boot (recomiendo Práctica 4)
- Crear una clase filtro con los usuarios en memoria y los atributos para segurizar de manera básica el servicio que obtiene token (solo la api Usuario) .
- Modificar una clase RestController agregando todos los del diseño REST FULL
  - Método /user POST que recibe 1 parámetros dentro del body JSON username y retorna token bearer
  - Método /hello que recibe 1 parámetro que se suma al “hola” + parámetro.
- El método hello no puede ser consultado sin el token entregado anteriormente por el rest /user
- Evidenciar pruebas postman paso a paso.
- Incluir las librerías necesarias en el POM.xml para ejecutar las pruebas

AIEP
de la Universidad
Andrés Bello

Certified
B
Corporation

MUCHAS GRACIAS