Crear primeros Servicios

AR

Utilizando : Spring WEB

MAVEN

Spring JPA

JAKARTA VALIDATION

Windows 10

PostgreSQL,

SWAGGER

Lombok, Spring Boot

JAVA

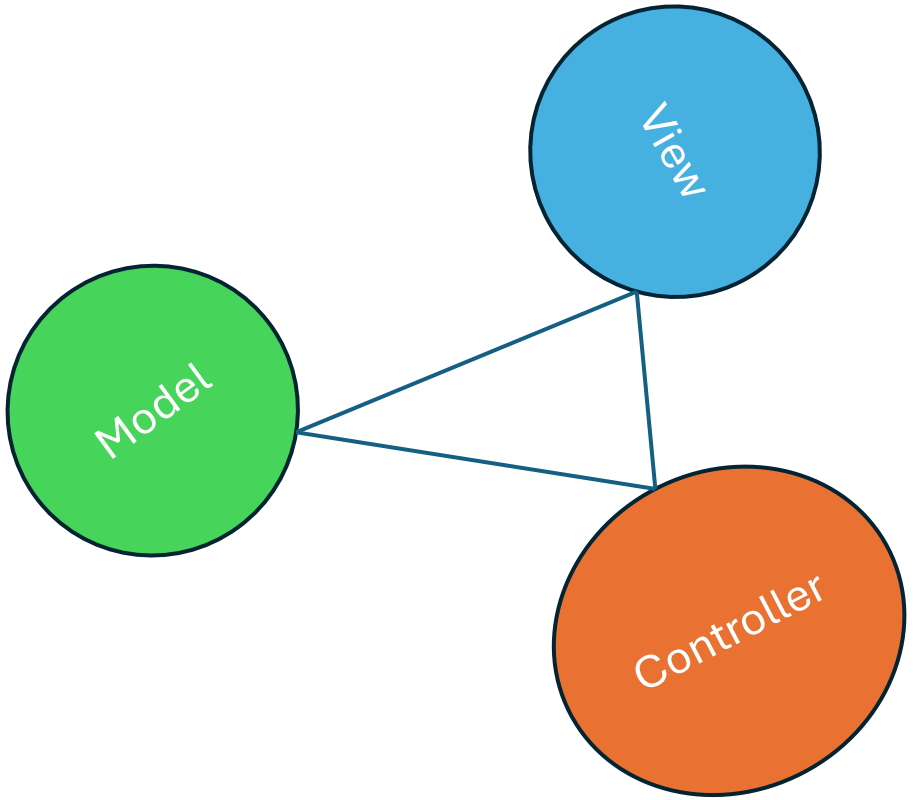
MVC

@Anotation´s

Spring Tools 4

PARTE 2

M V C



Componente	Rol principal
Modelo	Gestiona los datos y la lógica de negocio. Puede incluir entidades JPA, DAOs.
Vista	Presenta los datos al usuario. En Java web, puede ser HTML con Thymeleaf o JSP.
Controlador	Recibe las solicitudes del usuario, actualiza el modelo y selecciona la vista adecuada.

M V C

Componente	Rol principal
Modelo	Gestiona los datos y la lógica de negocio. Puede incluir entidades JPA, DAOs.
Vista	Presenta los datos al usuario. En Java web, puede ser HTML con Thymeleaf o JSP.
Controlador	Recibe las solicitudes del usuario, actualiza el modelo y selecciona la vista adecuada.

M V C

INTRO

En esta parte abarcaremos la parte modelo y controlador, dejando la vista para una tercera parte

Usaremos la base de datos postgres, con el esquema public y la tablas users

Column Name	#	Data type	Identity	Collation	Not Null
123 iduser	1	int4			[v]
A-Z name	2	varchar(50)		default	[v]
A-Z email	3	varchar		default	[v]
123 estatus	4	numeric			[v]
A-Z password	5	varchar		default	[v]
123 versión	6	numeric			[ ]

Crearemos la clase User para crear el modelo y el controlador, aplicando en ello, validaciones (Jakarta Validation) y documentación automática con swagger (OPENAPI).

El objetivo es crear un CRUD de esta entidad, permitiendo documentación con swagger y con ello poder permitir una visualización del desarrollo realizado.

Después de ver estos puntos, crearemos las pruebas unitarias usando Junit & Mockito.

```
@Entity
@Table(name="users")
@Data
@NoArgsConstructor
@AllArgsConstructor
@Schema(description="Entidad que representa usario en el sistema")
public class User {

        @Id
@GeneratedValue(strategy = GenerationType.AUTO)
        @Schema(description = "Identificador único del usuario", example = "101")
        private Integer iduser;

        @NotBlank(message = "El nombre no puede estar vacío")
@Size(min = 3, max = 50, message = "El nombre debe tener entre 2 y 50 caracteres")
@Schema(description = "Nombre completo del usuario", example = "Juan García")
        private String name;


        @Email(message = "El correo electrónico no tiene un formato válido")
@NotBlank(message = "El correo electrónico es obligatorio")
@Schema(description = "Correo electrónico del usuario", example = "alde@example.com")
private String email;

@Min(value = 0, message = "El estado debe ser igual o mayor a 0")
@Max(value = 1, message = "El estado debe ser 0 o 1")
@Schema(description = "Estado del usuario (1 = activo, 0 = inactivo)", example = "1")
private Integer status;

@NotBlank(message = "La contraseña es obligatoria")
@Size(min = 8, message = "La contraseña debe tener al menos 8 caracteres")
@Schema(description = "Contraseña del usuario (encriptada)", example = "abc123XYZ!")
private String password;

@Version
@Schema(description = "Versión de la entidad para control de concurrencia", example = "3")
private Integer version;
```

Primero vamos a ingresar las dependencias para swagger, Validation, la parte correspondiente a lombok, ya hemos hecho el registro en pasos anteriores.

En el archivo **pom.xml** declarar la siguiente dependencia:

```
<!-- Swagger Integration -->
<dependency>
	<groupId>org.springdoc</groupId>
	<artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
	<version>2.8.9</version>
</dependency>

<dependency>
	<groupId>org.hibernate.validator</groupId>
	<artifactId>hibernate-validator</artifactId>
	<!-- <version>9.0.1.Final</version> Usa una versión compatible con Jakarta -->
</dependency>


<dependency>
	<groupId>org.springframework.boot</groupId>
	<artifactId>spring-boot-starter-validation</artifactId>
</dependency>
```

Con estas dependencias continuaremos con la configuración de Swagger en la próxima diapositiva.

Pasos para configurar el Swagger OpenAPI 3.0

En el archivo **src/main/resources/application.properties** declarar las siguientes variables:

```
springdoc.swagger-ui.enabled=true
springdoc.api-docs.enabled=true
springdoc.api-docs.version=OPENAPI_3_0
springdoc.swagger-ui.path=/swagger-ui.html
```

Con esta configuración de Swagger, podemos continuar con nuestra generación de código

Properties

M V C

Manos a la obra, primero crearemos la base clase User, aquí su código.

Primero crearemos el paquete: com.curso.spring.entities

Y dentro de este paquete crearemos la clase User.java a un lado el código.

Como podrán notar, la clase cuenta con **ANOTACIONES** unas de **lombok**

```
@Data
@NoArgsConstructor
@AllArgsConstructor
```

Otras de Persistencia

```
@Id
@GeneratedValue(strategy = GenerationType.AUTO)
```

Otras de Validación

```
@NotBlank(message = "El nombre no puede estar vacío")
@Size(min = 3, max = 50, message = "El nombre debe tener entre 2 y 50 caracteres")
```

Otras de Swagger

```
@Schema(description = "Estado del usuario (1 = activo, 0 = inactivo)", example = "1")
```

Entity

```
package com.curso.spring.entities;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.Table;
import jakarta.persistence.Version;
import jakarta.validation.constraints.Email;
import jakarta.validation.constraints.Max;
import jakarta.validation.constraints.Min;
import jakarta.validation.constraints.NotBlank;
import jakarta.validation.constraints.Size;
import io.swagger.v3.oas.annotations.media.Schema;
import lombok.*;

@Entity
@Table(name="users")
@Data
@NoArgsConstructor
@AllArgsConstructor
@Schema(description="Entidad que representa usario en el sistema")
public class User {
@Id
@GeneratedValue(strategy = GenerationType.AUTO)
@Schema(description = "Identificador único del usuario", example = "101")
private Integer iduser;
@NotBlank(message = "El nombre no puede estar vacío")
@Size(min = 3, max = 50, message = "El nombre debe tener entre 2 y 50 caracteres")
@Schema(description = "Nombre completo del usuario", example = "Juan García")
private String name;
@Email(message = "El correo electrónico no tiene un formato válido")
@NotBlank(message = "El correo electrónico es obligatorio")
@Schema(description = "Correo electrónico del usuario", example = "alde@example.com")
private String email;


@Min(value = 0, message = "El estado debe ser igual o mayor a 0")
@Max(value = 1, message = "El estado debe ser 0 o 1")
@Schema(description = "Estado del usuario (1 = activo, 0 = inactivo)", example = "1")
private Integer status;


@NotBlank(message = "La contraseña es obligatoria")
@Size(min = 8, message = "La contraseña debe tener al menos 8 caracteres")
@Schema(description = "Contraseña del usuario (encriptada)", example = "abc123XYZ!")
private String password;


@Version
@Schema(description = "Versión de la entidad para control de concurrencia", example = "3")
private Integer version;

}
```

MVC

Repository

Ahora crearemos las clases relacionadas a los componentes Repositorios en Spring, para ello utilizaremos las anotaciones @Repository y aprovecharemos las bondades que nos da Spring y sus componentes.

En este caso hay Interfaces que cuentan con un comportamiento ya definido para realizar CRUD sobre una entidad, para nuestro caso el CREATE, READ, UPDATE y DELETE usando JPA.

Crearemos el paquete com.curso.spring.repository
Después dentro del paquete creado crearemos la interface UserRepository.java.

El código a un lado, de este texto, la anotación **@Repository** indica que es componente de acceso a una entidad de base de datos.

La interface JpaRepository **<?,?> de JPA contiene el comportamiento definido para realizar CRUD y muchos mas sobre la entidad, solo se especifica como parámetro la clase de la entidad asociada a la base de datos en este caso User y el tipo de dato de la llave primaria que en este caso es Integer.**

Con esto ya hemos definido nuestro Repositorio para la entidad User con el comportamiento(métodos) para hacer un CRUD de esta entidad. FACIL, NO CREES?

```java
package com.curso.spring.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;


import com.curso.spring.entities.User;

/**
* Clase para acceso a entidad User
*/
@Repository
public interface UserRepository extends JpaRepository<User, Integer> {


}
```

M V C

Exception

Antes de entrar de lleno a la creación de los servicios, que es donde se declaran las validaciones relacionadas a las reglas de negocio del servicios, es necesario como buena practica y casi siempre como requerimiento del caso de uso, manejar Excepciones, para este caso utilizaremos una excepción utilizando la clase llamada UsuarioExistenteException.java

Crearemos el paquete com.curso.spring.exceptions
Después dentro del paquete creado crearemos la interface UsuarioExistenteException.java

El código a un lado, esta Excepción se lanzara cuando se quiera crear un Usuario que ya este registrado en nuestra tablas Users y se lanzará la excepción EntityNotFoundException cuando se intente actualizar una entidad con Id Inexistente.

Si fuera otro el caso se lanza una excepción General Exception.

```
package com.curso.spring.exceptions;


public class UsuarioExistenteException extends RuntimeException {


/**
* serialVersionUID
*/
private static final long serialVersionUID = -7822491956217469436L;


public UsuarioExistenteException(String message) {
super(message);
}
}
```

M V C

Service

Ahora crearemos las clases relacionadas a los componentes Services en Spring, para ello utilizaremos las anotaciones @Service de Spring.

En este para realizar un CREATE, READ, UPDATE y DELETE se debe declarar por buena practica los métodos para cada caso y para ello declararemos la interface ServiceUser y la clase ServiceUserImpl que implementará los métodos definidos en la interface.

Crearemos el paquete com.curso.spring.services
Después dentro del paquete creado crearemos la interface ServiceUser.java y la clase ServiceUserImpl.java

El código a un lado, de este texto, la anotación **@Service sobre la clase** indica que es componente de servicio de Spring.

La clase ServiceUserImpl.java **contiene los métodos definidos para realizar CRUD, utilizando la interface JPA del Repositorio, misma que se conecta mediante @Autowired, los métodos se documentan usando JavaDoc utilizando las anotaciones @Param, @return, @throws.**

Lo único que no quisiera obviar es el método save que pregunta si dentro de la clase User que recibe como parámetro contiene el valor del IdUser, se lanza la excepción UsuarioExistenteException.

Con esto ya hemos definido nuestro componente de servicios Spring para la clase User. FACIL, NO CREES?

```
package com.curso.spring.services;


import java.util.List;
import java.util.Optional;
import com.curso.spring.entities.User;
import com.curso.spring.exceptions.UsuarioExistenteException;
import com.curso.spring.exceptions.UsuarioNoExistenteException;
import jakarta.persistence.EntityNotFoundException;

/**
* Clase para repositorio User
*/
public interface ServiceUser {
/**
* @param user Usuario a dar de alta
* @return Objeto User persistido
* @throws Exception Si ocurre un problema al momento de persistir el usuario
*/
public User save(User user) throws UsuarioExistenteException;
/**
* @return Lista de Usuarios encontrados
* @throws Exception Si ocurre un problema al momento de buscar usuarios
*/
public List<User> findAll() throws Exception;
/**
* @param id Identificador de usuario a buscar
* @return Objeto User si es econtrado una coincidencia con el ID utilizado
* @throws Exception Si ocurre un problema al momento de buscar el usuario
*/
public User getUserById(Integer id) throws UsuarioNoExistenteException;
/**
* @param id Identificador de usuario a eliminar
* @throws Exception Si ocurre un problema al momento de eliminar el usuario
*/
public void deleteUserById(Integer id) throws Exception;
/**
* @return Lista de todos los usuarios
* @throws Exception Si ocurre un problema al momento de consultar usuarios
*/
public List<User> getAllUser() throws Exception;
/**
* @return Lista de todos los usuarios
* @throws Exception Si ocurre un problema al momento de consultar usuarios
*/
public User updateUser(Integer id, User updatedUser) throws
EntityNotFoundException,Exception;
}
```

Por motivos de espacio, la clase completa la tendrán que ver el código del repositorio.

```
package com.curso.spring.services;

import java.util.List;
import java.util.Optional;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import com.curso.spring.entities.User;
import com.curso.spring.exceptions.UsuarioExistenteException;
import com.curso.spring.exceptions.UsuarioNoExistenteException;
import com.curso.spring.repository.UserRepository;
import jakarta.persistence.EntityNotFoundException;

@Service
public class ServiceUserImpl implements ServiceUser {

@Autowired
UserRepository repo;

@Override
/**
* @param user Usuario a dar de alta
* @return Objeto User persistido
* @throws Exception Si ocurre un problema al momento de persistir el usuario
*/
public User save(User user) throws UsuarioExistenteException {
if(user.getIduser() != null){
throw new UsuarioExistenteException("El usuario ya existe.");
}
return repo.save(user);
}

@Override
/**
* @return Lista de Usuarios encontrados
* @throws Exception Si ocurre un problema al momento de buscar usuarios
*/
public List<User> findAll() throws Exception {
return repo.findAll();
}

@Override
/**
* @param id Identificador de usuario a buscar
* @return Objeto User si es econtrado una coincidencia con el ID utilizado
* @throws Exception Si ocurre un problema al momento de buscar el usuario
*/
public User getUserById(Integer id) throws UsuarioNoExistenteException {
Optional<User> optionalUser = repo.findById(id);
if (optionalUser.isPresent()) {
return optionalUser.get();
} else {
throw new UsuarioNoExistenteException("Usuario con ID " + id + " no
encontrado");
}
.....
```

M V C

Controller

Ahora crearemos las clases relacionadas a los componentes Controller en Spring, para ello utilizaremos las anotaciones @RestController de Spring.

En este caso realizaremos un CREATE, READ, UPDATE y DELETE (CRUD)

Crearemos el paquete com.curso.spring.controller
Después dentro del paquete creado, generar la clase UserController.java

El código a un lado, de este texto, la anotación @RestController **sobre la clase** indica que es componente controlador de Spring.

La clase UserController.java **contiene los métodos definidos para realizar CRUD, utilizando las anotaciones WEB de Spring, misma que se conecta mediante @Autowired a el componente de servicios, los métodos se documentan usando anotaciones Swagger, JavaDoc, Spring Web. utilizando las anotaciones @RestController, @RequestMapping, @PostMapping, @GetMapping, @Tag, @Operation,@Param, @return.**

Lo único que no quisiera obviar es el método save que pregunta si dentro de la clase User que recibe como parámetro contiene el valor del IdUser, se lanza la excepción UsuarioExistenteException y en el método de Update lanza la excepción UsuarioNoExistenteException .

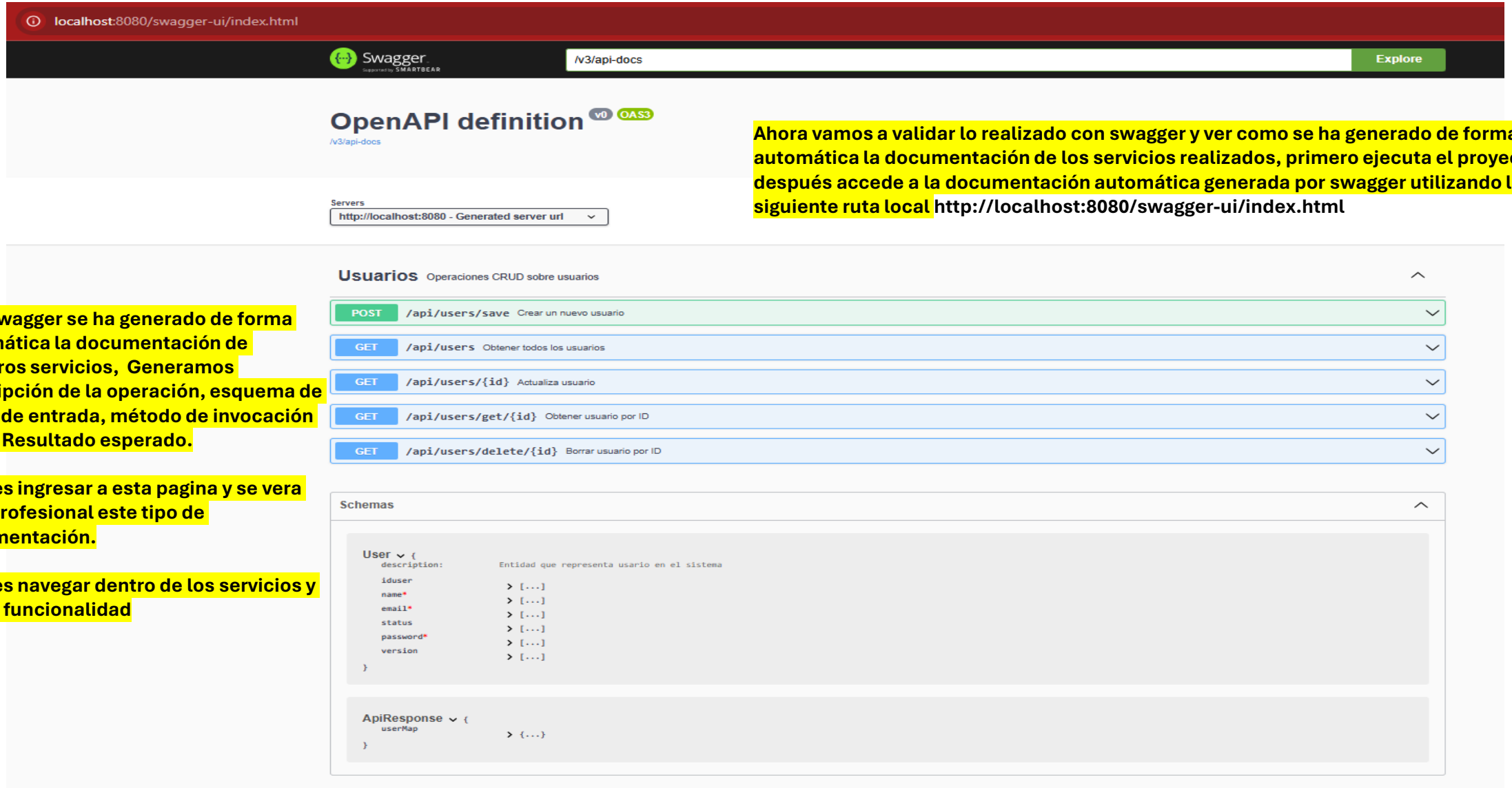
Con esto ya hemos definido nuestro componente de servicios Spring para la clase User. FACIL, NO CREES? Esto es parte de las bondades de Spring para este modelo de desarrollo y su fácil integración con otros frameworks como Swagger y Lombok. Y mas que se pueden utilizar y veremos en la siguiente parte del curso, cuando veamos Junit, Jacoco y Sonnar.

Por motivos de espacio, la clase completa la tendrán que ver el código del repositorio.

```
package com.curso.spring.controller;


@RestController
@RequestMapping("/api/users")
@Tag(name = "Usuarios", description = "Operaciones CRUD sobre usuarios")
public class UserController {

@Autowired
ServiceUser service;

/**
* Method for save record of entity User
*
* @param user is the object to save
* @return ApiResponse with Status Code Message
* Information about result of service invoked Data Object resulted
*/
@Operation(summary = "Crear un nuevo usuario")
@PostMapping("/save")
public ResponseEntity<ApiResponse> createUser(@RequestBody @Valid
User user) {
ApiResponse response = new ApiResponse();
Map<String, Object> retorno = new HashMap<>();
try {
User usr = service.save(user);
retorno.put("Estatus", 201);
retorno.put("Mensaje", "Usuario creado correctamente");
retorno.put("Data", usr);
} catch (UsuarioExistenteException e) {
retorno.put("Estatus", 400);
retorno.put("Mensaje", e.getMessage());
retorno.put("Data", user);
} catch (Exception e) {
retorno.put("Estatus", 500);
retorno.put("Mensaje", e.getCause());
retorno.put("Data", user);
}
response.setUserMap(retorno);
return ResponseEntity.badRequest().body(response);
}
```

M V C

Swagger



Ahora vamos a validar lo realizado con swagger y ver como se ha generado de forma automática la documentación de los servicios realizados, primero ejecuta el proyecto y después accede a la documentación automática generada por swagger utilizando la siguiente ruta local http://localhost:8080/swagger-ui/index.html

Con swagger se ha generado de forma automática la documentación de nuestros servicios, Generamos descripción de la operación, esquema de datos de entrada, método de invocación HTTP, Resultado esperado.

Puedes ingresar a esta pagina y se vera muy profesional este tipo de documentación.

Puedes navegar dentro de los servicios y ver su funcionalidad

M V C

Sumario

En esta parte del curso se ha realizado la generación de código para realizar un CRUD de la entidad User, Generando los componentes Spring Repositorios, Servicios y Controladores, manejo de Excepciones, Generación automática de documentación con Swagger, así como toda la configuración necesaria para ejecutarlo, todo un proyecto en forma, en la siguiente parte del curso veremos como probar los servicios realizados y como realizar las pruebas unitarias, así como generar la cobertura de pruebas realizadas al código……No te lo pierdas porque cada vez se formarán todas las capas para un desarrollo WEB con buenas practicas utilizando herramientas tecnológicas que harán tu trabajo mas ameno y profesional, hasta la vista, no olvides recomendar este curso y subscribirte para seguir recibiendo este tipo de videos y ayudar a este canal en crecimiento.

SALUDOS Y HASTA LA PROXIMA