Proyecto FPMADdigital

Recursos digitales y multimedia para Formación Profesional

Comunidad de Madrid

Dirección General de Educación Secundaria, Formación Profesional y Régimen Especial

CONSEJERÍA DE EDUCACIÓN, UNIVERSIDADES, CIENCIA Y PORTAVOCÍA

Unión Europea

Fondo Social Europeo

"El FSE invierte en tu futuro"

Financiado como parte de la respuesta de la Unión a la pandemia de COVID-19

FPMADdigital
Recursos digitales y multimedia para Formación Profesional

CFGS Desarrollo de Aplicaciones Multiplataforma

módulo profesional

0486 - Acceso a Datos

unidad didáctica

05 Programación de componentes de acceso a datos

resultados de aprendizaje

06 Programa componentes de acceso a datos identificando las características que debe poseer un componente y utilizando herramientas de desarrollo.
03 Gestiona la persistencia de los datos identificando herramientas de mapeo objeto relacional (ORM) y desarrollando aplicaciones que las utilizan.
04 Desarrolla aplicaciones que gestionan la información almacenada en bases de datos objeto relacionales y orientadas a objetos valorando sus características y utilizando los mecanismos de acceso incorporados.

Comunidad de Madrid
Dirección General de Educación Secundaria, Formación Profesional y Régimen Especial
CONSEJERÍA DE EDUCACIÓN, UNIVERSIDADES, CIENCIA Y PORTAVOCÍA

Unión Europea
Fondo Social Europeo
"El FSE invierte en tu futuro"
Financiado como parte de la respuesta de la Unión a la pandemia de COVID-19

Resultados de aprendizaje y unidades didácticas

Resultados de aprendizaje y unidades didácticas

RESULTADOS DE APRENDIZAJE						UNIDAD DIDÁCTICA
1	2	3	4	5	6	
X						1.- Ficheros, colecciones y data frames
				X		2.- Manejo de XML
	X	X				3.- Procesamiento de BBDD Relacionales
			X			4.- Uso de BBDD NoSQL
		X	X		X	5.- Programación de componentes de acceso a datos

Unidades didácticas y materiales asociados

Unidades didácticas y materiales multimedia

RRAA						UUDD	Material Multimedia
1	2	3	4	5	6		
X						1.- Ficheros, colecciones y data frames	1.1 Contenidos básicos 1.2 Ejemplos aplicados
				X		2.- Manejo de XML	2.1 Contenidos básicos 2.2 Ejemplos aplicados
	X	X				3.- Procesamiento de BBDD Relacionales	3.1 Contenidos básicos 3.2 Ejemplos aplicados
			X			4.- Uso de BBDD NoSQL	4.1 Contenidos básicos 4.2 Ejemplos aplicados
		X	X		X	5.- Programación de componentes de acceso a datos	5.1 Contenidos básicos 5.2 Ejemplos aplicados

Repositorios de materiales y prácticas

Repositorio de materiales y prácticas

Todos los proyectos mostrados, así como otros materiales utilizados en las unidades didácticas los podrás encontrar completos en:

https://github.com/joseluisgs/FP-NextGen-AccesoDatos

Cualquier error o propuestas de mejora se publicarán en el repositorio indicado.
Gracias por tu colaboración.

Contenidos

1. **Desarrollo**
2. **Securización**
3. **Testeo**
4. **Despliegue**

Desarrollo

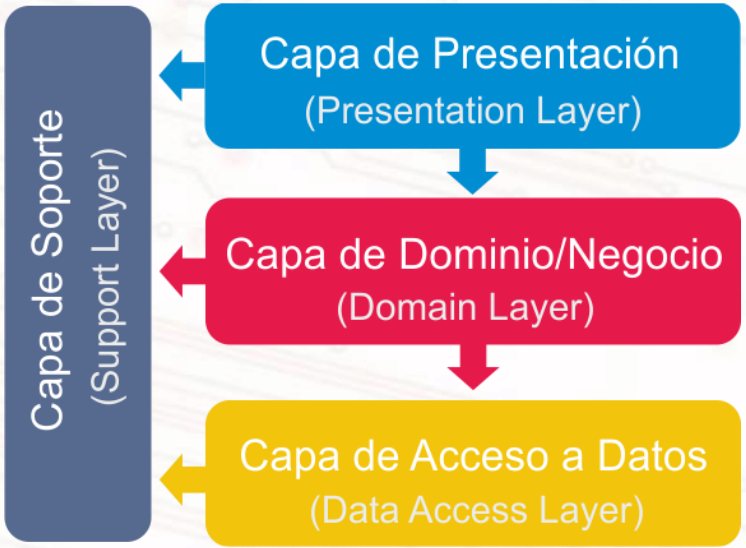
Componentes de acceso a datos

Llamamos **componentes de acceso a datos**, a cualquier componente software que explote una fuente de información, ya sea una base de datos, un fichero de intercambio, etc.

Es fundamental plantear el **diseño y desarrollo** teniendo en cuenta: ciclo de vida, securización, testeo y despliegue o difusión.

Aplicar **patrones** conocidos: DAO, Repositorio, DTO, etc.

Arquitectura en capas, ya sea en niveles o hexagonal.



Spring y Spring Boot

Spring es un marco de trabajo o Framework que nos permite desarrollar aplicaciones para JVM donde su punto fuerte es el contenedor de Inversión de Control y que tiene distintos módulos que facilitan los desarrollos: acceso a datos, seguridad, modelo vista controlador, orm, inyección de dependencias...

Spring Boot es una herramienta que nos permite crear un proyecto de Spring Framework, solo que Spring Boot elimina ciertas configuraciones repetitivas requeridas para desplegar la aplicación o proyecto.

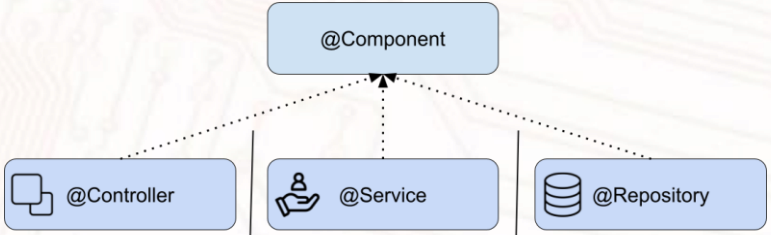
Spring Boot nos proporciona una serie de dependencias, llamadas **starters**, que podemos añadir a nuestro proyecto dependiendo de lo que necesitemos: crear un controlador REST, acceder a una base de datos usando JDBC o NoSQL. Una vez añadimos un starter, éste nos proporciona todas las dependencias que necesitamos, tanto de Spring como de terceros. Además, los starters vienen configurados con valores por defecto, que pretenden minimizar la necesidad de configuración a la hora de desarrollar.

Componentes

Controladores: se etiquetan como **@Controller** o en nuestro caso al ser una API REST como @RestController. Estos son los controladores que se encargan de recibir las peticiones de los usuarios y devolver respuestas.

Servicios: Se etiquetan como **@Service.** Se encargan de implementar la parte de negocio o infraestructura. En nuestro caso puede ser el sistema de almacenamiento o parte de la seguridad y perfiles de usuario.

Repositorios: Se etiquetan como **@Repository** e implementan la interfaz y operaciones de persistencia de la información. En nuestro caso, puede ser una base de datos o una API externa. Podemos extender de repositorios pre establecidos o diseñar el nuestro propio.

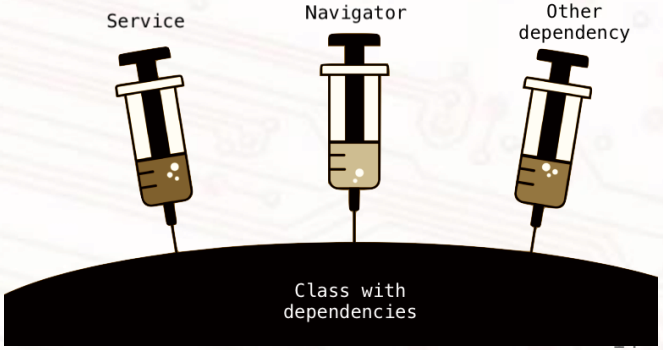


https://gustavopeiretti.com/spring-boot-componente-service/

Inyección de Dependencias

La **inyección de dependencias** (en inglés Dependency Injection, DI) es un patrón de diseño orientado a objetos, en el que se suministran objetos a una clase en lugar de ser la propia clase la que cree dichos objetos. Esos objetos cumplen contratos que necesitan nuestras clases para poder funcionar (de ahí el concepto de dependencia). Nuestras clases no crean los objetos que necesitan, sino que se los suministra otra clase 'contenedora' que inyectará la implementación deseada a nuestro contrato.

El contenedor **Spring IoC** (Inversión de Control) lee el elemento de configuración durante el tiempo de ejecución y luego ensambla el Bean a través de la configuración. La inyección de dependencia de Spring se puede lograr a través del constructor, el método Setter y el dominio de entidad. Podemos hacer uso de la anotación **@Autowired** para inyectar la dependencia en el contexto requerido. O si usamos ***Lombok***, podemos hacer uso de la anotación @Setter, @AllArgsConstructor, siempre y cuando declaremos como final las dependencias necesitadas.

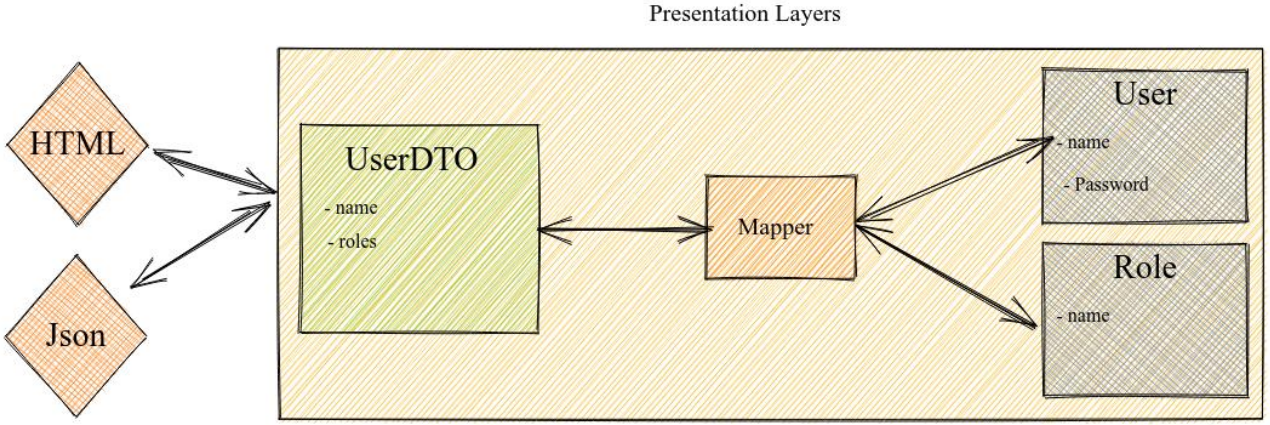


https://www.techmeet360.com/blog/introduction-to-dependency-injection-with-autofac/

DTO y Mappers

Usaremos los **DTO** (Data Transfer Objects), como objetos de transporte y con ellos facilitar la transición entre Request y objetos del modelo, y objetos de modelo y Responses. De esta manera podemos ensamblar y desensamblar los objetos de modelo y de transporte según nuestra necesidades.

Los **mapeadores** nos ayudarán en la misión de mapear los objetos de modelo a objetos de transporte y viceversa.



https://www.baeldung.com/java-dto-pattern

Spring Data

Spring Data es una librería de persistencia que nos permite acceder a bases de datos relacionales y no relacionales de forma sencilla obteniendo los repositorios necesarios para trabajar con nuestro modelo de dominio. Para ello Spring Data nos ofrece modelos de repositorio con funcionalidades, como pueden ser las operaciones de consulta, inserción, actualización y eliminación, así como las de paginación, ordenación o búsquedas. **Spring Data** permite realizar consultas usando JPQL, lenguaje nativo o en base a las propiedades del objeto. Los principales repositorios ofrecidos son:

- CrudRepository: tiene la mayoría de las funcionalidades CRUD.

- PagingAndSortingRepository: ofrece mecanismos de paginación, ordenación y búsqueda.

- JpaRepository: proporciona algunos métodos relacionados con JPA, como vaciar el contexto de persistencia y eliminar registros en un lote.

- MongoRepository: proporciona mecanismos para trabajar con MongoDB.

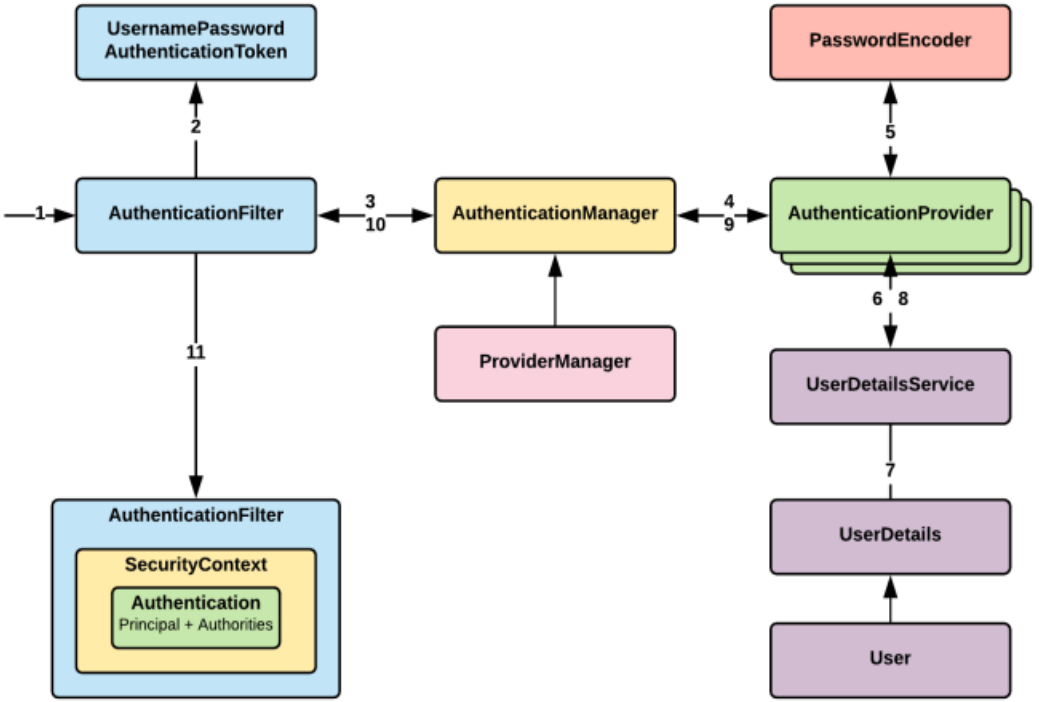
Securización

Spring Security

Spring Security es una librería de seguridad que nos permite controlar el acceso a nuestra aplicación permitiendo mecanismos de autenticación y autorización en base a roles.

Para ello haremos uso de UserDetailsService, un servicio que nos permitirá cargar datos específicos del usuario.

Además, actuará como middleware, analizando las rutas y con ellas a base de roles saber si se puede permitir el acceso a operar con ellas.



https://ducmanhphan.github.io/2019-02-09-The-mechanism-of-spring-security/

JWT

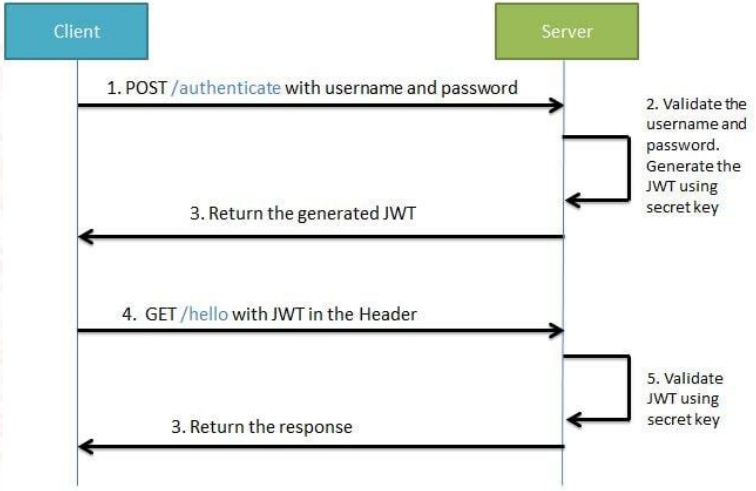
Spring Security ofrece una librería para generar y verificar JWT, entre otras. Gracias a ella podemos realizar el proceso de autenticación y autorización. **Json Web Token** es un estándar que define una forma auto contenida de transmitir información como JSON. Consta de tres partes separadas por puntos.

- Header: algoritmo (SHA256, HS512 …) y el tipo de token.

- Payload: contiene las propiedades (claims) del token.

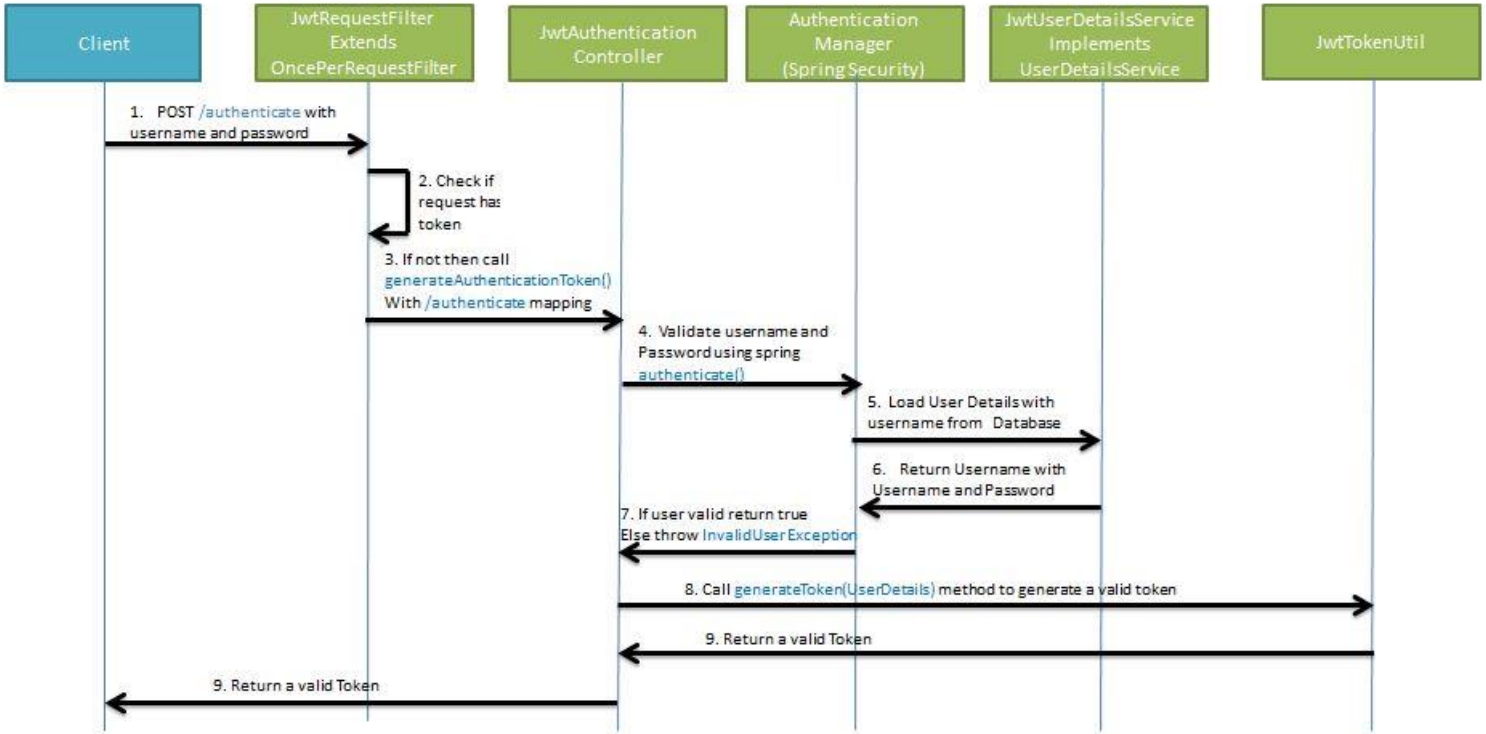
- Signature: header (codificado en base64), payload (codificado en base64), una clave, y todo firmado con el algoritmo del header.

- Claim: porción de información en el cuerpo del token.



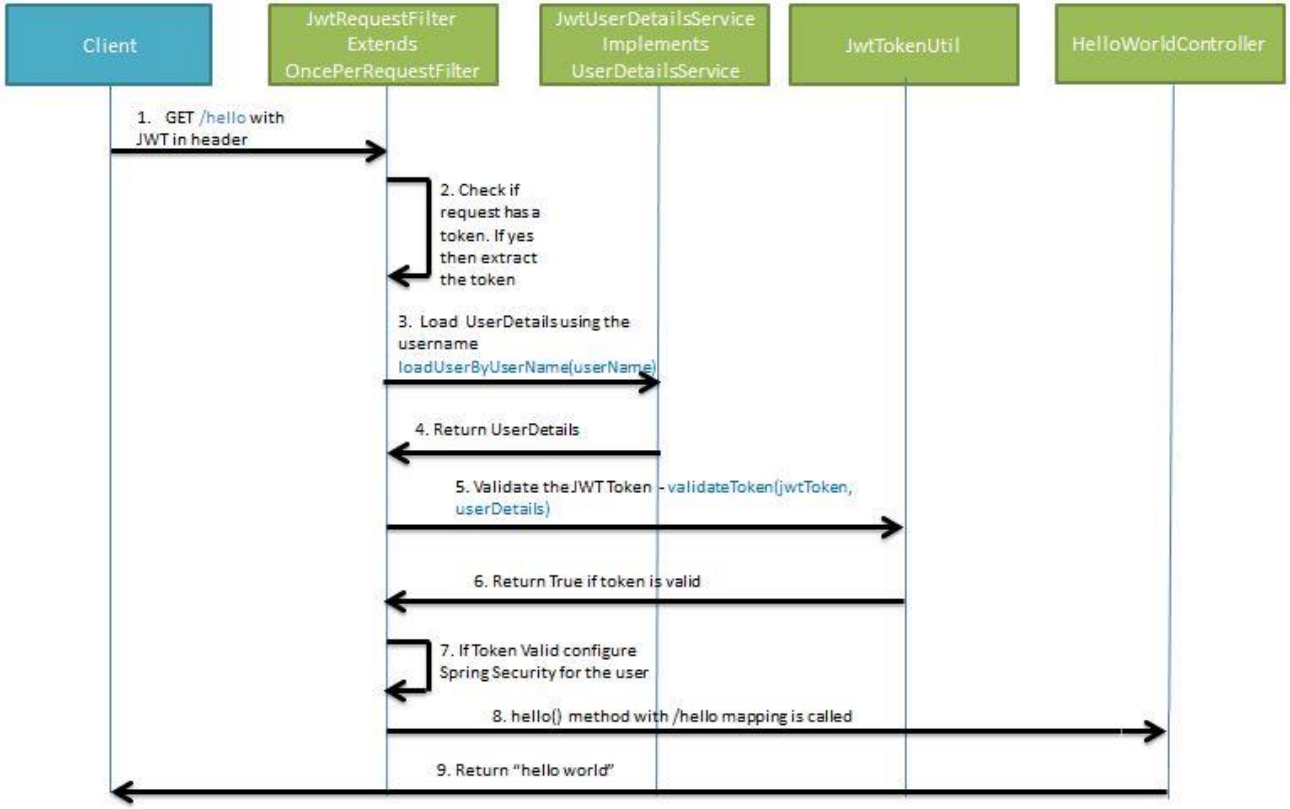
https://www.javainuse.com/spring/boot-jwt

Autenticación y Generación de JWT Token



https://www.javainuse.com/spring/boot-jwt

Autorización y Validación de JWT Token



https://www.javainuse.com/spring/boot-jwt

Testeo

JUnit y Mockito

JUnit es un framework que permite realizar la ejecución de clases Java de manera controlada, para poder evaluar si el funcionamiento de cada uno de los métodos de la clase se comporta como se espera. Es decir, nos permite crear pruebas unitarias. Para ello trabajamos con asercciones, donde para un valor dado comprobamos si coincide con lo que el método nos tiene que devolver.

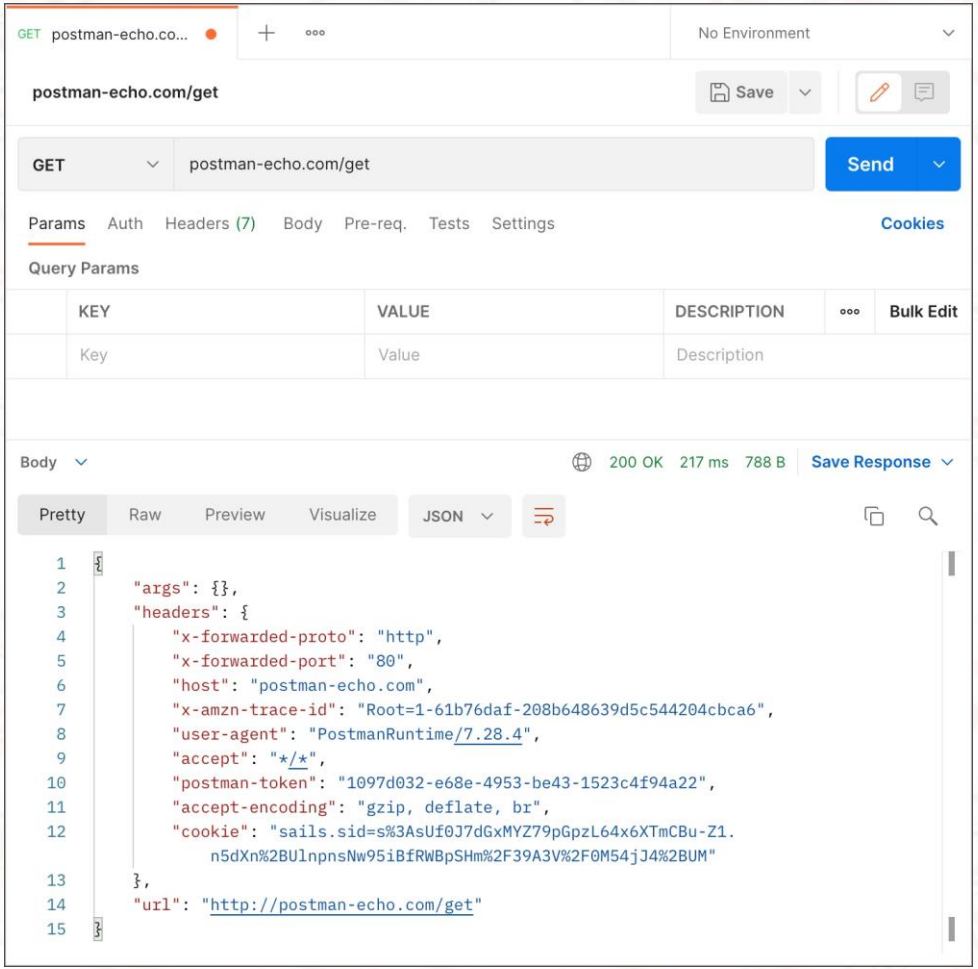
Mockito, nos permite hacer pruebas usando dobles o mocks, de esta manera podemos probar una clase creando un doble de su dependencia si esta está creada. Por ejemplo podemos crear un mock de un repositorio para ser usado en el test de un controlador.

A parte **Spring** nos ofrece gracias a su extensión de test poder simular el comportamiento JPA, con **@DataJpaTest** y con ello el de EntityManager y simular operaciones con JPA si acceder directamente a la Base de Datos. También con Spring **MockMvc** podemos mockear desde la propia aplicación las llamadas a la propia API y con ello analizar el funcionamiento de los controladores a nivel de integración con el resto del sistema. Esto nos hace que tengamos un cliente rest que interactúa con nuestro servicio, realizando las peticiones (resquest) para obtener los resultados de dicha petición al servicio (response).

Postman

Para probar nuestra API podemos usar **Postman**. Con ella podemos probar las llamadas y peticiones HTTP y analizar su comportamiento. Además, podemos documentarla y subirla a la nube para que pueda ser probada por otros usuarios.

Postman es muy útil a la hora de programar y hacer pruebas, puesto que nos ofrece la posibilidad de comprobar el correcto funcionamiento de nuestros desarrollos.

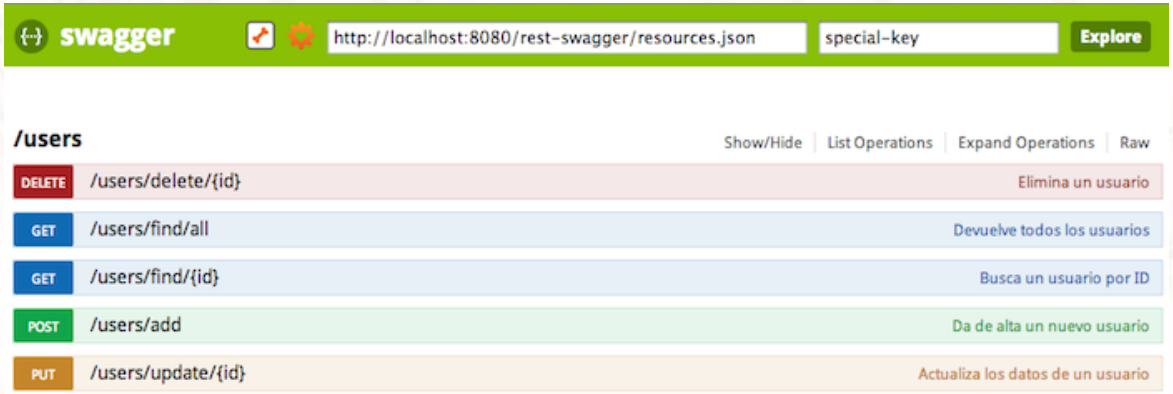


Despliegue

Swagger

Antes de desplegar es conveniente documentar nuestra API y/o componentes.

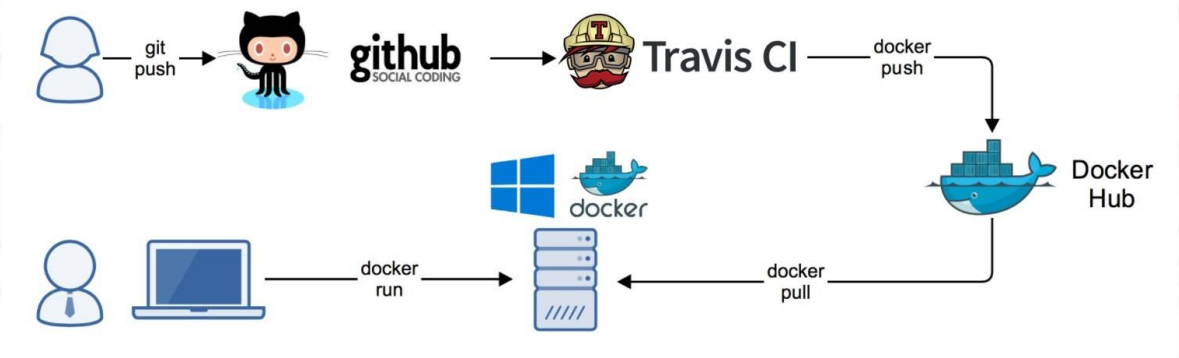
Podemos hacer uso de **Swagger**. Swagger es un conjunto de herramientas de software de código abierto para diseñar, construir, documentar, y utilizar servicios web RESTful. Además podemos usar su UI para testear nuestra API.



Docker y CI/CD

Para facilitar el despliegue de nuestra API podemos usar **Docker**. Podemos crear un contenedor de nuestra app y ejecutarlo. Además, podemos subir nuestra app a la nube para que pueda ser usada por otros usuarios usando Docker Hub.

Finalmente, gracias a este sistema podemos implementar **CI/CD** para desplegar bajo un **repositorio**, una vez se haga **cambios** y se pasen los **test** de manera automática **sea desplegada** en servicios como Heroku, Amazon AWS o similares **en la nube** en base a este contenedor desarrollado



https://www.linkedin.com/pulse/continuous-integration-delivery-linux-windows-docker-net-krastanov/

Conclusiones

¡Vamos con la práctica!

El buen código es su mejor documentación"

- Steve McConnell

Comunidad de Madrid
Dirección General de Educación Secundaria, Formación Profesional y Régimen Especial
CONSEJERÍA DE EDUCACIÓN, UNIVERSIDADES, CIENCIA Y PORTAVOCÍA

Unión Europea
Fondo Social Europeo
"El FSE invierte en tu futuro"
Financiado como parte de la respuesta de la Unión a la pandemia de COVID-19