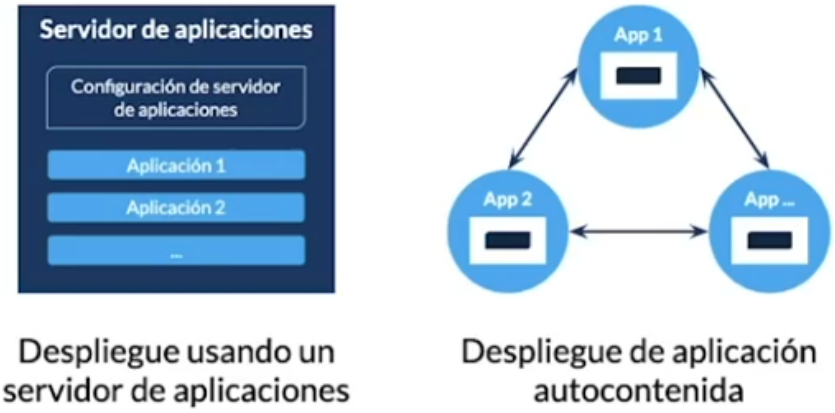
Framework Spring

¿Cómo funcionan las aplicaciones autocontenidas?



Despliegue usando un servidor de aplicaciones

Despliegue de aplicación autocontenida

> Aplicación autocontenida: cada aplicación tiene su propio servidor de aplicaciones con su propia configuración

Spring boot

> Es el proyecto de Spring para aplicaciones autocontenidas

- Nos centramos en el desarrollo
- Funciona por defecto con Tomcat
- Incluye gestor de dependencias como Gradle o Maven

¿Qué es JPA?

Jpa es una especificación de Java, standar, para un framework ORM. Quiere decir que son uan serie de reglas que Java define para que cualquier framework que quierea interactura con la BD de Java, tenga que seguir.

Los frameworks mas populares de Java para este fin son:

- Hibernate
- TopLink
- EclipseLink

- ObjectDB

Anotaciones JPA

JPA utiliza anotaciones para conectar clases a tablas de la BD y asi evitar hacerlo de manera nativa con SQL.

- `@Entity`. Indica a una clase de java que esta representando una tabla de nuestra BD.
- `@Table`. Recibe el nombre de la tabla a la cual esta mapeando la clase.
- `@column`. Se le pone a los atributos de la clase, no es obligatoria, se indica sólo cuando el nombre de la columna es diferente al nombre del atributo de la tabla.
- `@id` and `@EmbededID`. Es el atributo como clave primaria de la tabla dentro de la clase. `@id` se utiliza cuando es clave primaria sencilla y `@EmbededID` cuando es una clave primaria compuesta.
- `@GeneratedValue`. Permite generar automáticamente generar valores para las clases primarias en nuestras clases
- `@OneToMany` and `@ManyToOne`. Representar relaciones

Spring Data

Spring Data NO es una implementacion de JPA, sino mas bien es un proyecto que usa JPA para ofrecer funcionalidaes extra en la gestion de tareas desde JAVA a las base de datos.

Spring Data internamente tiene varios subproyectos, entre ellos: **Spring Data JPA y Spring Data JDBC,** para conectarnos a BD relacionales (SQL). Spring Data MongoDB y Spring Data Cassandra, son proyectos para conectarnos a BD no relacionales.

La tarea principal de Spring Data es optimizar tareas repitivas.

Spring data nos provee de respositorios sin codigo, nos permiten hacer todo tipo de operaciones en BD (CRUD) sin utilizar una linea de código.

También nos provee de auditorías transparentes, por ello, posee un motor de auditorias que nos permite saber cuadno se insertó un registro, cuando se borró, cuando se actualizo en la BD, etc.

Implementación en el proyecto Market.

Se busca en MAVEN el repositorio Spring Boot Starter Data JPA, se copia el group y el name en las dependencias del archivo build.gradle de nuestro proyecto quedando de la siguiente manera.

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

  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
  </dependency>

  <dependency>
    <groupId>org.postgresql</groupId>
    <artifactId>postgresql</artifactId>
  </dependency>

  <dependency>
    <groupId>org.mapstruct</groupId>
    <artifactId>mapstruct</artifactId>
    <version>${org.mapstruct.version}</version>
  </dependency>

  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
  </dependency>
</dependencies>
```

Patrón Data Mapper

> Consiste en convertir o traducir dos objetos que pueden hacer la misma labor

Nos ayuda a:

- No exponer la base de datos en el API
- Desacoplar nuestra API a una base de datos puntual
- No tener campos innecesarios en el API
- Sin mezclar idiomas en el dominio

Inyección de dependencias

> Consiste en pasar la dependencia a la clase que la va a usar, en lugar de crearla internamente en la clase, con el fin de no acoplar la clase a la implementación que está utilizando

Inversión de control (IoC)

> Un fragmento tiene el control sobre los objetos. Spring tiene un contenedor de inversión de control donde se administran y crean instancias de objetos.

Anotación @Autowired

> Anotación utilizada para la inyección de dependencias. La necesidad de instanciar objetos hace que el autowired sea necesario, debido a que si al declarar un objeto y no se inicializa, la aplicación producirá un nullpointer.

```
@Autowired
private GastoCrudRepository gastoRepository;

@Autowired
private SpendMapper spendMapper;
```

Uso de diferentes anotaciones

- La anotación @Service hace de Intermediario entre el controlador de la API y el repositorio
- Nuestra API se expone mediante la anotación @RestController

```
@RestController
@RequestMapping("/spend")
public class SpendController {

}
```

- Los métodos los expone con @GetMappings, @PostMapping o @DeleteMapping
- La anotación *@RequestBody* se utiliza a la hora de mandar el cuerpo de la petición (objeto)

```
@PostMapping("/save")
    public Spend save(@RequestBody Spend spend) {
        return spendService.save(spend);
    }
```