SPRING BOOT

ADOLFO SANZ DE DIEGO

PRONOIDE

2 ACERCA DE

2.1 AUTOR

- **Adolfo Sanz De Diego**
  - Blog: asanzdiego.com
  - Correo: asanzdiego@gmail.com
  - GitHub: github.com/asanzdiego
  - Twitter: twitter.com/asanzdiego
  - LinkedIn: in/asanzdiego
  - SlideShare: slideshare.net/asanzdiego

2.2 LICENCIA

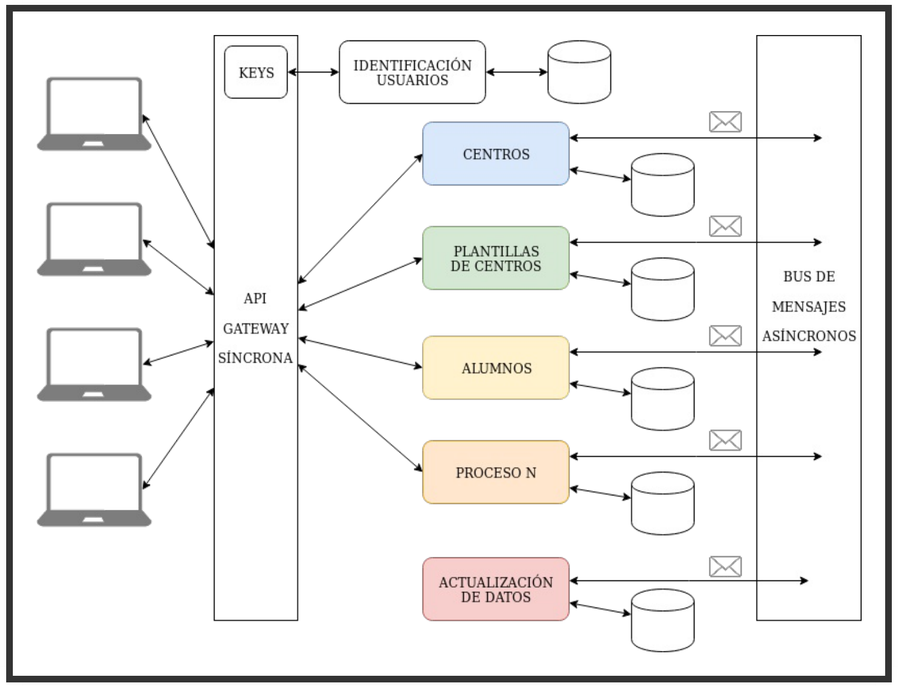
- **Esta obra está bajo una licencia:**
  - Creative Commons Reconocimiento-CompartirIgual 3.0

3 MICROSERVICIOS

3.1 INTRODUCCIÓN

- Son **son pequeños servicios autónomos que se comunican con APIs ligeros**, típicamente con APIs REST.
- Encierran **toda la lógica necesaria para cubrir una funcionalidad completa**, desde el API que expone que puede hacer hasta el acceso a la base de datos.
- La arquitectura de microservicios, se forma con microservicios que se ejecutan y despliegan independientemente, comunicándose con la parte cliente o si es necesario entre ellas **síncronamente a través de APIs o asíncronamente a través de servicios de mensajería**.

3.2 EJEMPLO



Arquitectura Microservicios

3.3 VENTAJAS

- **Desarrollos pequeños y rápidos**.
- **Escalado más granular**, permitiendo aprovechar de forma más eficiente los recursos.
- Aprovechamiento de los puntos fuertes de cada lenguaje de programación, ya que **permite el uso de distintas tecnologías** para la implementación de cada microservicio.

3.4 DESVENTAJAS

- **Monitorización** más compleja.
- **Despliegues** más complejos.
- Surgen problemas con la **compatibilidad entre versiones**.

4 REST

4.1 INTRODUCCIÓN

- Los servicios REST son **servicios basados en recursos**, montados sobre HTTP.
- Los recursos son **representados por una URL** (Uniform Resource Locator).
- Los recursos se pueden **acceder o modificar mediante los métodos del protocolo HTTP** (POST, GET, PUT, DELETE).

4.2 VENTAJAS

- Es **más sencillo** (tanto la API como la implementación).
- Es **más rápido** (peticiones más ligeras que se pueden cachear).
- Es **multiformato** (HTML, XML, JSON, etc.).

4.3 EJEMPLO

- **GET** a http://myhost.com/person
  - Devuelve todas las personas
- **POST** a http://myhost.com/person
  - Crear una nueva persona
- **GET** a http://myhost.com/person/123
  - Devuelve la persona con id=123
- **PUT** a http://myhost.com/person/123
  - Actualiza la persona con id=123
- **DELETE** a http://myhost.com/person/123
  - Borra la persona con id=123

4.4 ESTADOS HTTP

- 200 OK
- 201 Created
- 202 Accepted
- 301 Moved Permanently
- 400 Bad Request
- 401 Unauthorized
- 402 Payment Required
- 403 Forbidden
- 404 Not Found
- 405 Method Not Allowed
- 500 Internal Server Error
- 501 Not Implemented

4.5 JSON

- Acrónimo de **JavaScript Object Notation**.
- Sirve como **formato ligero** para el intercambio de datos.
- Su **simplicidad** ha generalizado su uso, especialmente como alternativa al XML.

4.6 EJEMPLO

```
{
    curso: "Spring Boot y Angular",
    profesor: "Adolfo",
    participantes: [
        { nombre: "Asunción", edad: 35 },
        { nombre: "Miguel", edad: 31 },
        { nombre: "Miky", edad: 27 },
        { nombre: "Alberto", edad: 25 }
    ]
}
```

5 SPRING

5.1 INTRODUCCIÓN

- **Framework de desarrollo** de aplicaciones java.
- Se encarga de la instanciación de POJOS mediante la **Inversión de Control** y la **Inyección de Dependencias**.
- Ha ido evolucionando y ahora tiene **muchos sub proyectos**: https://spring.io/projects
- **No es intrusivo**: puedes usar lo que el te proporciona u otras implementaciones.

5.2 DEFINICIONES

- **Inversión de control**:
  - Patrón de diseño que **delega la creación de objetos en un contenedor**.
- **Inyección de dependencias**:
  - Patrón de diseño que **permite desacoplar la Interfaz de su implementación**.

5.3 EJEMPLO

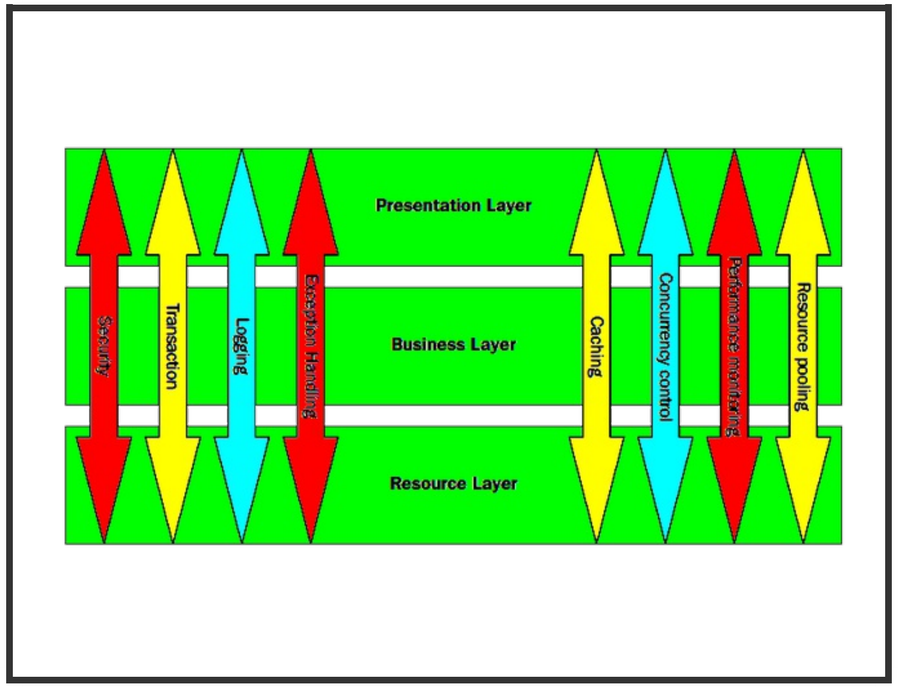
```
interface PersonaDao {
    void insertar(Persona persona);
}
class GestorPersonas {
    private PersonaDao dao;
    // Inyección por constructor
    public GestorPersonas(PersonaDao dao) {
        this.dao = dao;
    }
    // Inyección por setter
    public setDao(PersonaDao dao) {
        this.dao = dao;
    }
    public void alta(Persona persona) {
        dao.insertar(persona);
    }
}
```

6 SPRING AOP

6.1 INTRODUCCIÓN

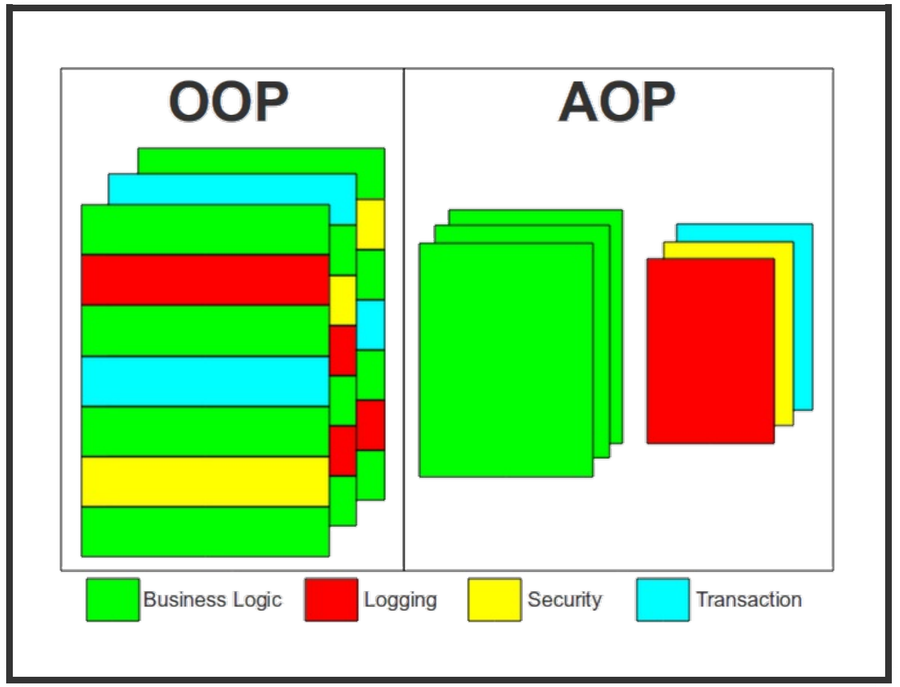
- Aunque es bastante transparente para el programador, **Spring hace uso intensivo de programación orientada a aspectos** cada vez que usamos una de sus anotaciones.
- Los programas tienen muchas funcionalidades trasversales, **que afectan a muchos los módulos de la aplicación**, como el logging, la transaccionalidad o la seguridad.
- La AOP, o **programación orientada a aspectos**, se encarga de extraer dichas funcionalidades transversales de los módulos de la aplicación.

6.2 FUNCIONALIDADES TRASVERSALES



Funcionalidades trasversales

6.3 AOP OVERVIEW



AOP Overview

7 SPRING MVC

7.1 INTRODUCCIÓN

- Framework de desarrollo de aplicaciones web que implementa el **Modelo Vista Controlador**.
- Se basa en anotaciones, las más importantes:
  - **@Controller**: indica que una clase es un controlador.
  - **@RequestMapping**: mapea un método de clase a una request en función de la URL y el método HTTP.
  - **@PathVariable**: obtiene el path de la URL.
  - **@RequestParam**: obtiene parámetros de la URL.
  - **@RequestBody**: obtiene el body de peticiones POST y PUT.
  - **@ResponseBody**: devuelve el body en el response.
  - **@SessionAttribute**: obtiene información de la sesión.

7.2 EJEMPLO

```
// http://myhost.com/saludar/adolfo?edad=40
// en el body del POST {"apellido1":"sanz", "apellido2":"diego"}
@Controller
public class SaludarController {
    @ResponseBody
    @RequestMapping("saludar/{nombre}", method=RequestMethod.POST)
    public Saludo saludar(@PathVariable("nombre") String nombre,
        @RequestParam("edad") Integer edad, @RequestBody Apellidos a,
        @SessionAttribute Login login){
        Saludo saludo = new Saludo()
        saludo.setNombre(nombre);
        saludo.setApellidos(a.getApellido1(), a.getApellido2());
        saludo.setEdad(edad);
        saludo.setLogin(login);
        return saludo;
    }
}
```

7.3 SPRING REST

- **@RestController**: junta las anotaciones @Controller y @ResponseBody.
- **@ResponseEntity**: para devolver el estado del response.

7.4 EJEMPLO

```
// http://myhost.com/saludar/adolfo?edad=40
// en el body del POST {"apellido1":"sanz", "apellido2":"diego"}
@RestController
public class SaludarController {
    @RequestMapping("saludar/{nombre}", method=RequestMethod.POST)
    public ResponseEntity<Saludo> saludar(
        @PathVariable("nombre") String nombre,
        @RequestParam("edad") String edad, @RequestBody Apellidos a,
        @SessionAttribute Login login){
        if (login == null) {
            return new ResponseEntity<Saludo>(HttpStatus.UNAUTHORIZED);
        }
        Saludo saludo = new Saludo(nombre,
            a.getApellido1(), a.getApellido2(), edad, login);
        return new ResponseEntity<Saludo>(saludo);
    }
}
```

8 SPRING JPA

8.1 INTRODUCCIÓN

- **Framework de persistencia** que se basa en extender interfaces que otorgan funcionalidades a las clases de acceso a datos.
- Lo más interesante es que no es necesario definir implementaciones pues estas se crean **en tiempo de ejecución**.

8.2 EJEMPLO

```java
public interface EntityRepository extends JpaRepository<Entity, Long> {

    // EntityRepository tiene ya estos métodos por heredar de JpaRepository

    Entity save(Entity entity);

    Optional<TEntity> findById(Long primaryKey);

    Iterable<TEntity> findAll();

    long count();

    void delete(Entity entity);

    boolean existsById(Long primaryKey);
}
```

8.3 QUERYS PERSONALIZADAS

- Se pueden crear métodos siguiendo las siguientes reglas:
  - Prefijo del nombre del método **findBy** para busquedas y ***countBy*** para conteo de coincidencias.
  - Seguido del nombre de los campos de búsqueda concatenados por los operadores correspondientes: And, Or, Between, LessThan, LessThanEqual, GreaterThan, GreaterThanEqual, After, Before, IsNull, IsNotNull, Like, NotLike, StartingWith, EndingWith, Containing, OrderBy, Not, In, NotIn, True, False, IgnoreCase
- Tambien se pueden definir consultas personalizadas con **JPQL**.

8.4 EJEMPLOS

```
List<Person> findByLastname(String lastname);

List<Person> findByEmailAndLastname(String email, String lastname);

List<Person> findDistinctByEmailOrLastname(String email, String lastname);

List<Person> findByLastnameLikeIgnoreCase(String lastname);

List<Person> findByLastnameOrderByFirstnameAsc(String lastname);

List<Person> findByLastnameOrderByFirstnameDesc(String lastname);
```

8.5 EJEMPLOS JPQL

```
@Query("from Country c where lower(c.name) like lower(:name)")
List<Country> getByNameWithQuery(String name);

@Query("from Country c where lower(c.name) like lower(:name)")
Country findByNameWithQuery(@Param("name") String name);

@Query("select case when (count(c) > 0) then true else false end"
    + " from Country c where c.name = :name)")
boolean exists(String name);
```

8.6 PAGINACIÓN Y ORDENACIÓN

- A las consultas se puede añadir un último parametro de tipo **Pageable** o **Sort** que permite definir el criterio de paginación o de ordenación.

```
Page<Person> findByLastname(String lastname, Pageable pageable);
```

8.7 INSERCIÓN / ACTUALIZACIÓN

- Se **devolverá void o un entero** (int/Integer) que contendrá el número de objetos modificados o eliminados.
- El método **deberá ser transaccional** o bien ser invocado desde otro que sí lo sea.
- Se pueden definir **con JPQL, siempre que método esté anotado con @Modifying** si no Spring interpretará que se trata de una select y la ejecutará como tal.

8.8 EJEMPLOS

```java
@Transactional
int deleteByName(String name);
```

```java
@Transactional
@Modifying
@Query("UPDATE Country set creation = :creation)"
int updateCreation(Calendar creation);
```

9 SPRING SECURITY

9.1 INTRODUCCIÓN

- Basada en **otorgar accesos**.
- Se pueden aplicar **distintos niveles de acceso** a los contenidos.
- **Fácil de migrar** entre servidores de aplicaciones.

9.2 PROTECCION DE RECURSOS

- Permite configurar las seguridad sobre las peticiones http.

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    http
        .authorizeRequests()
            .antMatchers("/all/*").permitAll()
            .antMatchers("/**").access(
                "hasAnyRole('AGENTE_ESPECIAL','DIRECTOR')");
}
```

9.3 SEGURIDAD DE MÉTODOS

```
@Secured("ROLE_AGENTE_ESPECIAL,ROLE_DIRECTOR")
void clasificar(Expediente expediente);
```

```
@PreAuthorize("hasRole('ROLE_DIRECTOR') " +
    "or #expediente.investigador == authentication.name")
void desclasificar(Expediente expediente);
```

```
@PostAuthorize("hasRole('ROLE_DIRECTOR') " +
    "or returnObject.investigador == authentication.name")
Expediente mostrar(Long id);
```

```
@PostFilter("(hasRole('ROLE_AGENTE') and not filterObject.clasificado) " +
    "or (hasAnyRole('ROLE_AGENTE_ESPECIAL','ROLE_DIRECTOR') " +
    "and not filterObject.informe.contains(principal.username))")
List<Expediente> listarTodos();
```

10 SPRING BOOT

10.1 INTRODUCCION

- Framework orientado a la **construcción de proyectos Spring**.
- Basado en **Convention-Over-Configuration** minimiza la configuración.
- Proporciona **Starters** para simplificar las dependencias.

10.2 DEPENDENCIA PRINCIPAL

```
<project>
    ...
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.1.4.RELEASE</version>
    </parent>
    ...
</project>
```

10.3 DEPENDENCIA WEB

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

10.4 CONFIGUTACIÓN INICIAL

- **@SpringBootApplication**: es la unión de:
  - **@Configuration**: clase como origen de definiciones de Beans.
  - **@ComponentScan**: buscar otras clases con anotaciones como @RestController.
  - **@EnableAutoConfiguration**: incluir configuración por defecto.

```
@SpringBootApplication
public class HolaMundoApplication {
    ...
}
```

10.5 JAR EJECUTABLE

```java
@SpringBootApplication
public class HolaMundoApplication {
    public static void main(String[] args) {
        SpringApplication.run(HolaMundoApplication.class, args);
    }
}
```

10.6 PRIMER RESTCONTROLLER

```java
@RestController
public class HolaMundoController {
    @RequestMapping("/")
    public String holaMundo() {
        return "¡Hola Mundo!";
    }
}
```

10.7 EJECUCIÓN

```
mvn spring-boot:run
```

```
mvn package
java -jar nombre-del-jar.jar
```

11 BIBLIOGRAFÍA

11.1 OFICIAL DE SPRING

- Building an Application with Spring Boot
- Building a RESTful Web Service
- Accessing Data with JPA
- Securing a Web Application
- Accessing data with MySQL
- Enabling Cross Origin Requests