



Desarrollo de aplicaciones con Spring **Spring**

Jesús Bernal Bermúdez

Desarrollo de Aplicaciones con Spring

Objetivo

Dar los conocimientos
y prácticas necesarias
para que un equipo
pueda desarrollar una
aplicación Back-End
con Spring

Temario

Cronograma

Introducción (4h-d16)

DevOps.

Entorno: IntelliJ,
Java 11, Maven.

Java y
Programación
funcional.

Test Unitarios &
Test de
Integración
(JUnit).

Git. Workflow

Introducción a Spring (4h-d18)

Arquitectura Web: 3
capas VS Hexagonal.

Spring Boot, Spring
Framework, Spring
Data, Spring
Security & Spring
Cloud.

Spring MVC VS
Spring Reactive.

Inyección de
dependencias.

Test con Spring:
Unitarios, Mocks
(Mockito) & Test de
Integración.

Persistencia (6h-d22&d24)

Spring Data: JPA.
Patrón DAO.
BBDD
relacionales.

Proyecto Lombok.
Modelo de
Entidades: diseño
de BD.

Queries por
nombres de
métodos.

JPQL.

Arquitectura:
persistencia

API Rest (6h-d24&d29)

Construcción de
API's.

Tratamiento de
errores.

Validación de
datos.

Spring docs.
OpenAPI.

Cliente de API
externo.

Seguridad (4h-d30)

Basic Auth.

JWT.

Proyecto final.

¿Qué es DevOps?

Development – Operations & QA

Prácticas para unificar el desarrollo de software y la operación de negocio

El objetivo es integrar los procesos y las herramientas para optimizar la entrega continua de software de calidad, siguiendo los objetivos empresariales

DevOps

Prácticas

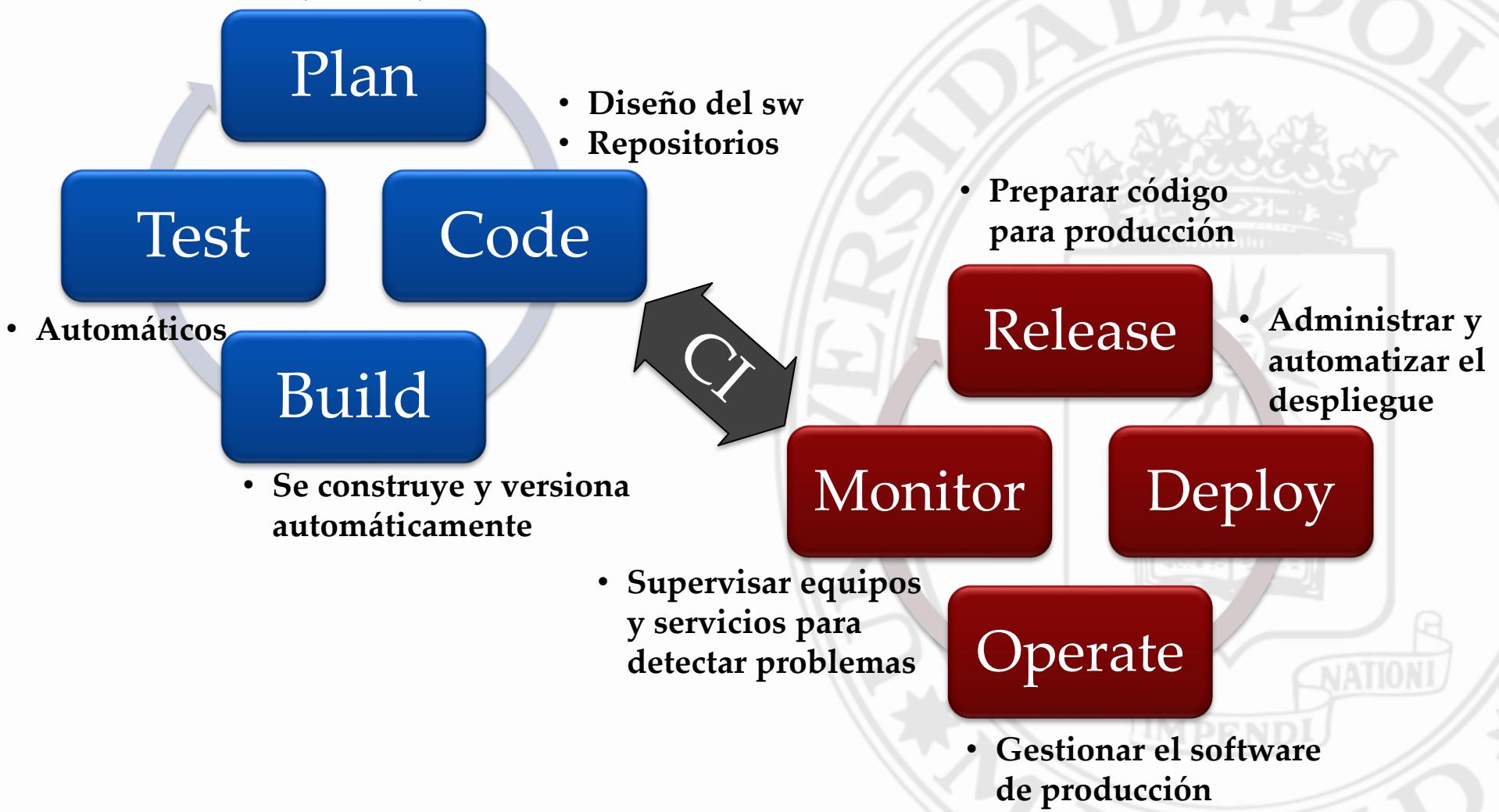
- Automatización de todos los procesos:
 - Integración Continua (*CI*).
 - Entrega/Despliegue Continuo (*CD*).
- Ciclos de desarrollo cortos.
- Microservicios.
- Lanzamientos confiables: sistema seguro, estable y rápido.
- Sistemas monitorizados.
- Comunicación y colaboración.

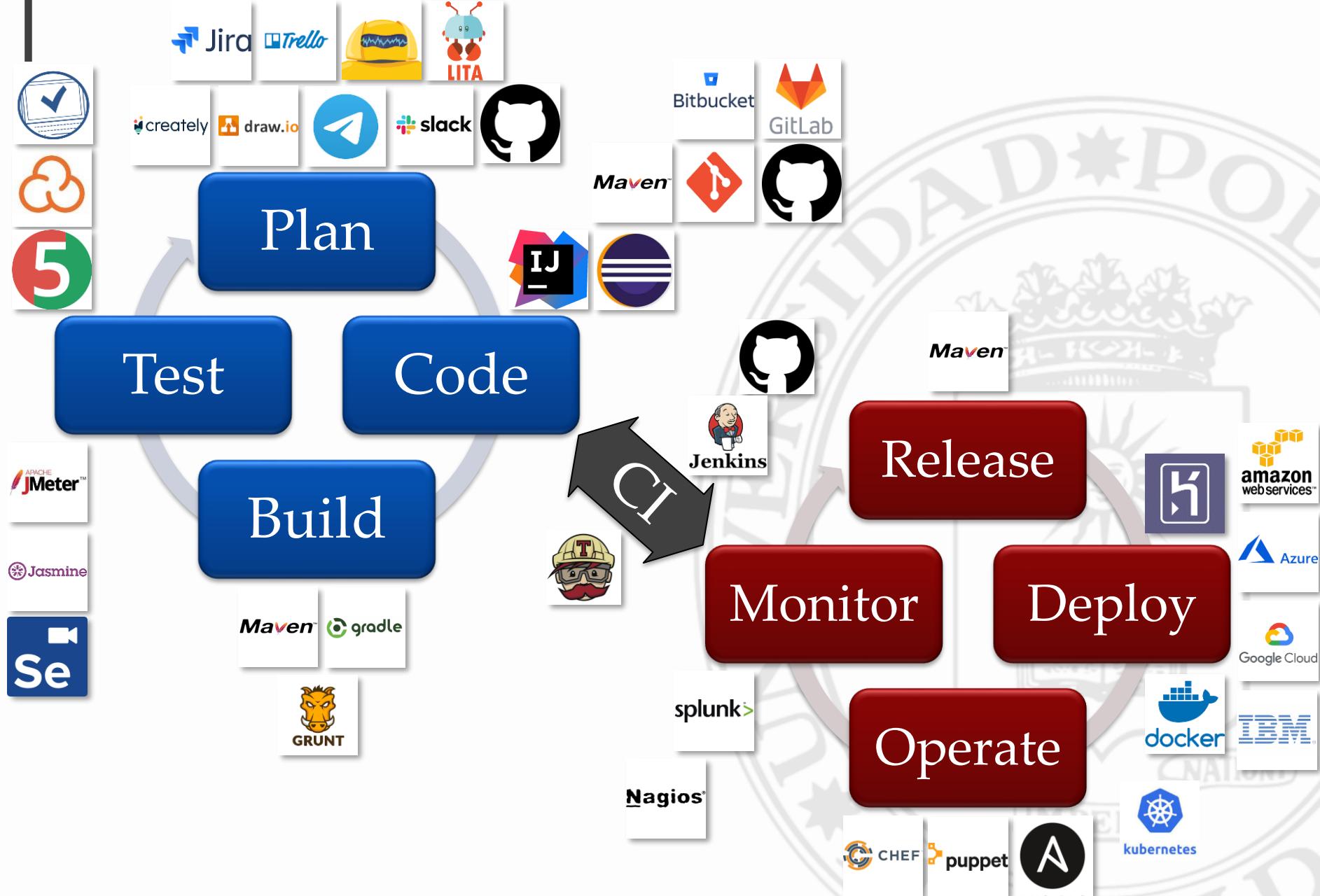
¿Qué es un Ecosistema Software?

Ecosistema Software

Se define como un espacio de trabajo donde un conjunto de herramientas interactúan y funcionan como una unidad para la colaboración, desarrollo, despliegue y supervisión del software en todas sus fases

- Valor comercial
- Gestión del proyecto
- XP, Scrum, Kanban...





¿Qué es Apache Maven?

Es una herramienta de gestión y construcción de proyectos de software con Java.

Identificar el componente

Resolver dependencias

Test

Empaquetar

...

Se basa en un modelo de objetos del proyecto (*POM*)

Fichero *pom.xml*

Se sitúa en la raíz del proyecto.



Maven Conceptos

Artefacto

Componente
software

Unidad mínima con
la que trabaja
Maven

Coordenadas

Sistema donde se determina de forma única a cada uno de los artefactos en Internet

Group Id

- Identificación del grupo.
Normalmente se utiliza el nombre del dominio, al revés: *es.upm.miw*

Artifact Id

- Identificación del artefacto: *devops*

Version

- 1.0.0-SNAPSHOT*
- 1.3.4-RC* (Release Candidate)
- 1.4.5-Release*

Empaquetado

Tipo de artefacto

JAR, POM

WAR, EAR, RAR...

Maven

Comandos

clean

- Elimina los ficheros generados en construcciones anteriores

validate

- Valida el proyecto si es correcto

compile

- Genera los ficheros *.class compilando los fuentes *.java

test

- Ejecuta los test unitarios (*Test) existentes

package

- Genera el empaquetado final (jar, war...)

integration-test

- Despliega el paquete y ejecuta los test de integración (*IT)

verify

- Verificar que el paquete cumpla los criterios de calidad

install

- Instala el paquete en el equipo local

deploy

- Instala el paquete en el repositorio remoto (lib)

```
C:\work-spaces\devops>mvn -v
C:\work-spaces\devops>mvn -help
C:\work-spaces\devops>mvn clean package
C:\work-spaces\devops>mvn clean -Dmaven.test.skip=true package
```

Maven Plugin

Plugin

jacoco

- Es una tarea específica, más pequeña que una fase de construcción, que contribuye a la construcción y gestión del proyecto.

sonar:sonar

- Genera un informe de cobertura de los test
- Se conecta con *Sonarcloud* para inspeccionar el código

spring-boot:run

- Arranca una aplicación en local realizada con *Spring Boot*

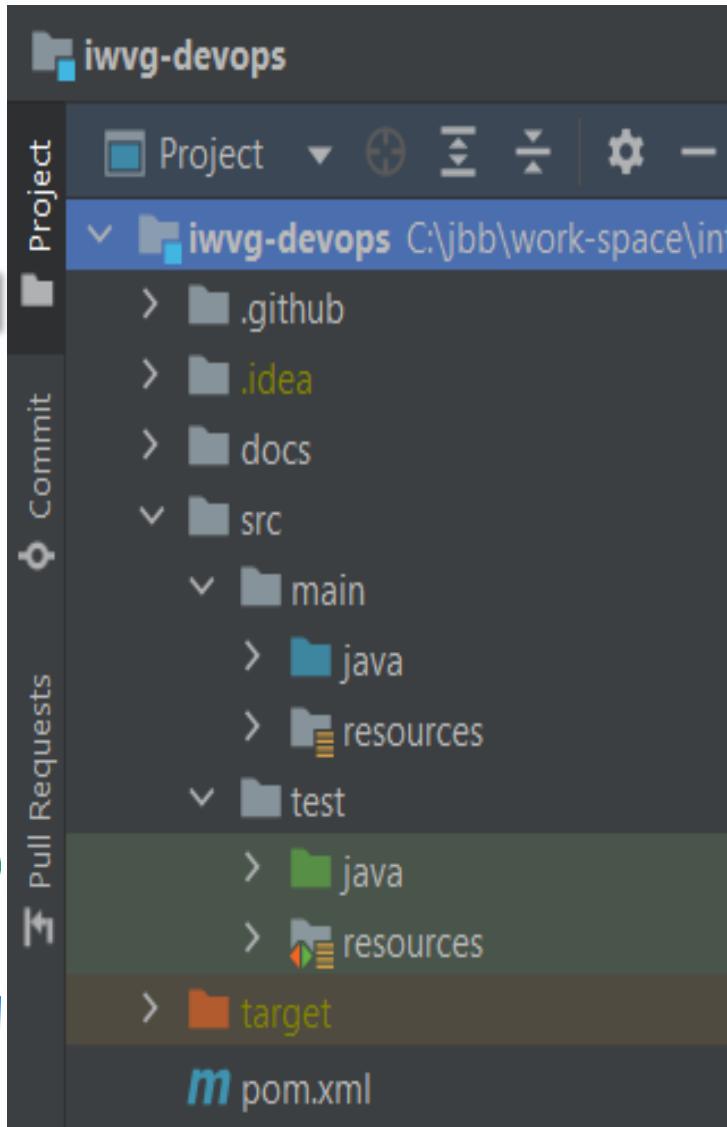
```
C:\work-spaces\betca-tpv-spring>mvn spring-boot:run
[INFO] Scanning for projects...
[INFO] -----
[INFO] < es.upm.miw:betca-tpv-spring >-----
[INFO] Building es.upm.miw.betca-tpv-spring 2.4.0-SNAPSHOT
...
...
```

Maven POM.XML

```
m iwg-devops x
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <project
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xmlns="http://maven.apache.org/POM/4.0.0"
5     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
6     <modelVersion>4.0.0</modelVersion>
7
8     <parent...>
14
15     <artifactId>iwg-devops</artifactId>
16     <groupId>es.upm.miw</groupId>
17     <version>1.4.0-SNAPSHOT</version>
18     <packaging>jar</packaging>
19
20     <name>${project.groupId}.${project.artifactId}</name>
21     <description>DevOps</description>
22     <url>http://github.com/miw-upm/${project.artifactId}</url>
23
24     <licenses...>
30
31     <developers...>
43
44     <properties...>
66
67     <dependencies...>
110    <build>
111        <plugins...>
178    </build>
179 </project>
```

Maven

Estructura del proyecto



Instalación externa Maven

Bajarse y descomprimir:

- *-bin.zip

Variable de entorno

- M2_HOME** (al raíz).
- PATH** (al bin).

Se necesita tener instalado el **OpenJDK** o **JDK**. Variables de entorno:

- JAVA_HOME**: Al raíz.
- PATH**: Al bin

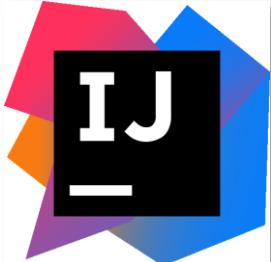
Maven IDE

IntelliJ IDEA (JetBrains)

- IDE para Java
- Tiene una versión gratuita: *Community Edition*.

Crear un proyecto nuevo

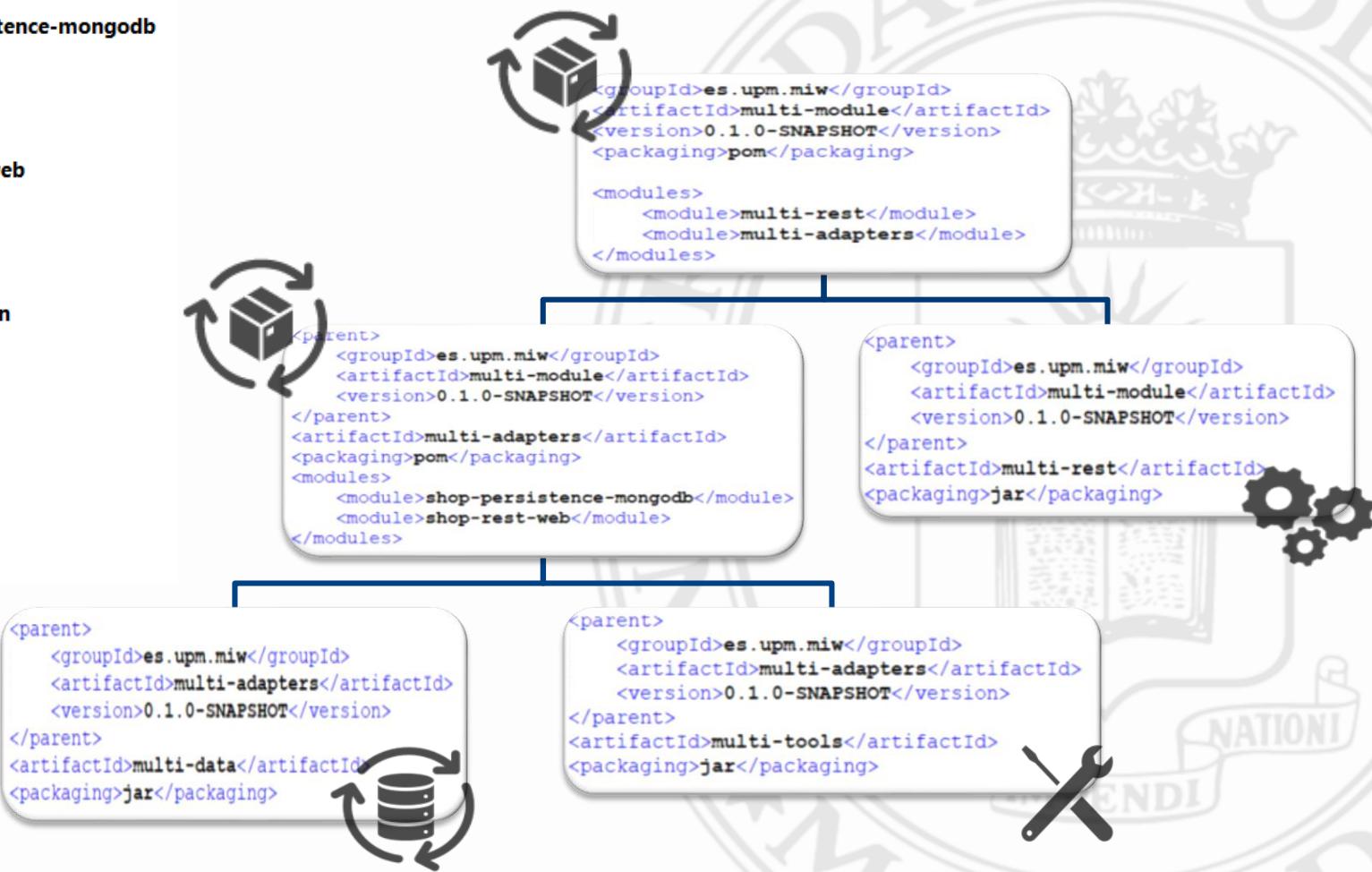
- Partir de un *Template*
- <https://github.com/miw-upm/iwvg-devops/docs>
 - Descomprimirla y cambiar coordenadas
- *Importar desde IntelliJ*, indicando que es de tipo *Maven*.



Maven Multi-módulo

```

apaw-shop-hexagonal
  > .idea
  > docs
  > shop-adapters
    > shop-persistence-mongodb
      > .idea
      > src
      m pom.xml
    > shop-rest-web
      > src
      m pom.xml
      m pom.xml
  > shop-application
    > src
    m pom.xml
  > shop-domain
    .gitignore
    .travis.yml
    LICENSE.md
    m pom.xml
  
```



Maven

Características

Repositorio

- Estructura de directorios y archivos que usa Maven para almacenar, organizar y recuperar artefactos.
- Existen repositorios *locales, privados y remotos*
 - Repositorio central de Maven: <https://mvnrepository.com/>
 - Repositorio local: %User%/.m2/repository.

Perfiles

- Permite cambiar la configuración de la aplicación dependiendo del entorno en el que se despliega.
- Se definido en el archivo *settings.xml*. y se utiliza en el *pom.xml*

Arquetipos

- Plantilla para crear proyectos.

Maven



Instalaciones

- Instalar *OpenJDK 11* de Java. Definir/actualizar variables de entorno (JAVA_HOME & PATH)
- Instalar *Maven* y definir las variables de entorno (M2_HOME & PATH)
- Instalar IntelliJ IDEA.

Maven in action

- Crear la carpeta de los *workspaces*.
- Crear un proyecto Java con *maven* en el *workspace*.
- Se ofrece una plantilla a modo de ejemplo: <https://github.com/miw-upm/betca-spring/docs>. Recordar cambiar el **nombre de la carpeta** y del **artefacto** en el fichero *pom.xml*.
- Importar el proyecto desde *IntelliJ IDEA*. *Cerrar proyecto si estuviese abierto*.
 - *Open Project*, y seleccionar la carpeta del proyecto.
 - Marcar *Create Project from external model*, elegir *Maven* .
 - *Next... Finish*.
- Comandos.
 - *Maven externo: maven: mvn -v, mvn package*

Ejercicios

- Instalar en equipos locales.
- Crear un proyecto Java con Maven

Programación Orientada a Objetos

Programación Funcional

POO

Programación imperativa:
¿**Cómo**?

Abstracción, encapsulamiento,
modularidad, jerarquía.

Clases relacionadas mediante
herencia, composición, agregación
y asociación.

Objetos con estado: los atributos.

**Unidad: Clases &
Objetos.**

PF

Programación declarativa: **¿Qué?**

Basado en Funciones: funciones
Lambda.

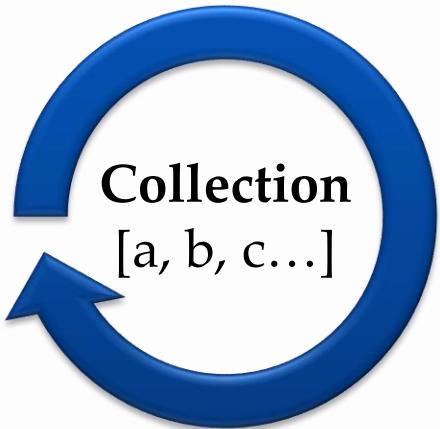
Sin estado, sin orden y sin efectos
colaterales.

Valores inmutables: paso de
parámetros por valor.

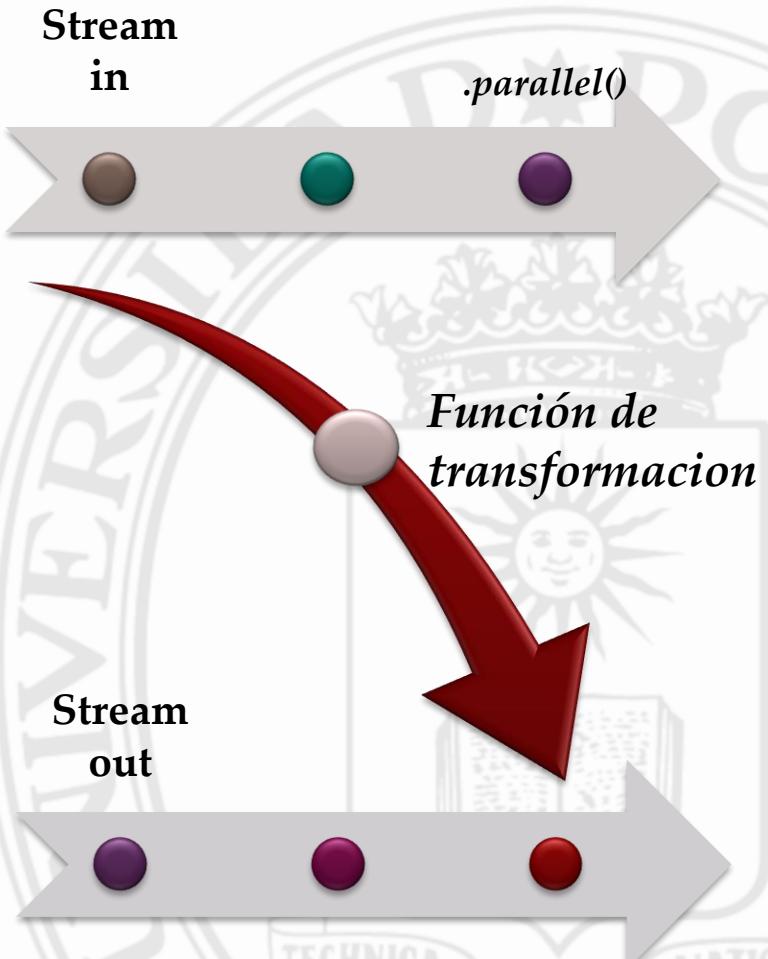
Unidad: Función.

Java

Collection & Stream



- Gestión bucle
- *Método de transformación*
- Paralelismo
- ¿Reutilización del código?



Java

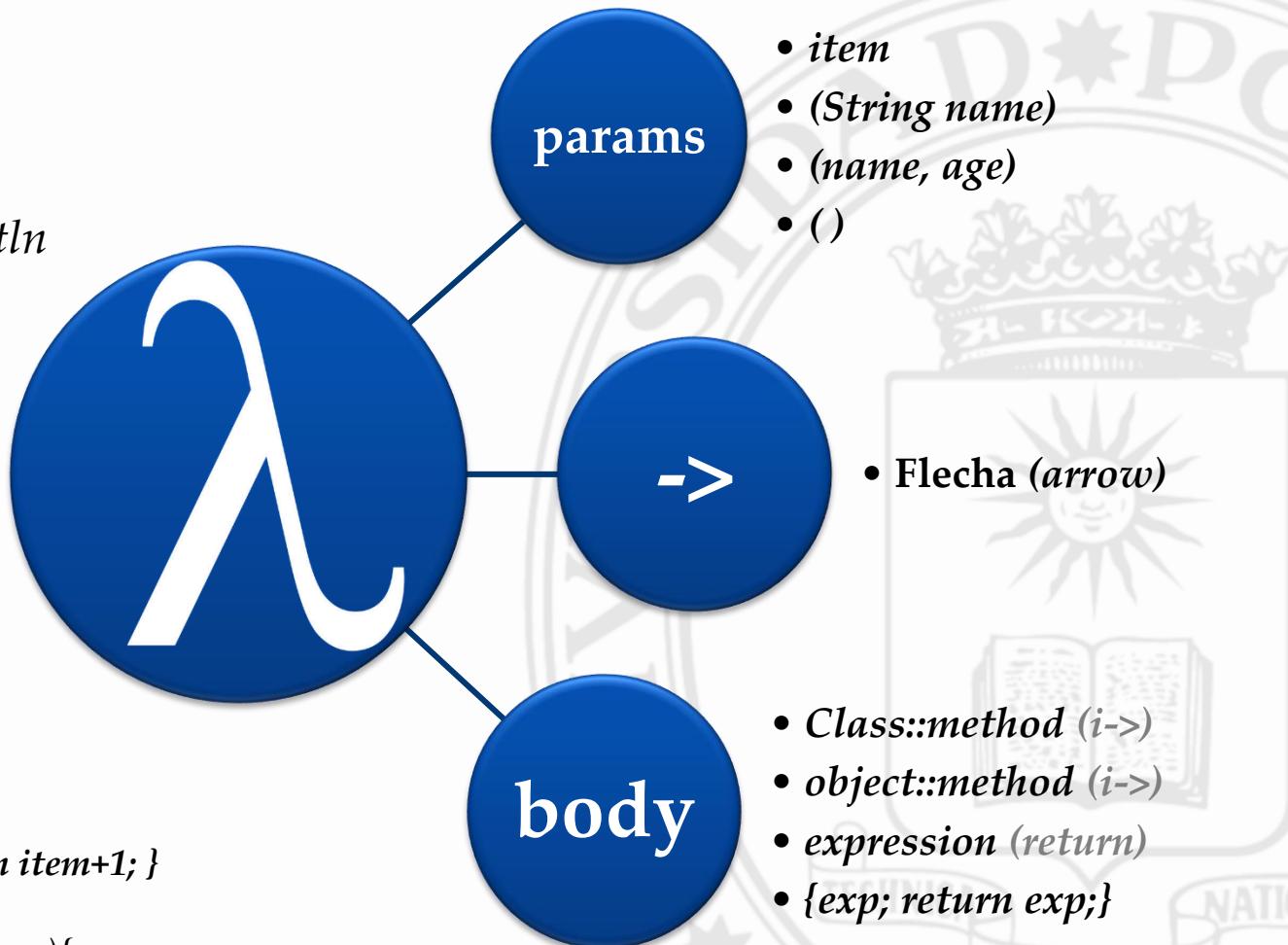
Función Lambda

Función anónima

- Alonzo Church
- $item \rightarrow item + 1$
- `System.out::println`



```
item -> item+1  
(Integer item) -> item+1  
(Integer item) -> { return item+1; }  
o::sumar  
public int sumar(int item){  
    return item+1;  
}
```



Java

Función Lambda (*java.util.function*)

Consumer<T> accept(T)

- System.out::println {item->System.out.println(item)}

Function<T,R> apply(T):R

- item -> item +1

Predicate<T> test(T):Boolean

- item -> item > 0

Supplier<T> get(): T

- ()-> "..."

BiConsumer<T,U> accept(T,U)

- (msg1, msg2) -> System.out.println(msg1 + "," + msg2)

BiFunction<T,U,R> apply(T,U):R

- (x, y) -> x + y

BiPredicate<T,U,R> apply(T,U):R

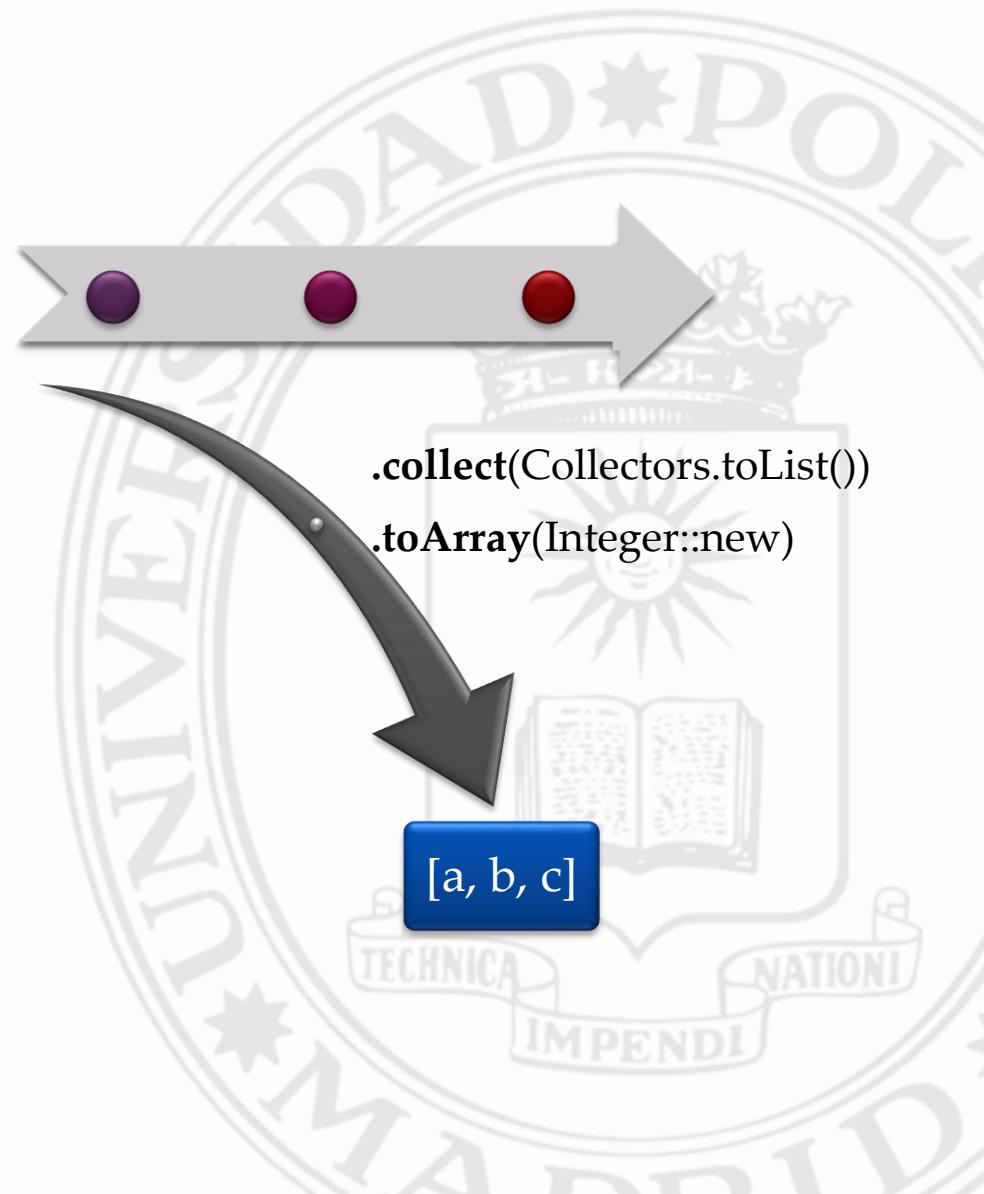
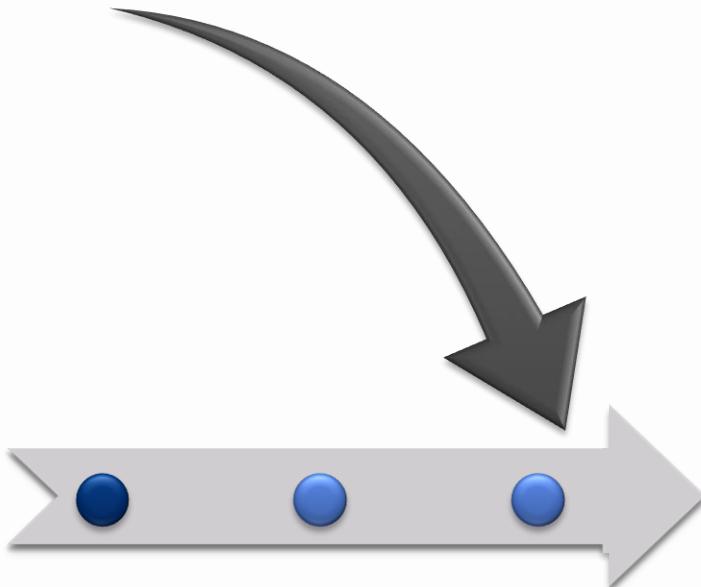
- String::equals

...

Java

Stream. Crear y colección

```
list.stream();  
IntStream.range();  
Stream.of("1", "2"...);  
Stream.generate();  
Stream.iterate();
```

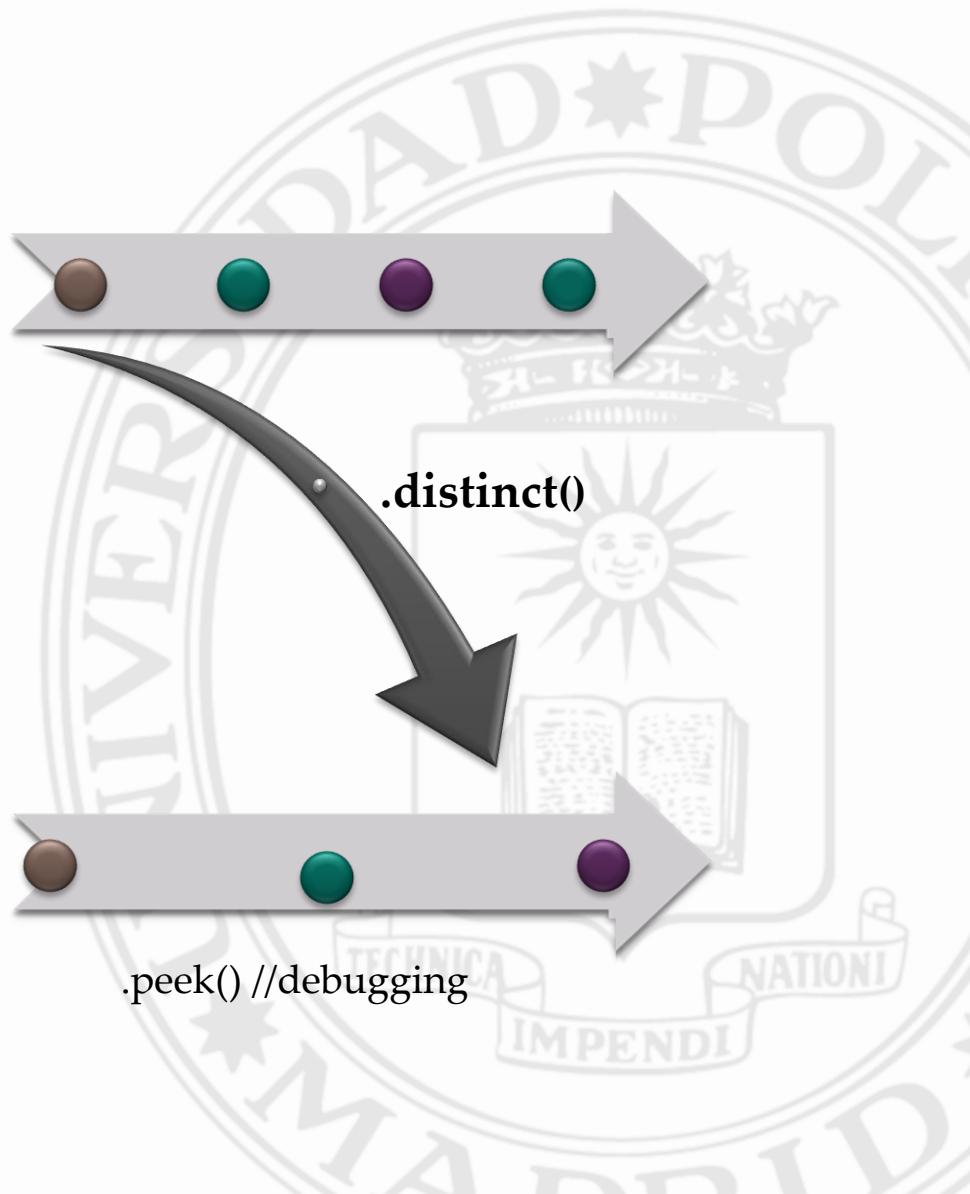
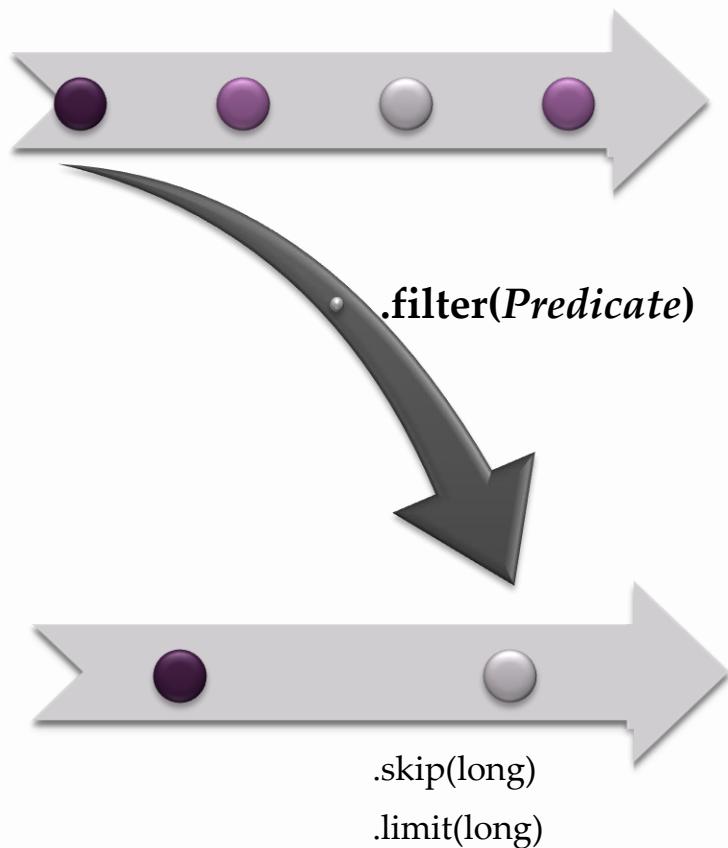


```
.collect(Collectors.toList())  
.toArray(Integer::new)
```

[a, b, c]

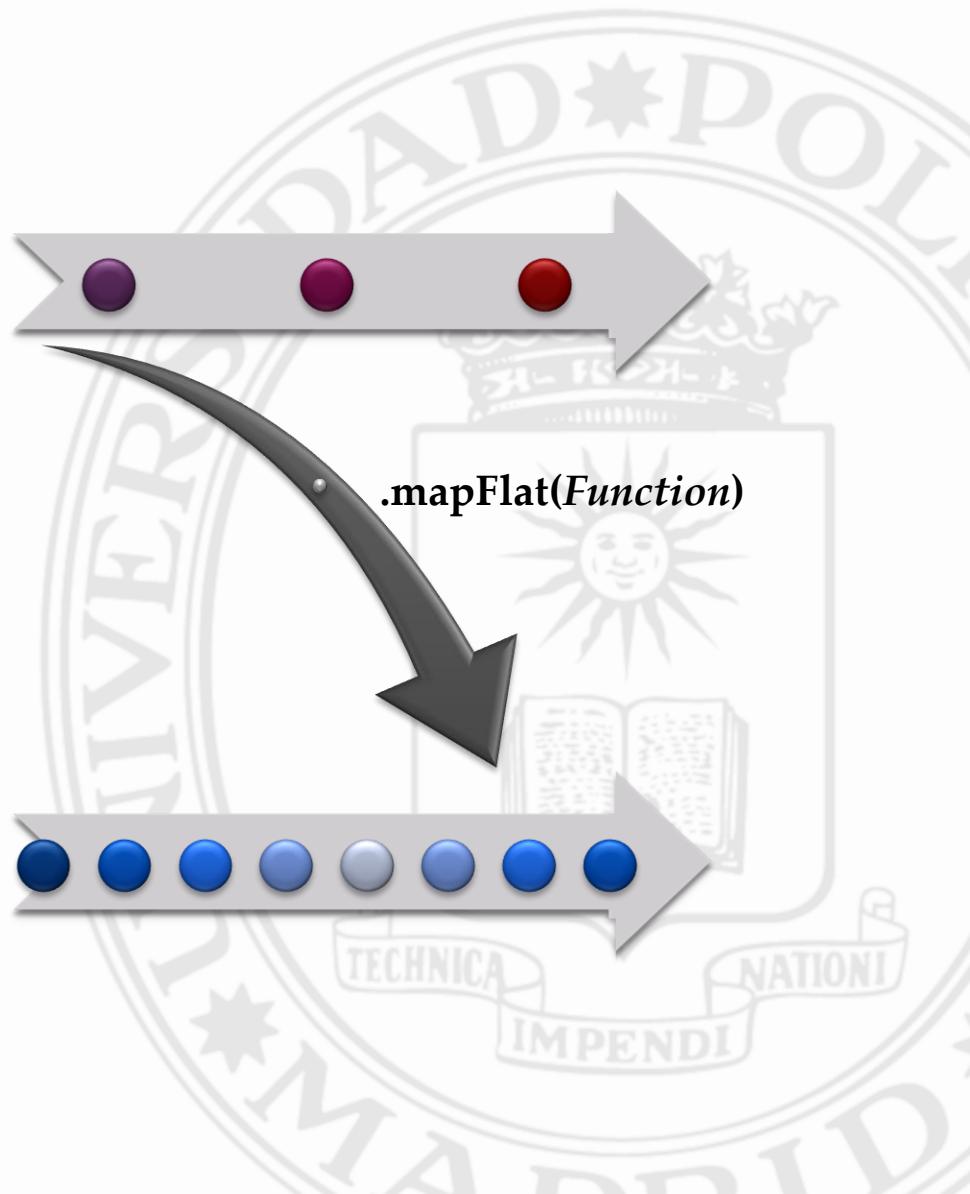
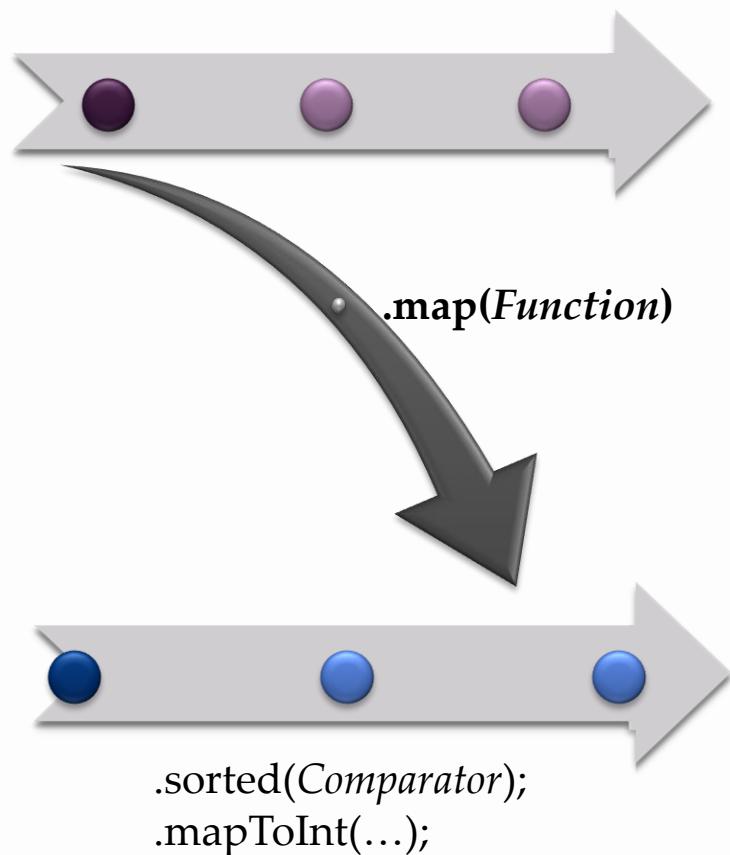
Java

Stream. Filtrado



Java

Stream. Transformación



Java

Stream. Operaciones terminales

- .reduce(Double::sum)**
- .reduce(Double::max)**
- .reduce(Double::min)**
- .reduce(ini,BiFun)**
- .forEach():void**

- .findFirst(Predicate)**
- .anyMatch(Predicate)**
- .noneMatch(Predicate)**
- .allMatch(Predicate)**



<https://github.com/miw-upm/betca-spring>

- Package: *functional*

Code in action

- *PointLackOfPOO, Point, DecimalCollection.*
- *Code, Lambda, Flow.*
- *DecimalCollectionWithStream, DecimalStreamWithFunctionalProgramming.*
- *User, Fraction, UsersDatabase.*
- *Searches.*

Ejercicios

- *Copiar los ficheros: User, Fraction, UsersDatabase en tu proyecto.*
- *Ampliar Searches.*

Test

Característica de calidad

Automática	Integración Continua	Cobertura	Independiente	Sencillez
<ul style="list-style-type: none">Clases que prueban clases.	<ul style="list-style-type: none">Cuando parte del código ha sido modificado, se vuelven a lanzar todas las pruebas.	<ul style="list-style-type: none">% de líneas de código ejecutadas en las pruebas.>80%.	<ul style="list-style-type: none">El orden de las pruebas es independiente.Las pruebas no alteran el sistema.Se prueban los módulos por separado.	<ul style="list-style-type: none">Prueba una cosa a la vez.

Pruebas Unitarias (**Test)

- Se prueba un módulo de código o clase independientemente del resto. Si existen dependencias entre componentes se rompen con los *mocks*.

Pruebas de Integración (**IT)

- Se prueban los diferentes componentes que dependen de otros componentes.

Pruebas Funcionales (**FT)

- Se prueba el sistema como un todo.

Pruebas de Aceptación

- Los clientes prueban la versión entregada.

Test

JUnit 5

JUnit

Es un framework que nos ayuda a la realización de pruebas unitarias

Fue creado por *Erich Gamma* y *Kent Beck*.

Sub-proyectos

Platform

Es el responsable de lanzar los test sobre la JVM

Jupiter

Es el nuevo modelo de programación

Vintage

Es para compatibilizar JUnit3 y 4 sobre JUnit5

Test

JUnit 5

@BeforeAll

- Se ejecuta una sola vez antes de la batería de pruebas definida en la clase y el método debe ser *static*

@BeforeEach

- Se ejecuta antes de cada uno de los marcados con @. Suele ser una inicialización por todas las pruebas de la clase

@Test

- Marca un método como prueba

@AfterEach

- Se ejecuta después de cada uno de los @Test. Suele ser una liberación de recursos

@AfterAll:

- Se ejecuta al final del proceso completo y el método debe ser static

@Test
assert**

assertEquals, assertNotEquals,
assertArrayEquals, assertTrue,
assertFalse, assertEquals,
assertNotSame, assertNull,
assertNotNull, fail(),
assertThrows,
assertDoesNotThrow...

JUnit



<https://github.com/miw-upm/betca-spring>

- Package: *functional*.

JUnit in action

- Point: *PointTest*, DecimalCollection: *DecimalCollectionTest*.
- *DecimalCollectionWithStreamTest*,
DecimalStreamWithFunctionalProgrammingTest.
- *FlowTest*, *SearchesTest*.
- Lanzamiento de test con *IntelliJ*: *todo*, *paquete*, *clase y test*.
- Cobertura con *IntelliJ*.

Ejercicios

- Crear los test para las clases del paquete: *User & Fraction*.
- Ampliar las funcionalidades de *Fraction*.

Git

Sistema de Control de Versiones

Tipos

- **Distribuidos.** Aumenta la flexibilidad pero complica la sincronización y gestión. Ejemplos: *Git, Mercurial...* En la actualidad se están imponiendo estos sistemas.
- **Centralizados.** Dependiente de un responsable. Facilita la gestión pero reduce la potencia y flexibilidad. Ejemplo *CVS*.

Git

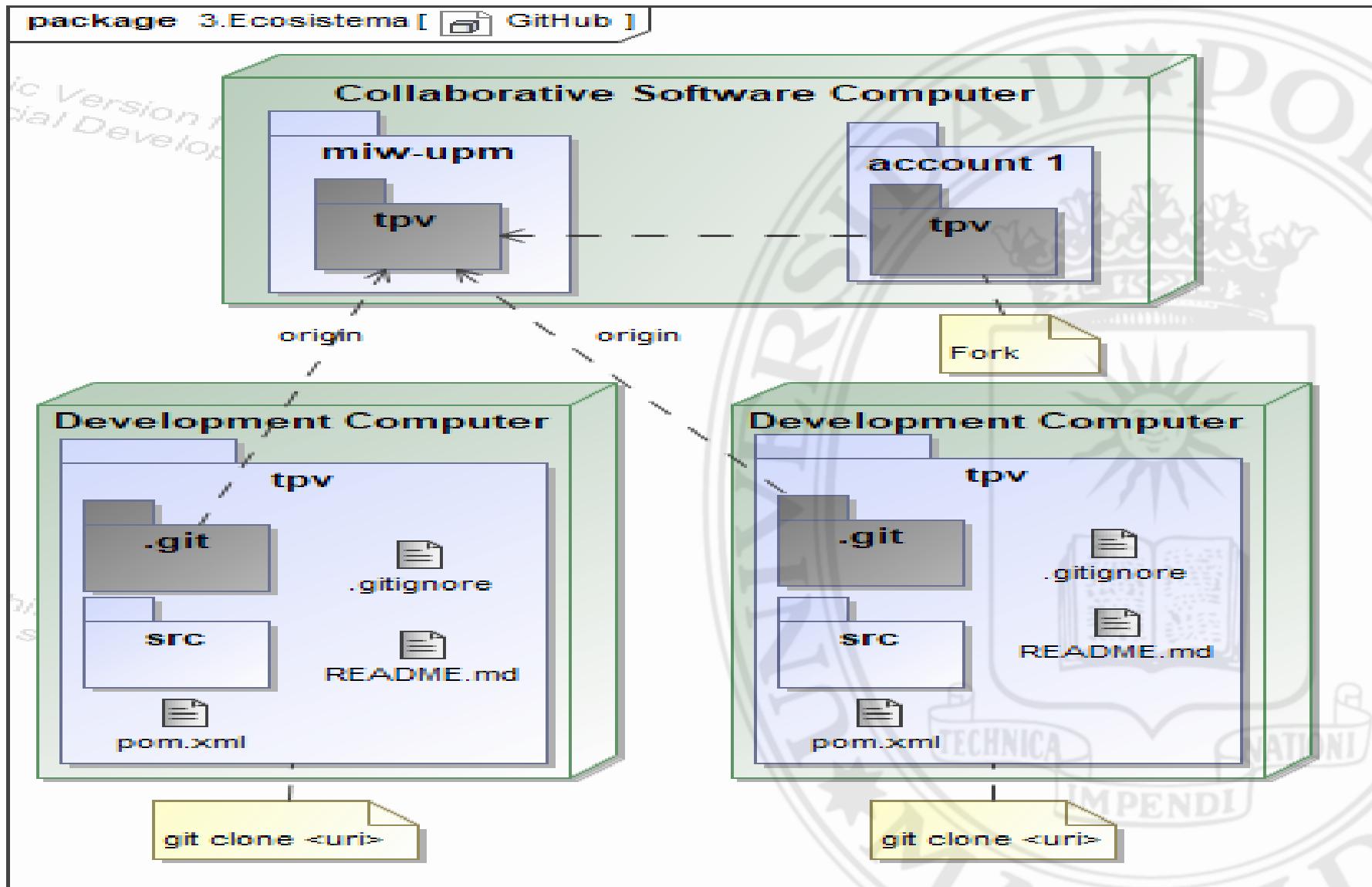
- Nace en 2005.
- Toma como experiencia el proyecto Bitkeeper. (propietario).
- En 2008 nace GitHub, Git Cloud.
- En el 2018 Microsoft adquiere GitHub por 7.500 millones de dólares.
- Software: CLI: <https://git-scm.com>

Características

- Control de versiones distribuido.
- Muy fiable, imposible perder el proyecto.
- Trabaja sin necesidad de conexión al remoto, muy rápido. Se podrá sincronizar con el remoto, pero con asistencia...
- **Snapshot:** instantánea (commits)

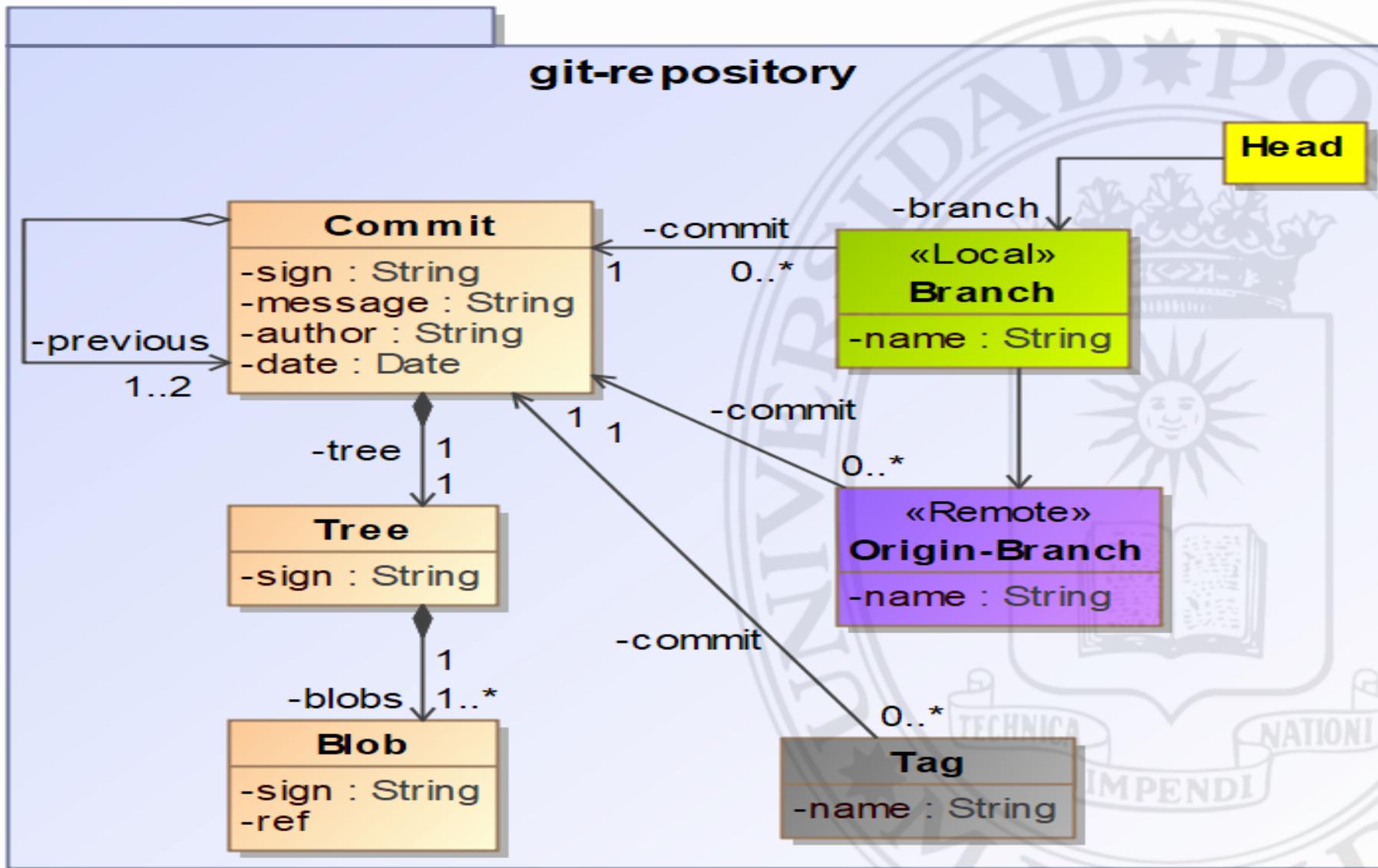
Git

Sistema de control de versiones



Git

Instantánea (snapshot)



Git IntelliJ

The screenshot shows the IntelliJ IDEA interface with the Version Control tab selected. The log displays a series of commits, each with a timestamp, author, and a detailed message. A context menu is open over a commit from May 19, 2019, at 16:22, which includes options like 'master', 'release-1.0', 'origin/master', 'origin/release-1.0', and 'release-1.0.2'. To the right of the log, there's a file tree for a project named 'etsisi-tpv-spring' containing '.travis.yml' and 'README.md'. A red arrow points to the 'Version Control' tab in the bottom navigation bar.

Commit Message	Date	Author
reformat all code	19/05/2019 17:20	jbernal
update Swagger page description #1 on develop	19/05/2019 16:50	jbernal
version 1.0.2 & Heroku Procfile	19/05/2019 16:31	jbernal
Merge bug #1 into release-1.0	19/05/2019 16:22	jbernal
update swagger page description #1	19/05/2019 16:19	jbernal
add mongodb embedded test scope	19/05/2019 13:19	jbernal
version 1.0.1	19/05/2019 13:00	jbernal
update sprig-boot version to 2.1.5	19/05/2019 11:34	jbernal
version 1.1.0-SNAPSHOT	19/05/2019 11:30	jbernal
prepare release: version 1.0.0 & prod profile	19/05/2019 11:29	jbernal
heroku add Procfile	19/05/2019 8:20	jbernal
heroku add deploy	19/05/2019 8:17	jbernal
better-code-hub depth to 7	19/05/2019 7:33	jbernal
ecosystem add sonarcloud & better-code-hub	19/05/2019 7:23	jbernal
ecosystem add Travis-CI	19/05/2019 7:07	jbernal
add docs & readme	19/05/2019 6:55	jbernal
Initial TPV	19/05/2019 6:40	jbernal

File tree on the right:

- etsisi-tpv-spring (2 files)
 - .travis.yml
 - README.md

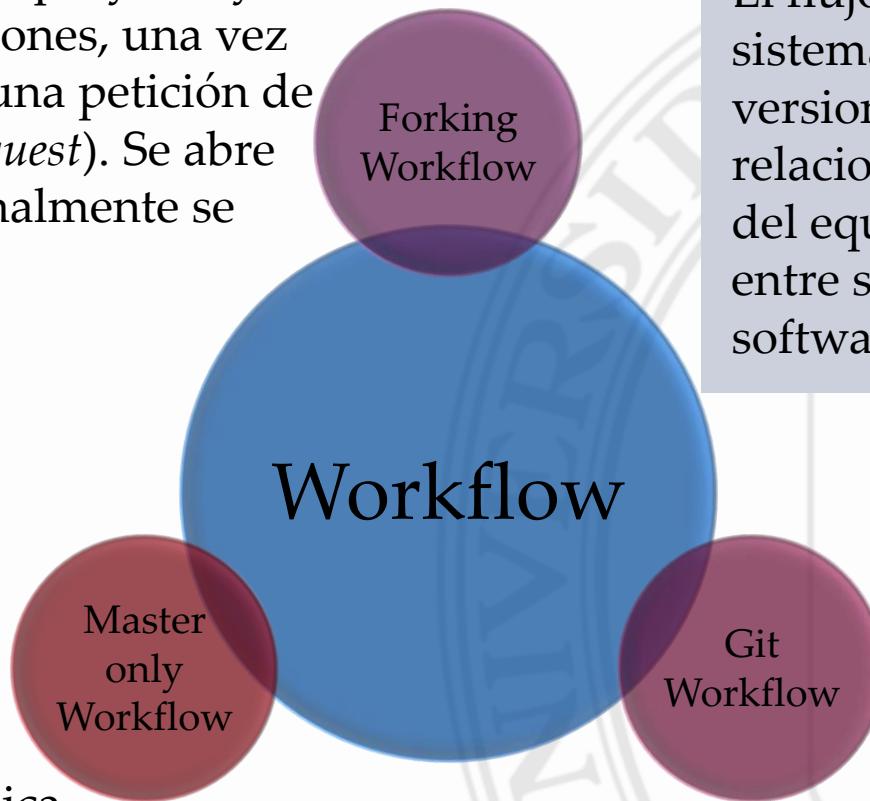
Bottom navigation bar:

- Version Control
- Terminal
- Java Enterprise
- Spring
- Messages
- Find
- Run
- TODO

Git

Flujo de Trabajo

El usuario bifurca el proyecto y realiza las ampliaciones, una vez finalizado, realiza una petición de agregación (*pull request*). Se abre una discusión, y finalmente se fusión.



Todos comparten la única rama... Fusión problemática!!! y difícil de mantener estable

El flujo de trabajo de un sistema de control de versiones indica cómo se relacionan los miembros del equipo para colaborar entre sí en el desarrollo del software colaborativo



Existen varias ramas con distintas funcionalidades dentro del equipo: *master*, *develop*, *feature*, *release*, *bug*

Git

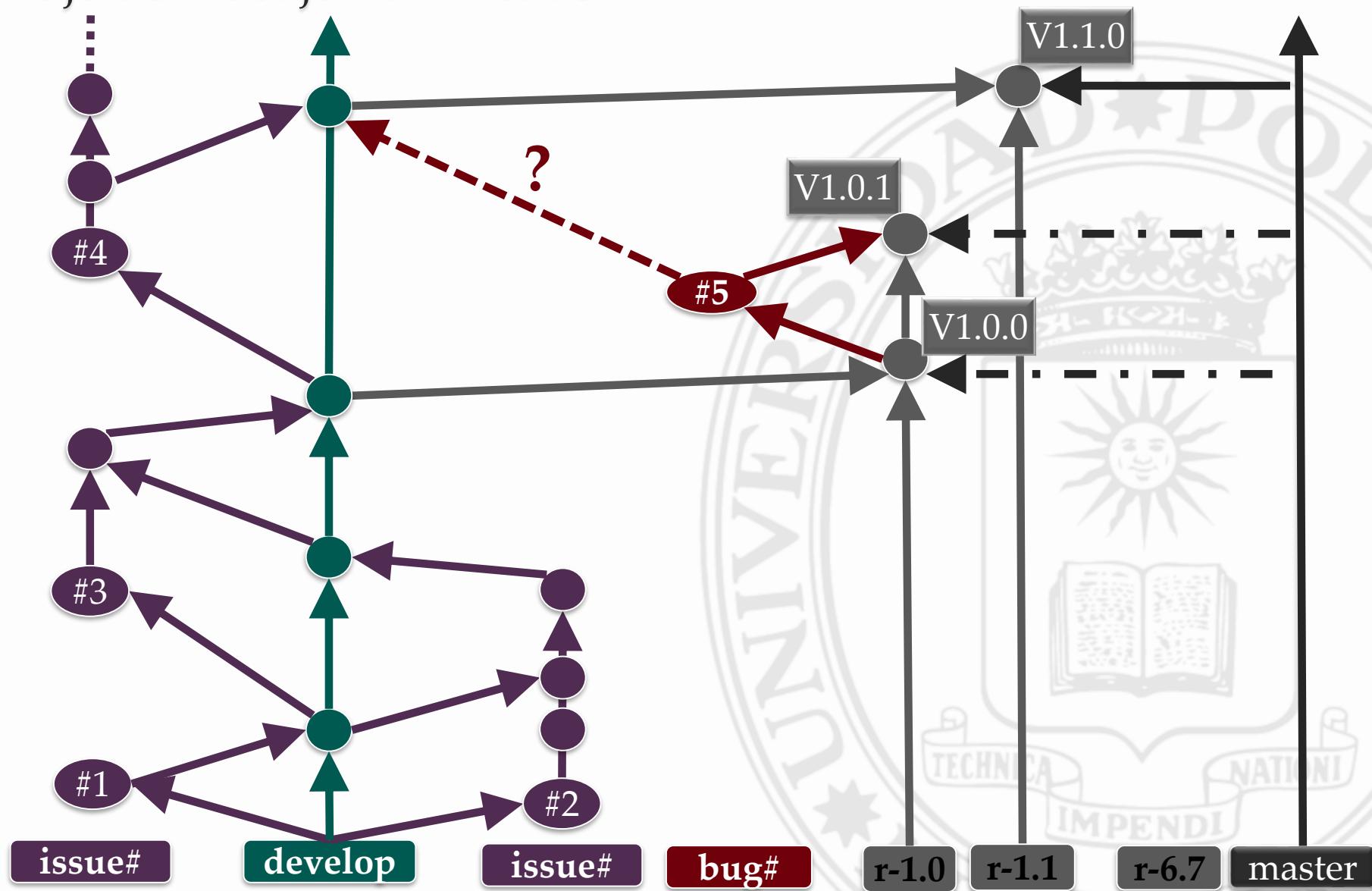
Flujo de Trabajo Ramificado

Se utilizan para corregir errores (bugs) en el código en producción



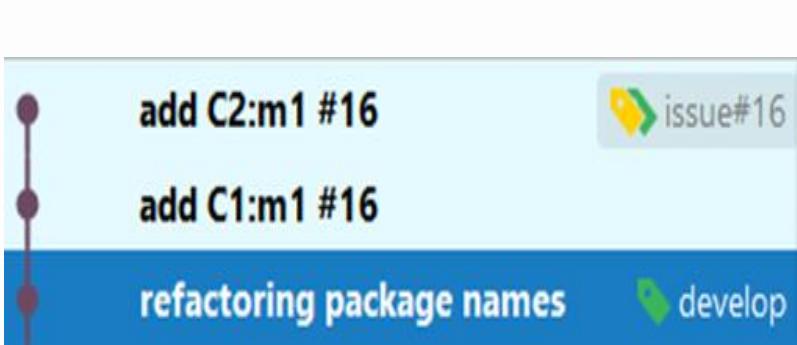
Git

Flujo de Trabajo Ramificado



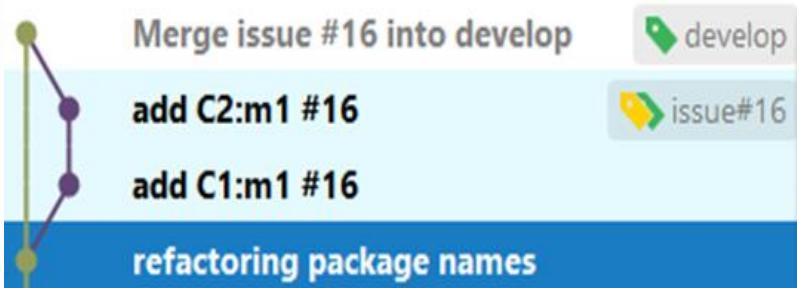
Git

Workflow: local



inicio
issue#

- git checkout -b issue#16
- git add -all
- git commit -m "... #16"
- git commit -m "... #16"



aportación
parcial

- git checkout develop
- git merge --no-ff -m ".. #16" issue#16
- git checkout issue#16

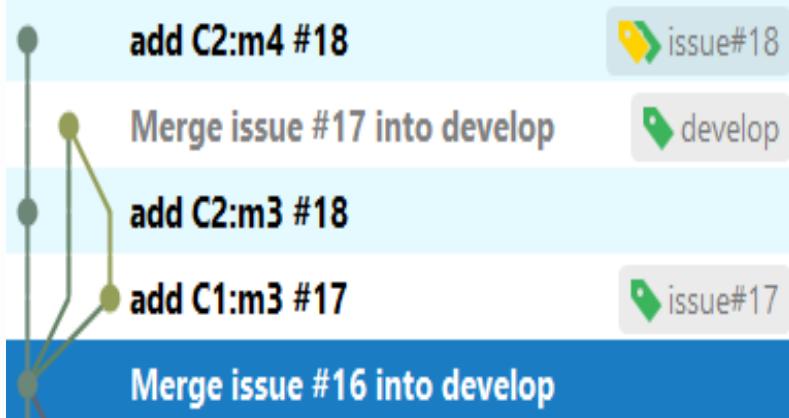


cierre
issue#

- git commit -m "..."
- git commit -m "..."
- git checkout develop
- git merge --no-ff -m "... #16" issue#16

Git

Workflow: local



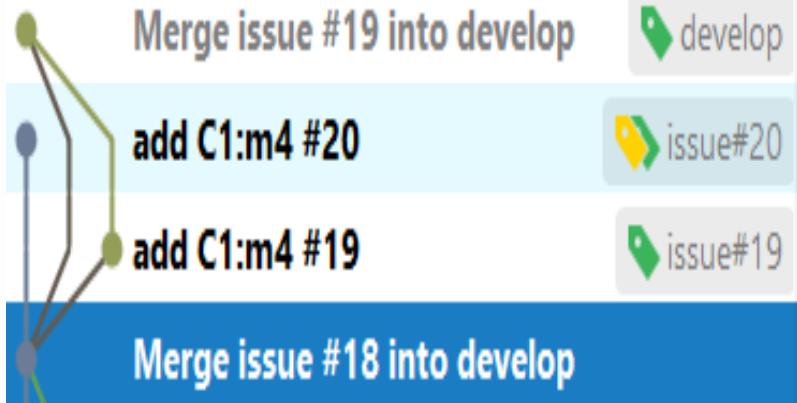
cierre
issue#18
SIN
conflicto

- git merge -m "..." develop
- git checkout develop
- git merge --no-ff -m "..." issue#18



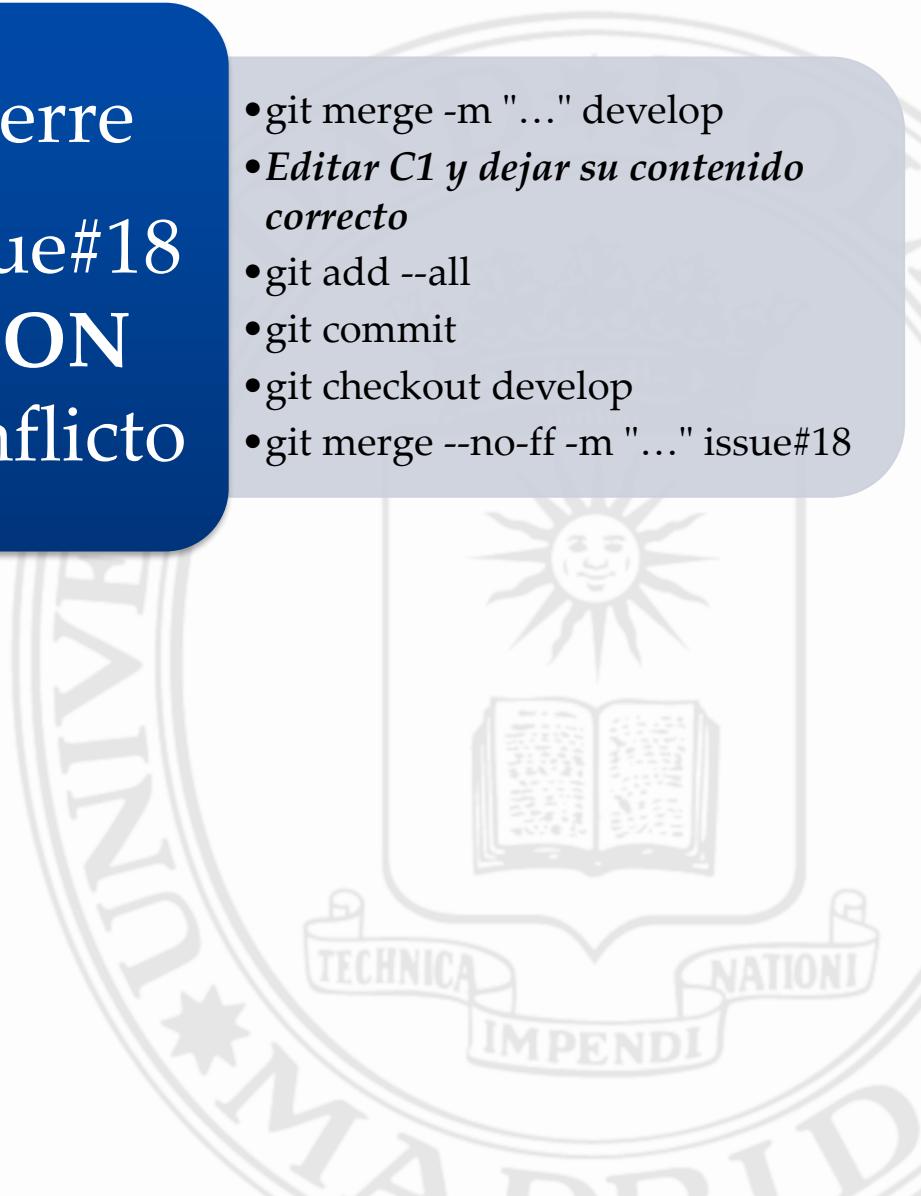
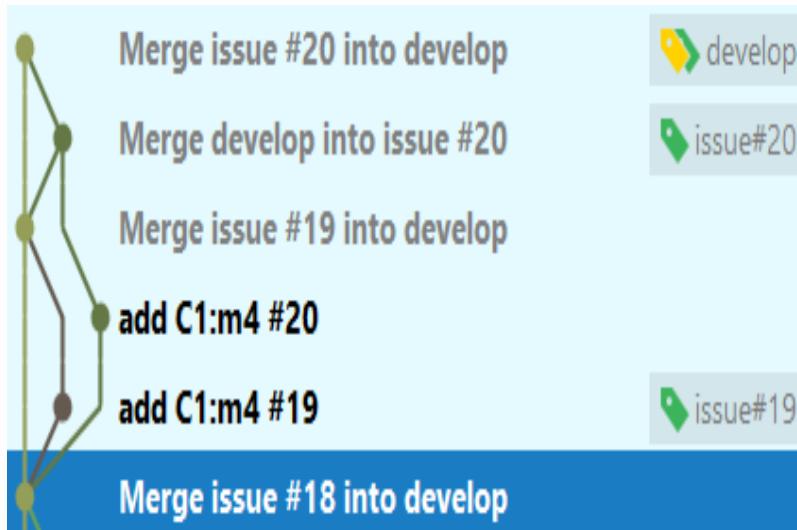
Git

Workflow: local



cierre
issue#18
CON
conflicto

- git merge -m "..." develop
- *Editar C1 y dejar su contenido correcto*
- git add --all
- git commit
- git checkout develop
- git merge --no-ff -m "..." issue#18



Git

Comandos

help

- `git help --all`

status

- `git status`

config

- `git config --global user.name "..."`
- `git config --global user.email "..."`
- `git config credential.helper store`
- `git config --list`

clone

- `git clone <uri>`

Git

Comandos

branch - checkout

- git branch <branch>
- git checkout -b <branch>
- git checkout <branch>

add - commit

- git add --all
- git commit -m "message #666"
- git commit --amend --no-edit
- git commit --amend -m "new message #666"

merge

- git merge -m "merge develop into #666" develop
- git merge --no-ff -m "merge #666 into develop. Details" issue#666
- git merge origin/develop

init

- git init
- git add --all
- git commit -m "mi mensaje de commit inicial"

reset

- git reset --hard HEAD
- git reset --hard <commit>
- Borrado de rama y creación en el commit correcto

Git



Clonar el repositorio

- <https://github.com/miw-upm/iwvg-devops>

Importar desde IntelliJ IDEA

Git in Action

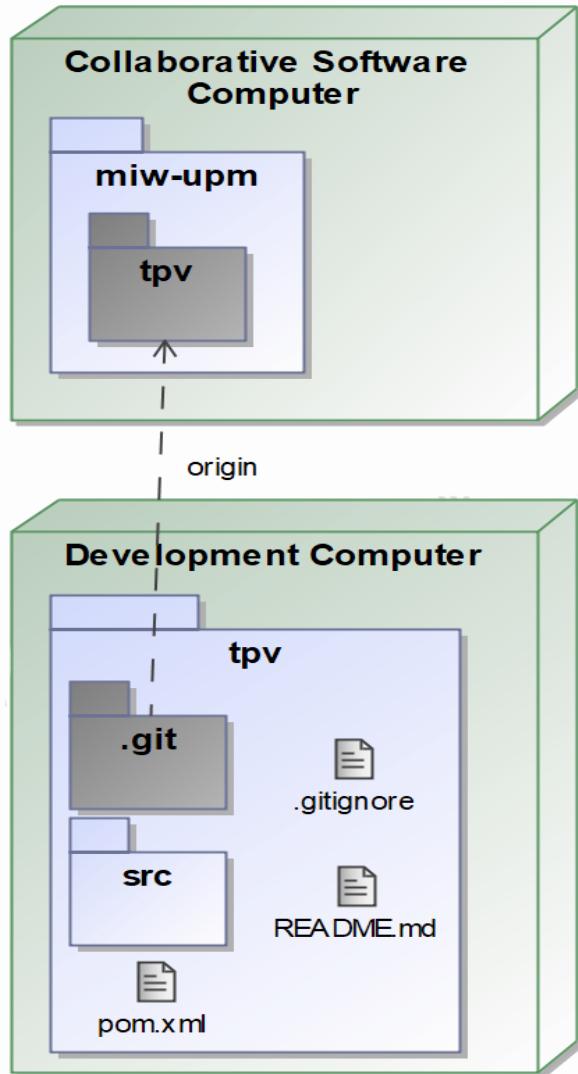
- Crear un proyecto *maven* a partir de la *plantilla* y crear un repositorio local
- Crear la ramas *develop* & *issue#1*
- Trabajar en la rama *issue#1*, creando clases y paquetes
- Incorporar los cambios a la rama *develop*, seguir trabajando en *issue#1* y al final, volver a incorporar a *develop*
- Dos *issues* sin conflictos
- Dos *issues* con conflictos

Ejercicios

- Crear la rama *issue#2* y repetir lo anterior... repetir con *issue#3...* & *repetir!*
- Crear las ramas *issue#10* & *issue#11*, creando una competencia sin conflicto.
- Crear las ramas *issue#12* & *issue#13*, creando una competencia con conflicto, trabajar en el mismo método.
- Repetir con #14 & #15... *repetir...* *repetir*

Git

Workflow: remoto



Creación

- ...*crear repositorio en GitHub*
- `git remote add origin <URI>`
- `git push origin --all`

Clonación

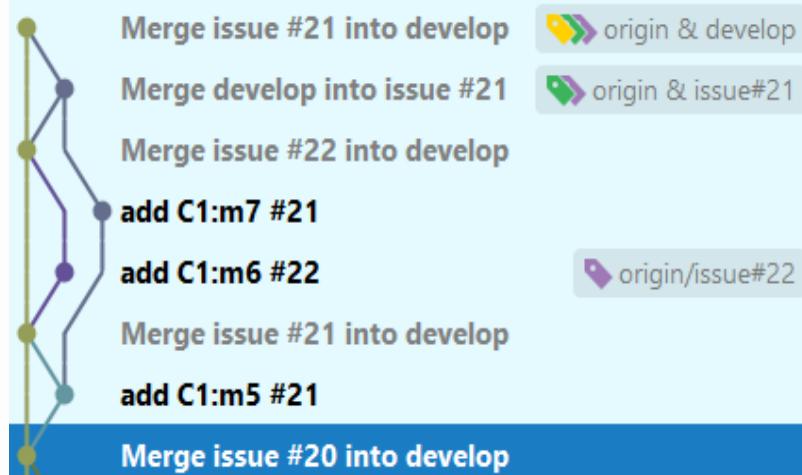
- ...*clonar repositorio ya existente*
- `git clone <URI>`

Git

Workflow: remoto



- git fetch origin
- git checkout develop
- git merge origin/develop
- git check issue#21
- git merge -m "..." develop
- git checkout develop
- git merge --no-ff -m "..." issue#21
- git push origin develop



Git

Workflow: remoto



subir
develop

- Solo FF (**FastForward**)
- git push origin develop

Git

Workflow: remoto

Consulta

- git remote -v
- git remote rm origin

Añadir - Borrar

- **Crear repositorio mediante la Web de GitHub**
- git remote add origin <url repository>
- git remote rm origin

Subir - Bajar

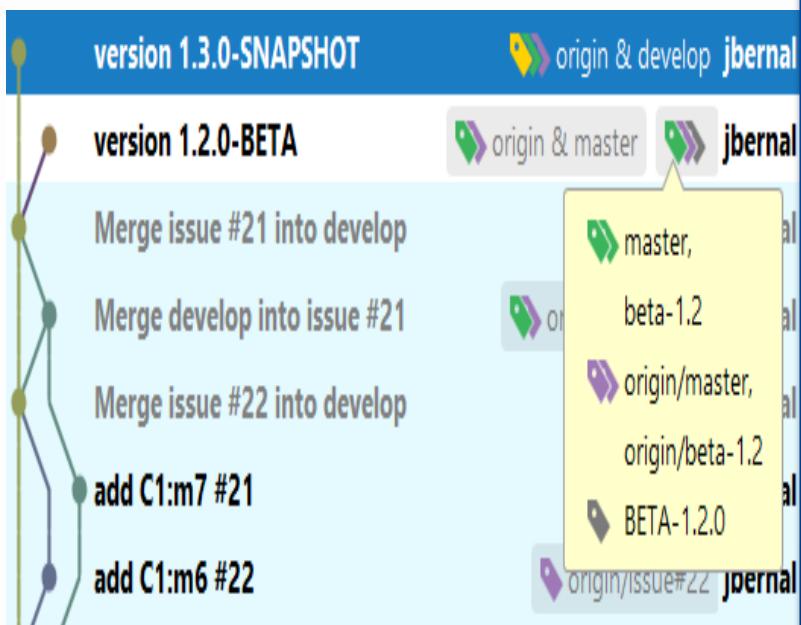
- git push origin <branch>
- git fetch origin

💀💀💀 Subir de manera forzada 💀💀💀

- git push origin <branch> --force

Git

Workflow: release



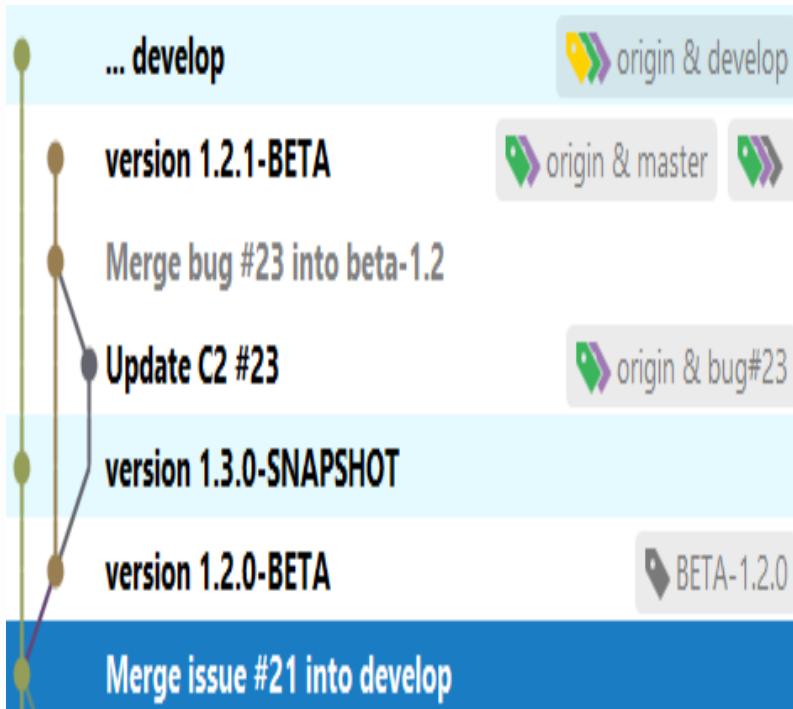
release

- git checkout -b beta-1.2
- ... *change pom.xml*
- git commit -m"..."
- git push origin beta-1.2

- git tag -a v1.2.0-beta -m"..."
- git push origin v1.2.0-beta
- git branch -d master
- git branch master
- git push origin master --force
- git checkout develop
- ... *change pom.xml*
- git commit -m"..."
- git push origin develop

Git

Workflow: bug



bug

- git checkout beta-1.2
- git checkout -b bug#23
- ... *corregir problema*
- git commit -m"..."
- git checkout beta-1.2
- git merge --no-ff -m "..." bug#23
- git push origin beta-1.2
- *Liberar nueva versión*

Git

workflow: release

Consulta de etiquetas

- **git tag**

Crear etiqueta

- **git tag -a <etiqueta> -m "mensaje"**

Borrado de etiqueta

- **git tag -d <etiqueta>**

Subir etiqueta al remoto

- **git push origin <etiqueta>**

Git

Workflow: test



Errores?

• Solución?



Error?

• Solución?

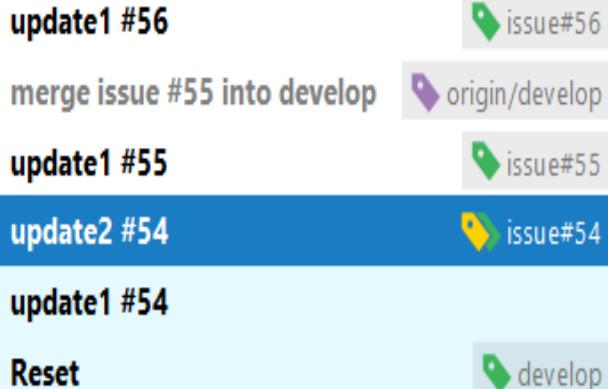


Error?

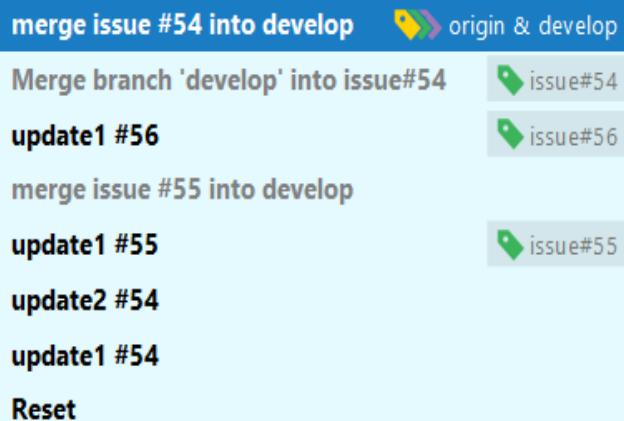
• Solución?

Git

Workflow: test



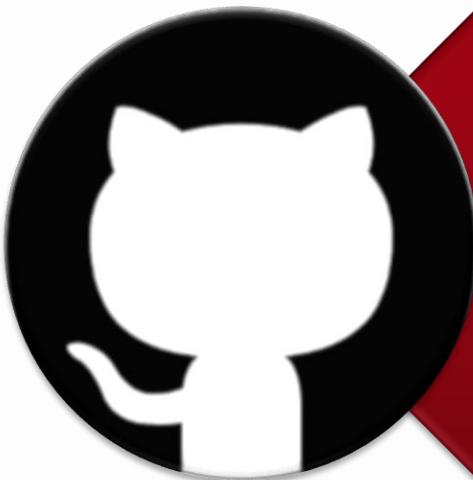
cierre
issue#54



Error?

- Quién #54, #55 o #56?
- Error cometido?
- Solución?

Plan



GitHub



Slack

GitHub Hitos

The screenshot shows a GitHub repository page for 'miw-upm / iwvg-devops'. The 'Issues' tab is selected, showing 11 issues. Below the tabs are 'Labels' and 'Milestones'. The 'Milestones' tab is active, showing three entries:

- Scrum: sprint 1**: Past due by 11 months, Last updated 12 months ago. Start: 9/01/2019 Team velocity: 50. Progress: 50% complete (2 open, 2 closed). Buttons: Edit, Close, Delete.
- Scrum: sprint 2**: Past due by 11 months, Last updated over 1 year ago. Start: 30/01/2019 Team velocity: 65. Progress: 0% complete (0 open, 0 closed). Buttons: Edit, Close, Delete.
- Curso 2019-20**: Past due by 12 months, Last updated over 1 year ago. Start: MIW-IWVG Software Ecosystem. Progress: 0% complete (0 open, 0 closed). Buttons: Edit, Close, Delete.

At the bottom of the screenshot, the word 'Milestone' is highlighted in blue.

Representa una de fecha de referencia, normalmente asociada a un lanzamiento o finalización de un módulo.

Puede estar abierto-cerrado.

Se les puede asociar tickets.

Tiene marcado un nivel de finalización, obtenido por los tickets asociados que se encuentran cerrados.

GitHub Issues

The screenshot shows a GitHub project named "Ecosystem Software" with the following columns:

- Product Backlog:** Contains 5 items:
 - #13 Find 1 (Medium priority, 2 points, User type)
 - #14 Find 2 (Medium priority, 2 points, User type)
 - #15 Find 3 (Medium priority, 2 points, User type)
 - #16 Find 4 (Medium priority, 2 points, User type)
 - #17 Bug (Low priority, 2 points, Bug type)
- Sprint Backlog:** Contains 6 items:
 - #7 User (High priority, 1 point, User type)
 - #8 UserTest (High priority, 1 point, User type)
 - #9 UsersDatabase (High priority, 1 point, User type)
 - #10 FractionTest (High priority, 1 point, Technical type)
 - #11 Fraction Enhancement (High priority, 1 point, User type)
 - #12 FractionTest Enhancement (High priority, 1 point, User type)
- In progress:** Contains 1 item:
 - #6 Fraction (High priority, 1 point, User type)
- Done:** Contains 4 items:
 - #3 Análisis del código con Sonarcloud (Urgent priority, 0.2 points, Technical type)
 - #2 Integración continua con GitHub Actions (Urgent priority, 0.5 points, Technical type)
 - #4 Desplegar en Heroku (Urgent priority, 0.5 points, Technical type)
 - #5 Liberación de la Release 3.0.0 (Urgent priority, 0.5 points, User type)

Issues

Disputa, asunto, tarea, error, ticket...

Se abre un foro de comentarios.

Etiquetas: facilitan su identificación y organización.

Etiquetas de prioridad:
priority: high, priority: médium...

Etiquetas de tipo: *type: test, type: documentation...*

Etiquetas de estimación:
points: 8

GitHub

Organización de un Issue

The screenshot shows a GitHub issue page for issue #355. The title is "Búsqueda 1 hotel #355". The status is "Closed" by "Susana" last month. The issue has 2 comments. The description is: "Dado nombre de Cliente, obtener los empleados que han limpiado la habitación donde reservó el cliente". The assignee is Susana, and the label is "story: hotel". The project is "APAW.Practice" with a milestone "Evaluación Continua. 2022-23" set to "Done". The commit history includes: "Susana commented last month · edited · Collaborator · ...", "Susana self-assigned this last month", "Susana added this to the Evaluación Continua. 2022-23 milestone last month", "Susana added a commit that referenced this issue 27 days ago (#355 added first search)", "Susana added a commit that referenced this issue 27 days ago (merge develop into issue #355)", "Susana added a commit that referenced this issue 27 days ago (#355 fixed problems with tests)", and "Susana added a commit that referenced this issue 27 days ago (merge #355 into develop)". Notifications show 1 participant.

Issue

Breve descripción.
Con la referencia en los commits de “#xx” aparece asociado al issue#

GitHub Scrum

The screenshot shows the GitHub Project Settings interface for a specific project. On the left, there's a sidebar with options like 'Project settings' and 'Manage access'. Under 'Custom fields', the 'Estimation' field is selected and highlighted with a blue border. The main panel displays the 'Estimation field settings' configuration. It includes a 'Field name' input set to 'Estimation', a 'Field type' dropdown set to 'Single select', and a list of estimation points: 11 Points, 8 Points, 5 Points, 3 Points, 2 Points, 1 Points, 0.5 Points, and 0.2 Points. Each item has a small icon next to it. At the bottom, there's a green 'Save options' button.

The screenshot shows the GitHub Project Settings interface for a specific project. It displays two field configurations side-by-side. The first is for the 'Priority' field, which has a 'Field name' of 'Priority' and a 'Field type' of 'Single select'. Below it is a list of priority levels: Urgent, High, Medium, and Low, each with a small icon. The second is for the 'Story' field, which has a 'Field name' of 'Story' and a 'Field type' of 'Single select'. Below it is a list of story types: Technical, User, and Bug, each with a small icon.

Consumed (horas) field settings

The screenshot shows the GitHub Project Settings interface for a specific project. It displays the configuration for the 'Consumed (horas)' field. The 'Field name' is set to 'Consumed (horas)' and the 'Field type' is set to 'Number'. There is also a small icon representing the field type.

Story

- Crear las historias (*issues*) en el *Product Backlog*. Asociarle *prioridad, estimación, tipo...*

Sprint Backlog

- Mover las historias a la columna *Sprint backlog*, son las que desarrollan en el próximo *sprint*, asociarles el *sprint* (hito).

In progress

- Cuando alguien inicia una historia, tarea (*issue*), se lo asigna y lo mueve a la columna *In progress*.

Branch: issue#

- Se crea la rama 'issue#xx' y se programa la historia, tarea... un *commit* debe ser reflejado en el historial del *issue*, se añade en el mensaje '#xx'

Fusión

- Cuando la historia se termina, se fusiona con *develop*, siempre debiera ser posible el avance rápido, pero no se hace con avance rápido:
`git merge -no-ff -m "Merge #xx into develop. Detalles" issue#xx`

Close

- Se cierra la historia (*issue#xx*), se establece el tiempo consumido y se abandona la rama

Slack

Comunicación

The screenshot shows a Slack workspace interface. The left sidebar has a dark purple background with various sections: 'miw-upm' (with a dropdown menu), 'Mensajes sin leer', 'Hilos de conversaciones', 'Todos los mensajes directos', 'Menciones y reacciones', 'Slack Connect', 'Más', 'Canales' (expanded), 'iwvg-devops' (selected, highlighted in blue), 'apaw-practice', '# general', '+ Añadir canales', 'Mensajes directos' (with Slackbot, miw-jbernal, Wael Louati, Yosbel Brooks), 'Aplicaciones' (+ Añadir aplicaciones), and 'Canales' (with 'iwvg-devops' expanded). The main area shows the 'iwvg-devops' channel. At the top, there's a search bar with 'Buscar en miw-upm' and a magnifying glass icon. Below it, there's a dropdown menu for 'iwvg-devops' with options like 'Añadir un marcador'. The channel header says 'iwvg-devops' with a lock icon. It displays a message from 'miw-jbernal' at 19:07: 'se ha unido a iwvg-devops.' followed by a GitHub integration message from 'incoming-webhook' at 19:48: 'añadió una integración a este canal: incoming-webhook'. This message includes a screenshot of a GitHub commit log for '8398a7@action-slack' from 'miw-upm/iwvg-devops' with author 'jbernal<j.bernal@upm.es>'. Below this, another message from 'incoming-webhook' at 20:07: 'Hola desde spring!!' is shown. Further down, a message from 'miw-jbernal' at 19:34: 'Lambda.java' is displayed with code snippets for 'Lambda.java' and 'pom.xml'. At the bottom, a PDF file titled 'miw-iwvg-devops-slides.pdf' (10 MB) is previewed, showing the title 'Ingeniería Web: Visión General -IWVG-' and the MIW logo.

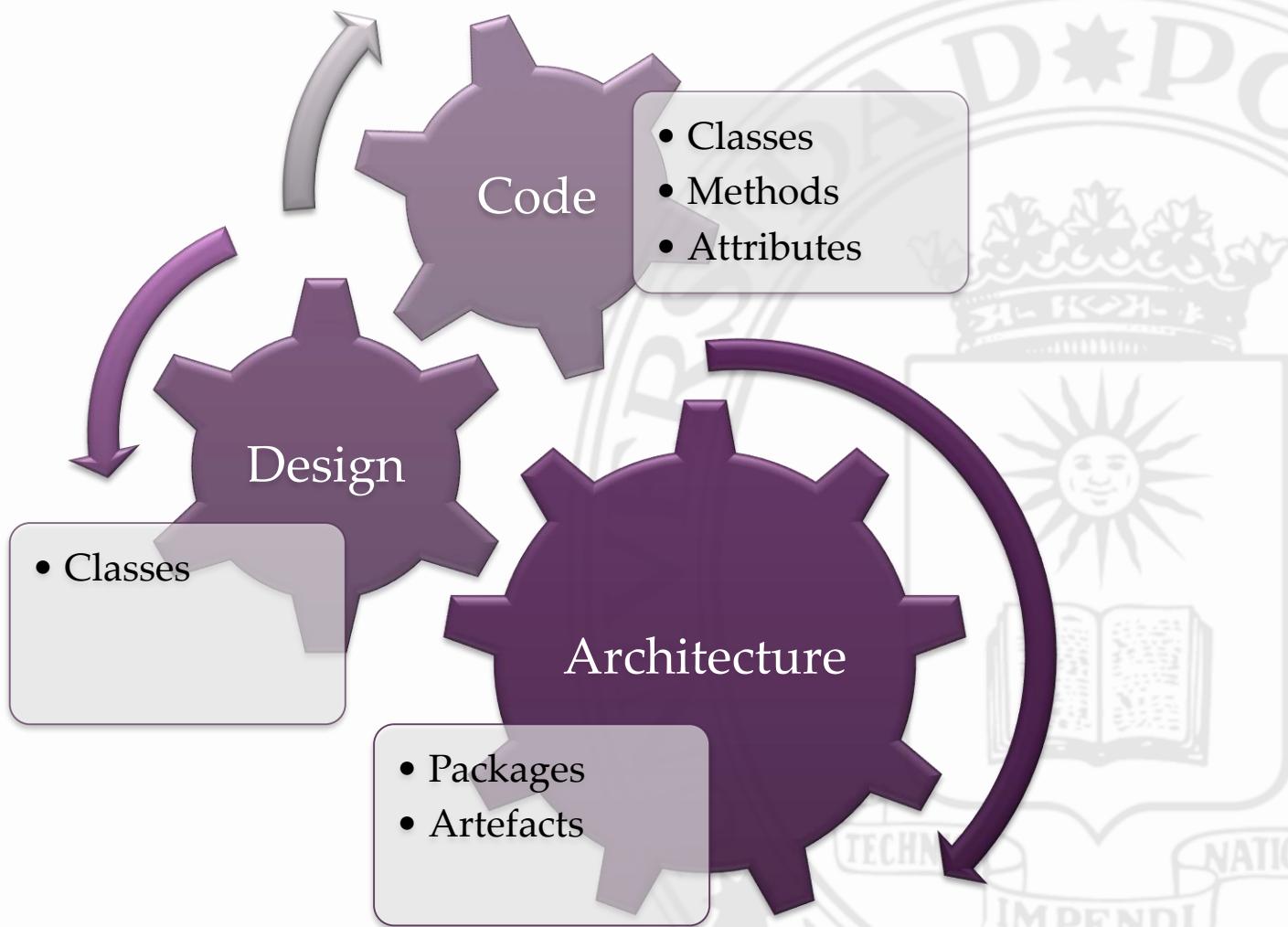
Arquitectura Web

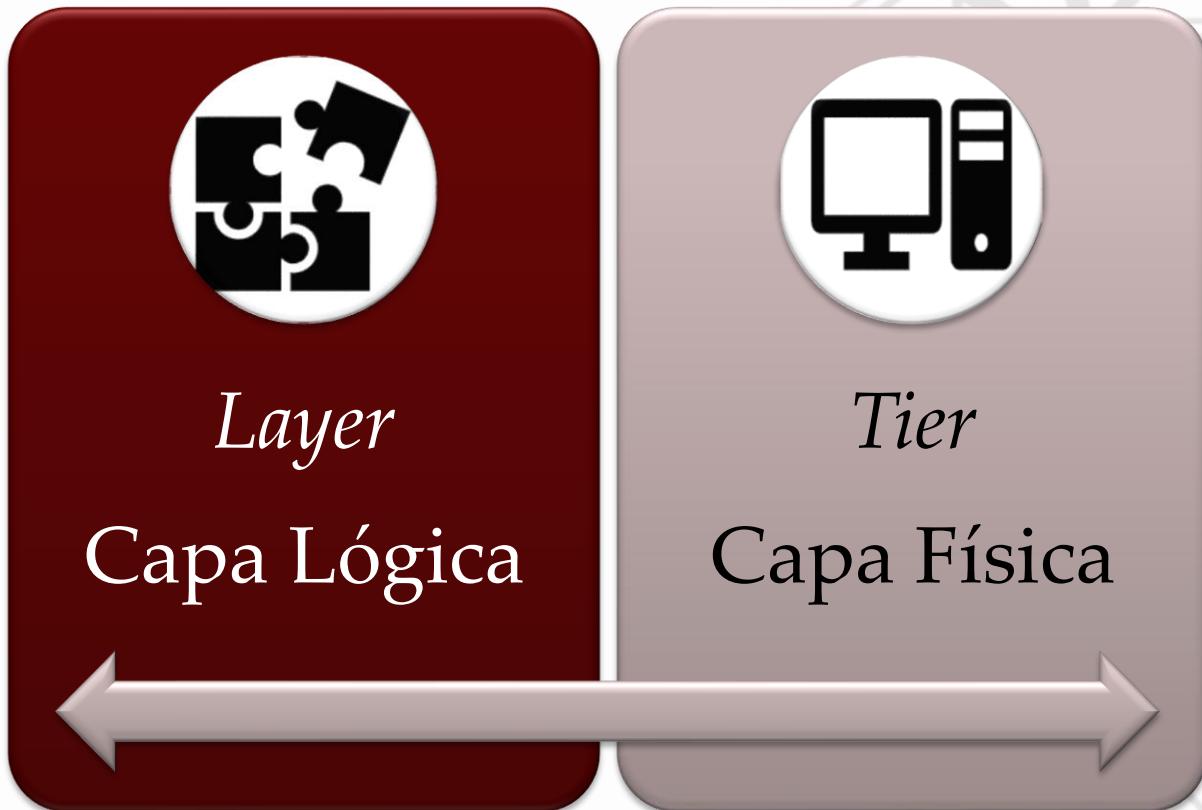
¿Qué es la Arquitectura software?

The IEEE logo is displayed in white text on a dark red circular background.

IEEE

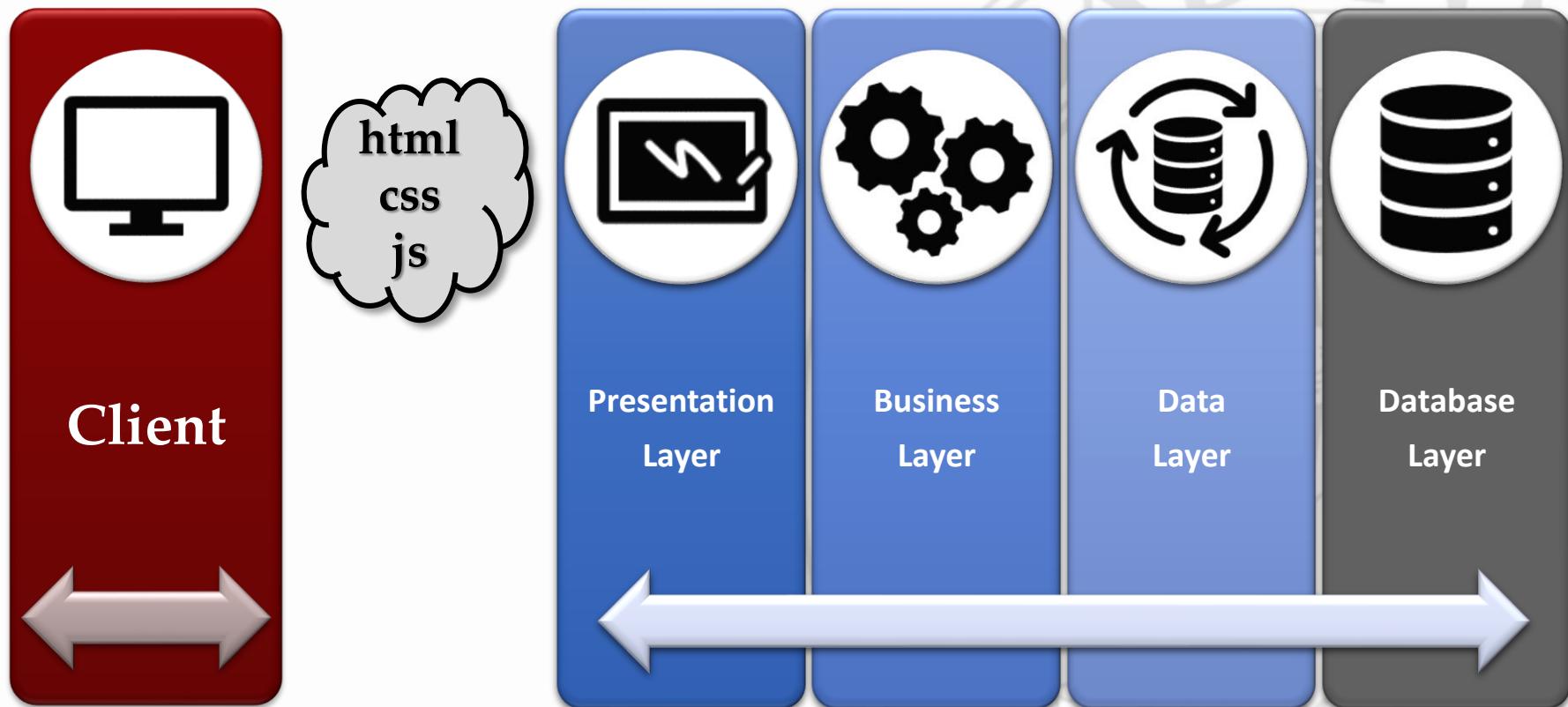
Estructura fundamental de un sistema. Comprende sus **componentes**, sus **relaciones** con otros, su **entorno** y las **principales guías** de su diseño y evolución





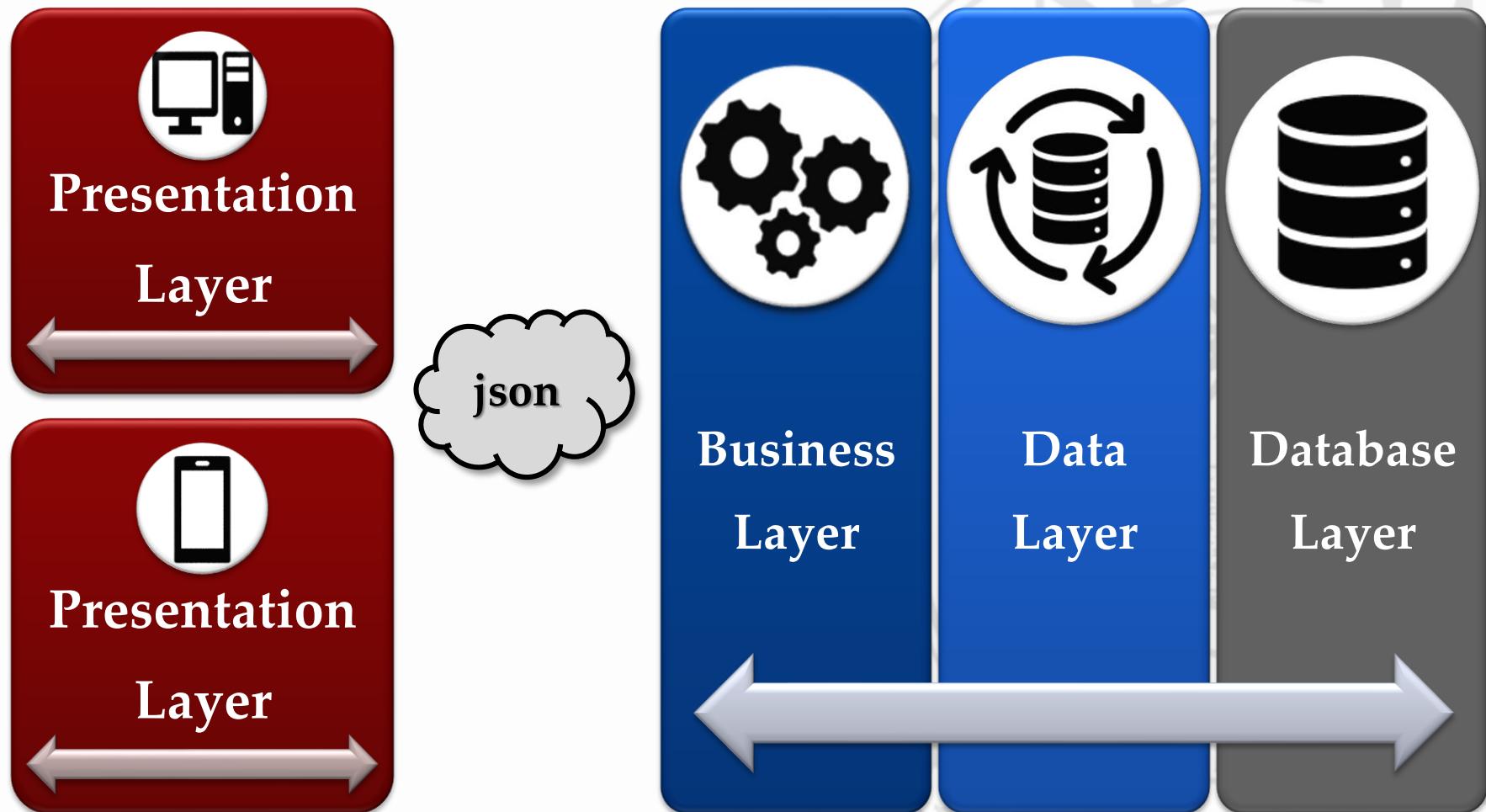
Arquitectura por Capas

Aplicación de Múltiples Páginas

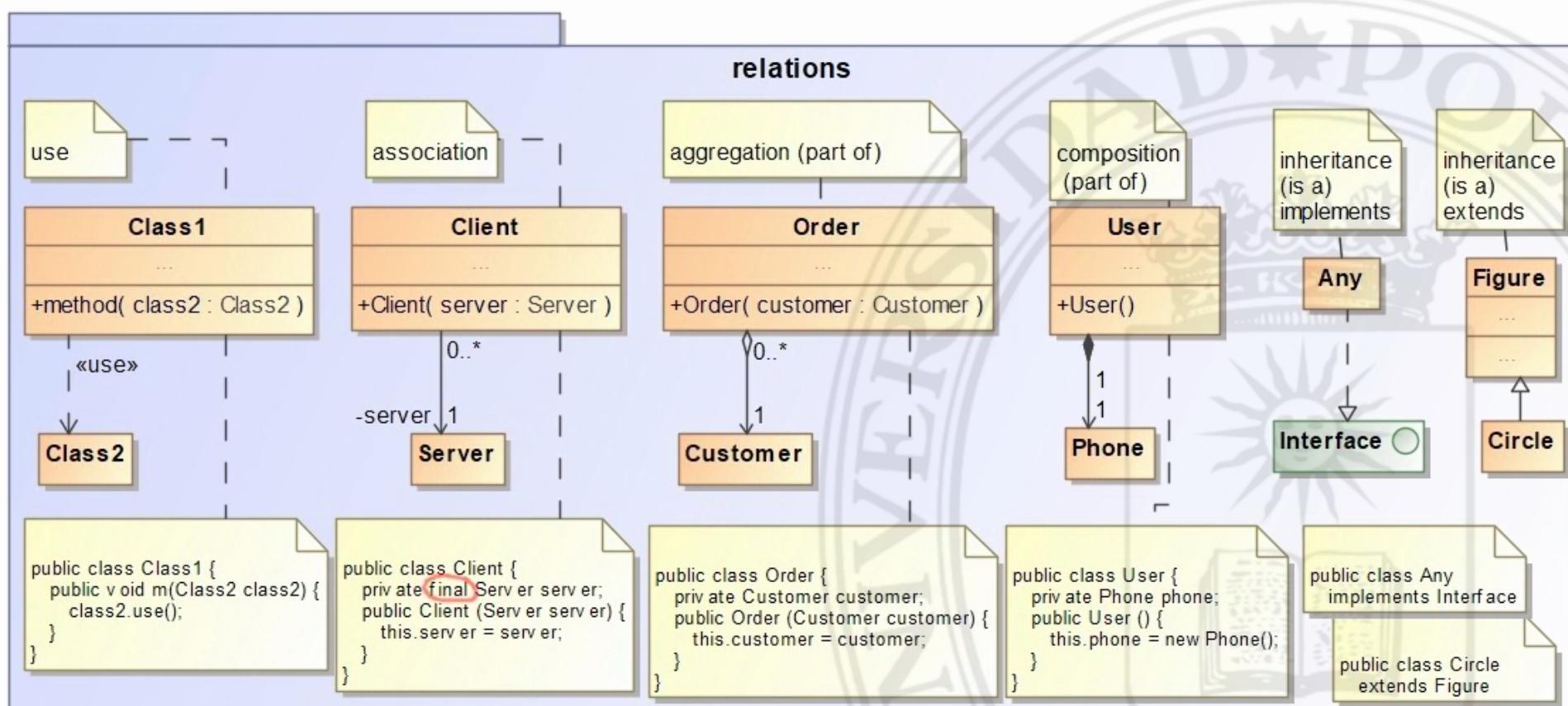


Arquitectura por Capas

Aplicación de Página Única (*SPA*)

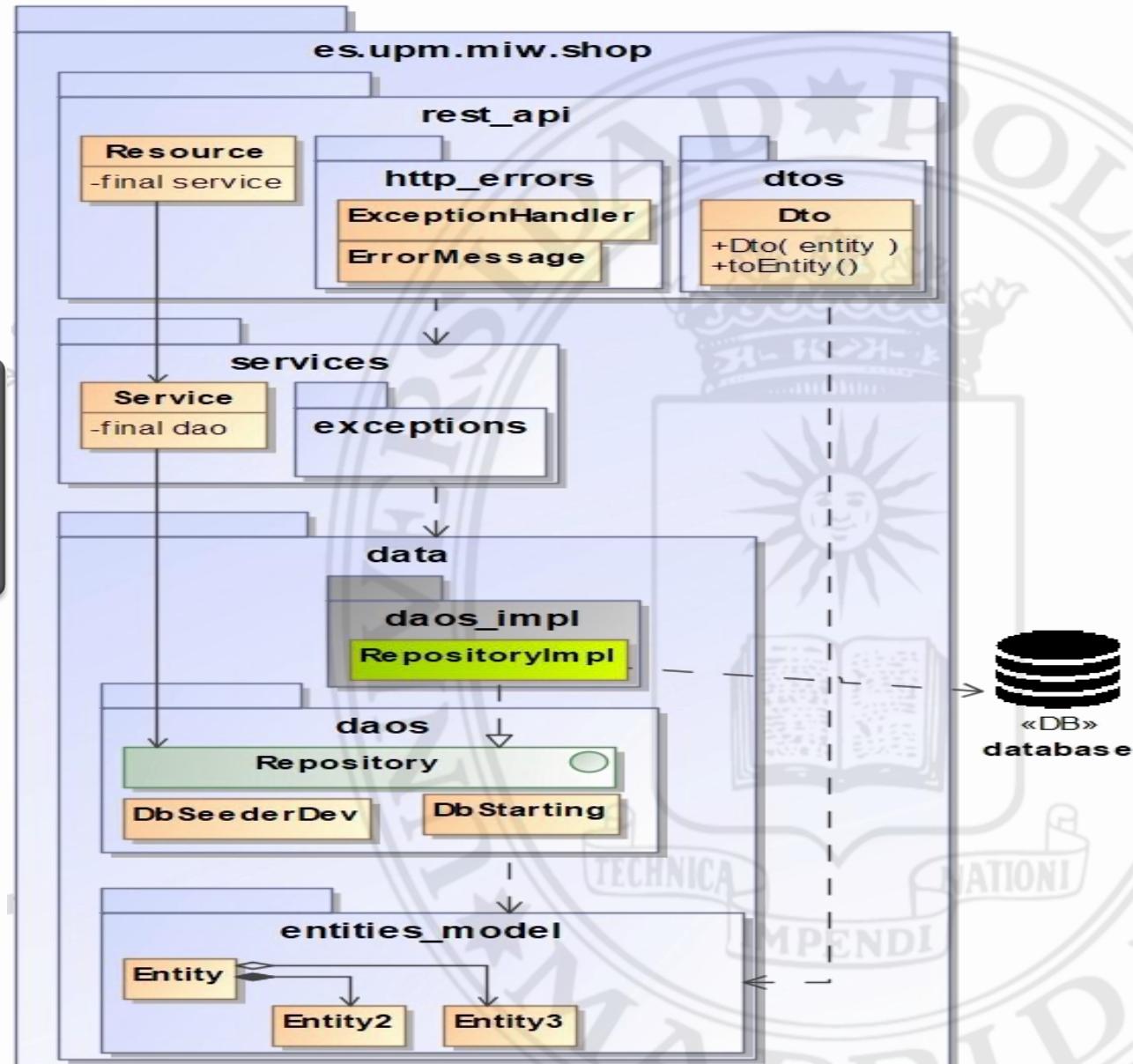


UML



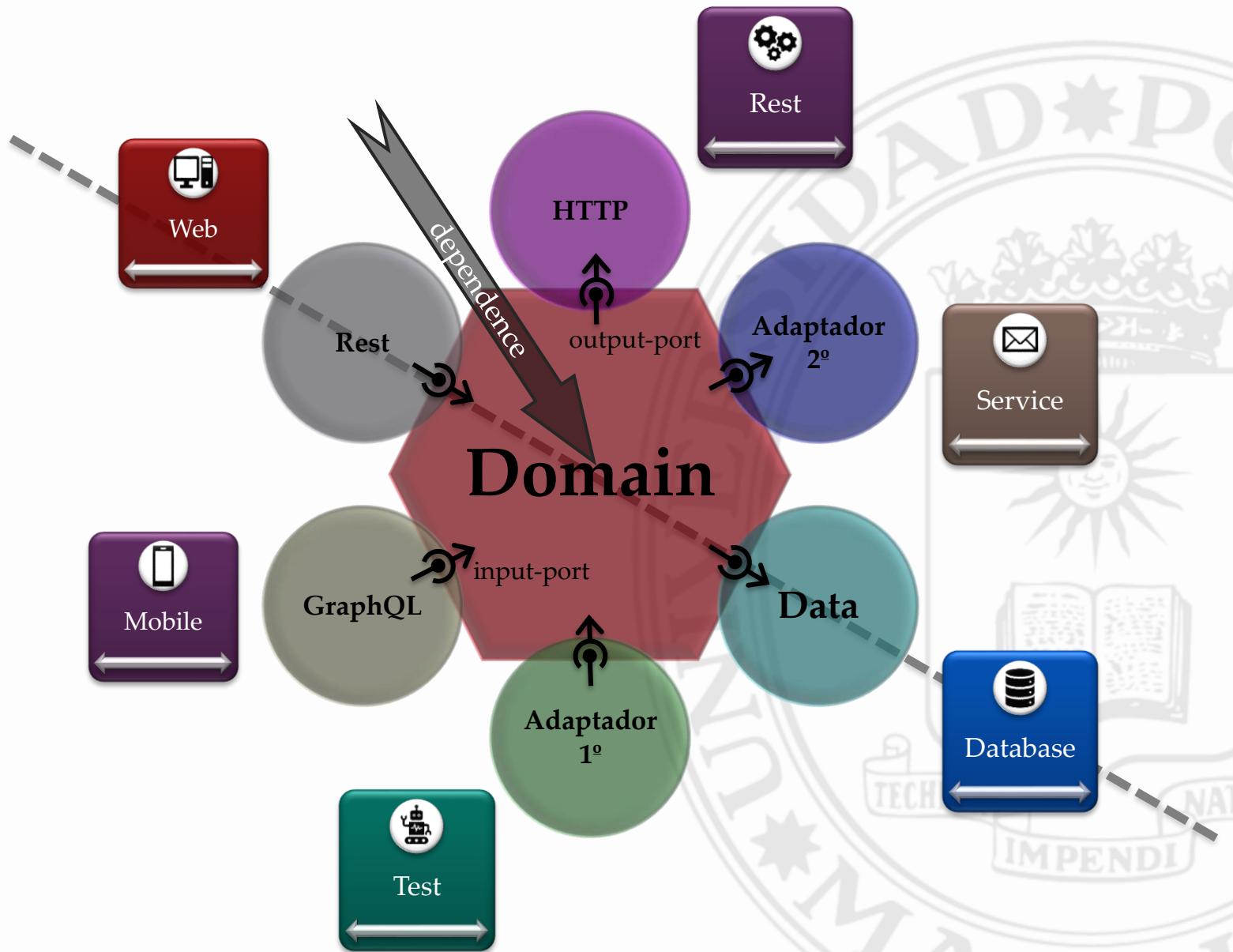
Arquitectura por Capas

Aplicación de Página Única (*SPA*)



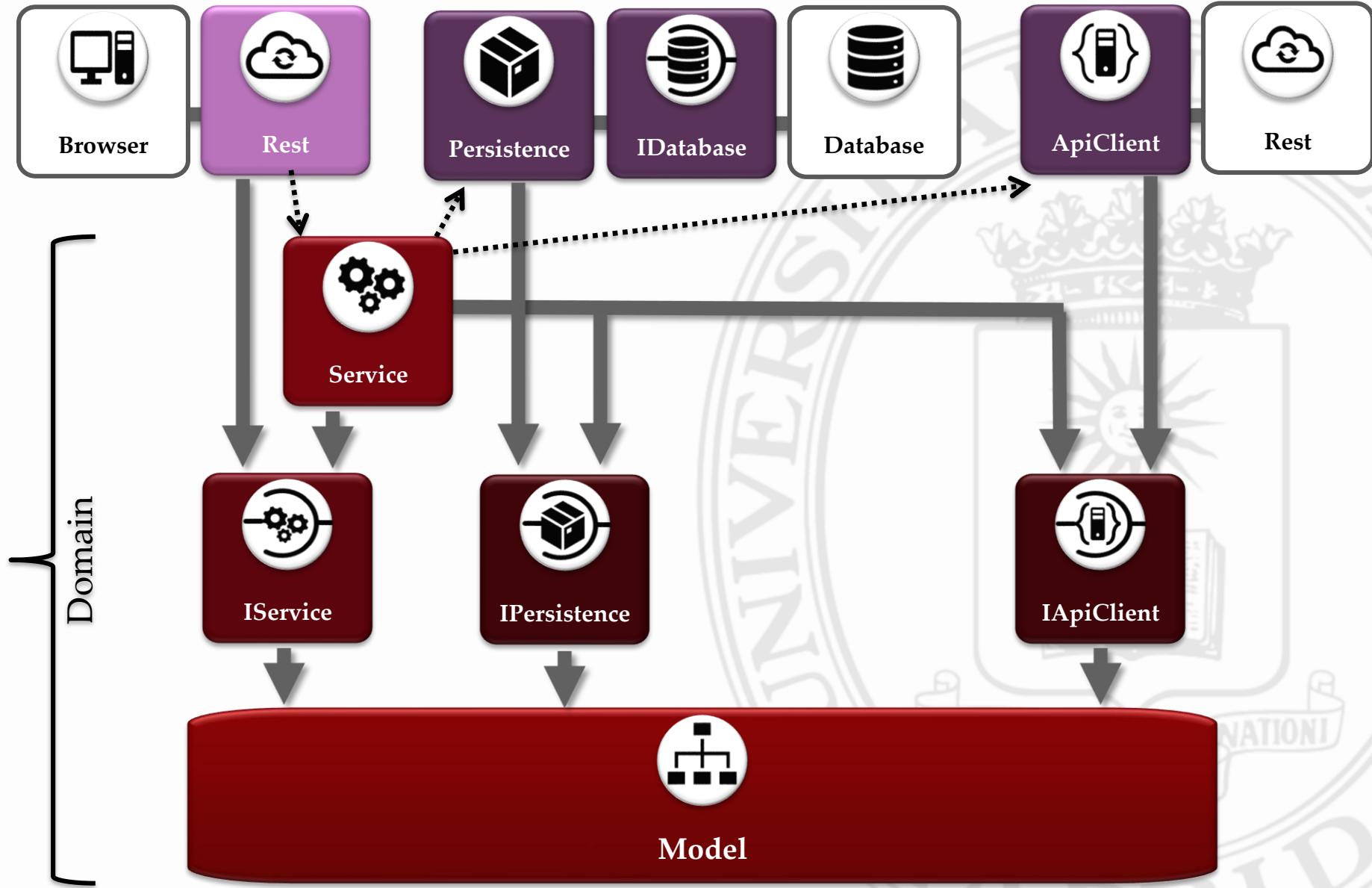
Hexagonal Architecture

Alistair Cockburn



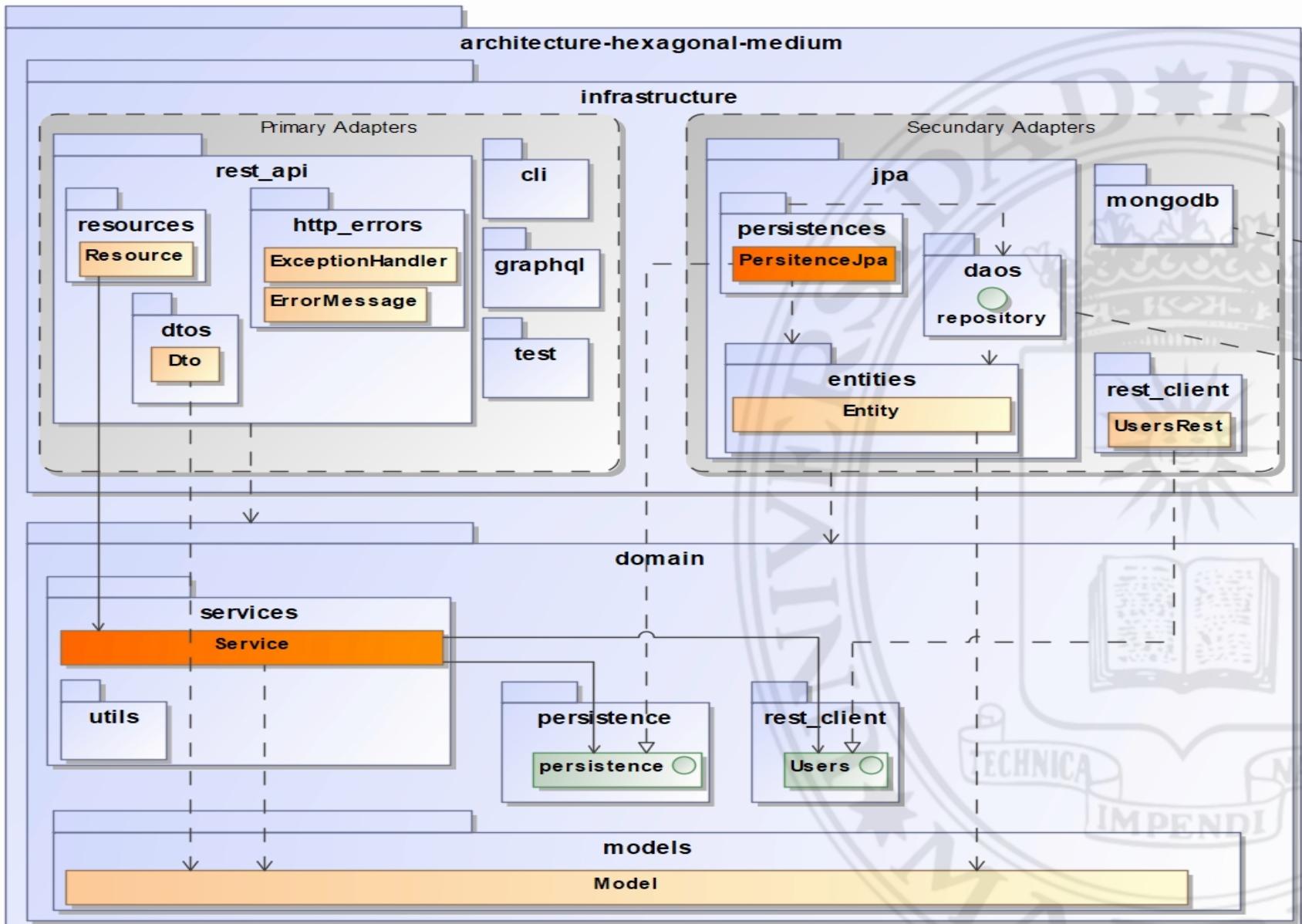
Hexagonal Architecture

Dependencias



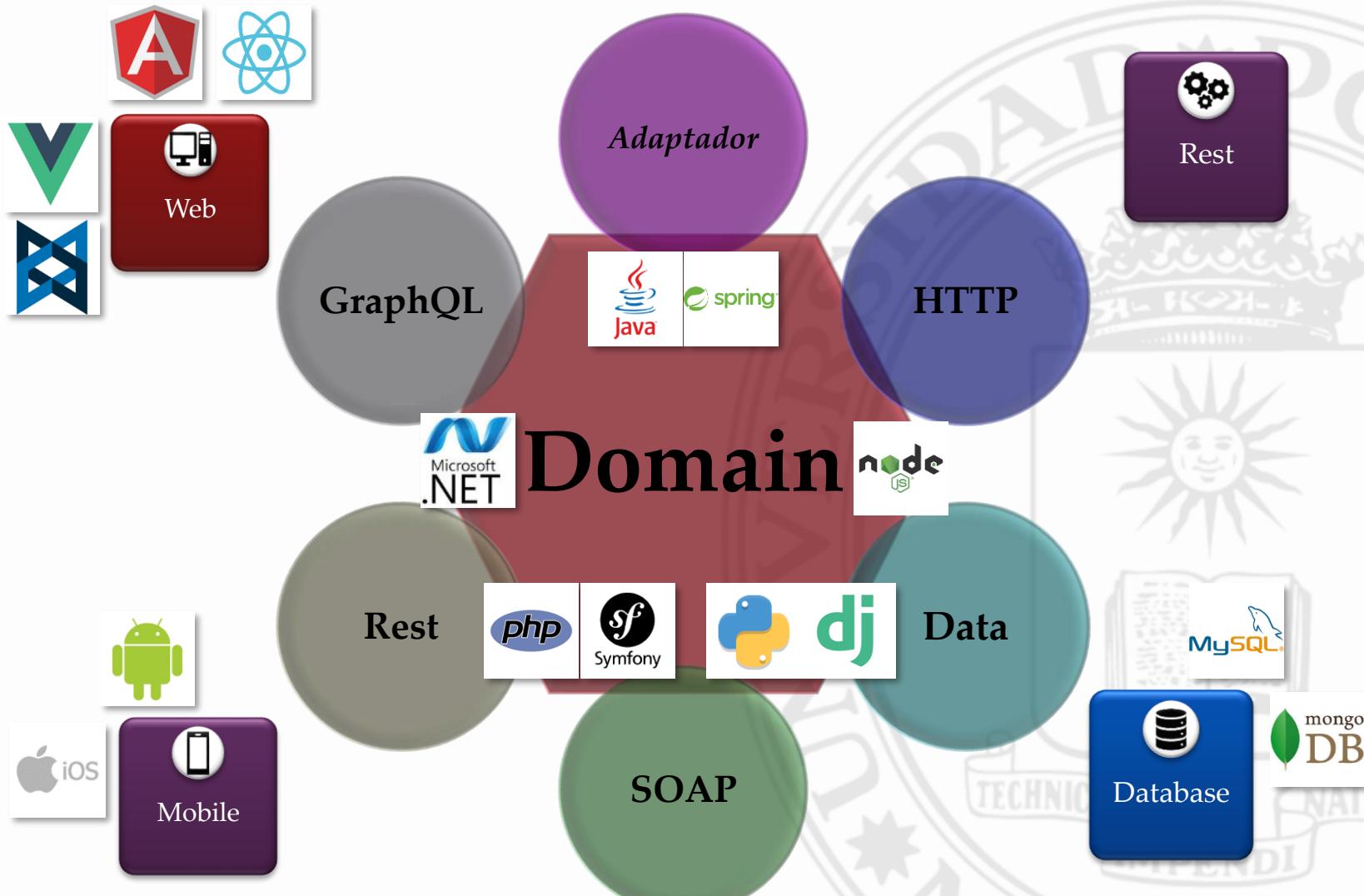
Arquitectura

Paquetes

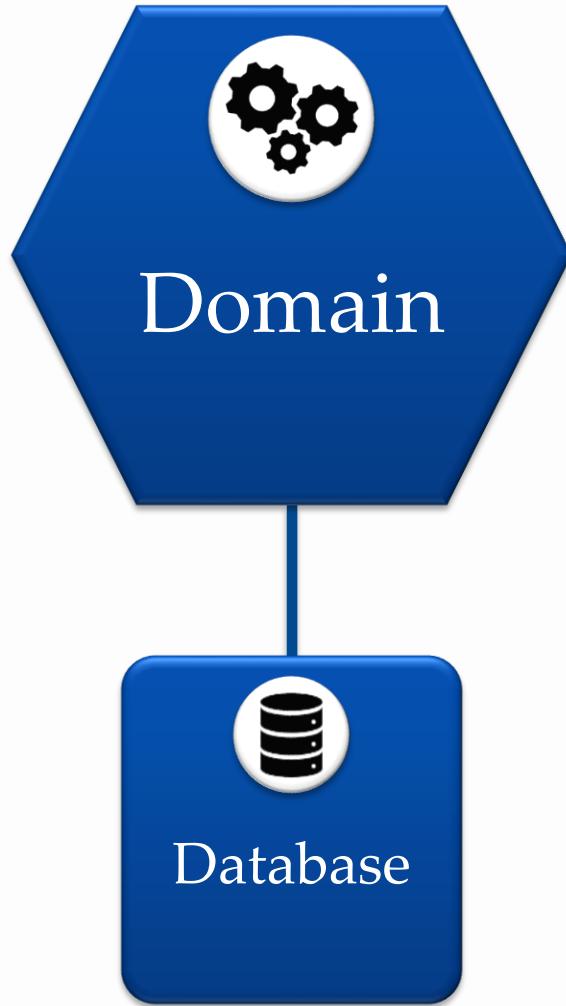


Arquitectura Hexagonal

Tecnologías



Aplicación Monolítica



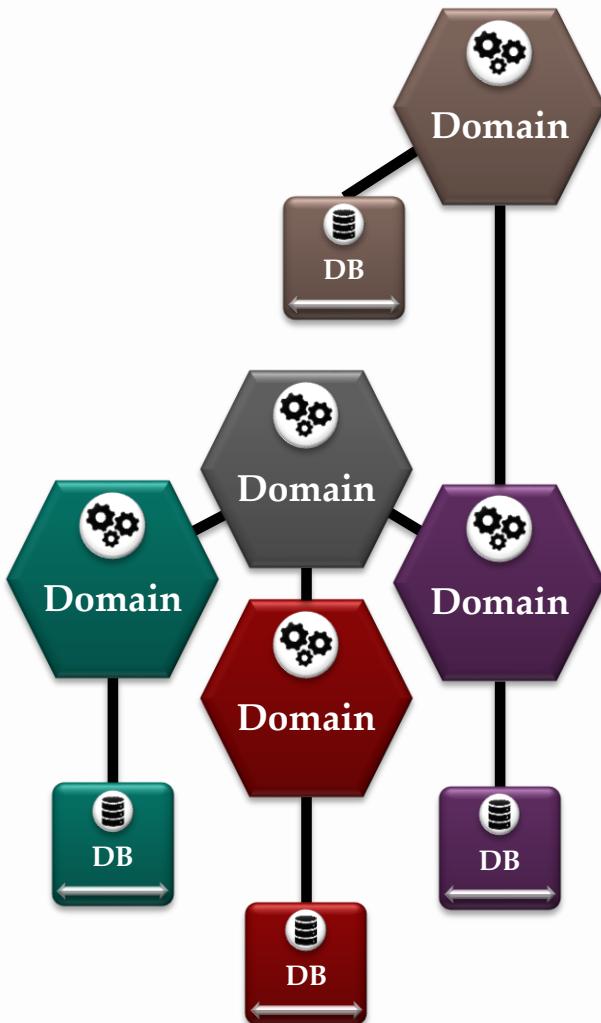
Ventajas

- Un solo artefacto: fácil despliegue
- Fácil gestión de versiones

Inconvenientes

- Complejidad aumenta en el tiempo: perdida de producción
- Grandes equipos de desarrollo: especialización y complejidad en la interacción entre los equipos
- Replicación costosa con difícil balanceo de carga
- Grandes BD: migraciones complejas
- Difícil reusabilidad de partes del proyecto
- Una sola tecnología y difícil migración tecnológica

Microservicios



Ventajas

- **Fuertemente modular.**

- Permite tener equipos más pequeños, multidisciplinarios.
- Filosofía de productos y no proyectos.
- Descentralización de BD y elimina la integración de BD.

- **Despliegues independientes.**

- Permite la evolución rápida y con menor riesgo.
- Replicar despliegues y balanceo de carga.
- Reusabilidad en diferentes proyectos.

- **Diversidad tecnológica.**

- Permite cambiar de tecnología en la evolución del proyecto con nuevos servicios.

Inconvenientes

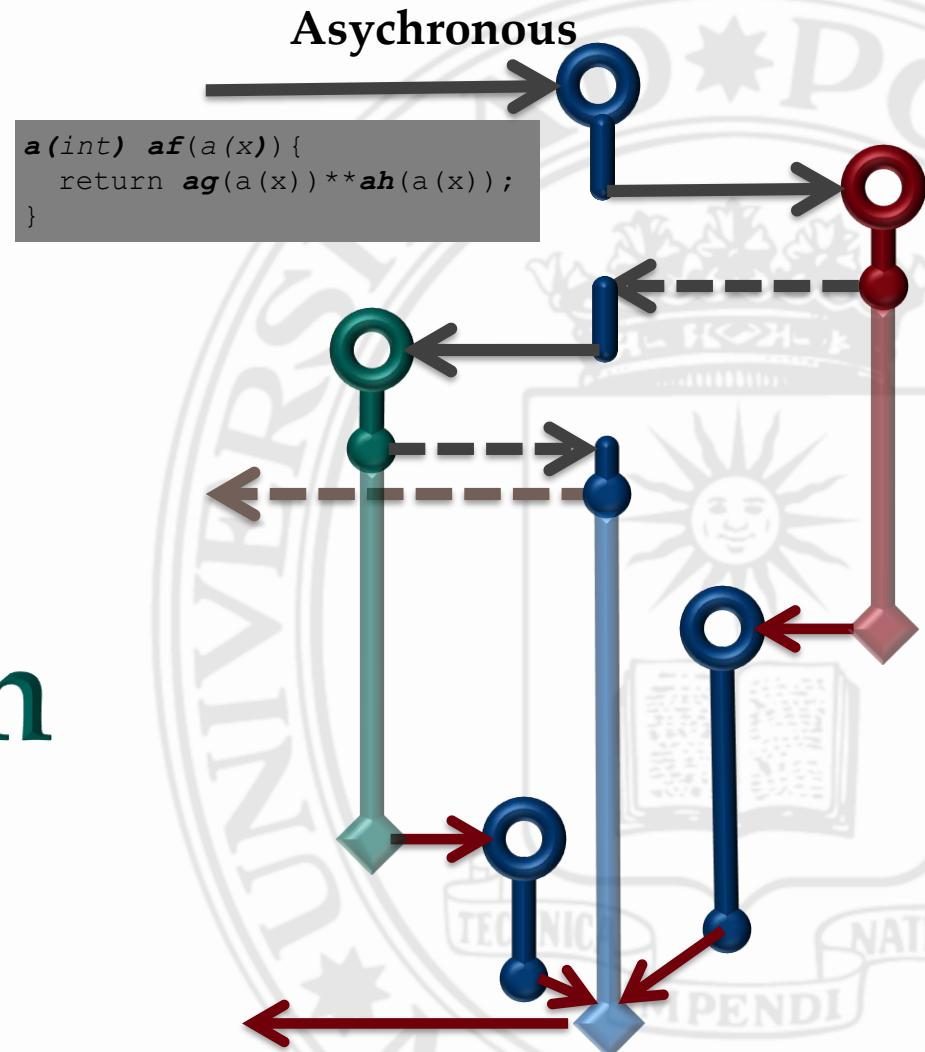
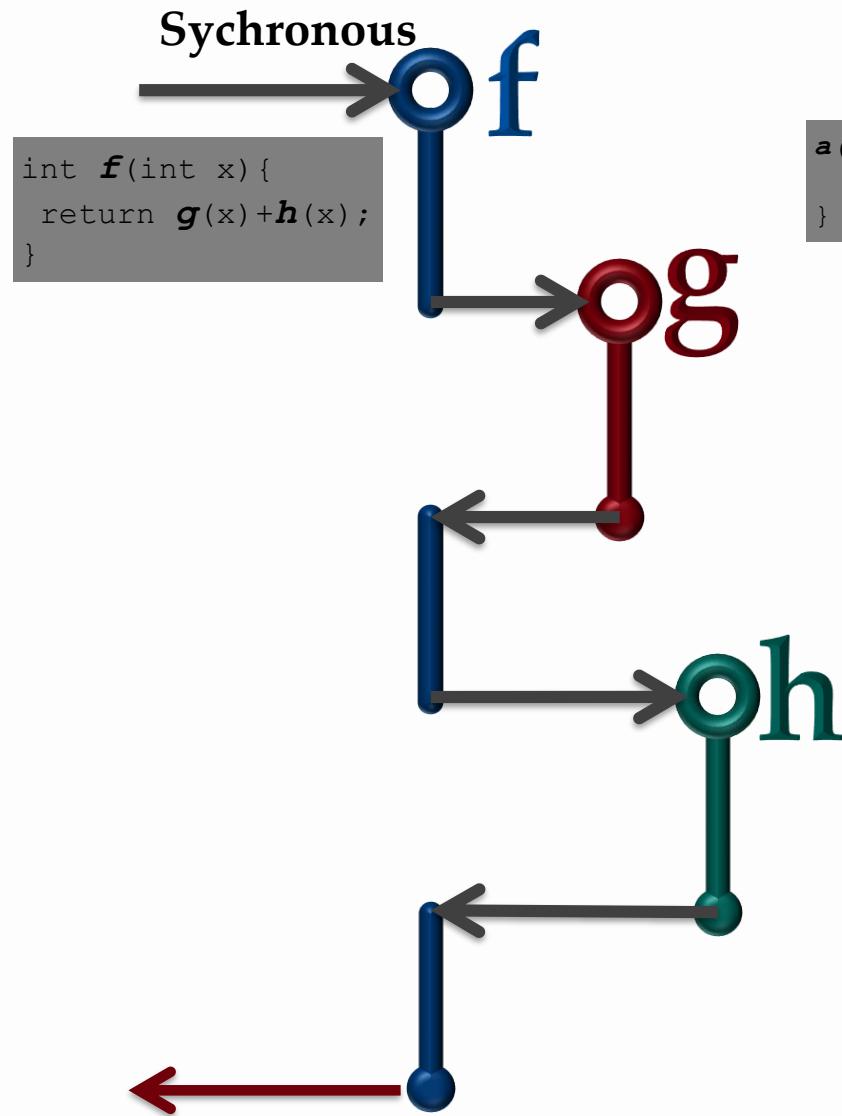
- **Distribución.** Al ser un sistema distribuido, es más complejo su despliegue

- **Consistencia.** Es más complicado mantener la consistencia del sistema

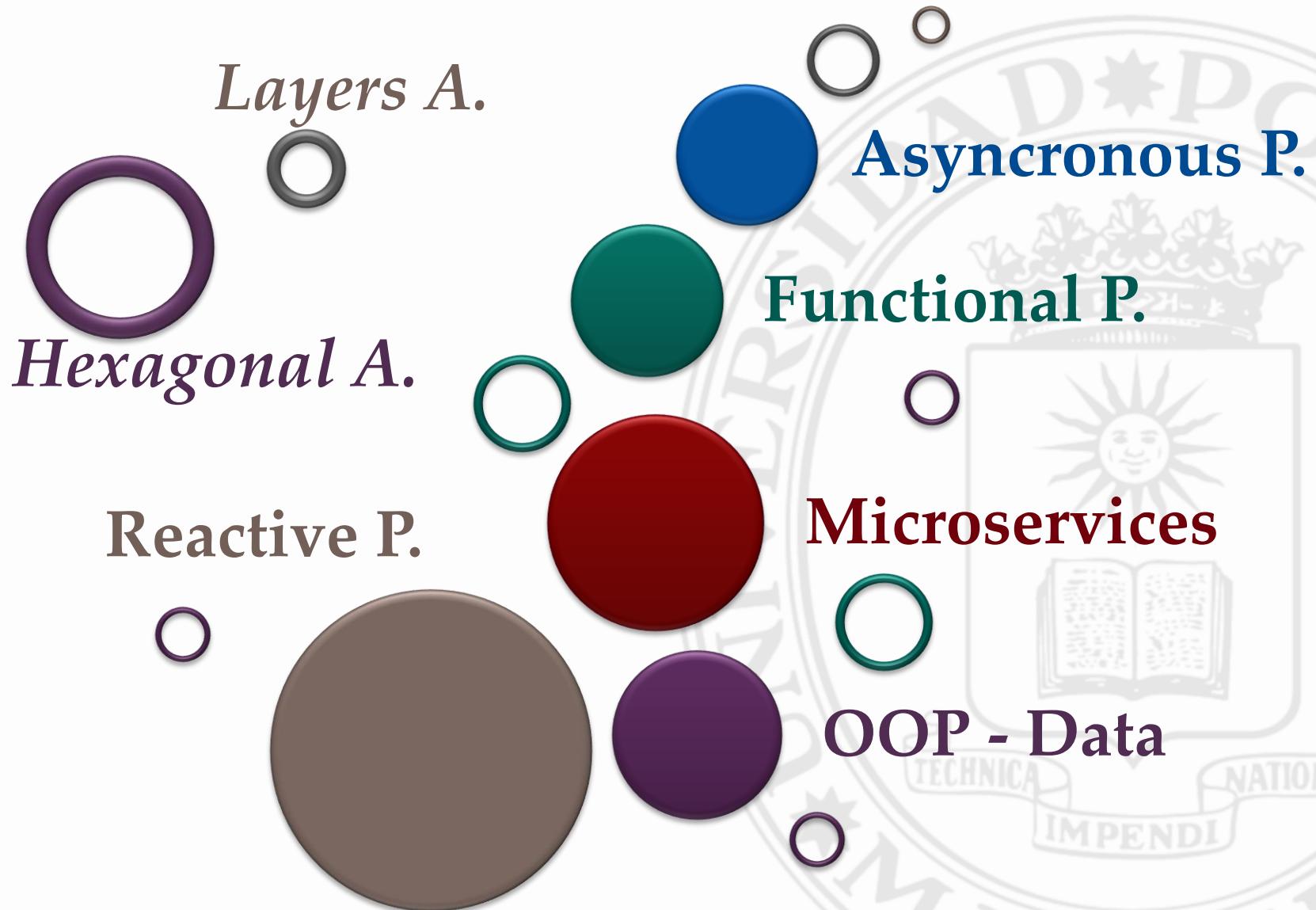
- **Complejidad operacional.** Se necesita acceder a varios servicios para realizar una operación

Arquitectura dirigida por Eventos

Síncrono - Asíncrono



Tendencias...



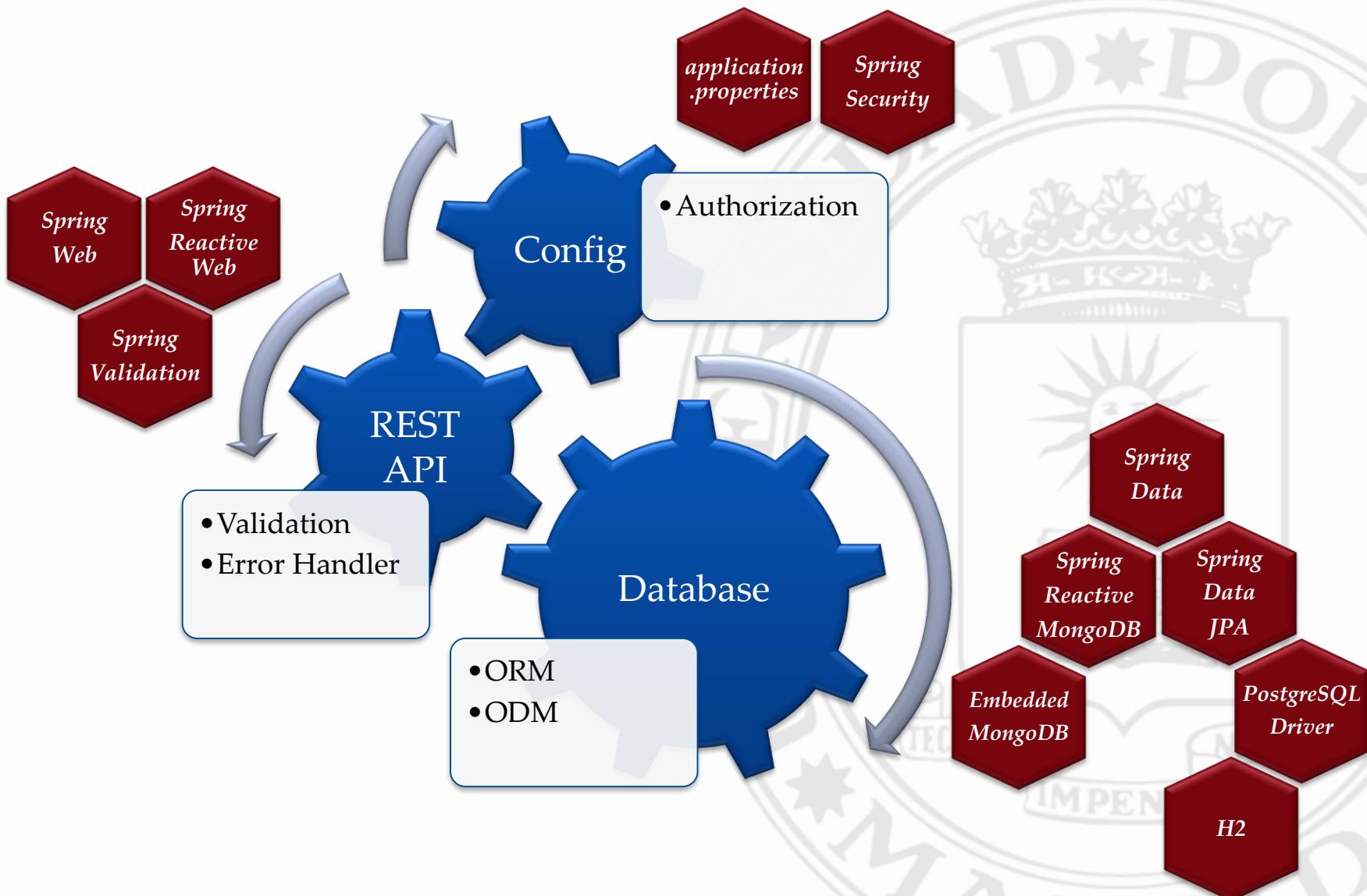
Spring

<https://spring.io/>

Spring

Spring es un framework ligero de código abierto, creado por Rod Jhonson, para facilitar el desarrollo de Aplicaciones Empresariales modernas (*Java o kotlin*)

Spring Web



Spring

<https://spring.io/>

Spring Boot

- Construye y arranca rápidamente la aplicación con una configuración inicial mínima de Spring.

Spring Framework

- Proporciona un modelo completo de programación y configuración para aplicaciones empresariales modernas.

Spring Data

- Proporciona acceso a diferentes tipos de bases de datos.

Spring Security

- Protege la aplicación con diferentes sistemas de autenticación y autorización.

Spring Cloud

- Proporciona herramientas que implementan los patrones comunes de sistemas distribuidos.

Spring Cloud Data Flow

Spring for GraphQL

Spring Integration

Spring REST Docs

Spring HATEOAS

Spring Batch

...

Spring Framework

<https://spring.io/projects/spring-framework>

Spring Reactive

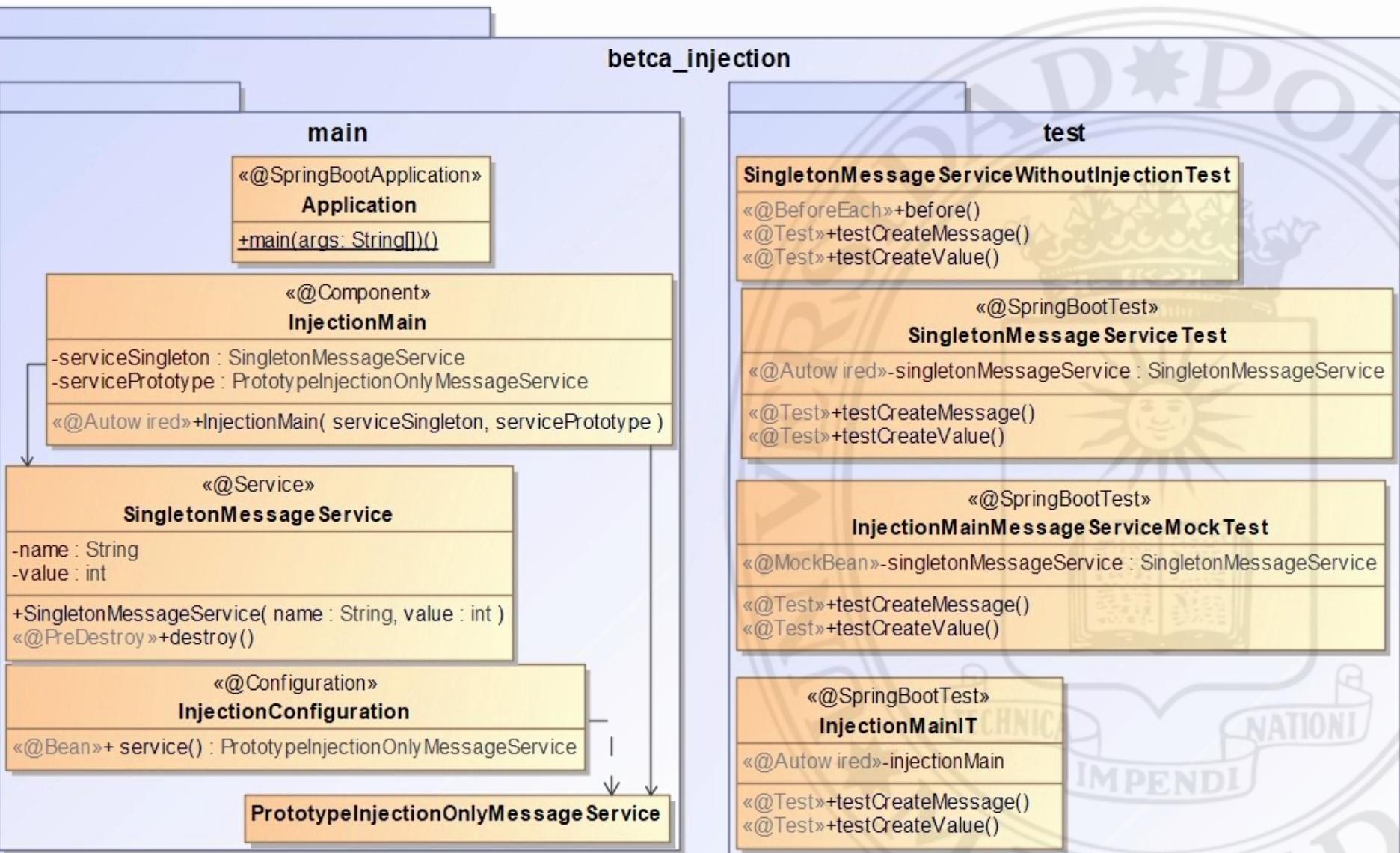
- Web asíncrona, no bloqueante.
- Pocos hilos (procesadores multi-core) pero masivo de conexiones.
- Contenedor Servlets 3.1+: *Netty*.
- Repositorios: *Reactive MongoDB, Cassandra, Redis, Couchbase, R2DBC* (*Postgres, MySQL, SQL Server*)

Spring MVC

- Web síncrona, bloqueante.
- Un hilo por petición.
- Contenedor Servlets: *Tomcat, Jetty...*
- Repositorios: *JDBC (Postgres, MySQL, SQL Server), JPA, MongoDB*.

Spring Framework

Inyección de dependencias



Inyección de dependencias



<https://github.com/miw-upm/betca-spring>

- **betca-injection**
- <https://start.spring.io/>

Inyección de Dependencias en práctica

- Crear un servicio con un método que devuelve un *String* con el día de la semana, crear un test para probarlo (*LocalDate.now().getDayOfWeek().toString()*)
- Crear un componente que utiliza el servicio anterior y tiene un método que indica si es fin de semana, crear *test unitario* y de *integración*.
- Añadir la propiedad “*miw.author=****” con el nombre del autor en el fichero *application.properties* y añadir al servicio un segundo método que devuelve un *String* con el nombre del autor, ampliar test de prueba.

Persistencia

Spring Framework

JTA

- Java Transaction API
- Soporta el manejo de transacciones tanto de forma declarativa como programado.

JDBC

- Java Database Connectivity.
- Nos proporciona una capa de abstracción para acceder mediante SQL.
- `String name = jdbcTemplate.queryForObject("SELECT name FROM actor WHERE id = ?", 1212L);`

R2DBC

- Reactive Relational Database Connectivity.
- Nos da acceso a BD SQL utilizando patrones reactivos.
- `Flux<String> names = client.sql("SELECT name FROM person") .all();`

ORM

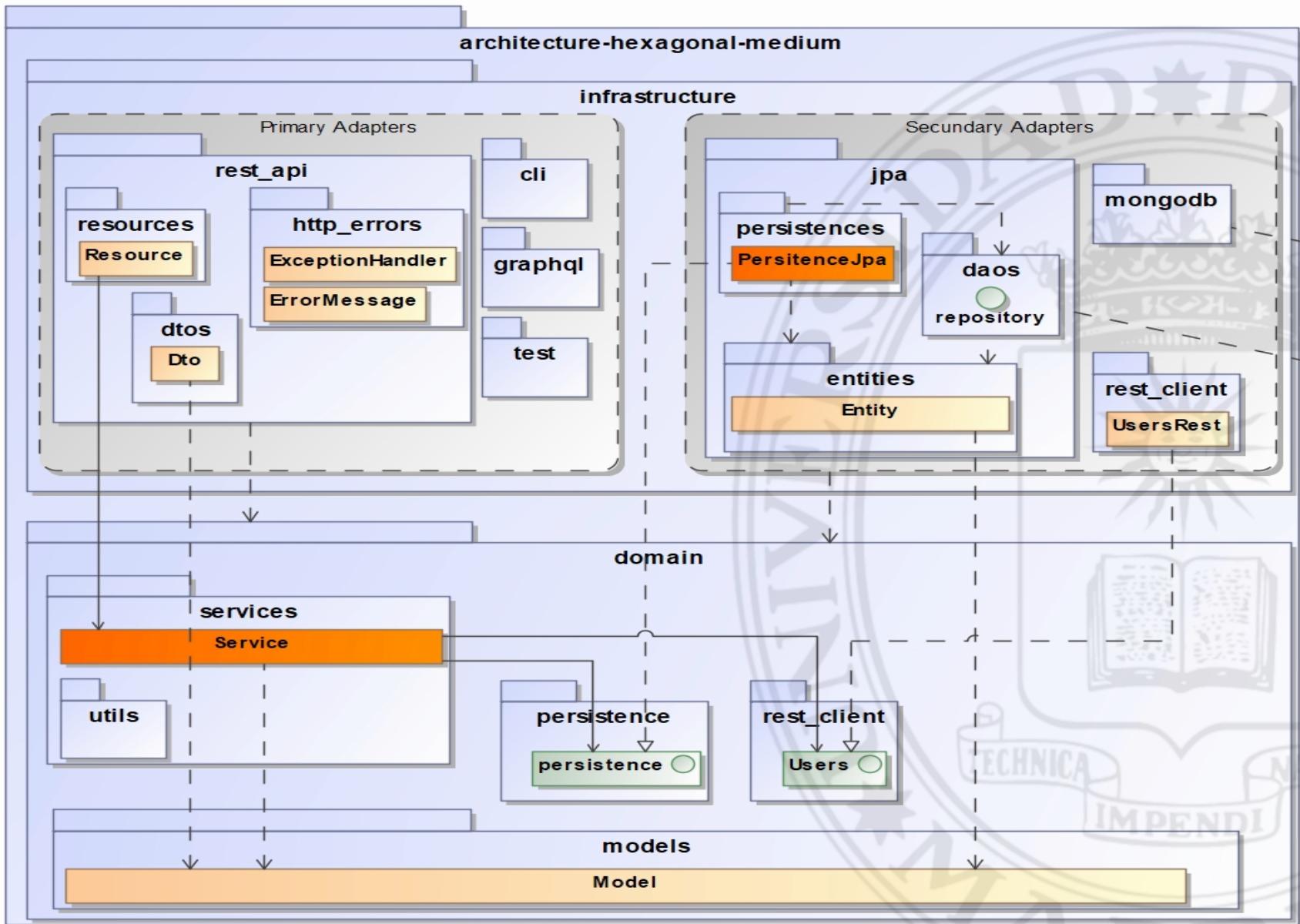
- Object Relational Mapping
- Soporta Java Persistence API (JPA) y Hibernate nativo

OXML

- Object-XML Mapping
- Nos proporciona la serialización XML (Marshaller & Unmarshaller)

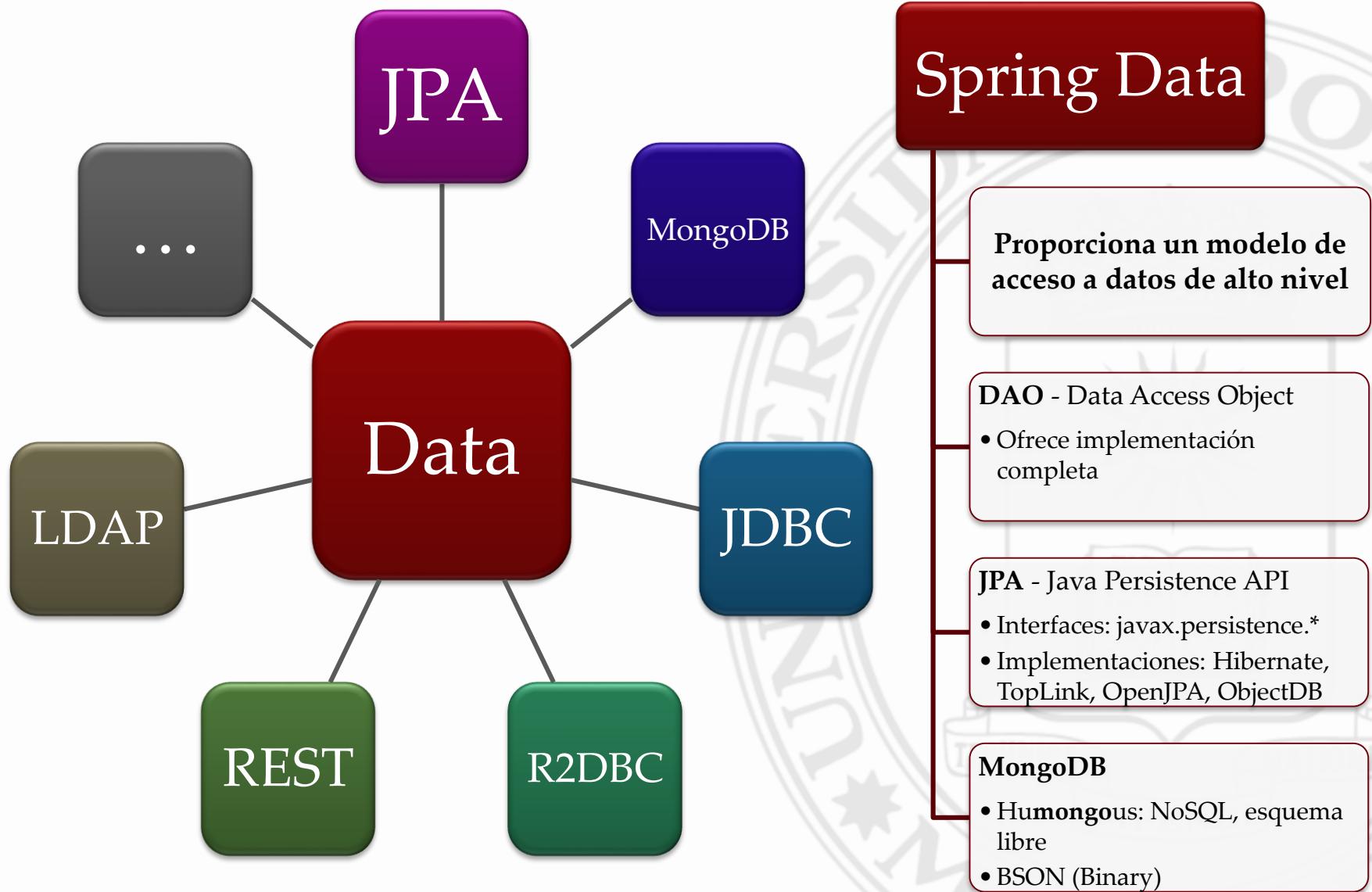
Arquitectura

Paquetes



Persistencia

Spring Data



Spring Data JPA. Dependencias



Dependencies

[ADD DEPENDENCIES... CTRL + B](#)

Lombok DEVELOPER TOOLS

Java annotation library which helps to reduce boilerplate code.

Spring Data JPA SQL

Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

PostgreSQL Driver SQL

A JDBC and R2DBC driver that allows Java programs to connect to a PostgreSQL database using standard, database independent Java code.

H2 Database SQL

Provides a fast in-memory database that supports JDBC API and R2DBC access, with a small (2mb) footprint. Supports embedded and server modes as well as a browser based console application.

MariaDB Driver SQL

MariaDB JDBC and R2DBC driver.

MySQL Driver SQL

MySQL JDBC and R2DBC driver.

Oracle Driver SQL

A JDBC driver to Oracle.

src/*/resources

- application-dev.properties
- test.properties
 - *@TestPropertySource*

Library: JAR

- Sin Application (NO *@SpringBootApplication*)
- JpaConfig
 - *@TestConfig*

Spring Data Database. Configuración

H2 (test.properties)

- **Database H2 Embedded**

```
spring.datasource.driver-class-name=org.h2.Driver  
spring.datasource.url=jdbc:h2:mem:testdb  
#spring.datasource.username=sa  
#spring.datasource.password=
```

Postgresql (application-dev.properties)???

- **Database Postgresql external**

```
spring.jpa.hibernate.ddl-auto=update  
spring.datasource.url=jdbc:postgresql://localhost:5432/betca  
spring.datasource.username=postgres  
spring.datasource.password=
```

Postgresql (application-prod.properties)

- **Database Postgresql in server**

```
spring.jpa.hibernate.ddl-auto=update  
spring.datasource.driverClassName=org.postgresql.Driver  
spring.datasource.url=${DATABASE_URL}
```

Spring Data

Proyecto Lombok

```
«@Builder»  
«@AllArgsConstructor»  
«@Data»  
«@NoArgsConstructor»  
Dto
```

```
-id : String  
-name : String  
-surname : String  
-email : String  
«@Singular»-tags : List<String>
```

```
Dto.builder().id("1").tag("tag1").tag("tag2").build()
```

Proyecto Lombok

<https://projectlombok.org/>

- @Data
 - @Getter
 - @Setter
 - @ToString
 - @EqualsAndHashCode
 - @RequiredArgsConstructor
- @NoArgsConstructor
- @AllArgsConstructor
- @EqualsAndHashCode.Include
 - Solo los referenciados
- @Builder
- @Singular
- @Value
- @Log

Spring Data

Proyecto Lombok

@Builder

@Data // **@ToString**, **@EqualsAndHashCode**, **@Getter**, and
@Setter on all non-final fields, **@RequiredArgsConstructor**

@EqualsAndHashCode(onlyExplicitlyIncluded = true)

@NoArgsConstructor

@AllArgsConstructor

```
public class DtoWithLombok {
```

@EqualsAndHashCode.Include // only id

private String id; //**@Setter(AccessLevel.NONE)** for exceptions

private String name;

private String surName;

private String email;

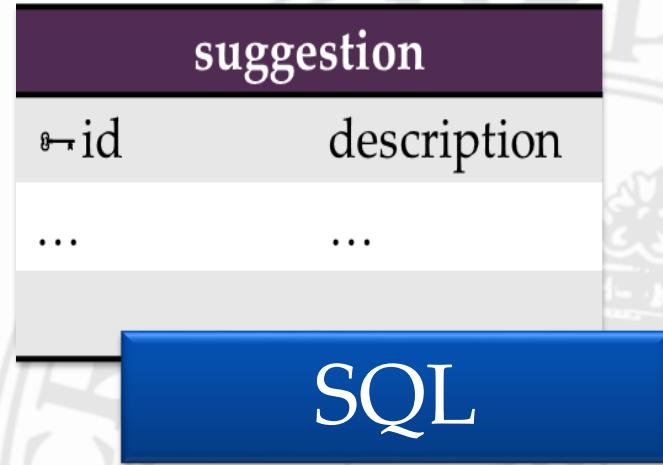
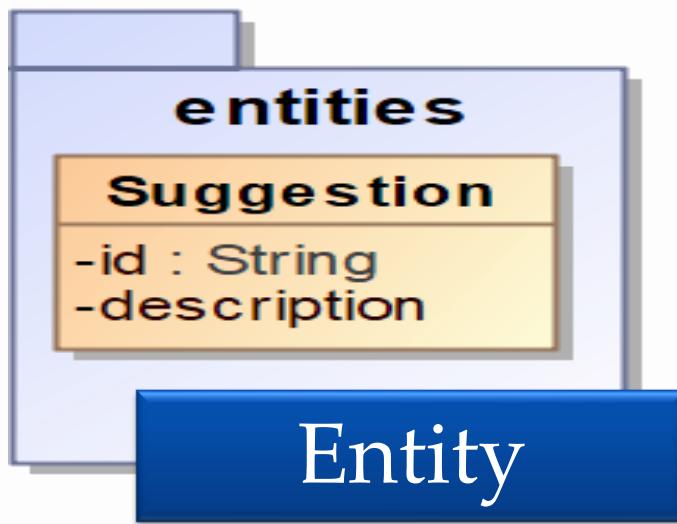
@Singular // .tag().tag().tag()

private List<String> tags;

```
}
```

Spring Data

Mapeo de objetos: sin relación



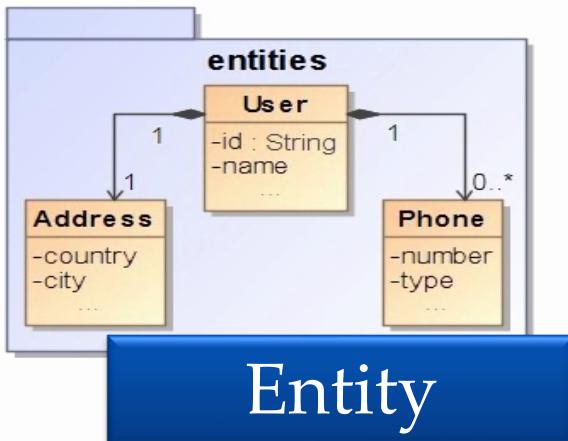
A screenshot of a code editor window titled "suggestion.json" shows the following JSON data:

```
1  [ ]  
2   { }  
3     "_id": "one-key",  
4     "description": "one-description"  
5   ,  
6   { }  
7     "_id": "two-key"  
8   ]
```

A blue button below the code editor is labeled "MongoDB".

Spring Data

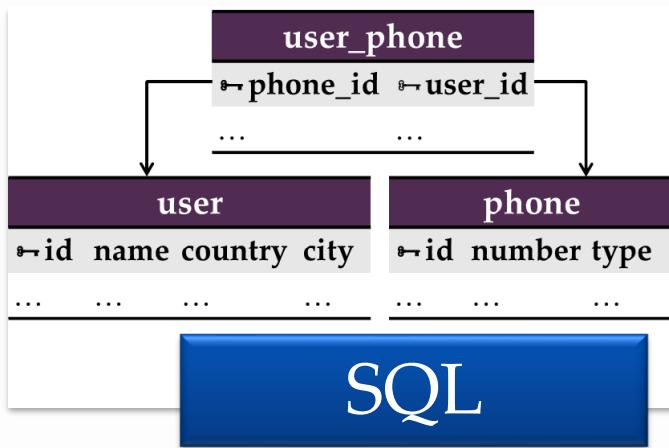
Mapeo de objetos: composición



SQL Table Structure:

user				
id	name	phones	country	city
...	...	[tinyblob]

SQL



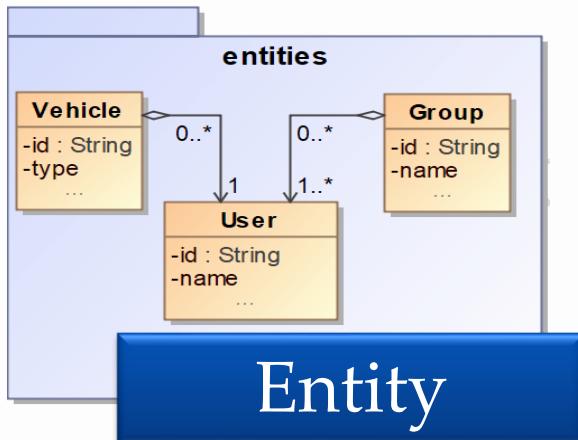
MongoDB JSON Document:

```
user.json x | 1 { 2   "_id": "key-1", 3   "name": "nombre", 4   "address": { 5     "country": "pa\u00eds", "city": "ciudad" 6   }, 7   "phones": [ 8     { "number": "6666666666", "type": "movil" }, 9     { "number": "999666999", "type": "casa" } 10  ], 11 }, 12   { 13     "_id": "key-2", 14     "address": { 15       "cou 16     } 17   } 18 }
```

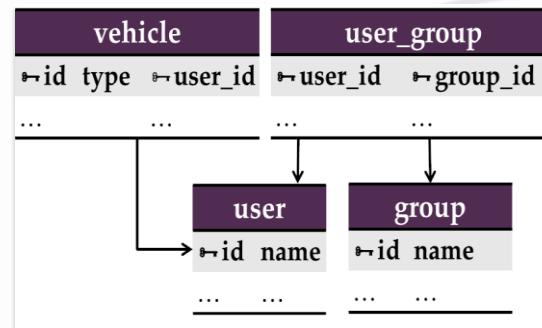
MongoDB

Spring Data

Mapeo de objetos: agregación



Entity



SQL

```
user.json
1 {
2     "_id": "key-1",
3     "name": "nombre",
4     "address": {
5         "country": "pais", "city": "ciudad"
6     },
7     "phones": [
8         { "number": "6666666666", "type": "movil" },
9         { "number": "999666999", "type": "casa" }
10    ]
11 },
12 {
13     "_id": "key-2",
14     "name": "nombre",
15     "address": {
16         "country": "pais"
17     }
18 }
```

A MongoDB document named user.json containing two user objects. Each user has an _id, name, address (with country and city), and phones (with number and type).

MongoDB

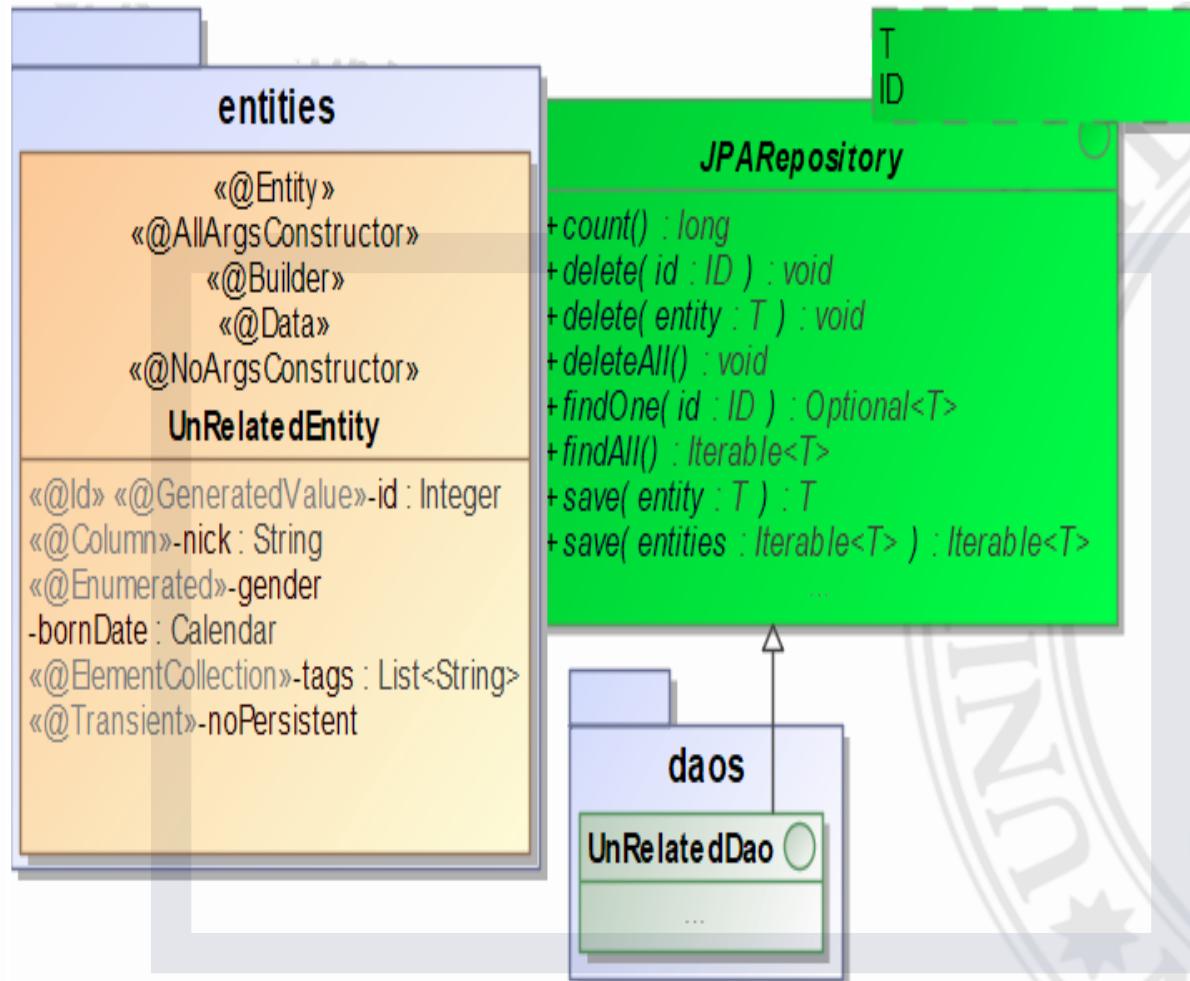
```
vehicle.json
1 {
2     "_id": "key-1",
3     "type": "tipo",
4     "user": DBRef("user", "user-key-1")
5 }

group.json
1 {
2     "_id": "key-1",
3     "name": "nombre",
4     "users": [
5         DBRef("user", "user-key-1" ),
6         DBRef("user", "user-key-2" )
7     ]
8 }
9
10
11
```

Two MongoDB documents: vehicle.json and group.json. vehicle.json contains an _id, type, and a user reference. group.json contains an _id, name, and a users array containing references to users.

MongoDB

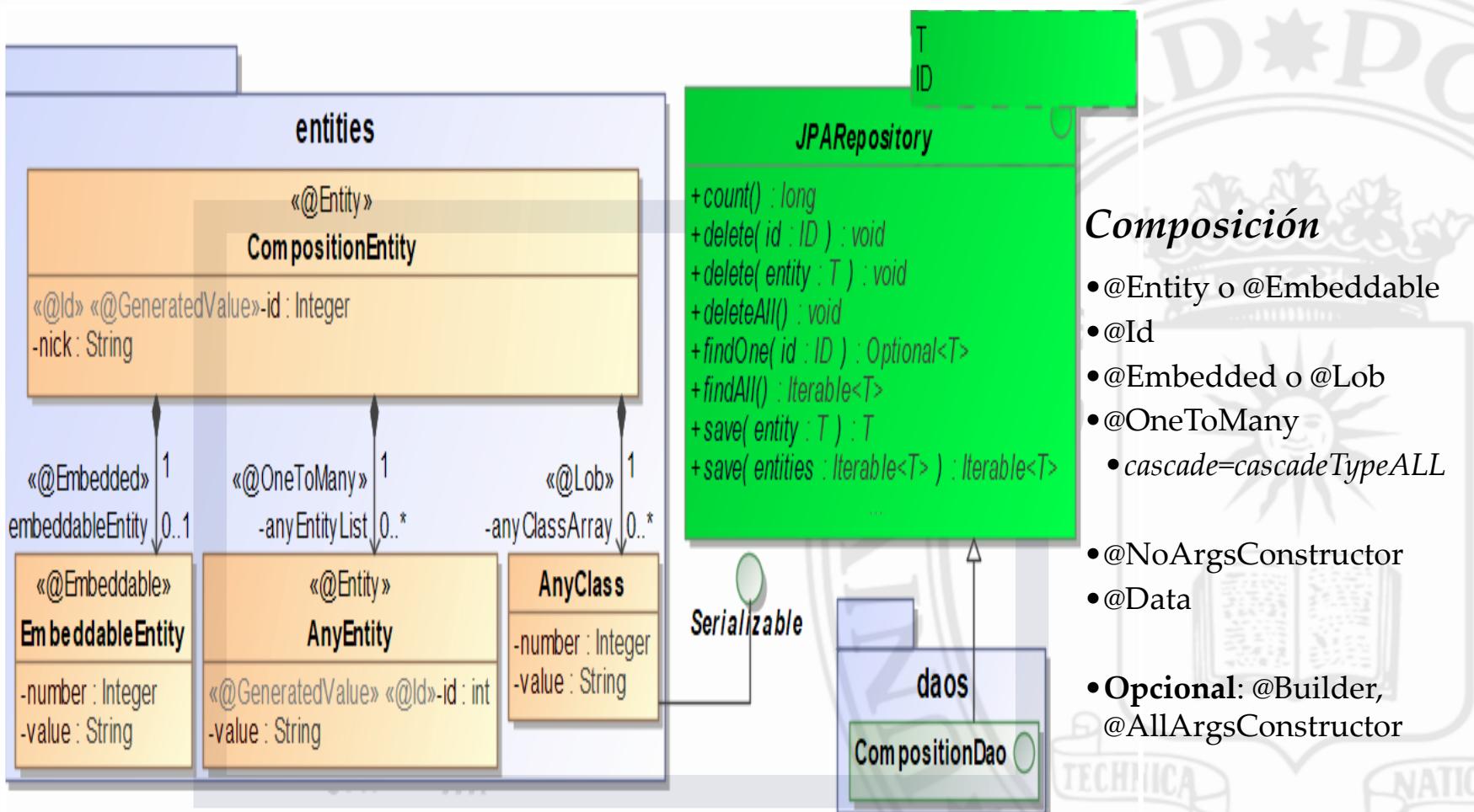
Spring Data JPA



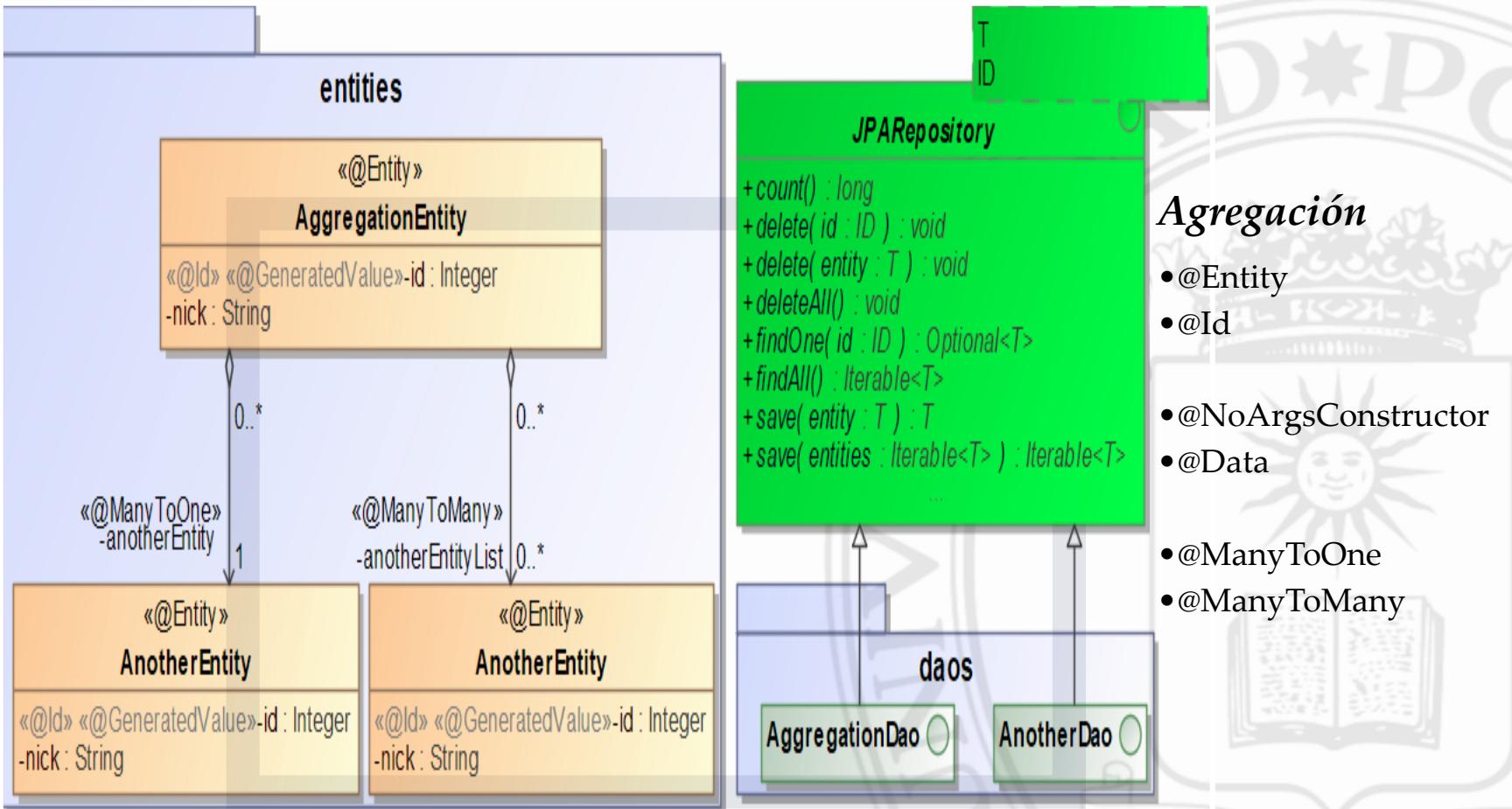
Sin Relación

- **@Entity**
- **@Id**
- **@NoArgsConstructor**
- **@Data**

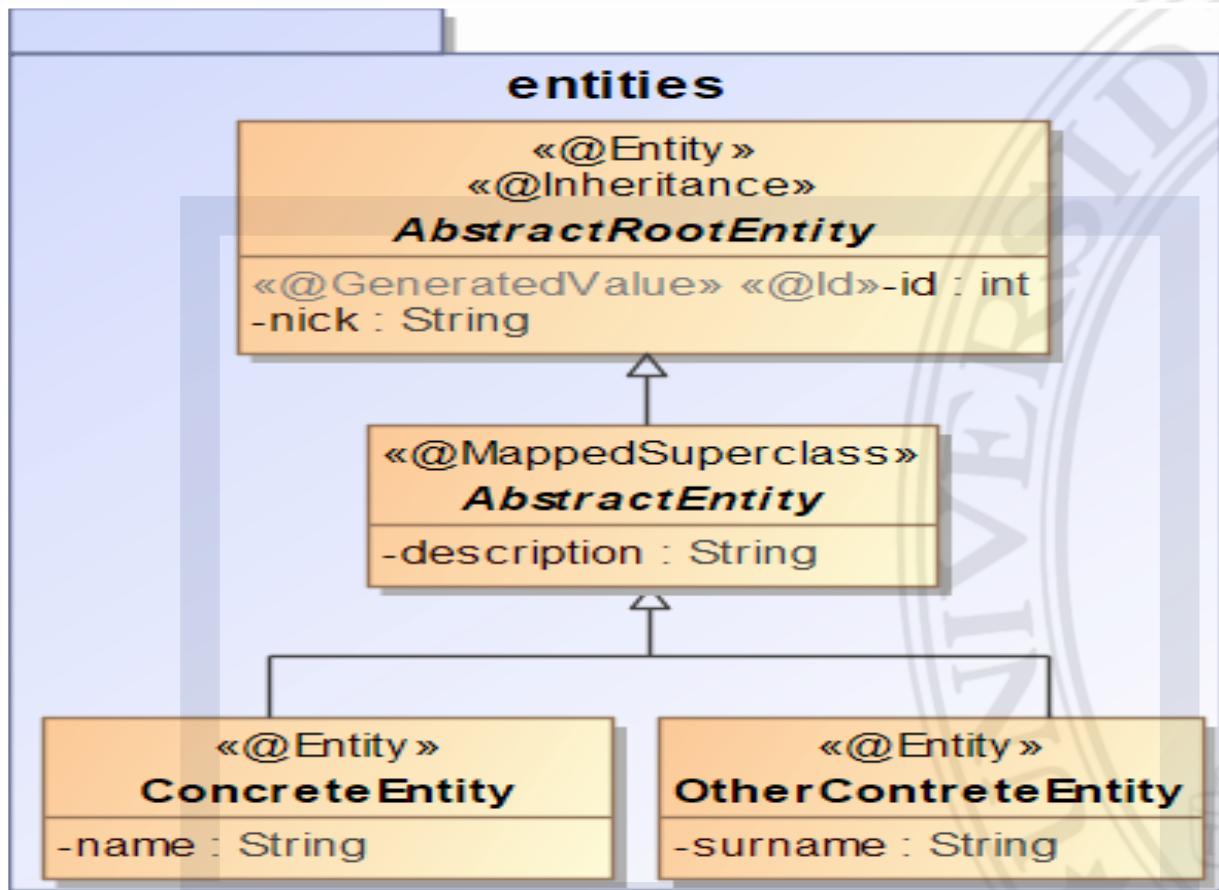
Spring Data JPA



Spring Data JPA



Spring Data JPA



Herencia

- `@Entity`
- `@Id`
- `@Inheritance`
 - `SINGLE_TABLE`
 - `JOINED`
- `@MappedSuperclass`

Spring Data

Postgresql. <https://www.postgresql.org/>

Bajarse el fichero *.zip

<https://www.enterprisedb.com/download-postgresql-binaries>

Descomprimir en el equipo, FUERA del proyecto

Inicializar las Bases de Datos

>bin> .\initdb -D ./data -U postgres -E UTF8

Arrancar – Parar

>bin> .\pg_ctl -D ./data -l logs start |>bin> .\pg_ctl -D ./data -l logs stop

Consola

>bin> chcp 1252 >bin> .\psql -U postgres

Crear Database “das” & “tpv”

Consola

- \? *Help*
- \l *Database list*
- \connect <db> <usr>
- \dt *Table list*
- \d <table> *describe*
- \q *exit*
- create database <db>;
- drop database <db>;
- \timing *activa*
- select * from <table>

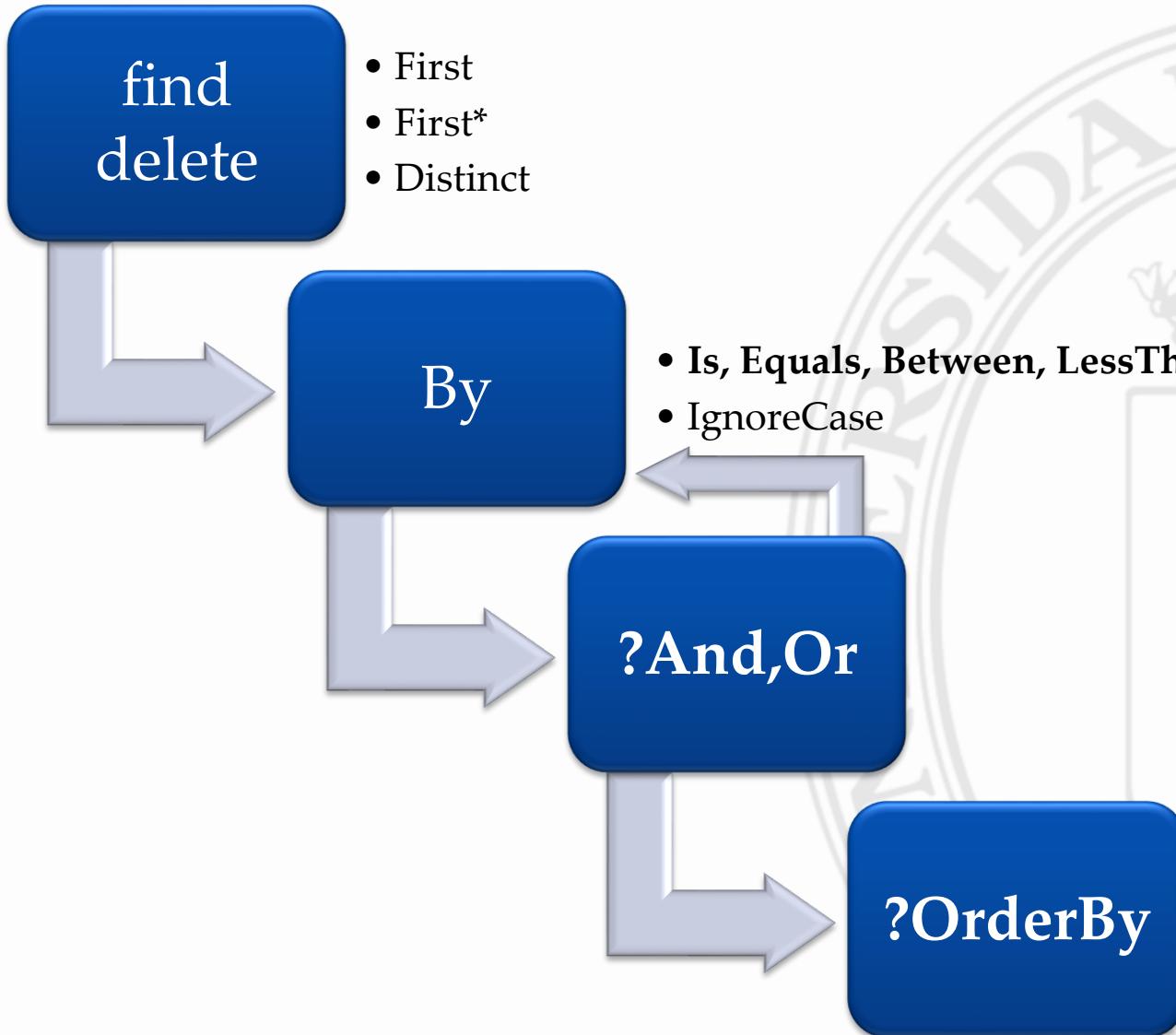
<https://github.com/miw-upm/betca-spring>

- {→} Demo de Postgres
- Proyecto: betca-jpa

✍ JPA en práctica

- Crear nuevas entidades con tests. Lanzar los Test con mucha frecuencia!!!
 - Sin relación
 - Con Composición
 - Con Agregación
- Postgres
 - Instalar Postgres y crear las BD “betca” & “tpv”
 - Comentar la activación de H2 en *test.properties*
 - Arrancar el motor de Postgres y lanzar los test y mirar las tablas creadas

Spring Data JPA



Spring Data JPA

Búsquedas por nombre de método

- **findByLastnameAndFirstname**, **findByLastnameOrFirstname**
- **findByFirstname**, **findByFirstnameIs**, **findByFirstnameEquals**
- **findByStartDateBetween**
- **findByAgeLessThan**, **findByAgeLessThanEqual**, **findByAgeGreaterThan**,
findByAgeGreaterThanOrEqual
- **findByStartDateAfter**, **findByStartDateBefore**
- **findByAgeIsNull**, **findByAgeNotNull**, **findByAgeIsNotNull**
- **findByFirstnameLike**, **findByFirstnameNotLike**
- **findByFirstnameStartingWith**, **findByFirstnameEndingWith**
- **findByFirstnameContaining**
- **findByAgeOrderByLastnameDesc**
- **findByLastnameNot**
- **findByAgeIn(Collection<Age> ages)**, **findByAgeNotIn(Collection<Age> ages)**
- **findByActiveTrue()**, **findByActiveFalse()**
- **findByFirstnameIgnoreCase**

- List<Entity> **findByAtr1(String atr1)**;
- int **deleteByAtr1GreaterThanOrEqual(int value)**;

Spring Data JPA

Paginación

- `List<UnRelatedEntity> findByIdGreaterThan(int id, Pageable pageable);`
- `dao.findByIdGreaterThan(90, PageRequest.of(1, 5));`
- `dao.count();`

JPQL (Java Persistence Query Language)

- `@Query("select u.id from other_name_for_unrelatedentity u where u.id > ?1 and u.id < ?2")`
- `List<Integer> findIdByIdBetween(int initial, int end);`
- `@Modifying @Query(value="delete from other_name_for_unrelatedentity u where u.nick = ?1")`
- `int deleteByNickQuery(String nick);`

SQL

- `@Query(value = "SELECT * FROM un_related_entity WHERE NICK = ?1", nativeQuery = true)`

Spring Data JPA

JPQL

- `FindByLastnameAndFirstname ...where x.lastname = ?1 and x.firstname = ?2`
- `findByLastnameOrFirstname ...where x.lastname = ?1 or x.firstname = ?2`
- `findByFirstname, findByFirstnameIs, findByFirstnameEquals ...where x.firstname = ?1`
- `findByStartDateBetween ...where x.startDate between ?1 and ?2`
- `findByAgeLessThan ...where x.age < ?1`
- `findByAgeLessThanEqual ...where x.age <= ?1`
- `findByAgeGreaterThan ...where x.age > ?1`
- `findByAgeGreaterThanOrEqual ...where x.age >= ?1`
- `findByStartDateAfter ...where x.startDate > ?1`
- `findByStartDateBefore ...where x.startDate < ?1`
- `findByAgeIsNull ...where x.age is null`
- `findByAgeNotNull, findByAgeIsNotNull ...where x.age not null`
- `findByFirstnameLike ...where x.firstname like ?1`
- `findByFirstnameNotLike ...where x.firstname not like ?1`
- `findByFirstnameStartingWith ...where x.firstname like ?1(parameter bound with appended %)`
- `findByFirstnameEndingWith ...where x.firstname like ?1(parameter bound with prepended %)`
- `findByFirstnameContaining ...where x.firstname like ?1(parameter bound wrapped in %)`
- `findByAgeOrderByLastnameDesc ...where x.age = ?1 order by x.lastname desc`
- `findByLastnameNot ...where x.lastname <> ?1`
- `findByAgeIn(Collection<Age> ages) ...where x.age in ?1`
- `findByAgeNotIn(Collection<Age> ages) ...where x.age not in ?1`
- `findByActiveTrue() ...where x.active = true`
- `findByActiveFalse() ...where x.active = false`
- `findByFirstnameIgnoreCase ...where UPPER(x.firstname) = UPPER(?1)`

Spring Data JPA

JPQL - JOIN

- SELECT agg.nick FROM AggregationEntity agg WHERE agg.anotherEntity.value = ?1
- SELECT anotherList.value FROM AggregationEntity agg JOIN agg.anotherEntityList anotherList
- SELECT another.value FROM AggregationEntity agg JOIN agg.anotherEntity another JOIN agg.anotherEntityList anotherList WHERE anotherList.value = ?1

JPA. Consultas



<https://github.com/miw-upm/betca-spring>

- **betca-jpa**

JPA en práctica

- Definir varias consultas para la entidad: *UnRelatedEntity*
- Definir una consulta con paginación para la entidad: *UnRelatedEntity*
- Definir varias consultas mediante JPQL para la entidad *UnRelatedEntity*
- Definir una consulta en SQL para la entidad *UnRelatedEntity*
- **Realizar los test correspondientes**

JPA. betca-tpv-user



<https://github.com/miw-upm/betca-tpv-user>

- **betca-tpv-user::data**

JPA en práctica

- Capa Data de la práctica TPV-user

JPA. spring-practice



<https://github.com/miw-upm/spring-practice>

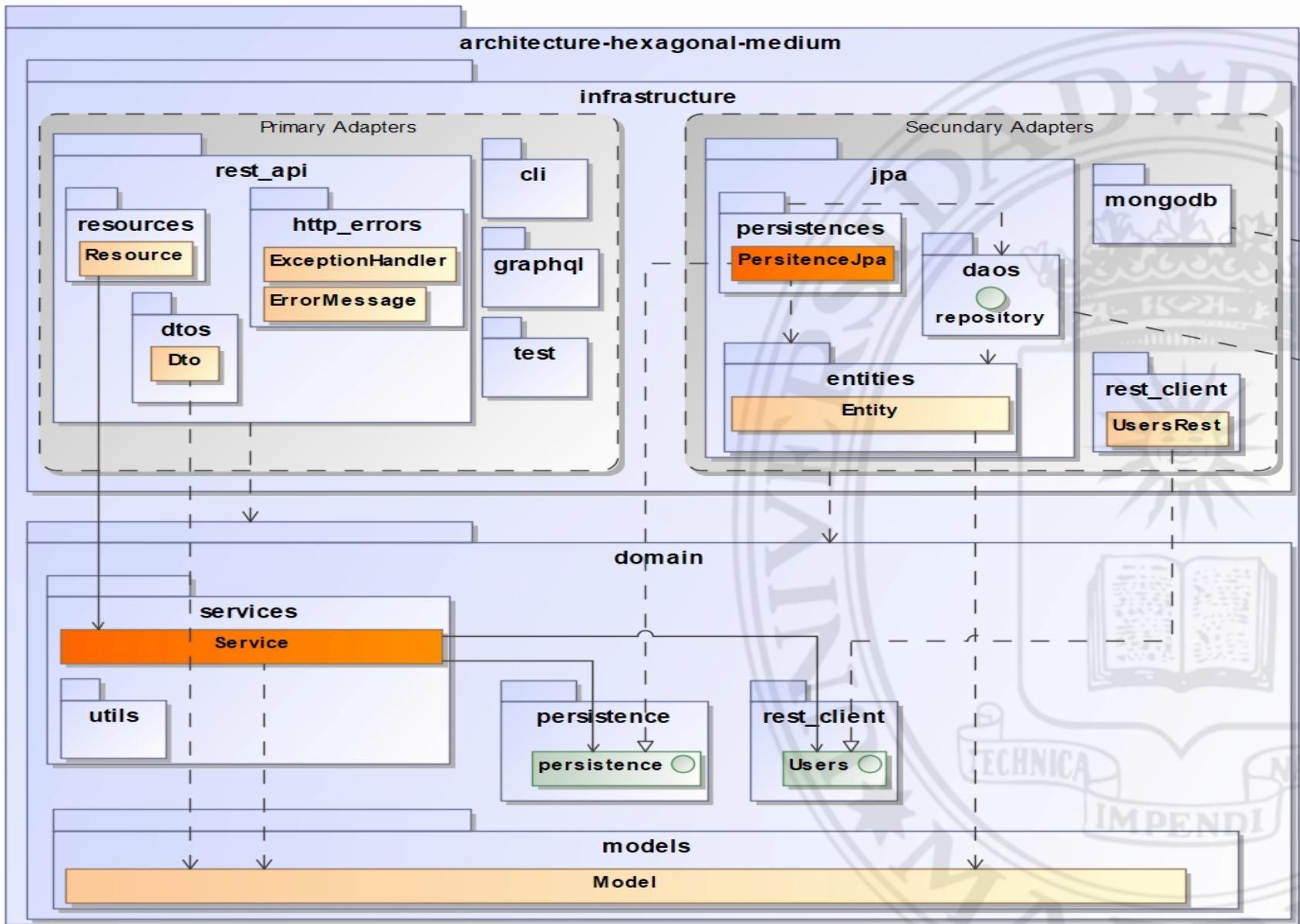
- **Spring-practice::data**

JPA en práctica

- Capa Data de la práctica **spring-practice**

Arquitectura

Paquetes



Rest

HTTP- *Hyper Text Transfer Protocol*

URI

- Server
- Path?
- Params?

Header

- Method
- Params?

Body?

HTTP - stateless

Header

- State
- Params?

Body?

¿Qué es REST?

Representational State Transfer

REST

*Representational
State Transfer*

Es un estilo de
arquitectura software
sobre HTTP

API

*Application
Programming
Interface*

Describe un interfaz

End-point

Cada una de las
posibles peticiones

¿Qué es REST?

Estilo de arquitectura software sobre HTTP

HTTP

- Utiliza el protocolo HTTP, sin ninguna capa adicional.

Relación Cliente-Servidor

- El cliente realiza una petición y el servidor da una respuesta.

Sin estado

- No se guardan datos de la petición del cliente en el servidor.
- En cada petición el cliente debe enviar toda la información.

Tipo de Representación

- Se permite la elección del tipo de representación (JSON, HTML, XML,...) a través de los tipos MIME

Cacheable

- En las respuestas se indica si es cacheable.

Operaciones CRUD

- Se permiten cuatro operaciones: Create (POST), Read (GET), Update (PUT/PATCH) y Delete (DELETE).

REST

Buenas prácticas

SSL

- Siempre!!!

Documentar

- OpenAPI

Versionado

- Para versiones utilizar v* en el nivel mas alto

Recursos

- Autodescriptivos
- Sustantivos en plural

ID

- Procurar id's no deducibles

Granularidad

- Controlar el tamaño de la respuesta

Error

- Devolver mensaje de error en el cuerpo

Pretty print

- Con gzip

HATEOAS?

- Hypermedia as the Engine of Application State

REST

Buenas prácticas

URI's

“/” jerarquía de los recursos

“,” y “;” para partes no jerárquicas

NO referenciar acciones /v0/get-orders → v0/orders

NO referenciar formatos /v0/orders/{id}/pdf → v0/orders/{id}/print

Búsquedas:

GET /orders/fixed-search

GET

GET /orders/search?q=name:jes

GET /search/repo?q=tetris;language:java&sort=stars

Respuestas

Respuestas parciales: ...?fields=id,description

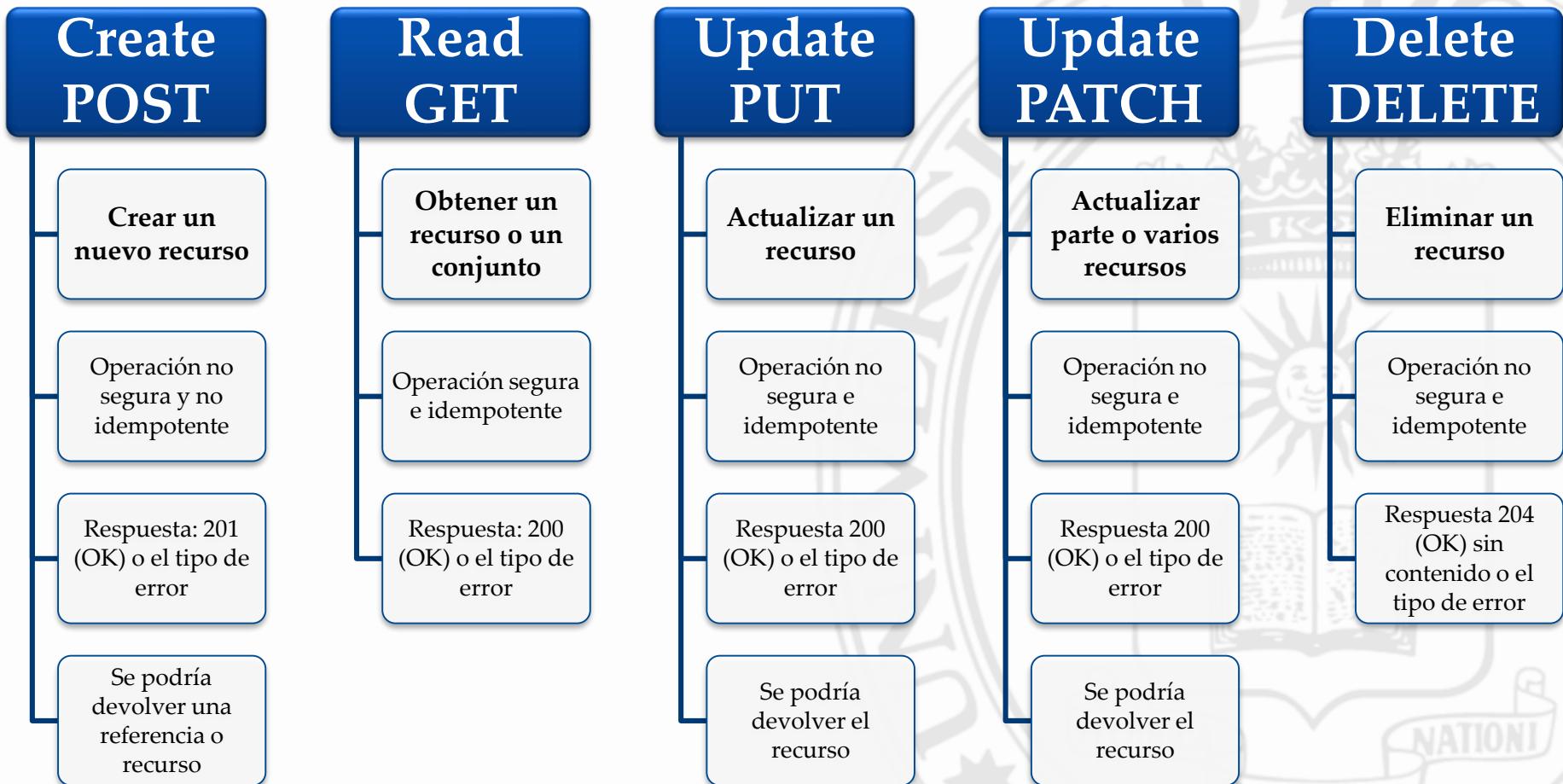
Paginación: ...?page=1&size=30

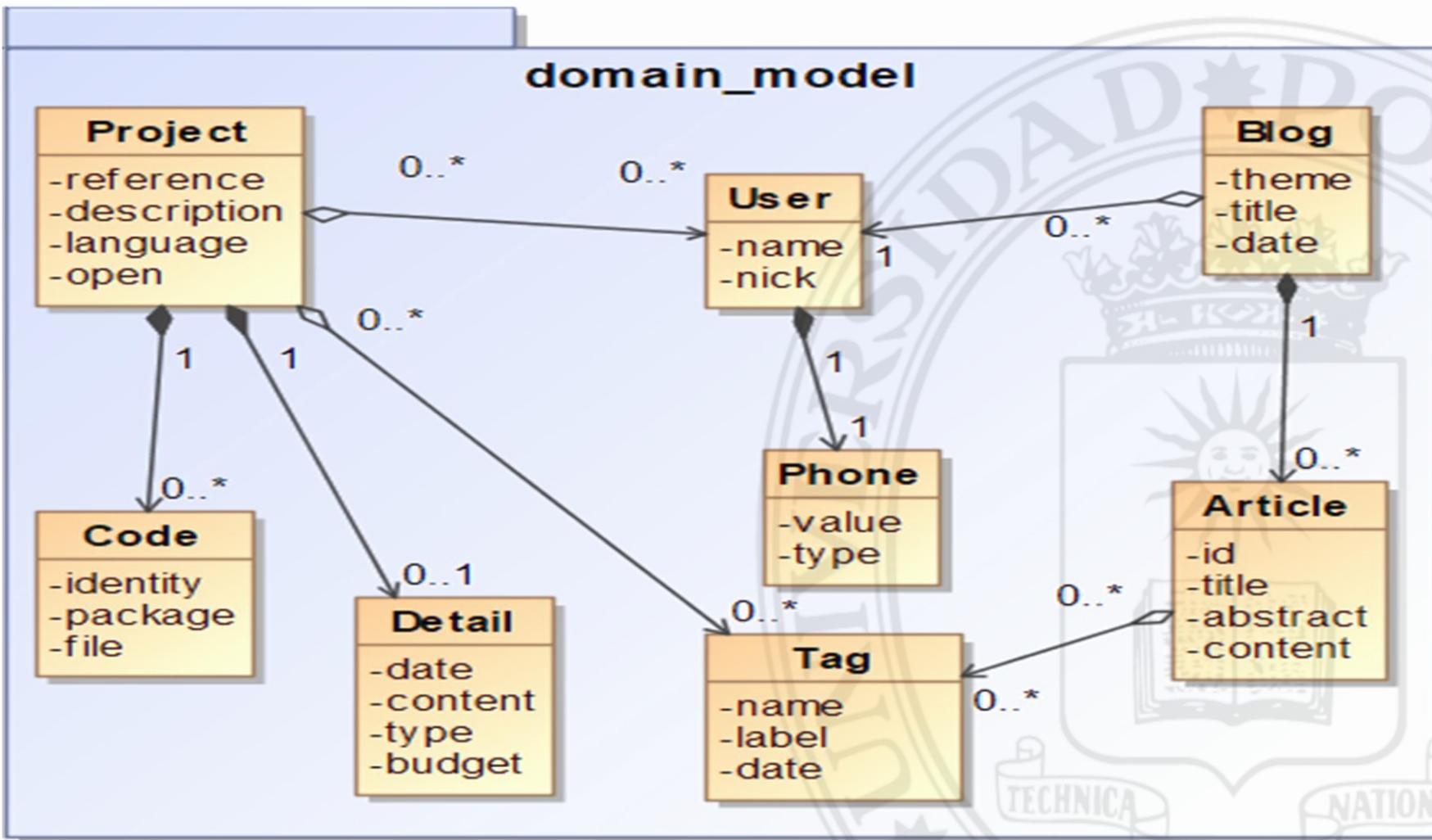
Filtros: ...?type=3,5

Orden: ...?sort=name,surname&desc=id

REST

Buenas prácticas





Crear un API Rest

Rest



Server URI

- `http://server.com/api/v0/*`
- `http://api.server.com/v0/**`

End-points

- `GET /users` response: [{name}]
- `POST /users` request: {nick, name...}
- `GET /users/{name}` response:{name, nick, phone{value, type}}
- `PUT /users/{name}` request:{...}
- `PATCH /users/{name}` request:[{nick:new-nick, phone/value:new-value}] Atomico
- `DELETE /users/{name}`
- `GET /blogs` response:{id,title}
- `DELETE /blogs/{theme}`
- `GET /blogs/{theme}/user`
- `POST /blogs` request: {...}
- `GET /blogs/{theme}/articles/{id}/abstract` response:{abstract}
- `GET /tags` response:{...}
- `GET /projects/search?q=language:java&sort=reference`
- `GET /projects/{reference}/tags?sort=-date,label`
- `GET /projects?fields=language,description`
- `GET /users?page=2&size=20`
- `GET /search?q=language:java+language:typescript&sort=identity`
- `GET /projects/opened`
- `GET /projects/search?q>tag:id`
- `GET/projects/{reference}/open`

Rest

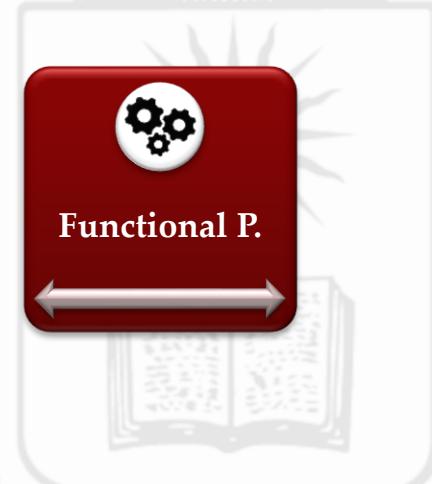
Responsabilidades

Define el *path* y el
método del recurso

Gestionar la validación de los
dtos, debe delegar en el *modelo*
del *dominio* o en los *dtos*

Organizar la recepción de datos
de la petición

Delega en el *Servicio* la
ejecución de la petición



resources

```
«@RequestMapping(uri)»  
«@RestController»  
BasicResource
```

```
«@PostMapping»+create( @RequestBody dto : Dto ) : Mono<Dto>  
«@GetMapping»+read( @PathVariable id : String ) : Mono<Dto>  
«@PutMapping»+update( @PathVariable id : String, @RequestBody dto : Dto ) : Mono<Dto>  
«@DeleteMapping»+delete( @PathVariable id : String ) : Mono<Void>  
«@GetMapping»+findByName( @RequestParam name : String ) : Flux<Dto>
```

```
«@JsonInclude»  
«@Getter»  
«@Setter»  
«@ToString»  
«@NoArgsConstructor»
```

Dto

```
-id  
-name  
-gender  
-borndate
```



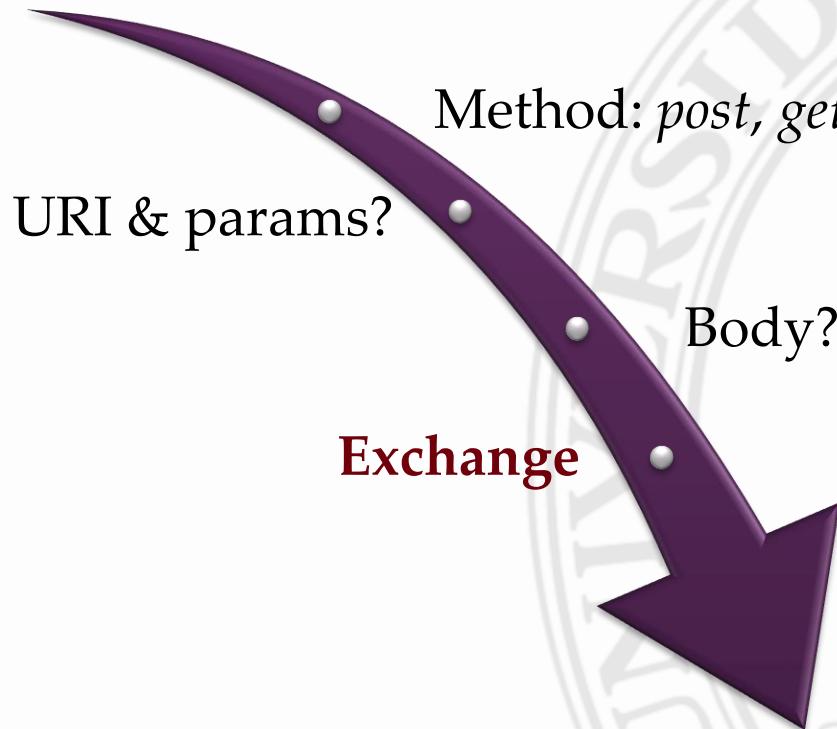
PUT /basic/666 {dto}

GET /basic/search?name=Miw&q=other

Rest

Spring - Test

*Spring-Test: WebTestClient
(Builder)*



- expectStatus, expectBody...

Spring Rest



Dependencies

[ADD DEPENDENCIES... CTRL + B](#)

Lombok DEVELOPER TOOLS

Java annotation library which helps to reduce boilerplate code.

Spring Reactive Web WEB

Build reactive web applications with Spring WebFlux and Netty.

Spring Security SECURITY

Highly customizable authentication and access-control framework for Spring applications.

Validation I/O

JSR-303 validation with Hibernate validator.

src/*/resources

- application.properties
- test.properties

@RestTestConfig

- @SpringBootTest
- @AutoConfigureWebTestClient
- @TestPropertySource

Rest. betca-spring



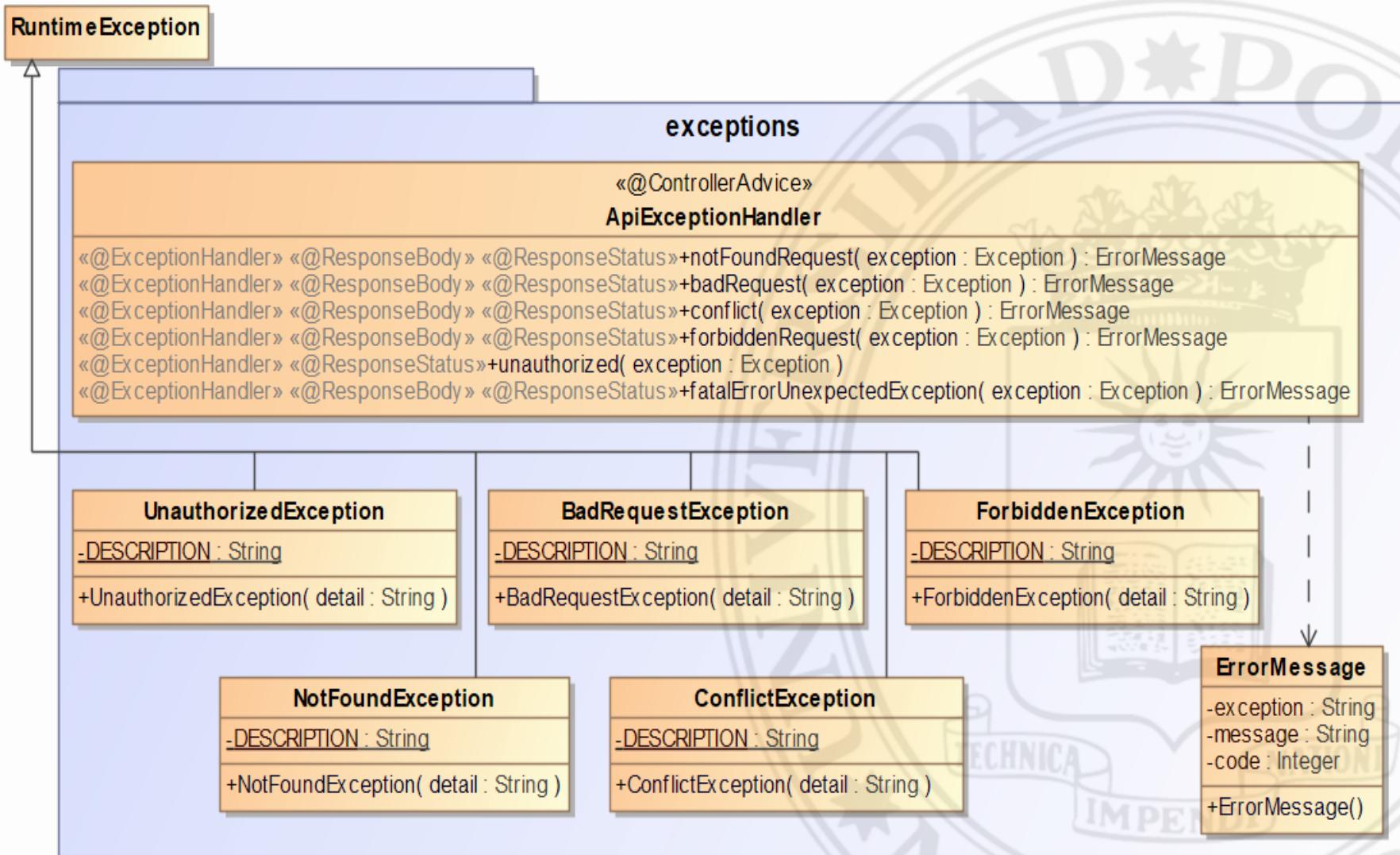
<https://github.com/miw-upm/betca-spring>

- **betca-rest**

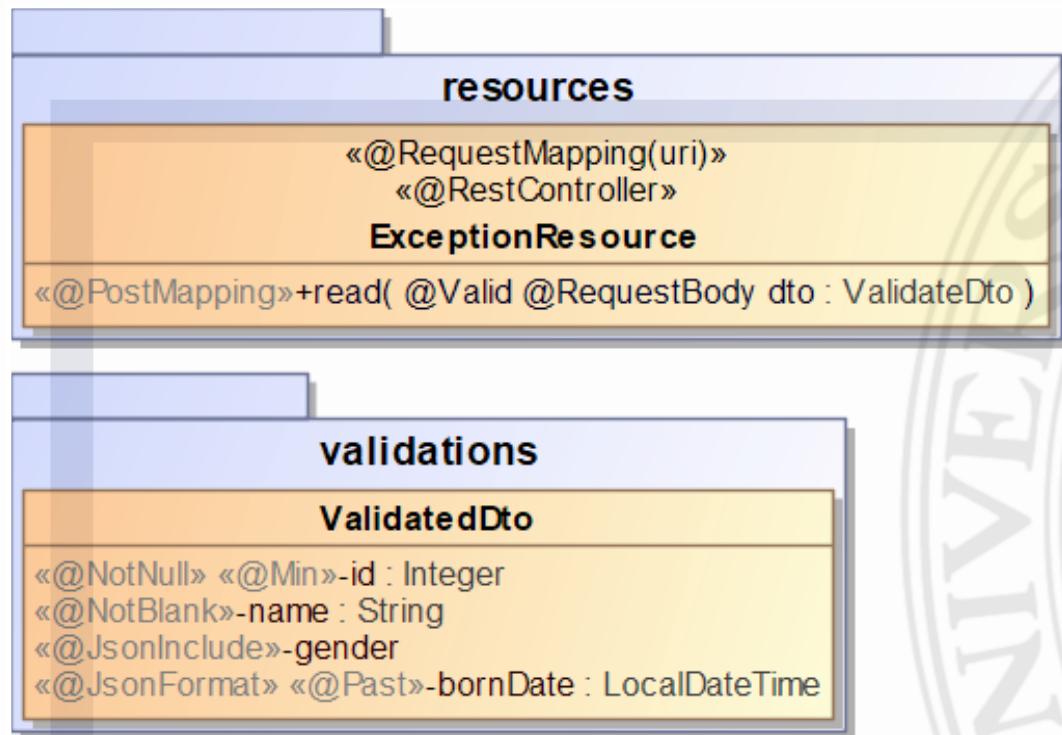
Rest en práctica

- Basic Resource
- Añadir end-point GET /basic/calculator, reciben dos parámetros(dividendo,divisor), devuelve un String: {"division":xxx}
- Añadir end-point POST /basic/calculator2, recibe un DTO que representa la división (atributos dividendo, divisor) y devuelve un DTO con el resultado.
- Añadir end-point POST /basic/calculator3, recibe una lista de DTO's (dividendo,divisor) y devuelve una List<DTO> con el resultado.

Spring Excepciones



Spring Validaciones



Validador propio: *PositiveBigDecimal*

Validadores

- @DecimalMax, @DecimalMin
- @Digits
- @Email
- @Future, @FutureOrPresent
- @Max, @Min
- @Negative, @NegativeOrZero
- @Null, @NotNull
- @NotEmpty, @NotBlank
- @Past, @PastOrPresent
- @Positive, @PositiveOrZero
- @Pattern
- @Size

Exceptions. betca-spring



<https://github.com/miw-upm/betca-spring>

- **betca-rest**

Rest en práctica

- *ExceptionResource, ValidatedDto*
- Añadir...

Rol

- Es un nombre abstracto para configurar el permiso de acceso de un conjunto de recursos de una aplicación.

Usuario

- Es una identidad única reconocida por el servidor. Un usuario puede tener varios roles.

Recursos

- Se controla las acciones de diferentes roles.
- Distribuido: anotaciones en el recurso.
- Centralizado: Uri's.

Autenticación

- Memoria
- Propio
- API Key o Token
- LDAP
- Auth2
- OpenIp
- ...

AuthenticationWebFilter

- Es un filtro de seguridad, que se procesa y se decide si se continua.
- Depende del *AuthenticationManager* para procesar.

AuthenticationManager

- Aquí se procesa si los datos de autenticación son correctos, dependiendo del tipo de autenticación (HttpBasic, Bearer...)
- Depende del *AuthenticationConverter* para extraer los datos de la cabecera de la petición.

AuthenticationConverter

- Extrae de la cabecera los datos de la autenticación y los prepara para ser procesados.

Seguridad. Spring

Basic Auth

HTTP

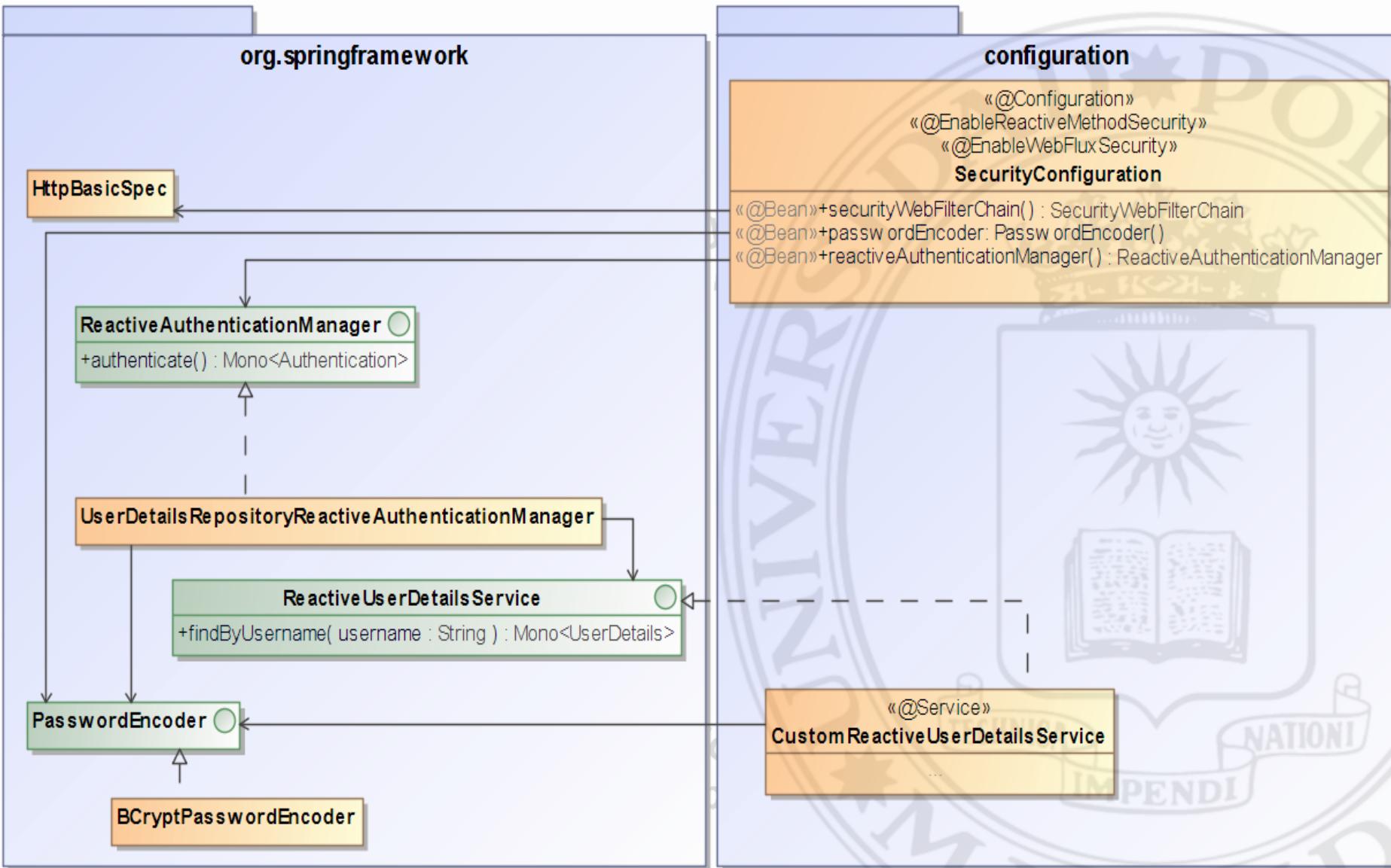
Header

Authorization=Basic (*user:password*)_{code64}

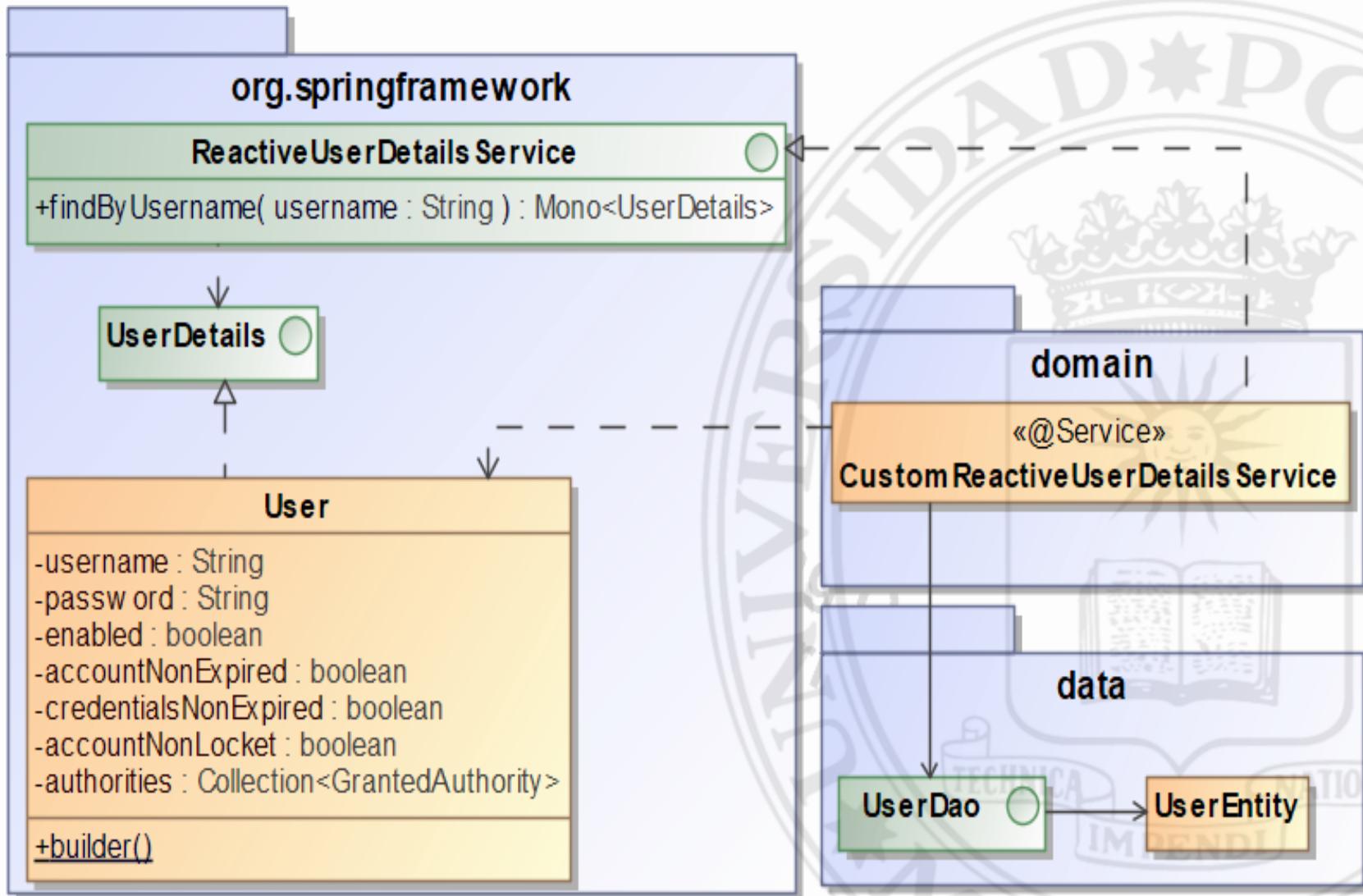
Configuración de Spring

- Bean de la cadena de filtros:
SecurityWebFilterChain
 - Filtro Basic Auth: *.httpBasic()*
- Bean de encriptación de claves:
PasswordEncoder
 - *BCryptPasswordEncoder*
- Bean del manejador de seguridad:
ReactiveAuthenticationManager
 - *CustomReactiveUserDetailsService*

Seguridad. Spring Basic Auth



Seguridad. Spring Basic Auth



Security. betca-spring



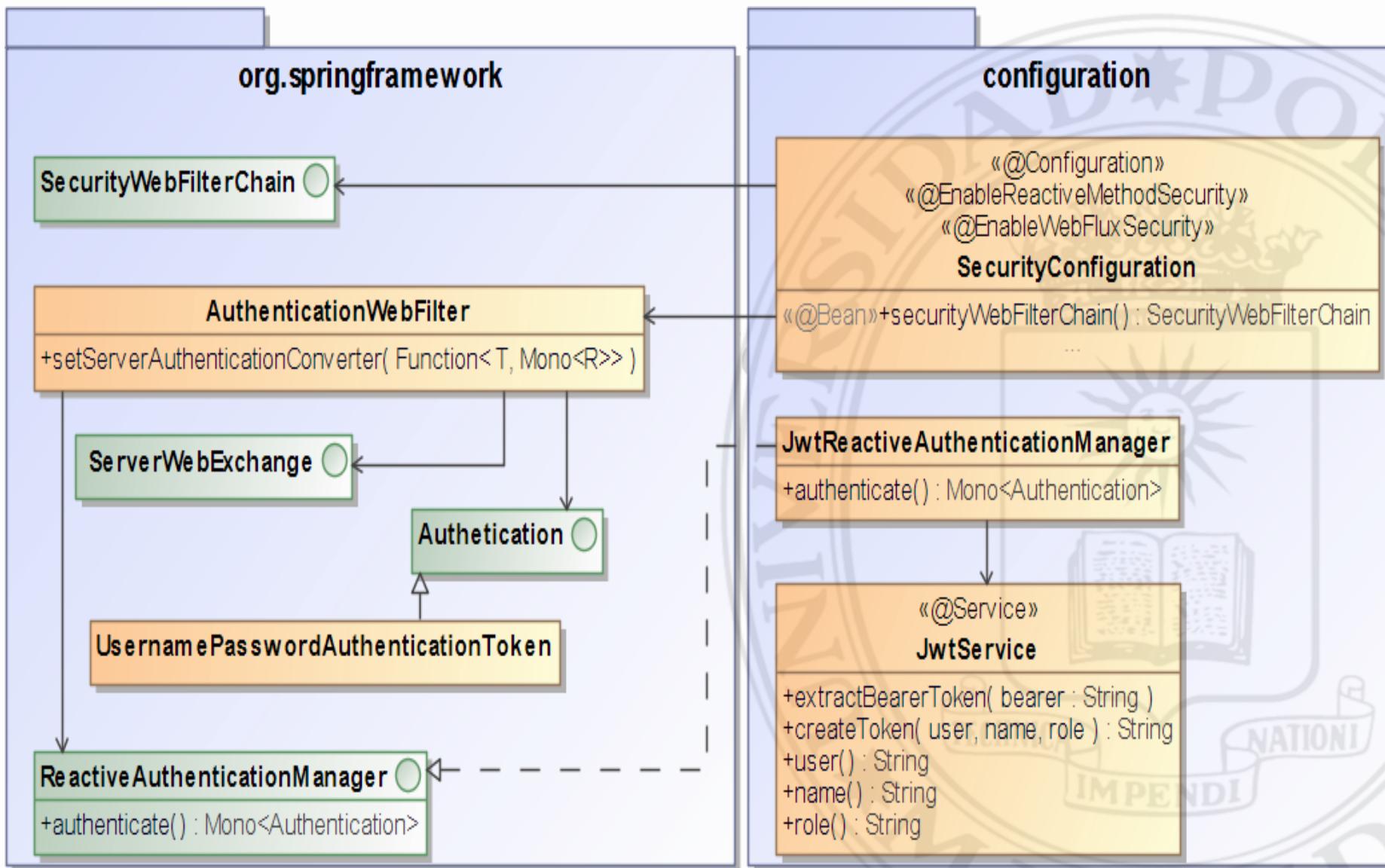
<https://github.com/miw-upm/betca-spring>

- **betca-rest**

Rest en práctica

- *SecurityResource, CustomReactiveUserDetailsService*
- Crear el end-point DELETE */basic-auth/{ID}*, sólo podrá borrar el usuario “3” con el rol de “ADMIN”, se configura con anotaciones.

Seguridad. Spring JWT



Security. betca-spring



<https://github.com/miw-upm/betca-spring>

- **betca-rest**

Rest en práctica

- *JwtReactiveAuthenticationManager, JwtService*
- Añadir...

Spring OpenAPI

OpenApi

- Es un estándar para la descripción de las interfaces de programación, o *Application Programming Interfaces (API)*.
- La especificación **OpenAPI** define un formato de descripción abierto e independiente de los fabricantes para los servicios de **API**.
- La actual especificación *OpenAPI* surgió del proyecto *Swagger*.

Configuración

- @Configuration
- @OpenAPIDefinition
- @SecurityScheme
 - basicAuth
 - bearerAuth

End-points

- @SecurityRequirement
 - basicAuth
 - bearerAuth

Spring OpenAPI

Dependencia

org.springdoc : springdoc-openapi-webflux-ui : 1.6.13

Cliente Rest

- <http://localhost:8080/swagger-ui.html>

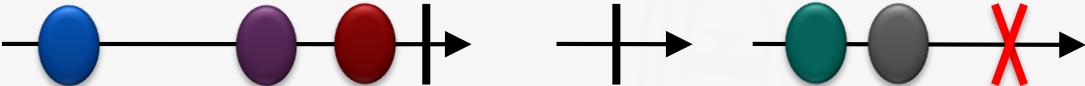
Especificación: JSON

- <http://localhost:8080/v3/api-do>

Spring

Programación Reactiva

Reactive Programming

- Flujo de datos asíncronos
- Programación Funcional
- Programación Asíncrona

Programación Reactiva

Proyectos



ReactiveX

- JavaScript, Java, Python, C#, PHP, C++, Swift, Go, Kotlin, Ruby, Scala,...
- <http://reactivex.io/>

RxJS



RxJS - Javascript

- Angular
- Clase: *Observable*
- <https://rxjs.dev/guide/overview>

Reactive

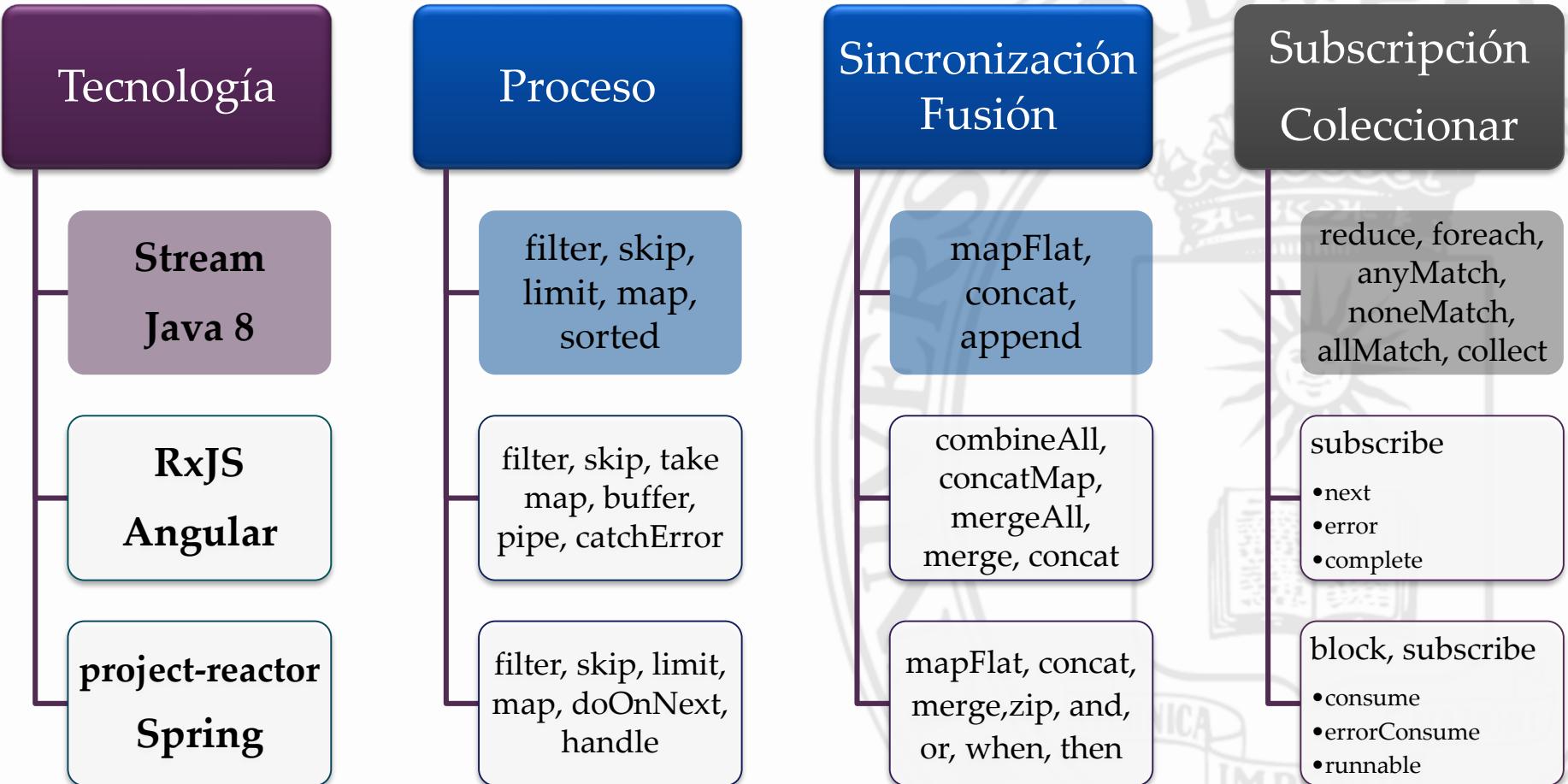


Project-reactor - Java

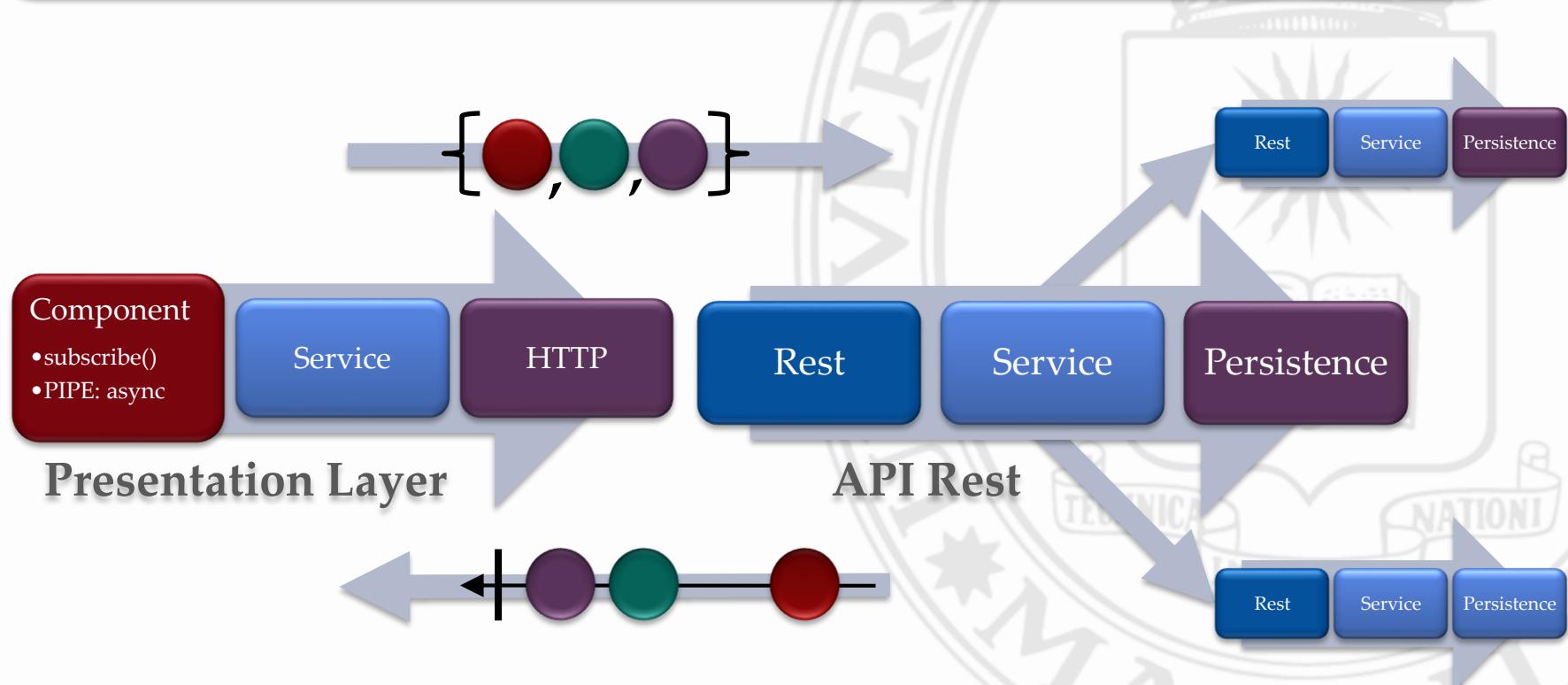
- Java - Spring
- Clases: *Mono & Flux*
- <https://projectreactor.io/docs/core/release/reference>
- <https://projectreactor.io/docs/core/release/api>

Programación Reactiva

Proyectos



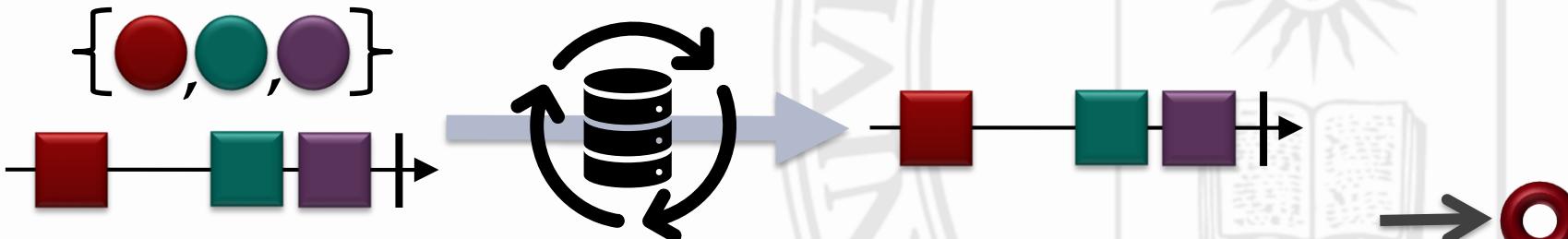
Métodos que devuelven un tipo *Publisher* no deberían subscribirse porque ello *podría romper la cadena del publicador*



Spring. Reactive BD

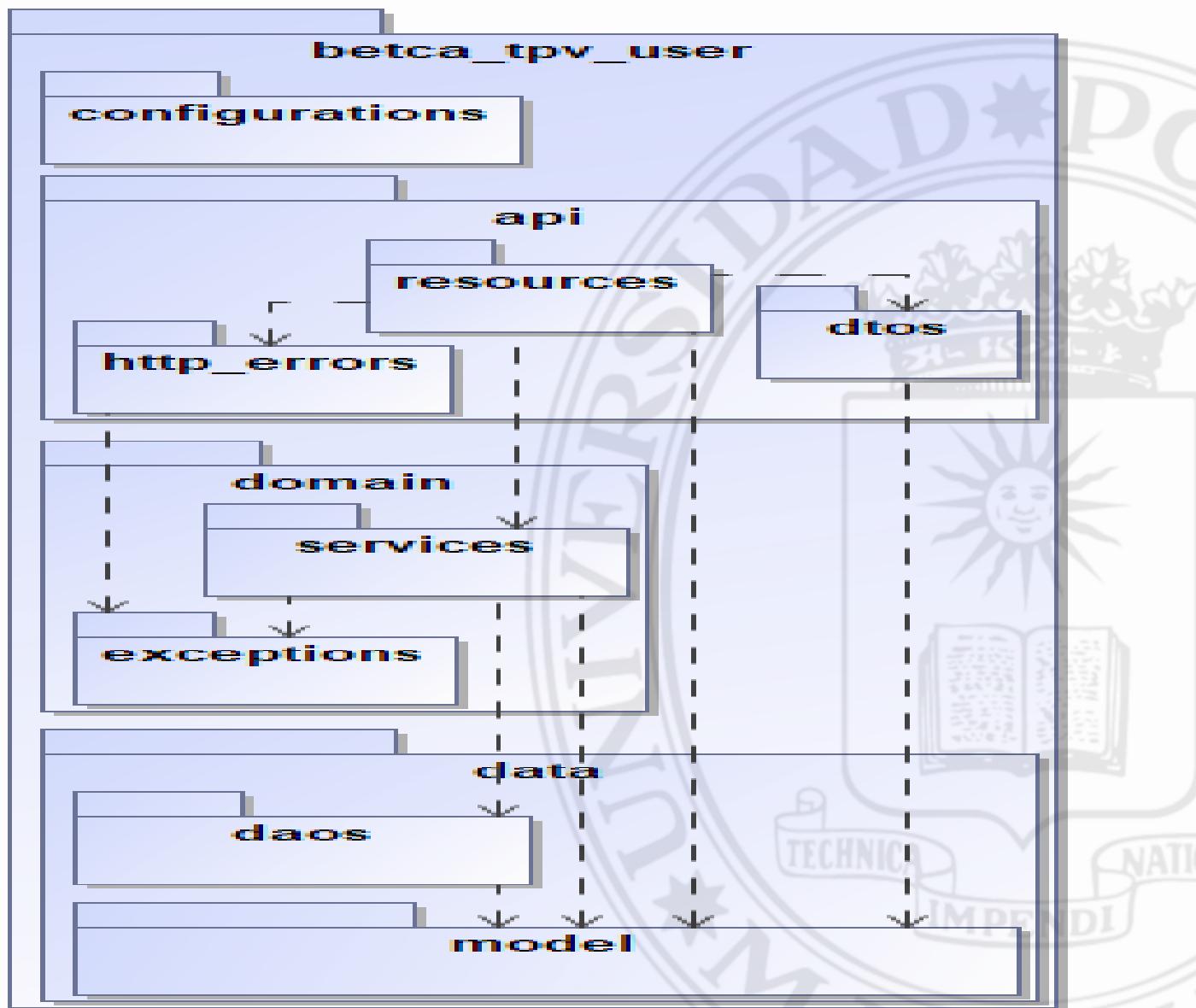


```
public ArticleDto readArticleSynchronous(String code) {  
    return new ArticleDto(this.articleRepository.findById(code)  
        .orElseThrow(() -> new NotFoundException("Article code (" + code + ")"))  
    );  
}
```

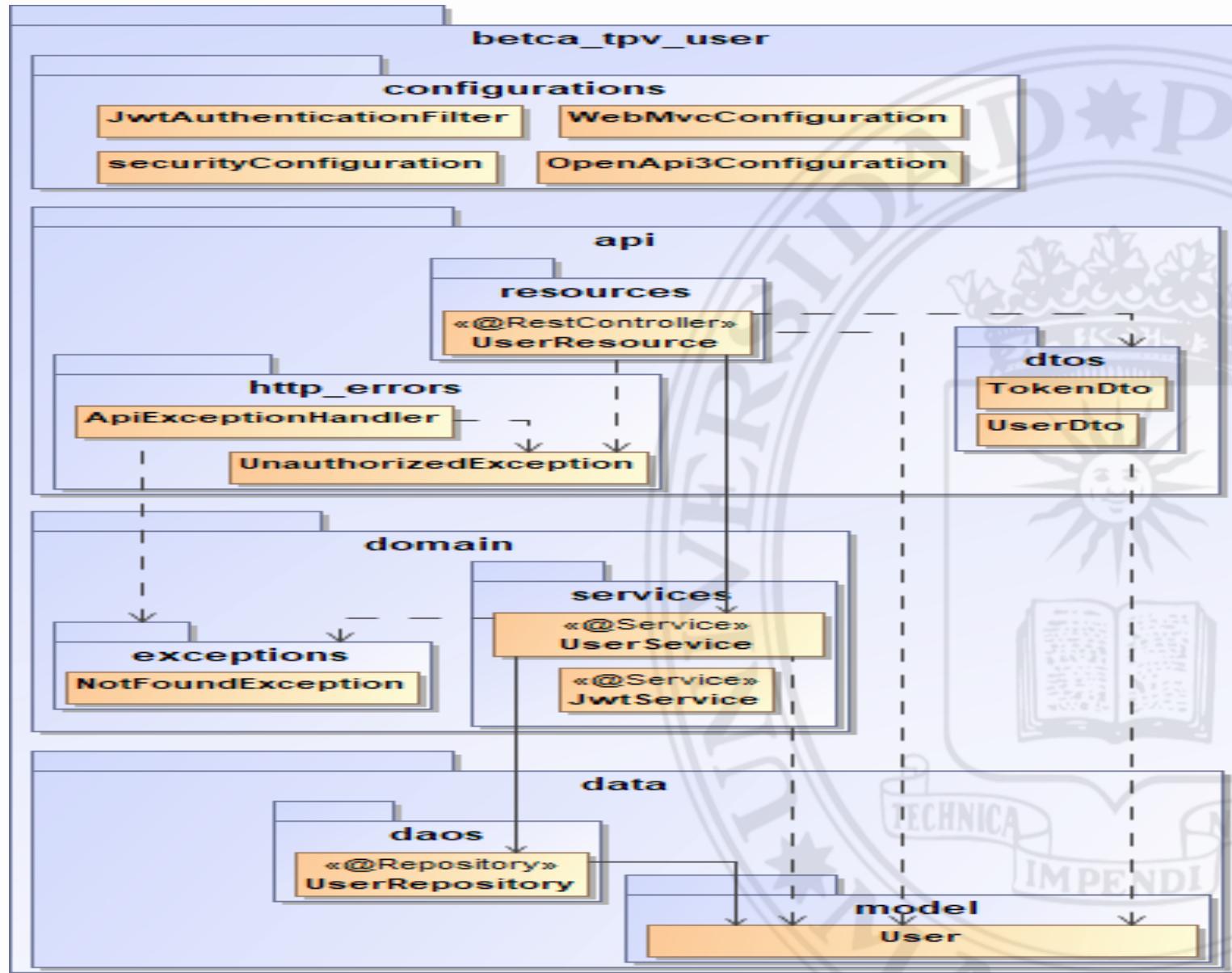


```
public Mono<ArticleDto> readArticle(String code) {  
    return this.articleReactRepository.findById(code)  
        .switchIfEmpty(Mono.error(new NotFoundException("Article code (" + code + ")")))  
        .map(ArticleDto::new);  
}
```

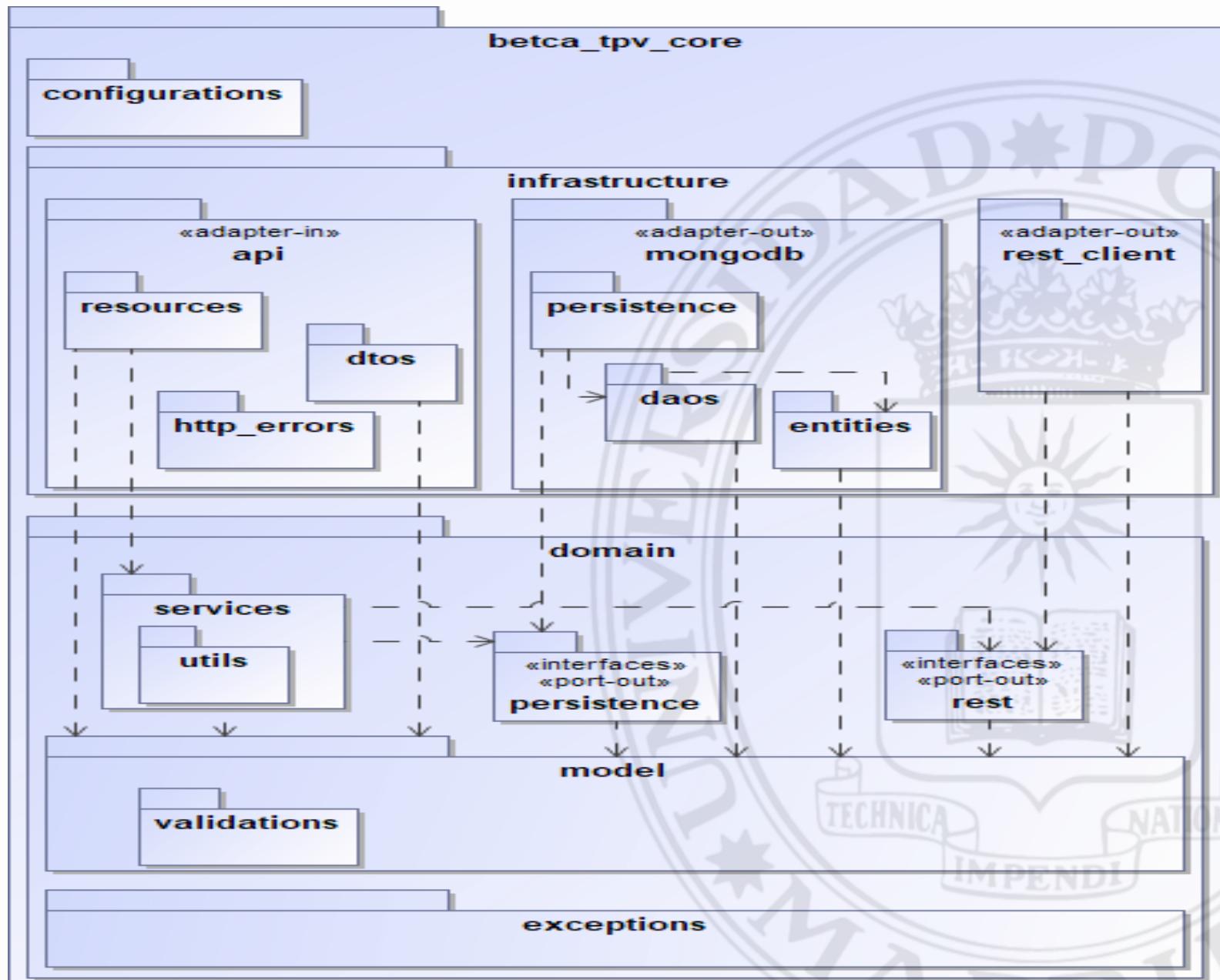
Spring. Reactive BD



Spring. Reactive BD



Spring. Reactive BD



Spring. Reactive BD

