



# Arquitectura y Patrones para Aplicaciones Web -APAW-

## WebArchitecture

Jesús Bernal Bermúdez

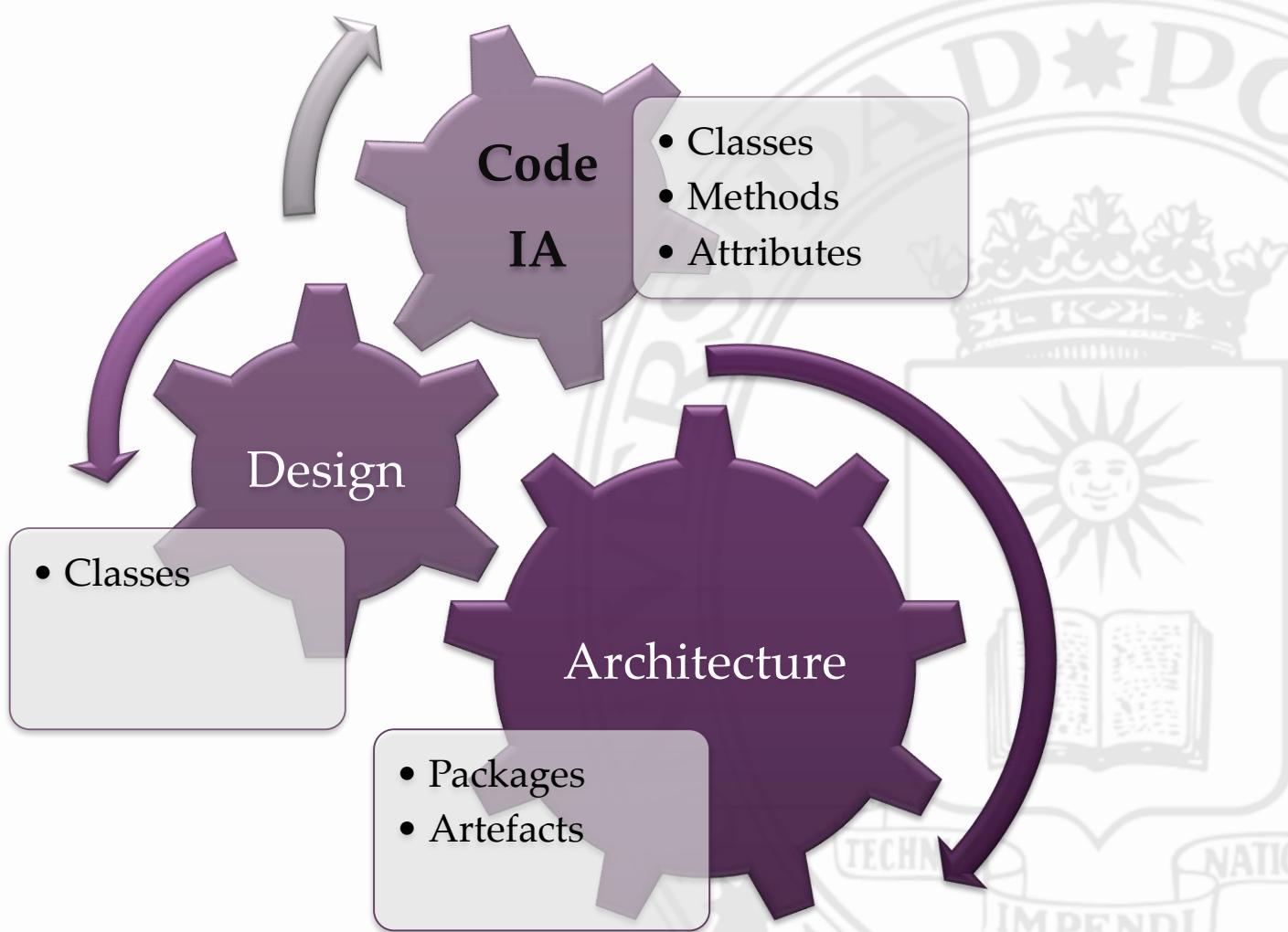
# Arquitectura Web

## ¿Qué es la Arquitectura software?

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

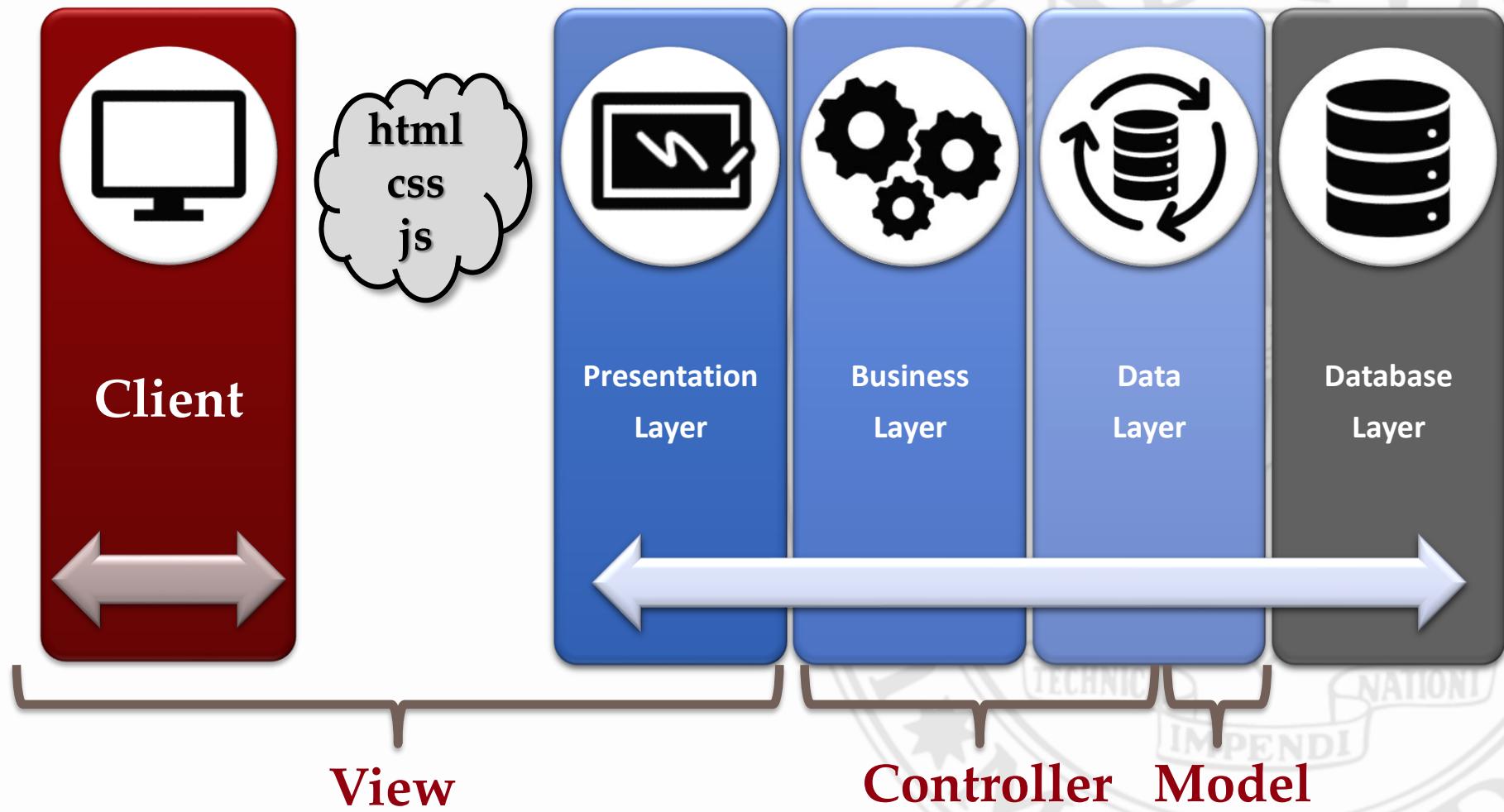
IEEE

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



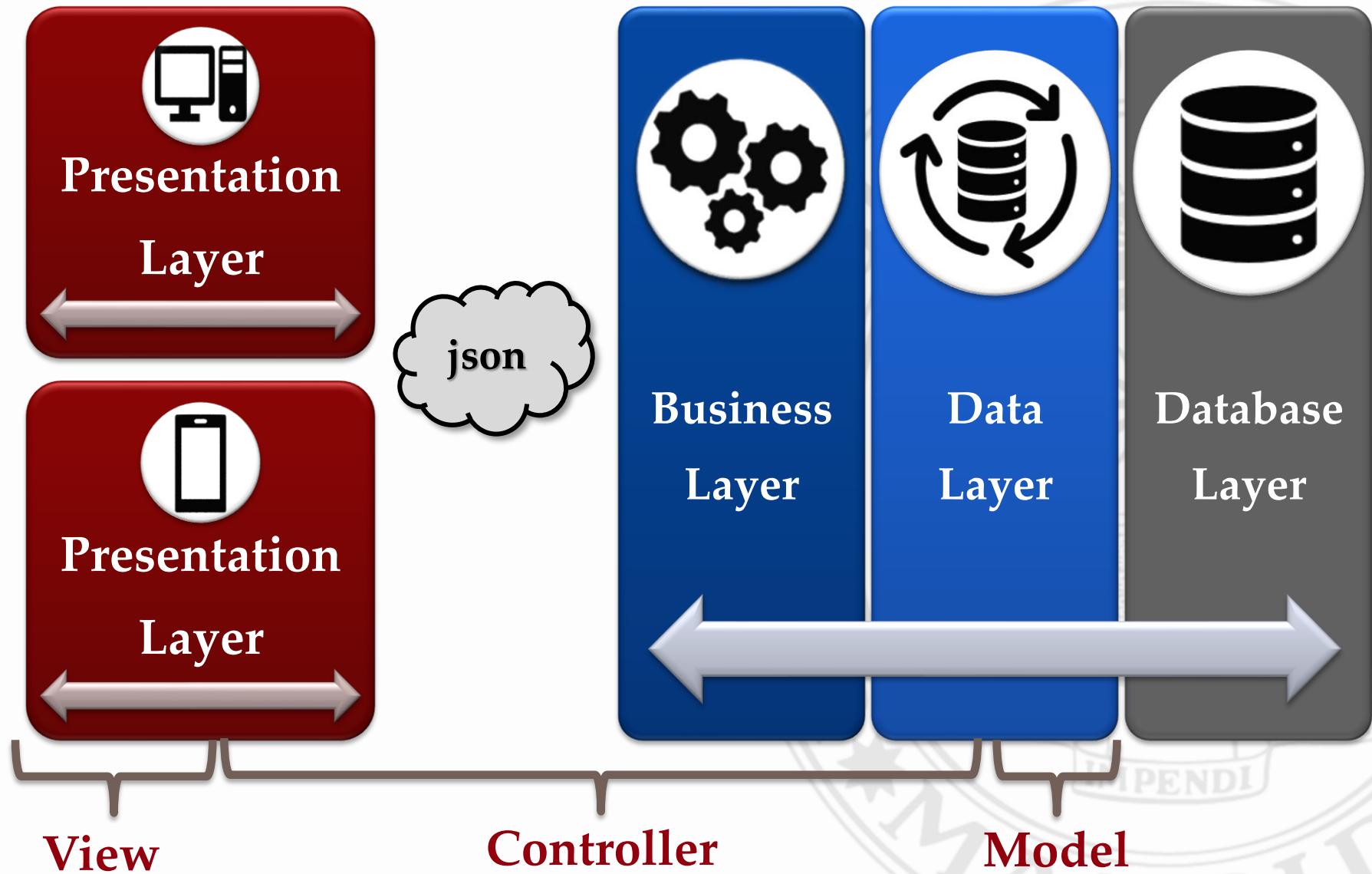
# Arquitectura por Capas

## Aplicación de Múltiples Páginas



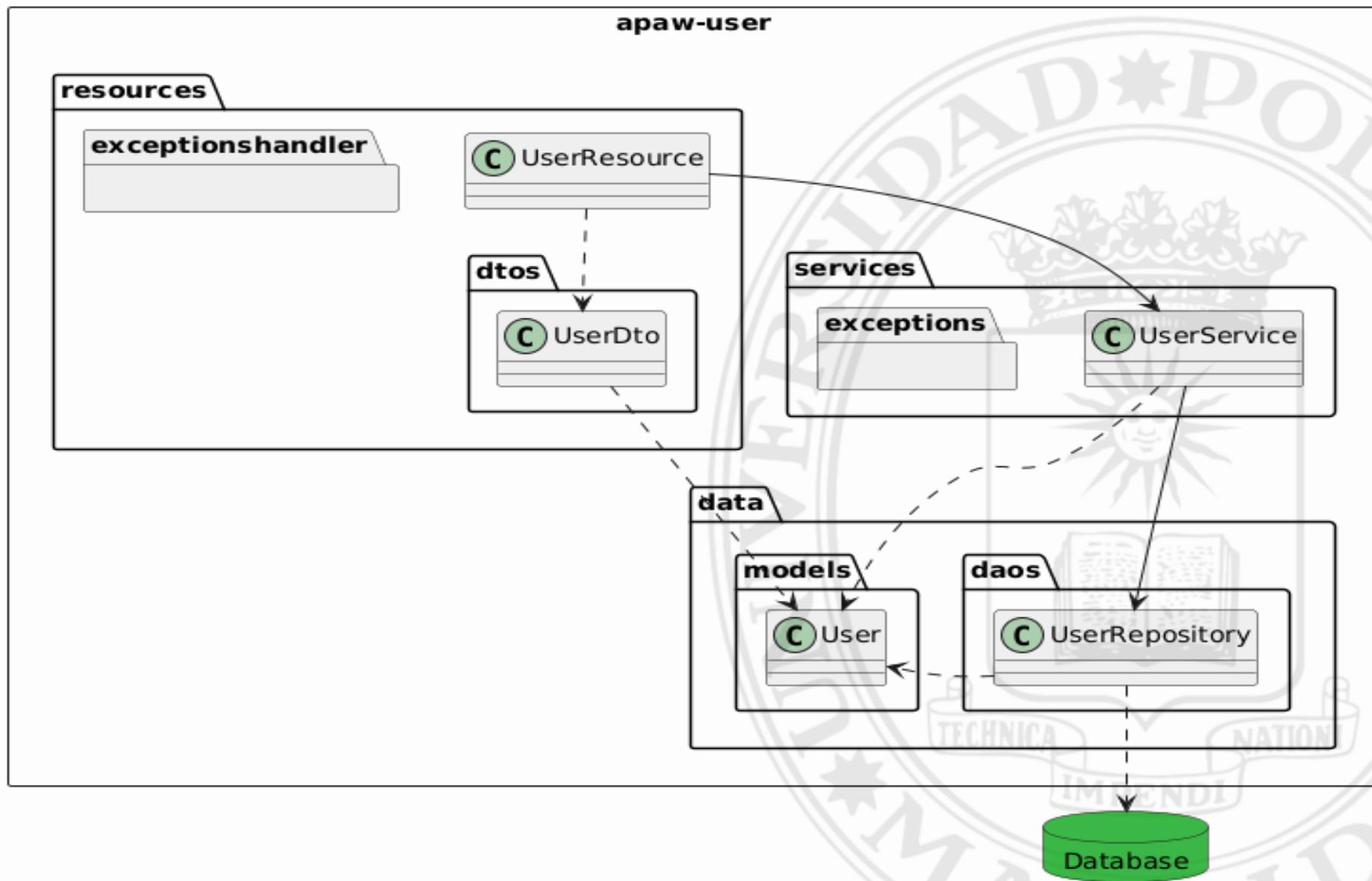
# Arquitectura por Capas

## Aplicación de Página Única (*SPA*)



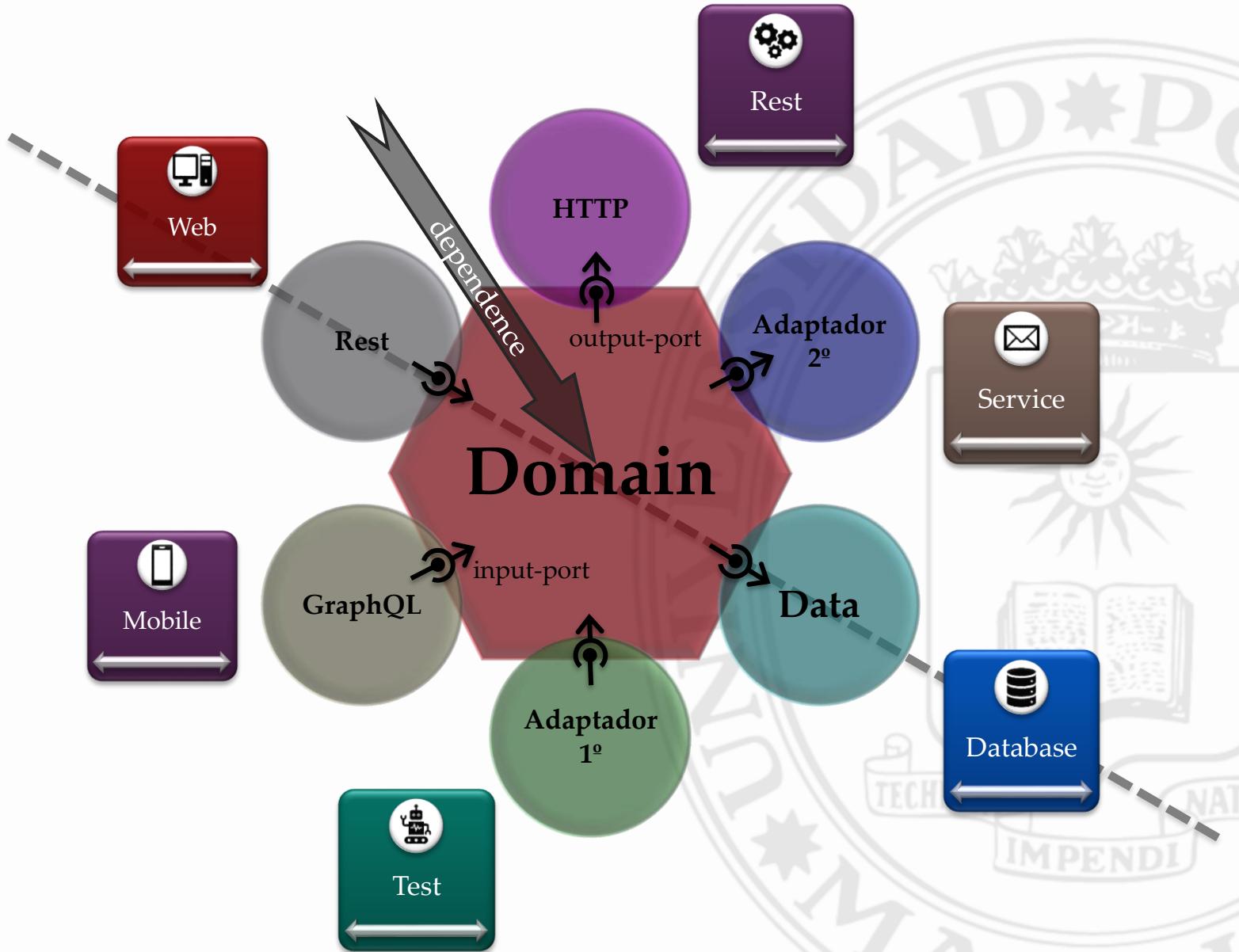
# Arquitectura por Capas

## Aplicación de Página Única (*SPA*)



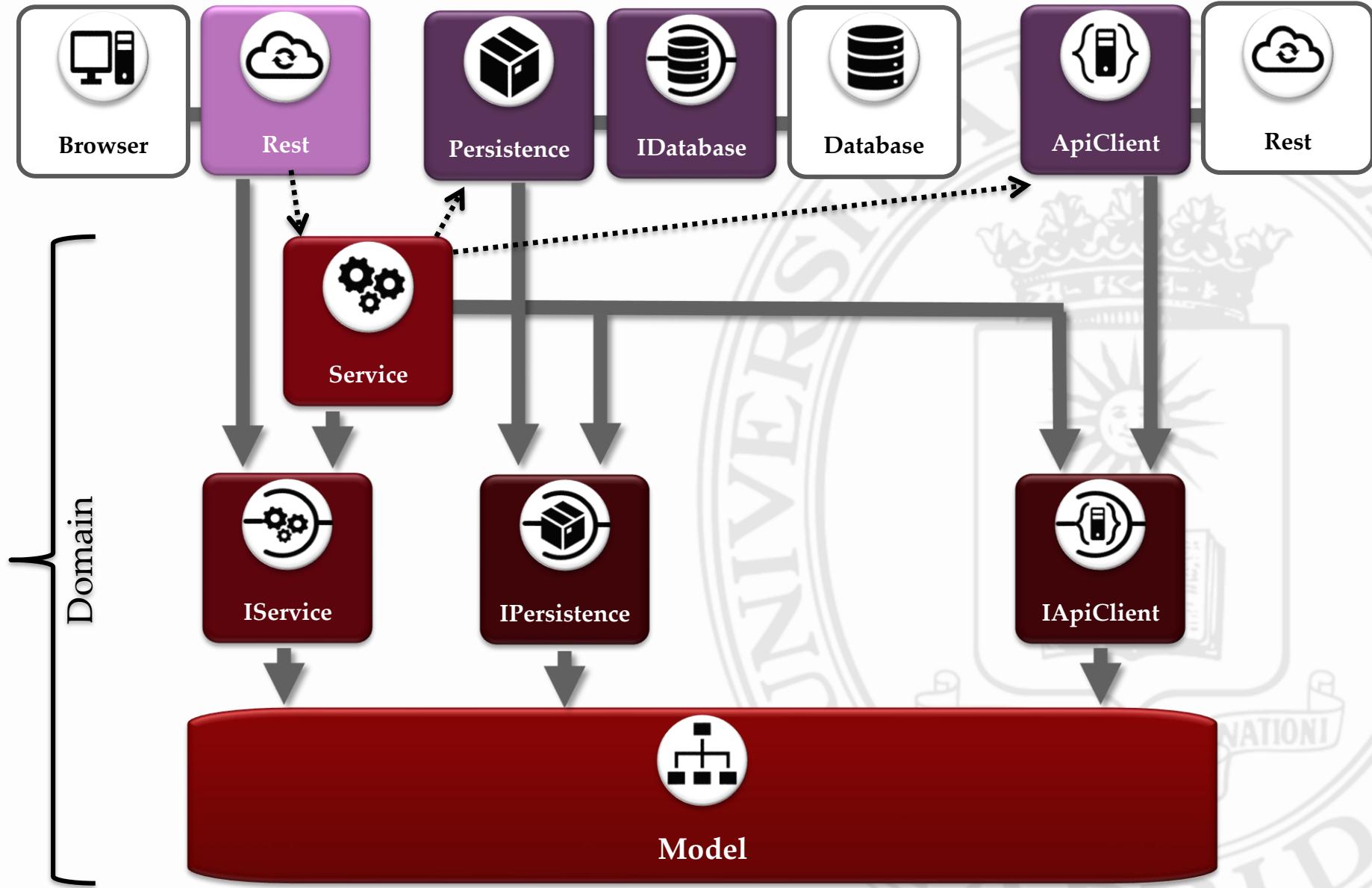
# Hexagonal Architecture

## Alistair Cockburn



# Hexagonal Architecture

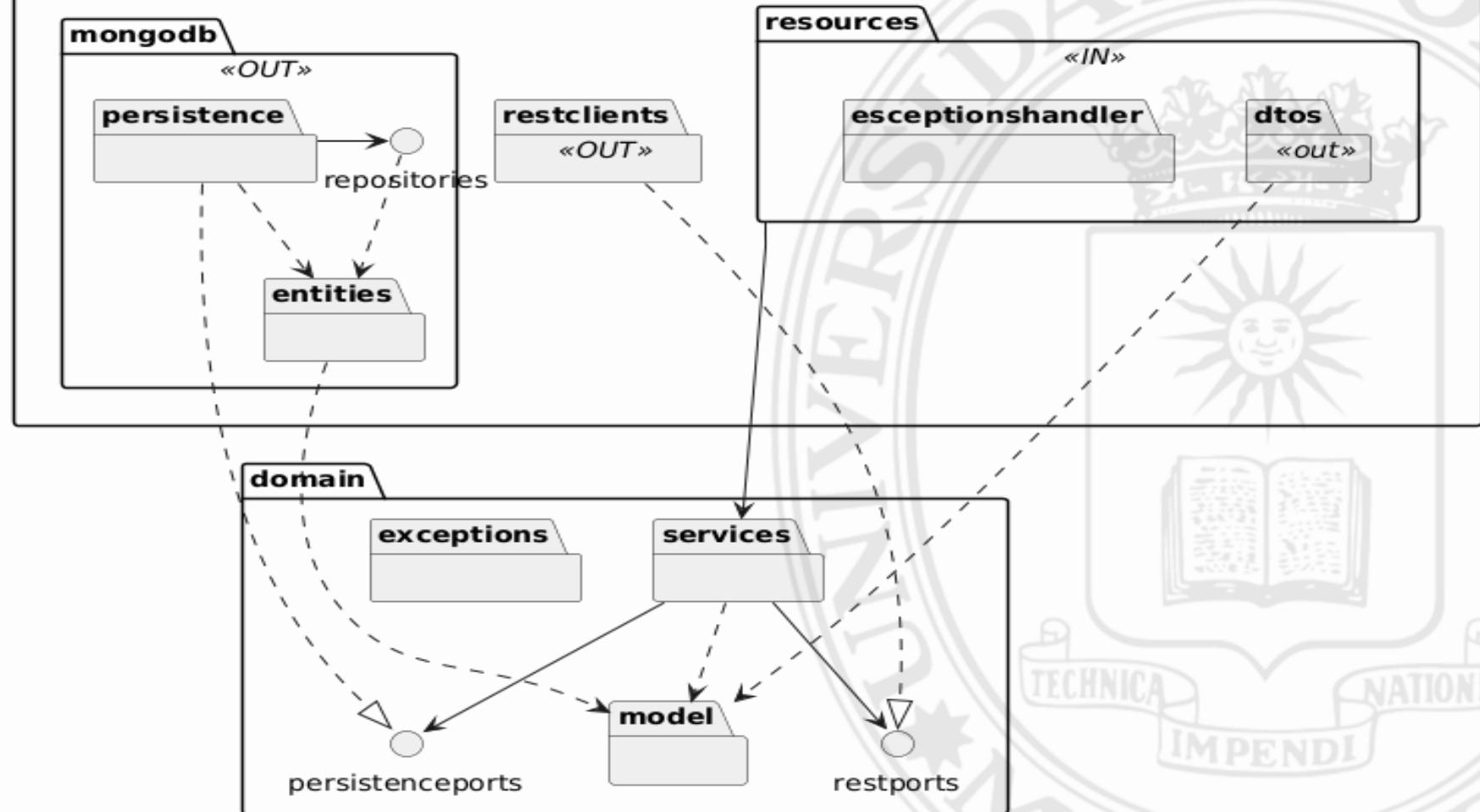
## Dependencias



# Arquitectura Paquetes

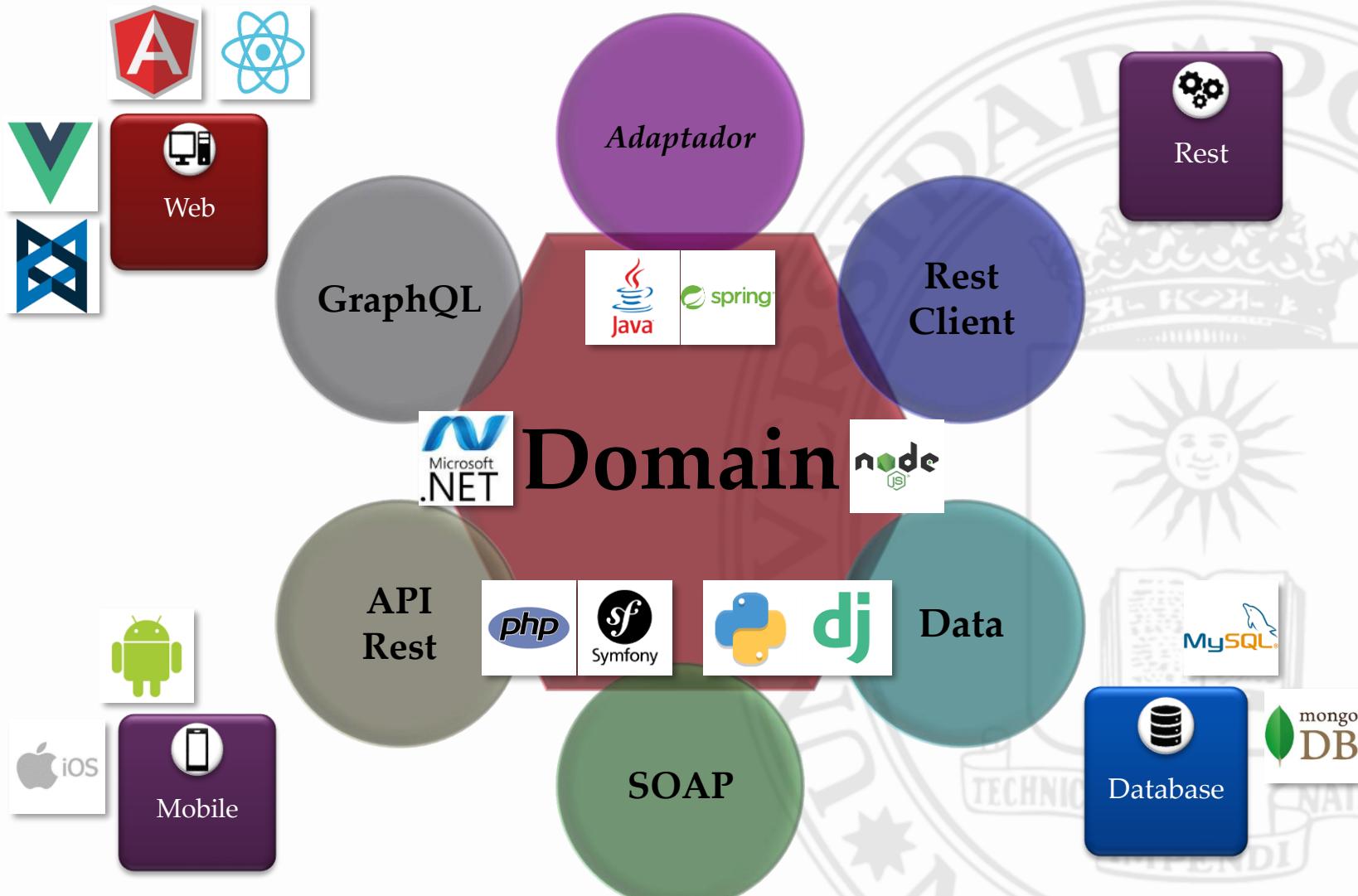
apaw-practice

adapters



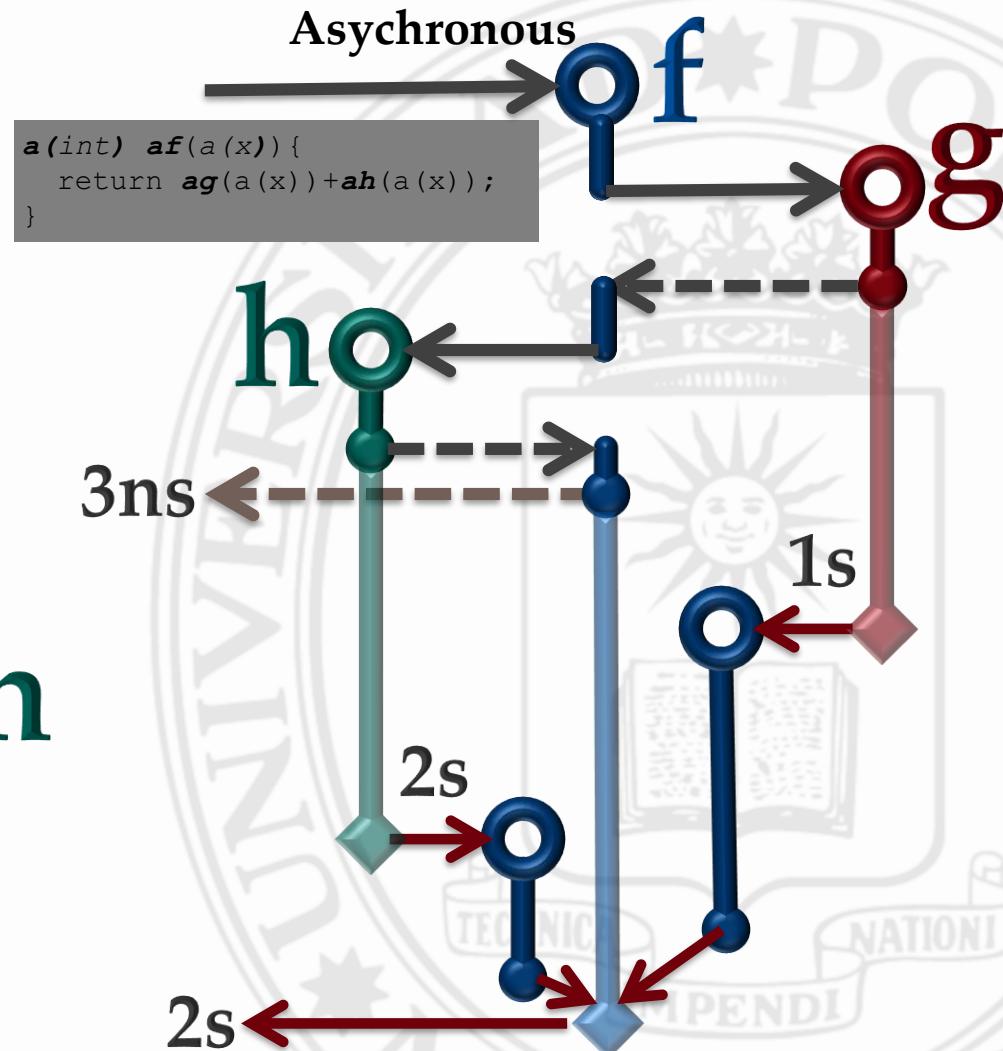
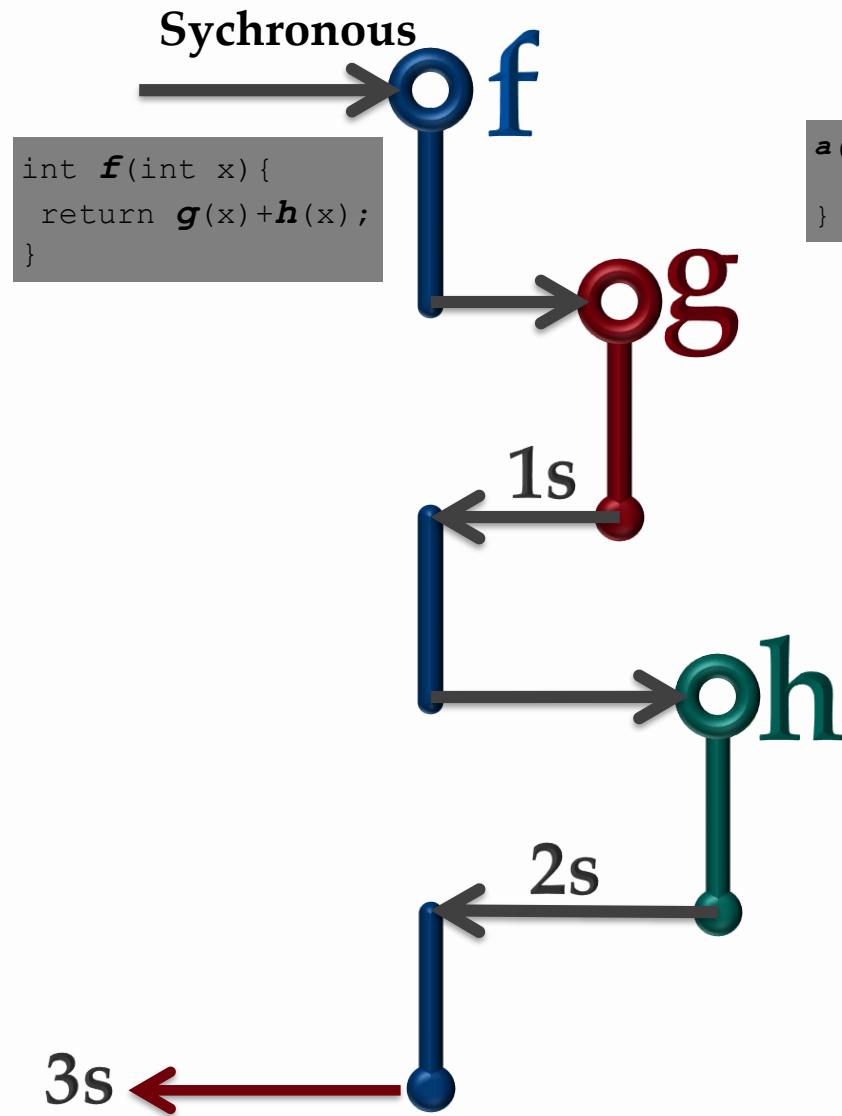
# Arquitectura Hexagonal

## Tecnologías



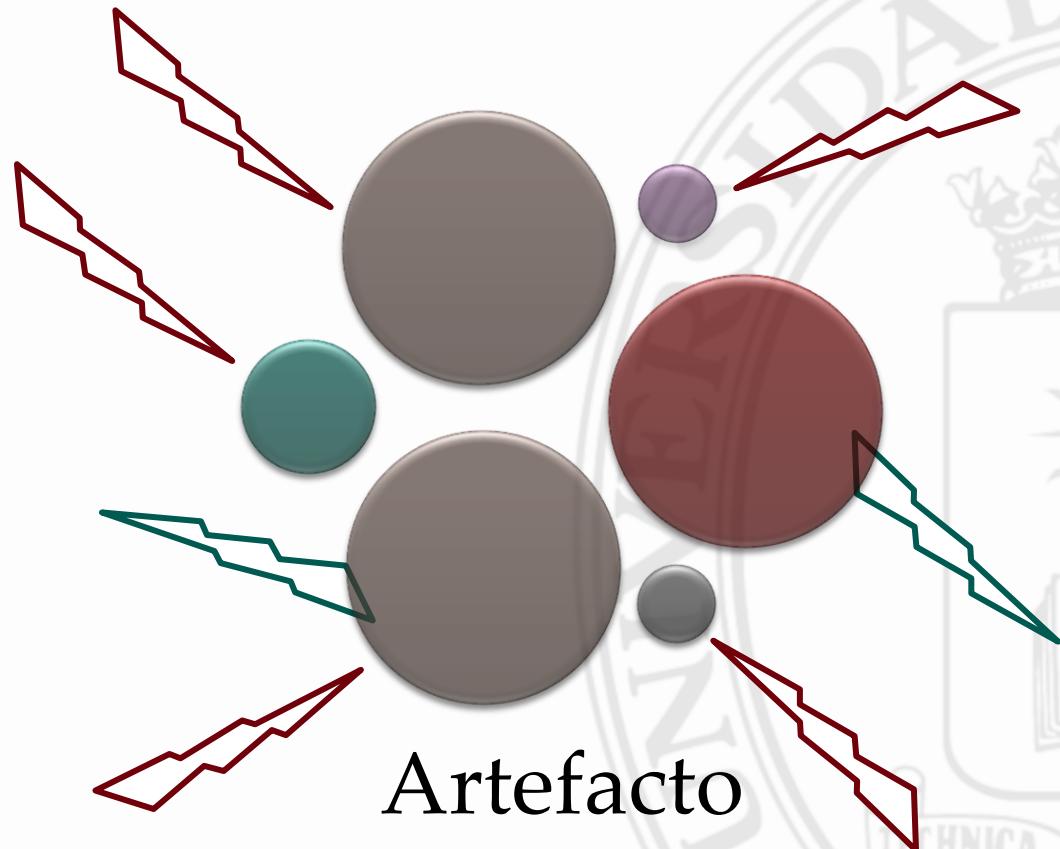
# Arquitectura dirigida por Eventos

## Síncrono - Asíncrono

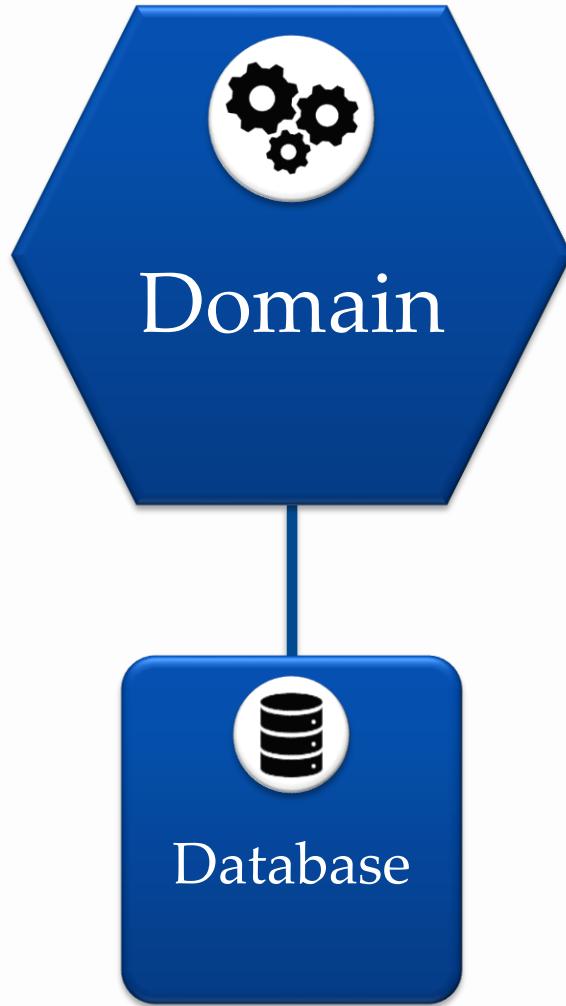


# Arquitectura dirigida por Eventos

*Event Driven Architecture*



# Aplicación Monolítica



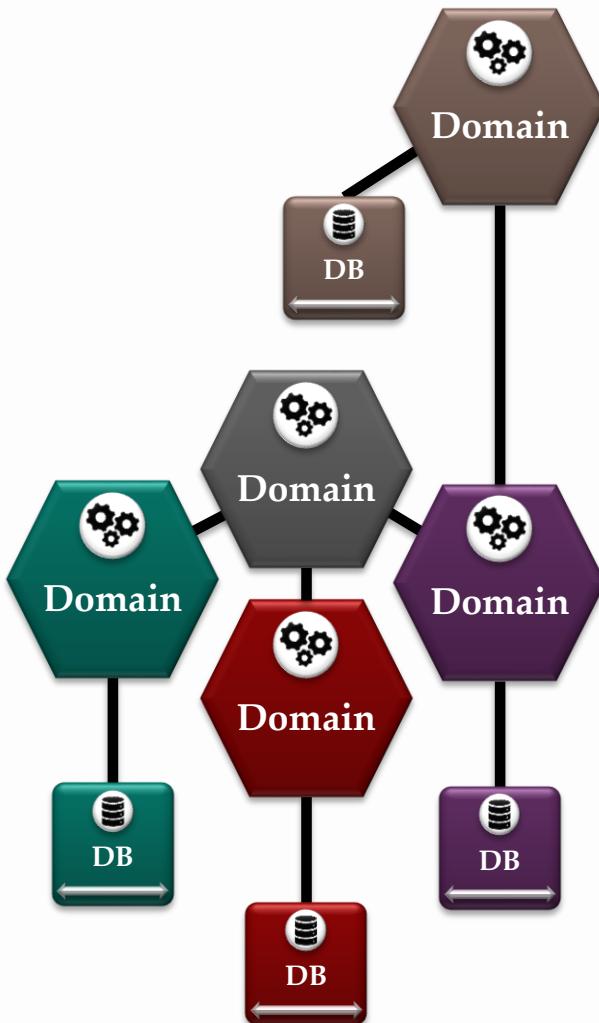
## Ventajas

- 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, multidisciplinares.
- 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

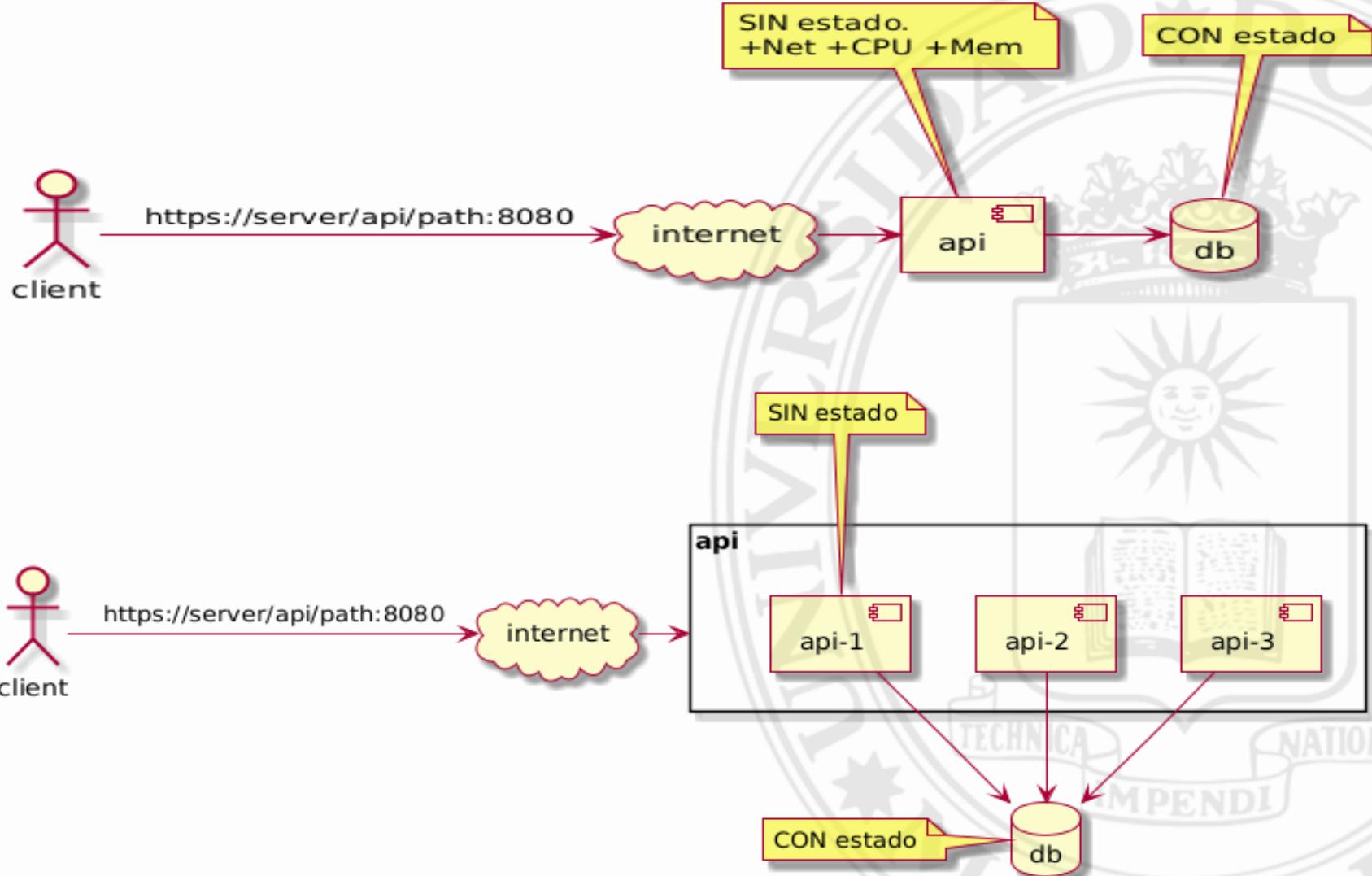
- **Distribución.** Al ser un sistema distribuido, es más complejo su despliegue

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

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

# API

## Réplica de despliegue: balanceo de carga



# Microservicios

## Patrones de diseño

### Service Discovery

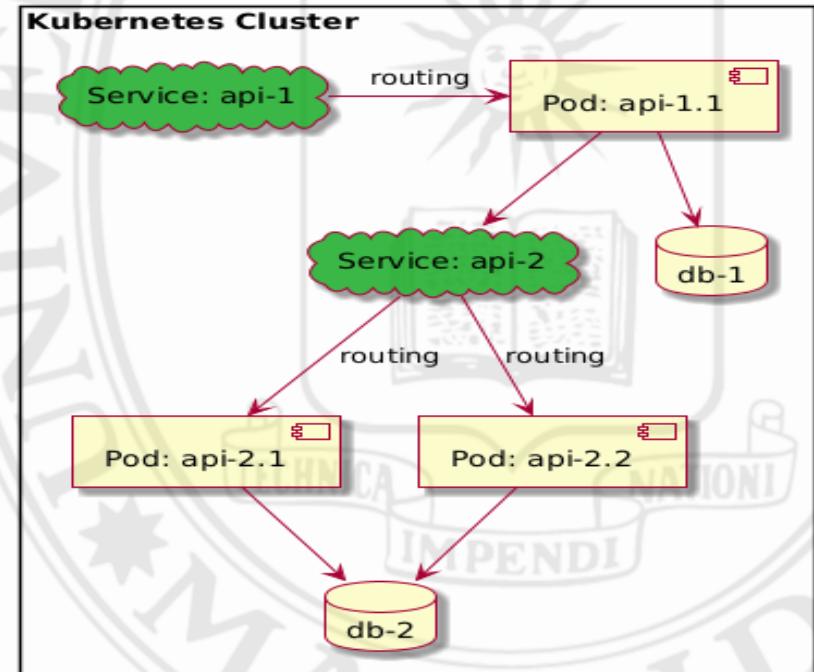
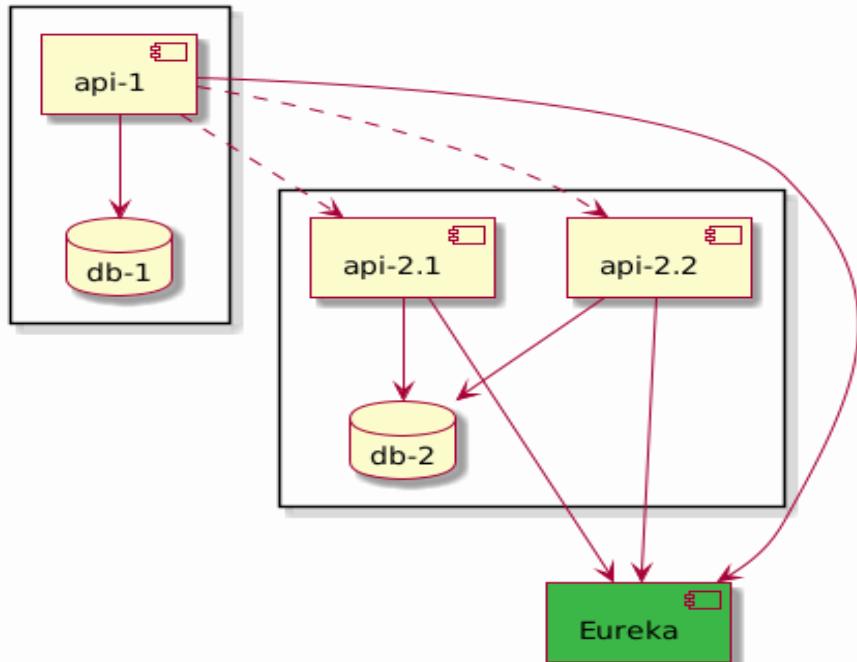
- Permite que los microservicios descubran entre sí sus ubicaciones (*IP y puerto*) sin necesidad de configuraciones estáticas.

### Client-Side Discovery

- El cliente consulta un registro de servicios y elige una instancia (balanceo en el cliente).
- Spring Cloud + Netflix Eureka*

### Server-Side Discovery

- Un proxy o *load balancer* intermedio consulta el registro y enruta la solicitud.
- Kubernetes Service, AWS Elastic Load Balancer*



# Microservicios

## Patrones de diseño

### API Gateway

- Es un único punto de entrada para todas las llamadas a los microservicios.

### Routing

- Redirige cada petición al microservicio correspondiente.

### CORS / Seguridad

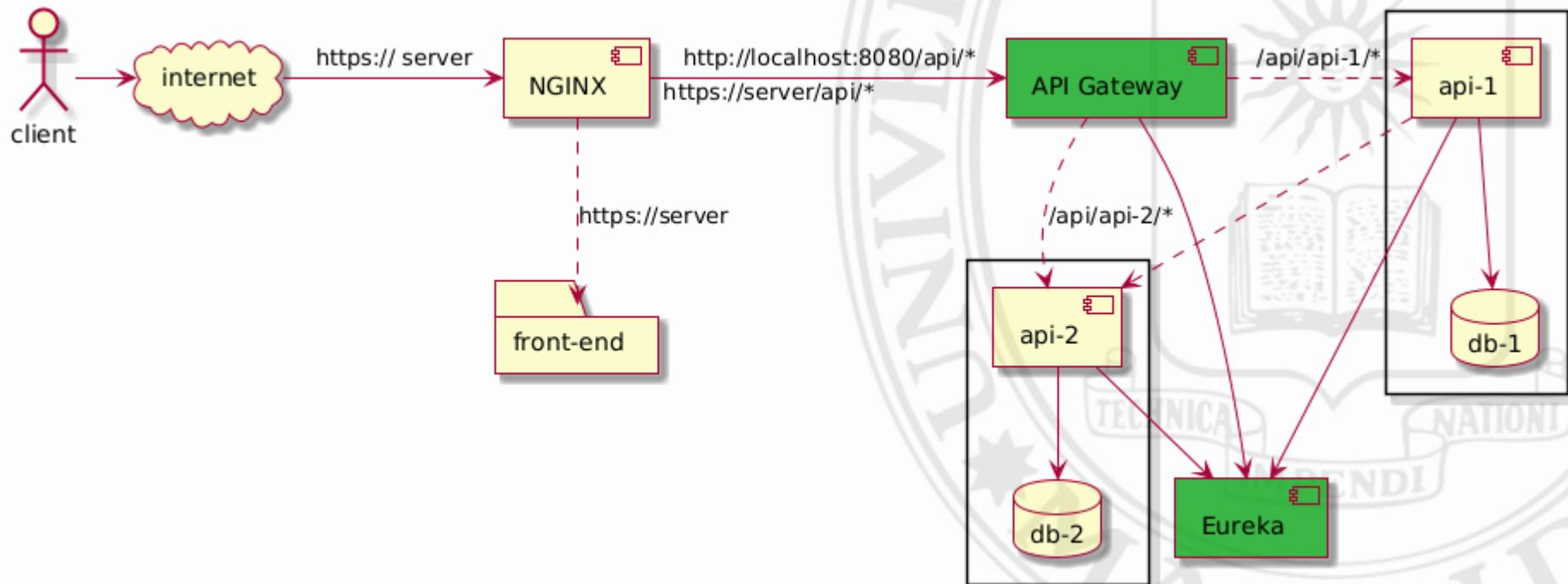
- Aplica políticas de seguridad, encabezados y filtros CORS.

### Autenticación/autorización

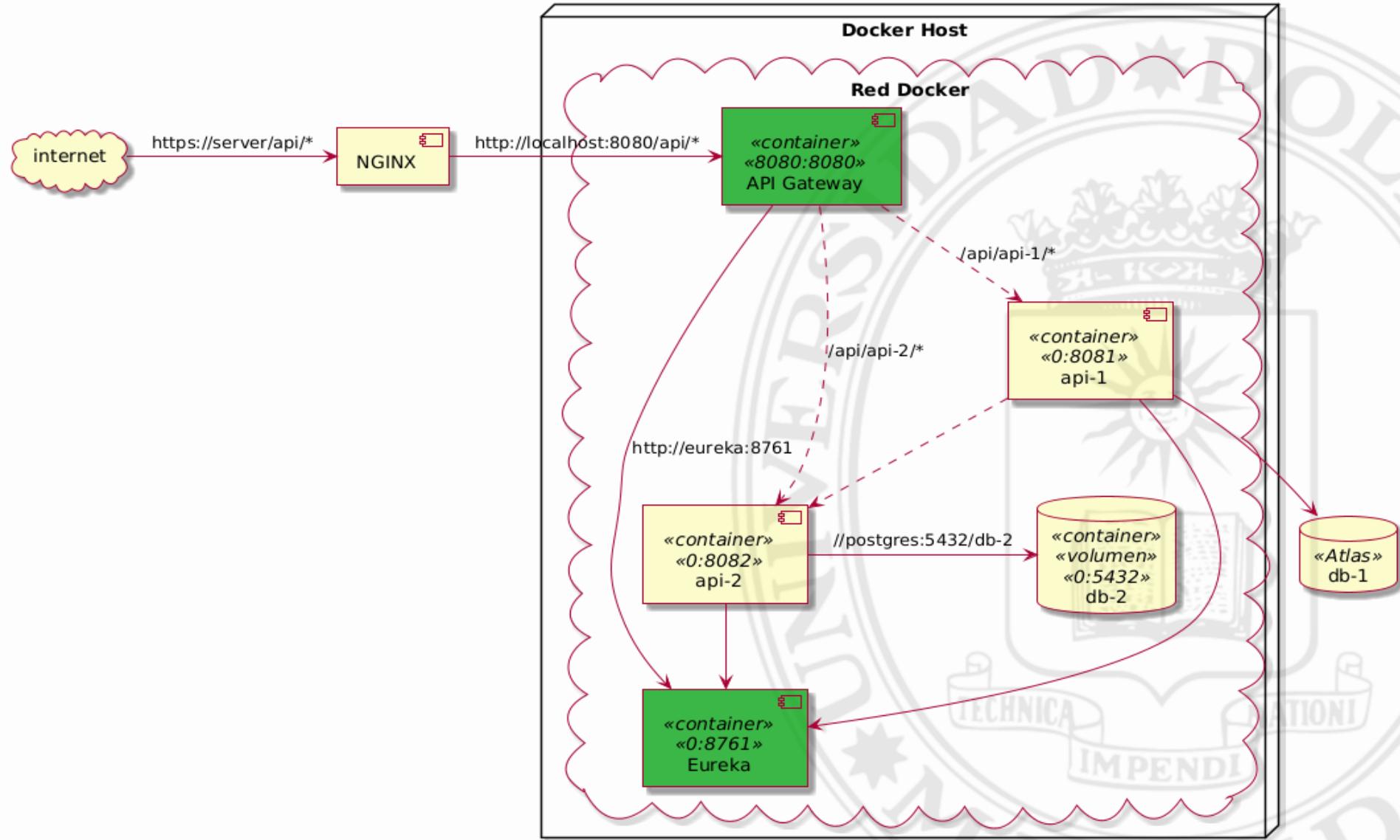
- Verifica tokens (OAuth2, JWT, etc.) antes de permitir acceso.

### Tecnologías

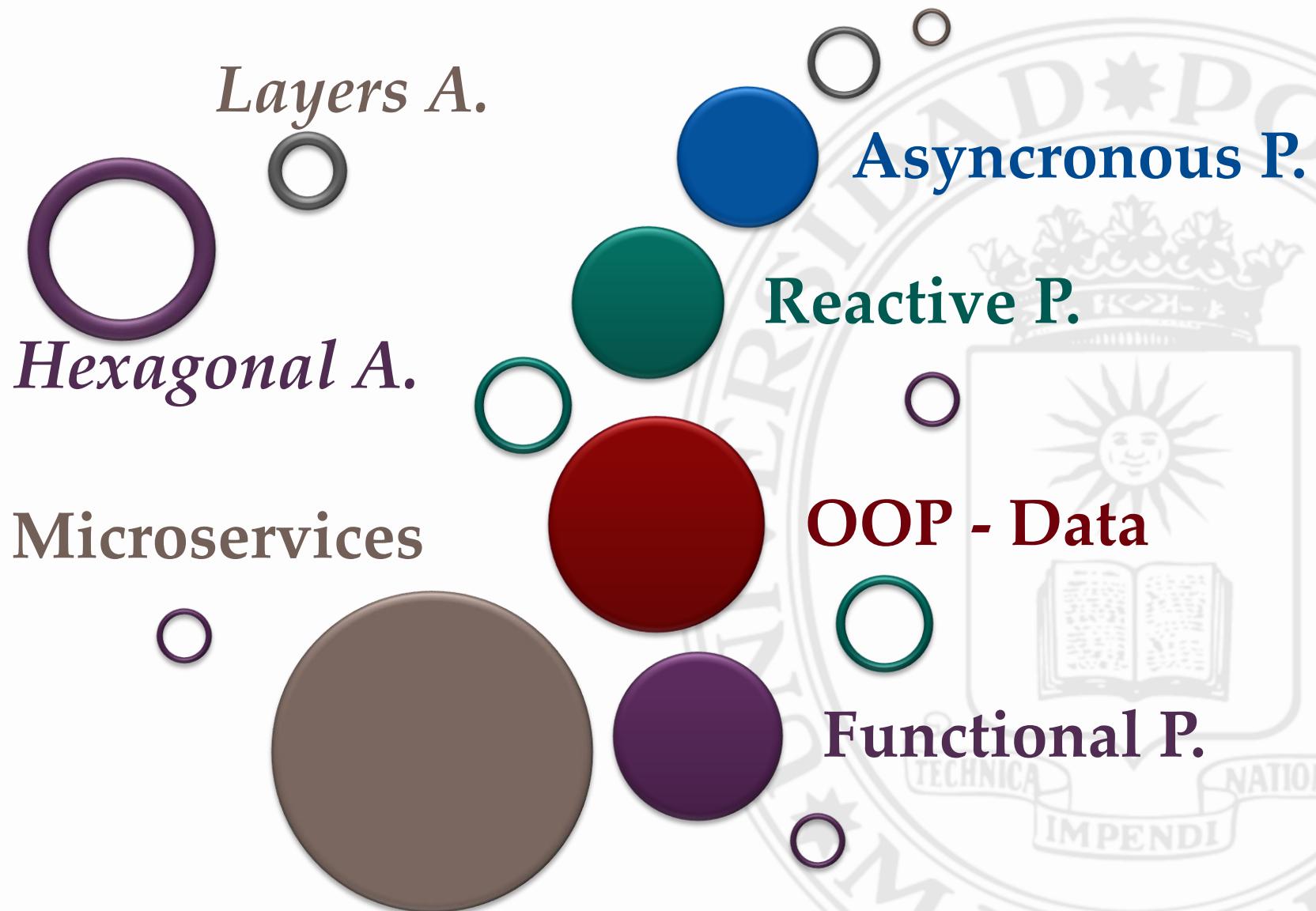
- *Spring Cloud Gateway, AWS API Gateway...*



### Docker Deploy

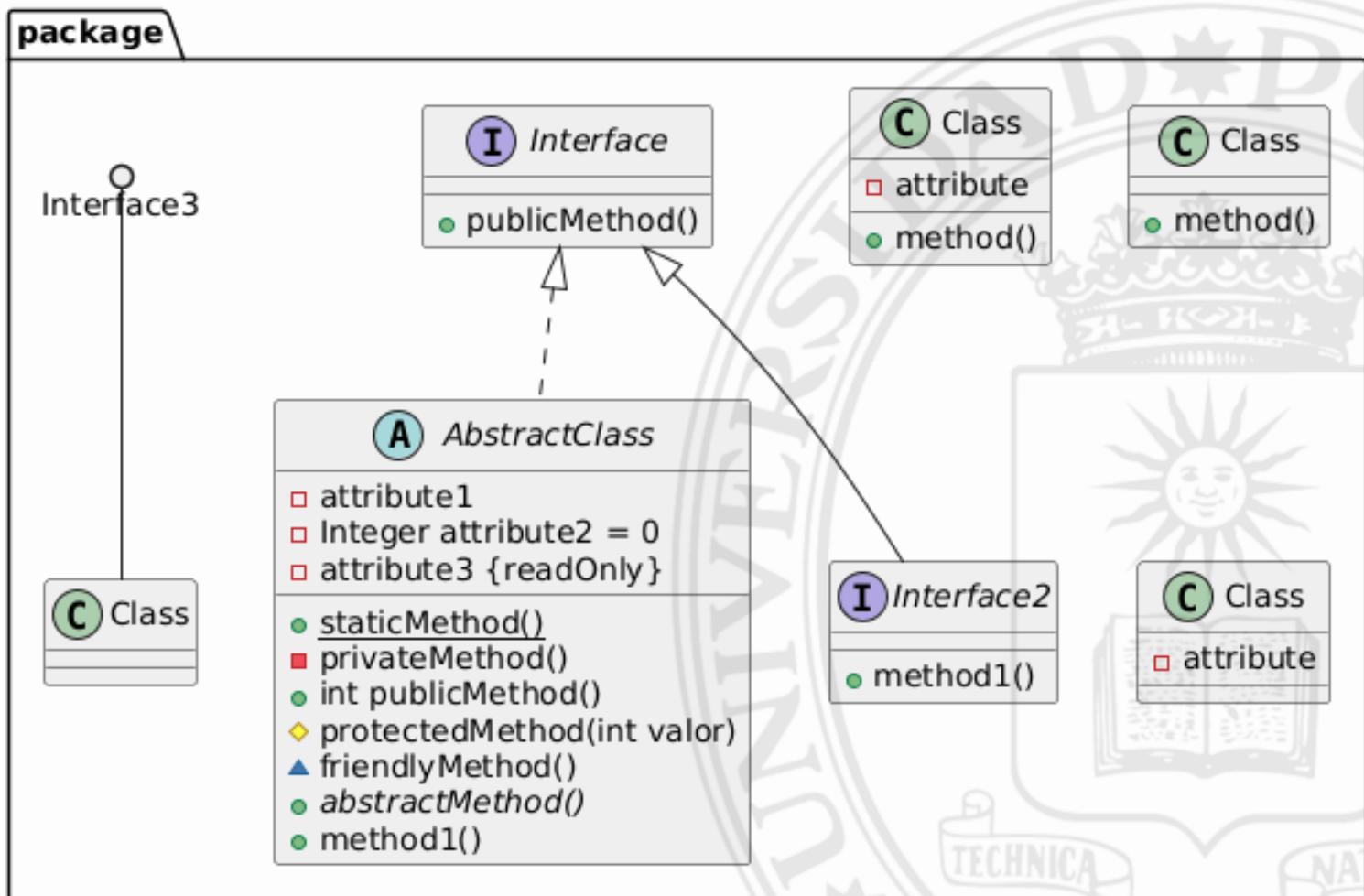


# Tendencias...



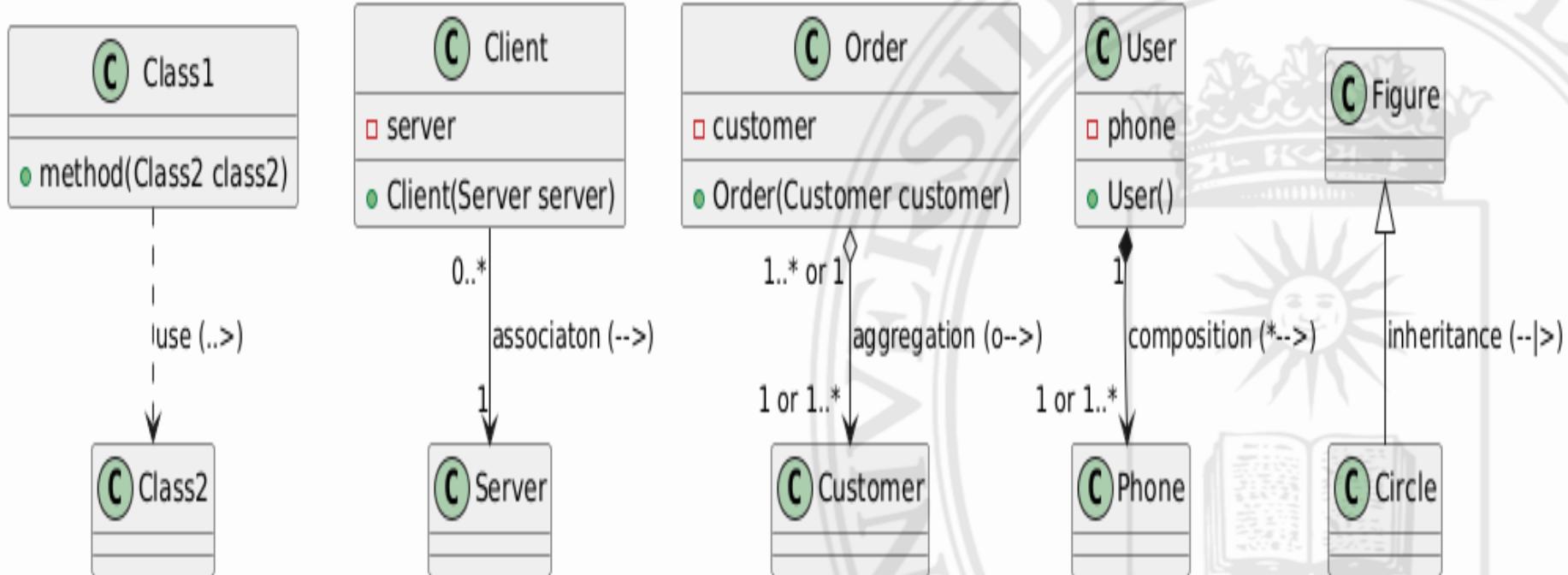
# Arquitectura

UML: *PlantUML* (on-line: *Planttext*)



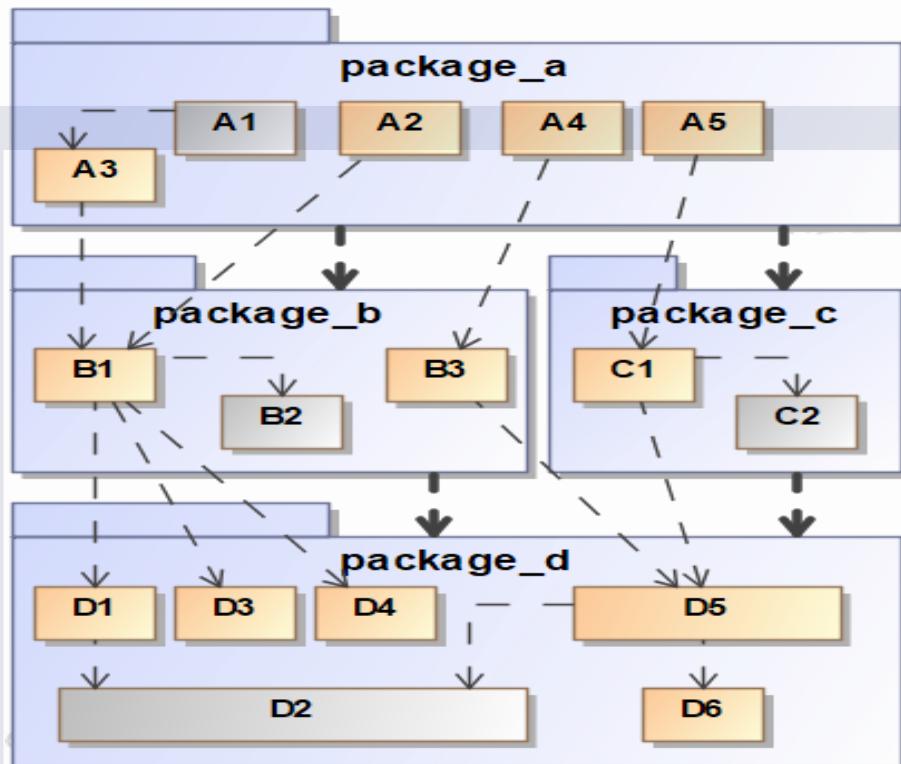
# Arquitectura UML

## Relations: Use, Association, Aggregation, Composition, Inheritance



# Arquitectura

## Principios de Robert Martin



Dependencias

### Principio de Equivalencia entre Reusable y Entregable

- Se reúsa si no se mira el código fuente.

### Principio de Reusabilidad Común

- Las clases de un paquete se reutilizan juntas.

### Principio de Cierre Común

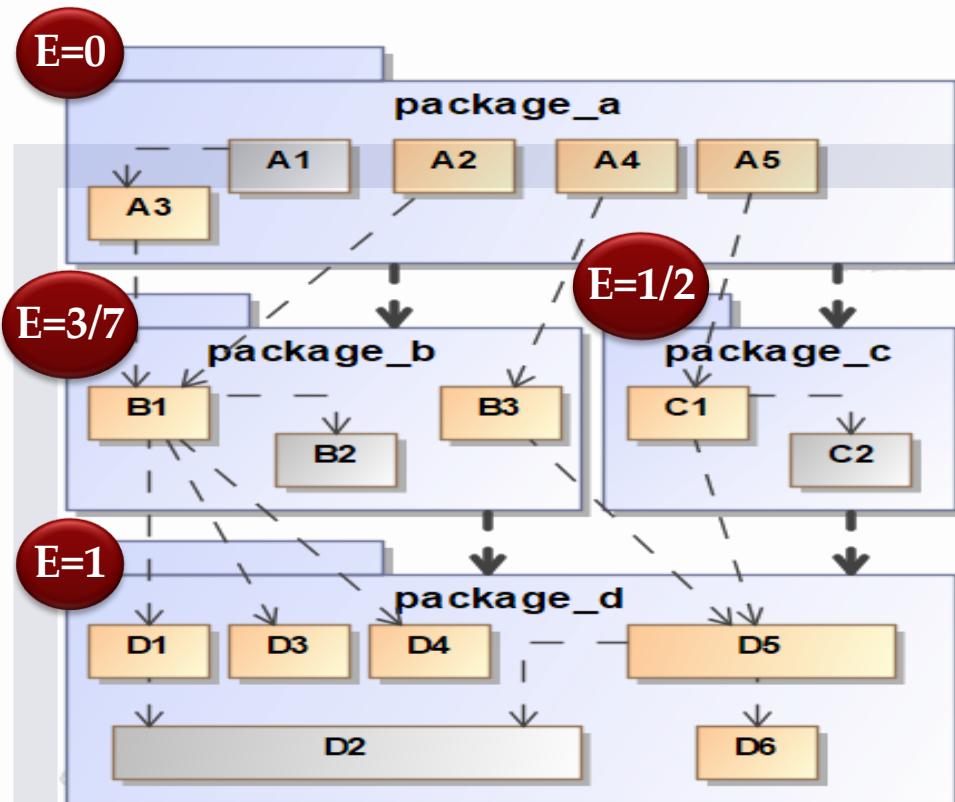
- Un cambio que afecte a una clase, afecta a todo el paquete.

### Principio de dependencias acíclicas

- Ciclo de dependencias **sin bucle**

# Arquitectura

## Principios de Robert Martin



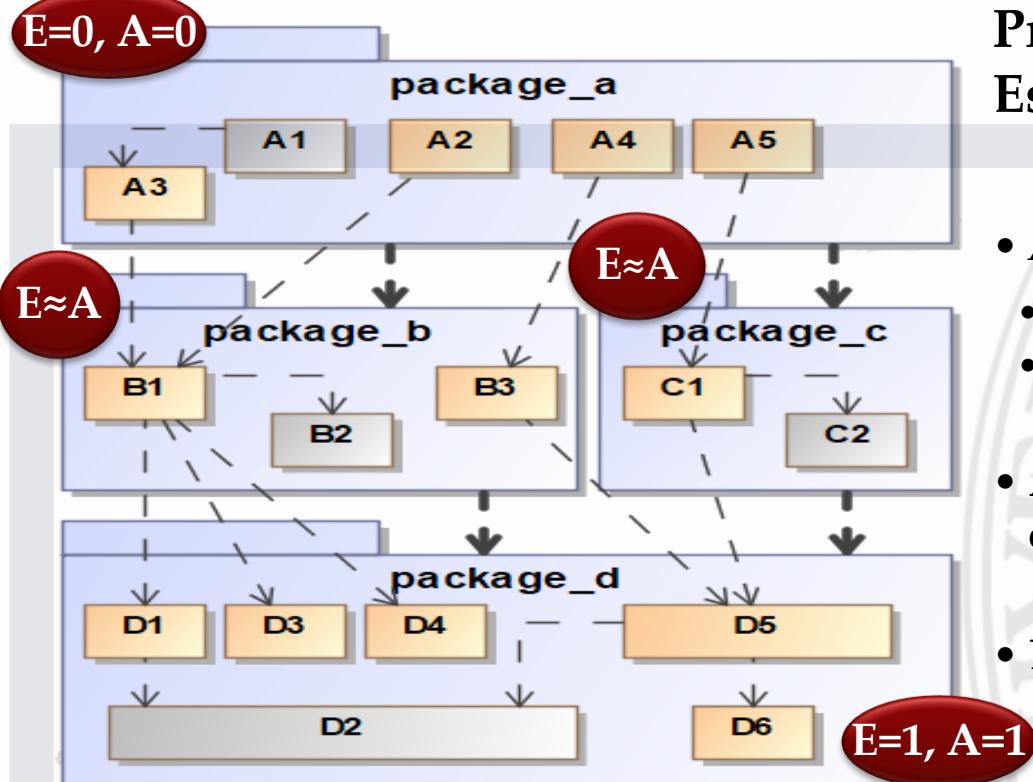
Dependencias

### Principio de Dependencias Estables

- Estable en el sentido de pesado, difícil de cambiar.
- Inestable en el sentido de ligero, fácil de cambiar.
- **Estabilidad** =  $\frac{Aa}{Aa+Ae} \rightarrow [1..0]$ , 1 es totalmente estable, 0 nada estable, es decir, inestable.
  - $Aa$  - Acoplamiento aferente: N° de clases de afuera que dependen del paquete.
  - $Ae$  - Acoplamiento eferente: el paquete depende de un N° de clases de fuera.
- **Inestabilidad** =  $1 - \text{Estabilidad}$

# Arquitectura

## Principios de Robert Martin

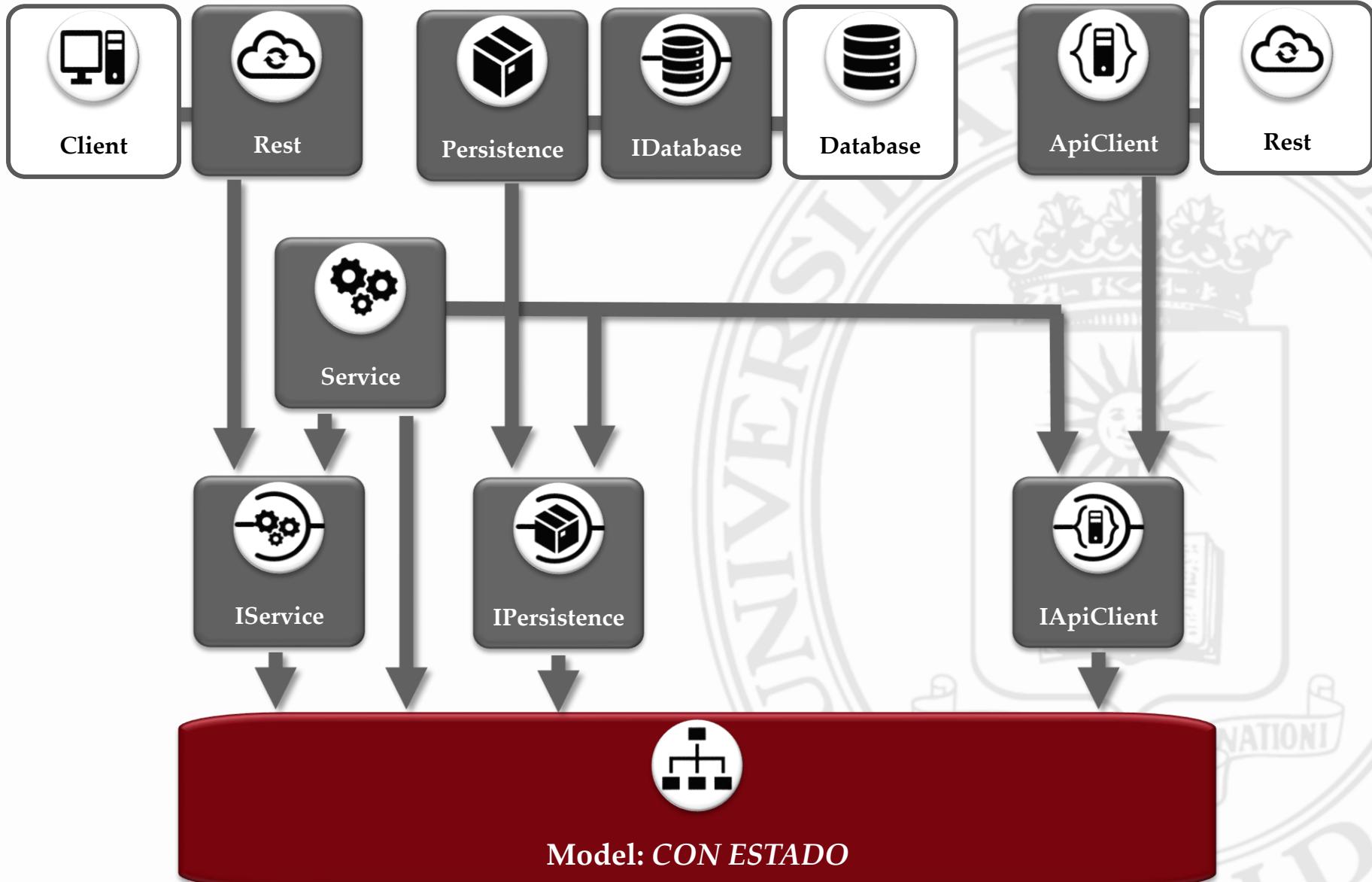


### Principio de Abstracciones Estables

- $\text{Abstracción} = \frac{Ca}{Ct}$ 
  - Ca = N° de clases abstractas
  - Ct = N° total de clases
- Abstracción debe ser  $\geq$  que la anterior, en cada capa
- Estabilidad  $\approx$  Abstracción

Dependencias

# Modelo del Dominio POO



# Modelo

## Modelización

### 1. Clases

- Dirección
- Tipo
- Multiplicidad

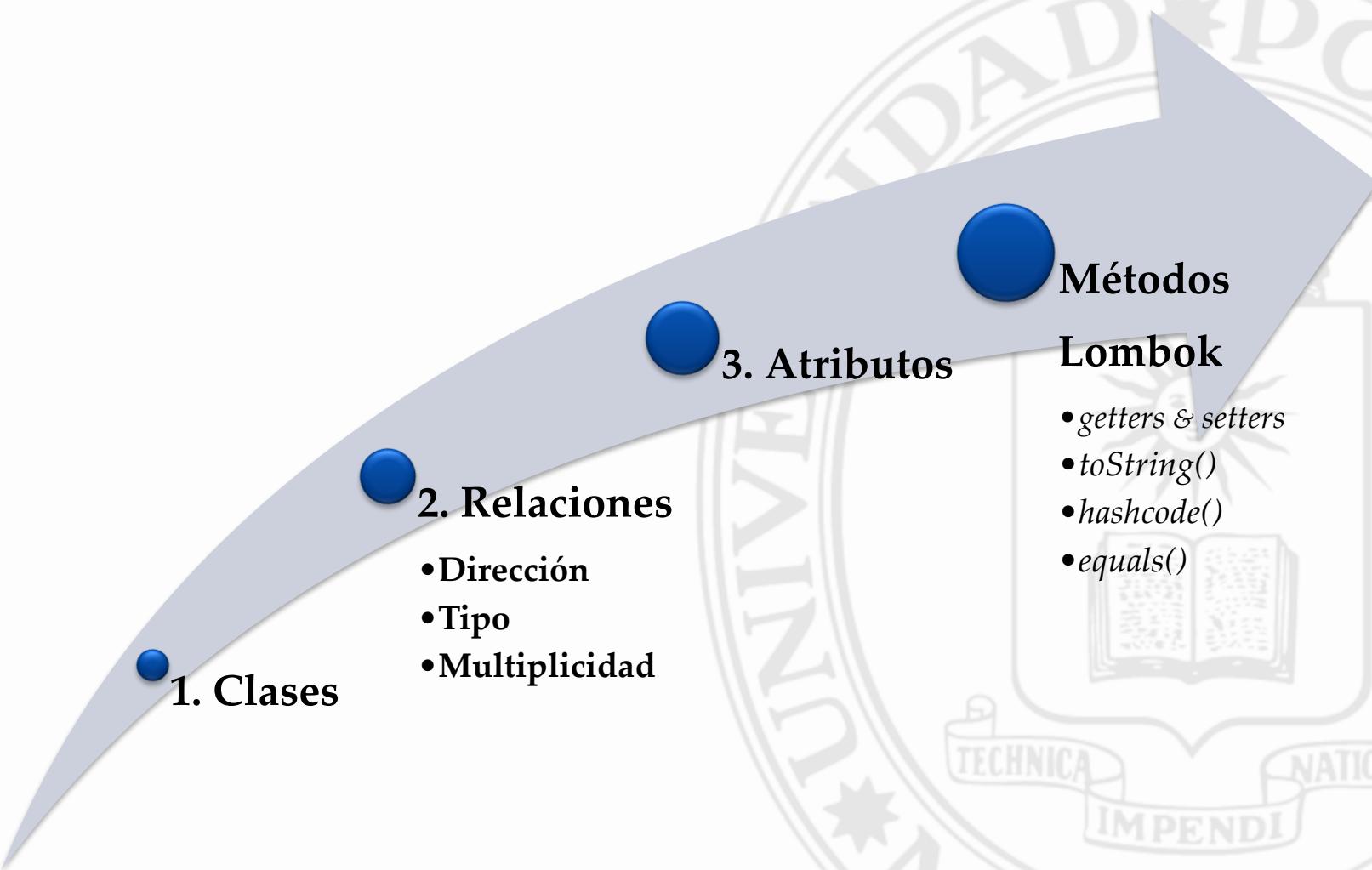
### 2. Relaciones

### 3. Atributos

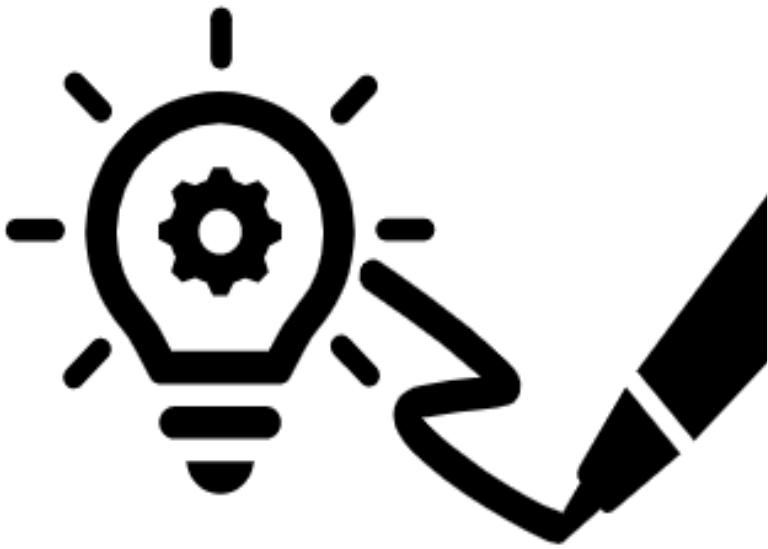
### Métodos

### Lombok

- *getters & setters*
- *toString()*
- *hashcode()*
- *equals()*



# Modelo Modelización



Biblioteca

Nos piden modelizar una **biblioteca**.

Existen libros, con su título, su ISBN y una lista de autores. Los libros están asociados a un conjunto de temas.

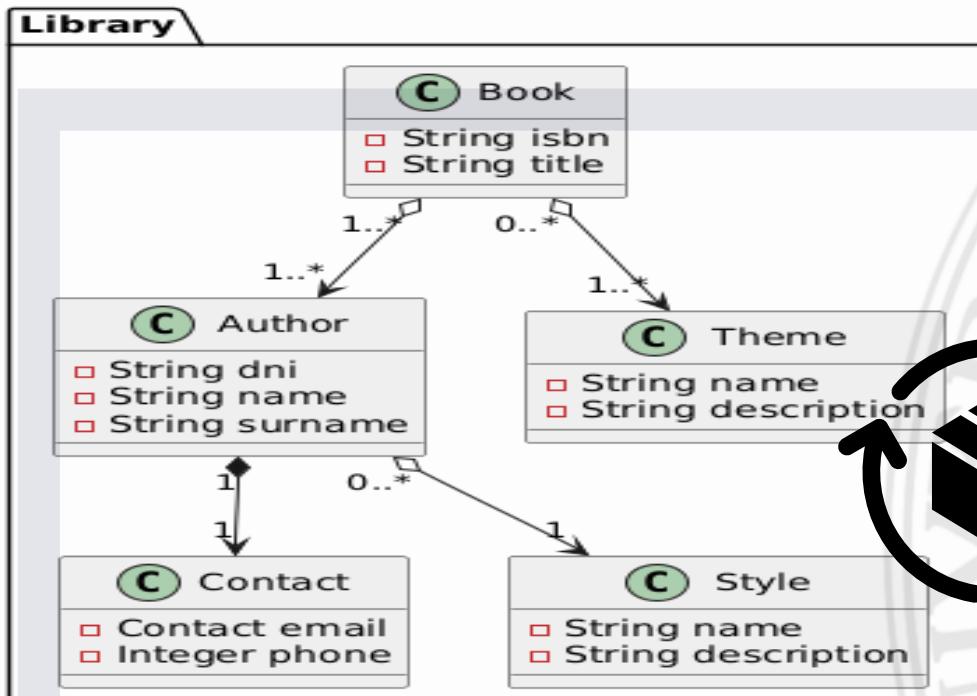
Un tema tiene un nombre y un tema puede estar asociado a muchos libros.

De cada autor queremos guardar su nombre y su apellido y sus datos de contacto, que están formados por un email y un teléfono.

Un autor tiene un estilo, pero un estilo puede ser utilizado por muchos autores. De un estilo se guarda el nombre y la descripción.

El cliente quiere poder gestionar libros, los autores...

# Modelo Modelización



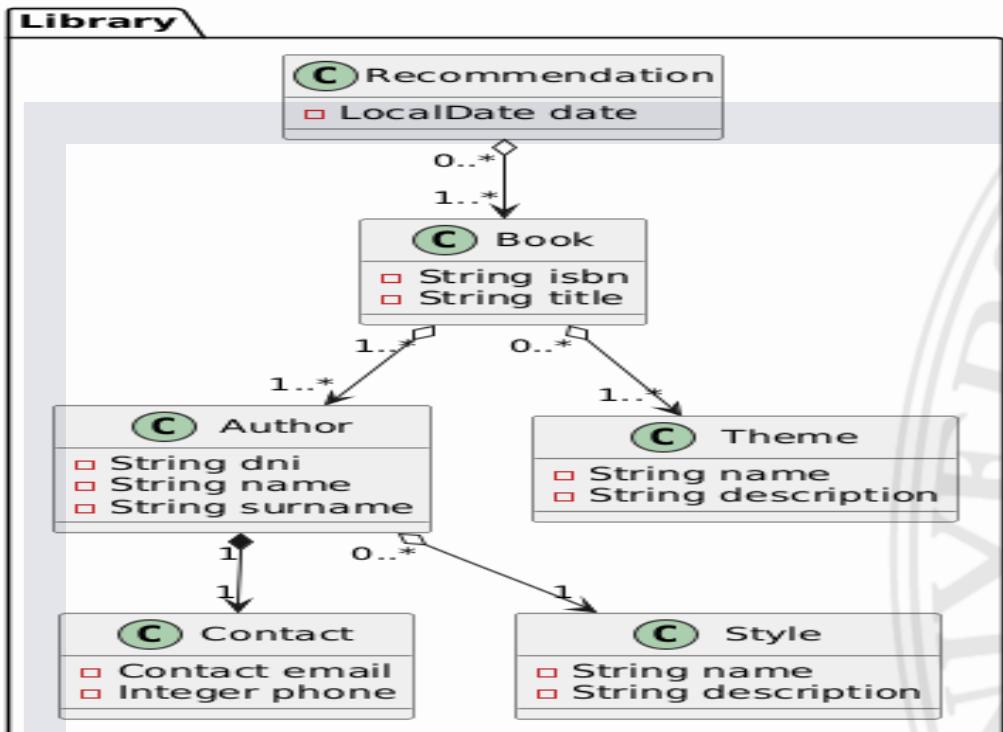
Biblioteca

Nos piden modelizar una biblioteca.

Existen libros, con su título, su ISBN ...

Después de LIBERAR (en producción) la primera versión, el cliente nos pide ahora que en un libro se indique si está recomendado. El pretende recomendar cada mes una serie de libros.

# Modelo Modelización

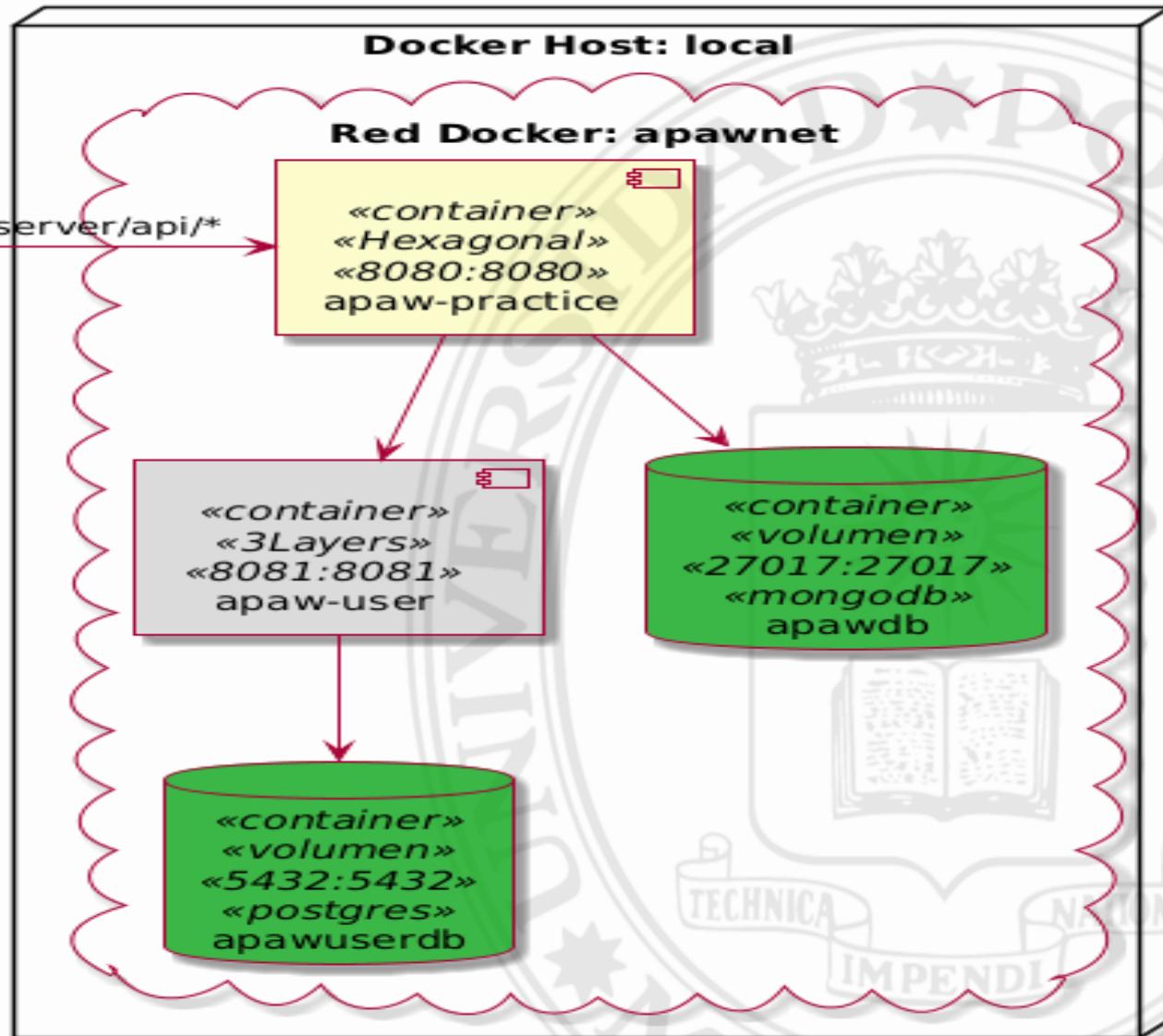


Biblioteca

Nos piden modelizar una biblioteca.

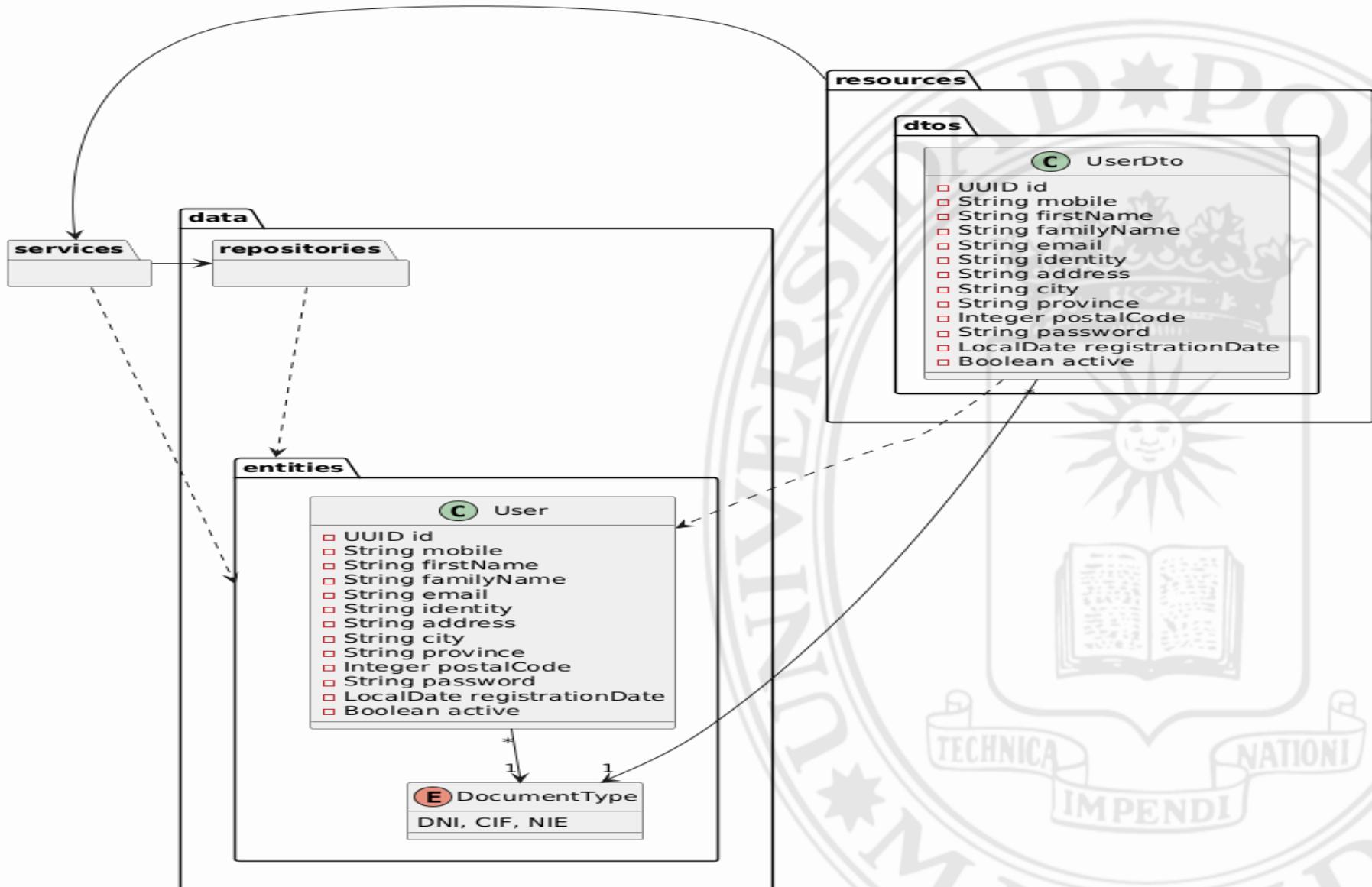
Existen libros, con su título, su ISBN ...

Después de liberar la primera versión, el cliente nos pide ahora que en un libro se indique si está recomendado. El pretende recomendar cada mes una serie de libros.



# Práctica

## Microservicio: apaw-user

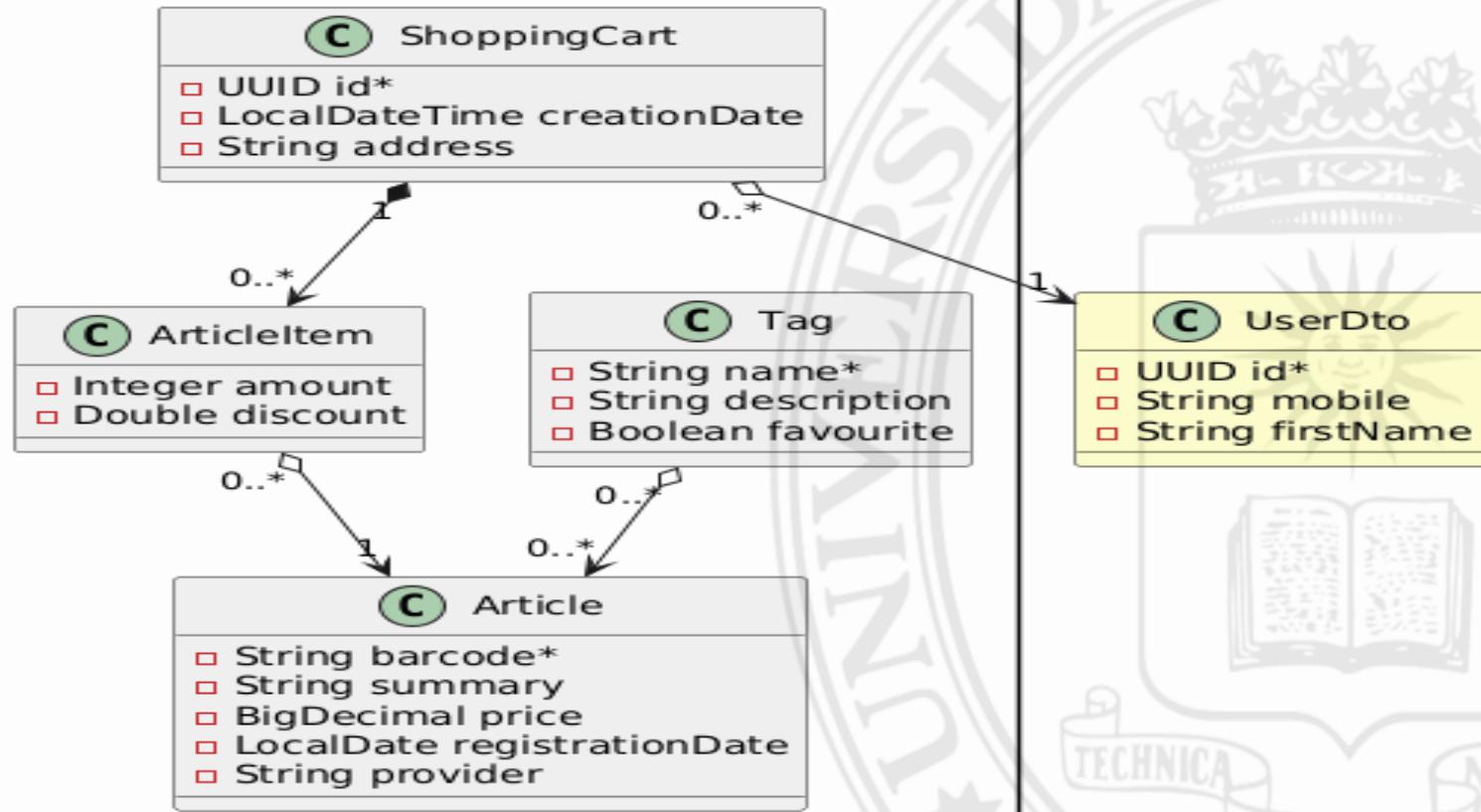


# Practica apaw-practice (shop)

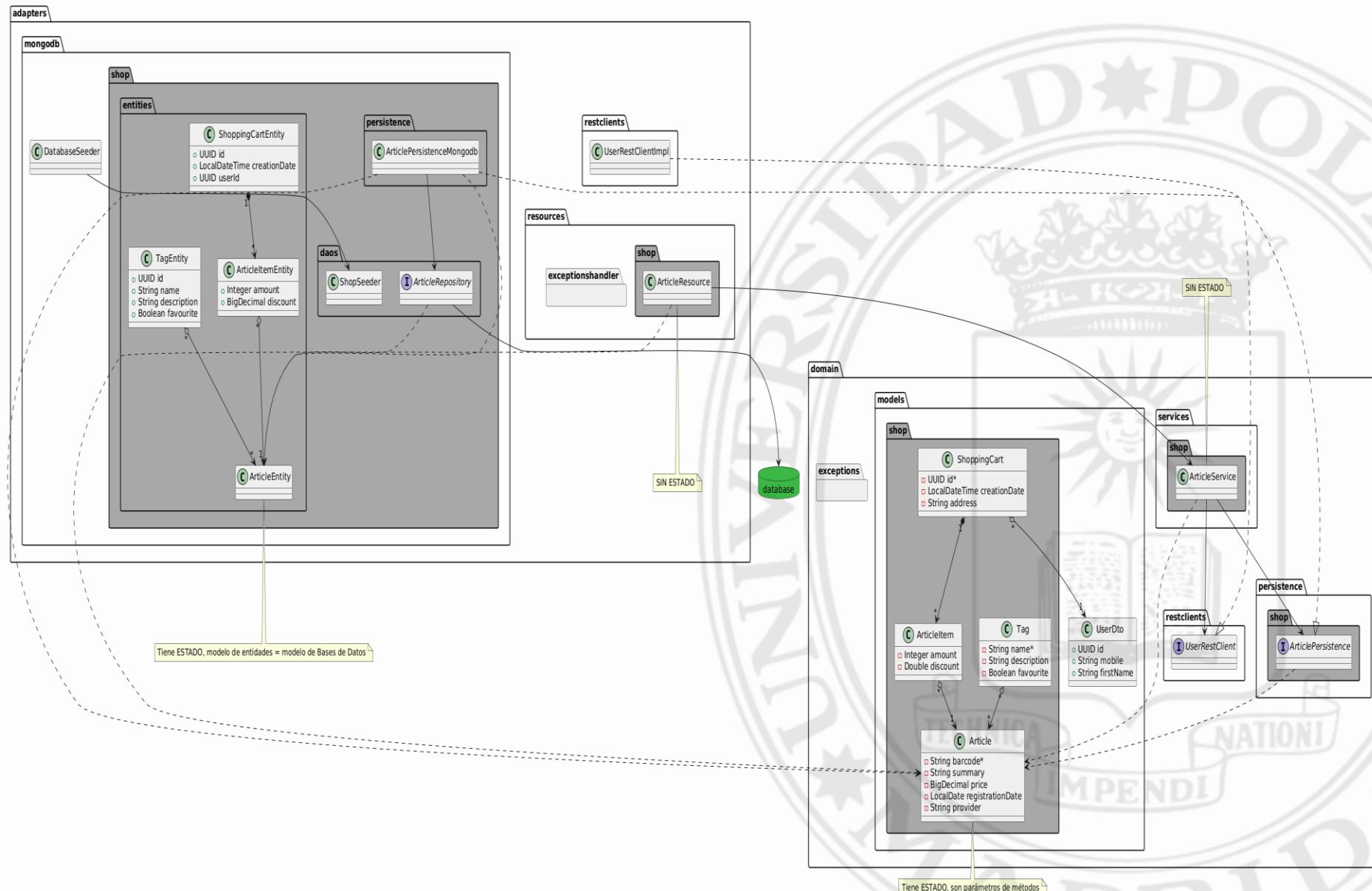
domain

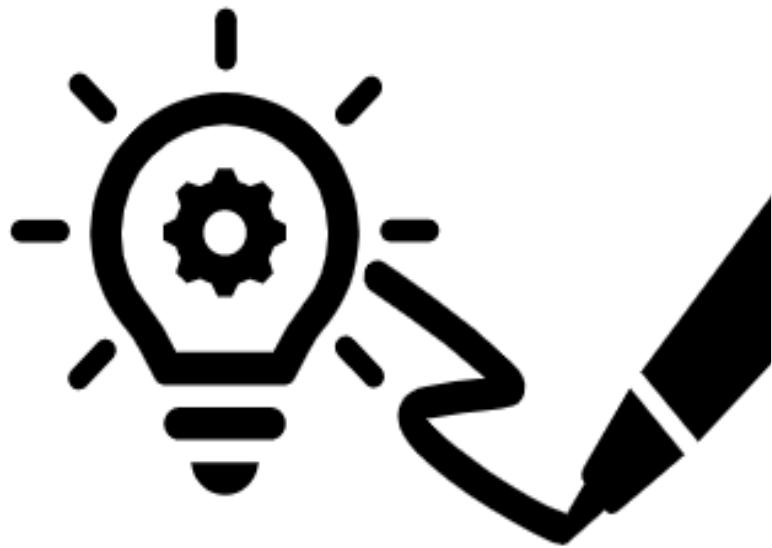
models

shop



# Practica apaw-practice (shop)





## Ejercicio

Nos piden modelizar:

- Company
- Doctor
- Car Hire
- Hotel
- Sport
- Shop
- Veterinary Clinic
- Department
- Movie

...

Con 4 clases + *UserDto*, cada una con al menos 3 atributos y un total de al menos de 15 atributos. Relaciones 1..n, n..1 ó 1..1 y n..n, con *UserDto* que es n..1. Proyecto y clases únicas para todos los grupos. Dentro de un proyecto, nombre de atributos únicos

# 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 Validations

```
public class UserDto {  
    private UUID id;  
    @NotNull  
    @NotBlank  
    @Pattern(regexp = Validations.NINE_DIGITS)  
    private String mobile;  
    @NotNull  
    @NotBlank  
    private String firstName;
```

```
@NotNull jakarta.validation.constraints  
@NotBlank jakarta.validation.constraints  
@Pattern jakarta.validation.constraints  
@Max jakarta.validation.constraints  
@Min jakarta.validation.constraints  
*  
@AssertFalse jakarta.validation.constraints  
@AssertTrue jakarta.validation.constraints  
@DecimalMax jakarta.validation.constraints  
@DecimalMin jakarta.validation.constraints  
@Digits jakarta.validation.constraints  
@Email jakarta.validation.constraints  
@Future jakarta.validation.constraints  
@FutureOrPresent jakarta.validation.constraints  
@Negative jakarta.validation.constraints  
@NegativeOrZero jakarta.validation.constraints  
@NotEmpty jakarta.validation.constraints  
@Null jakarta.validation.constraints  
@Past jakarta.validation.constraints  
@PastOrPresent jakarta.validation.constraints  
@Positive jakarta.validation.constraints  
@PositiveOrZero jakarta.validation.constraints  
@Size jakarta.validation.constraints
```

## Singleton

- Se garantiza que una clase sólo tenga una instancia y se proporciona un acceso global a ella.

## Builder

- Separa la construcción de objeto complejo de su representación, permitiendo diferentes construcciones.

## Composite

- Permite estructuras en árbol tratando por igual a las hojas que a los elementos compuestos.

## Iterator

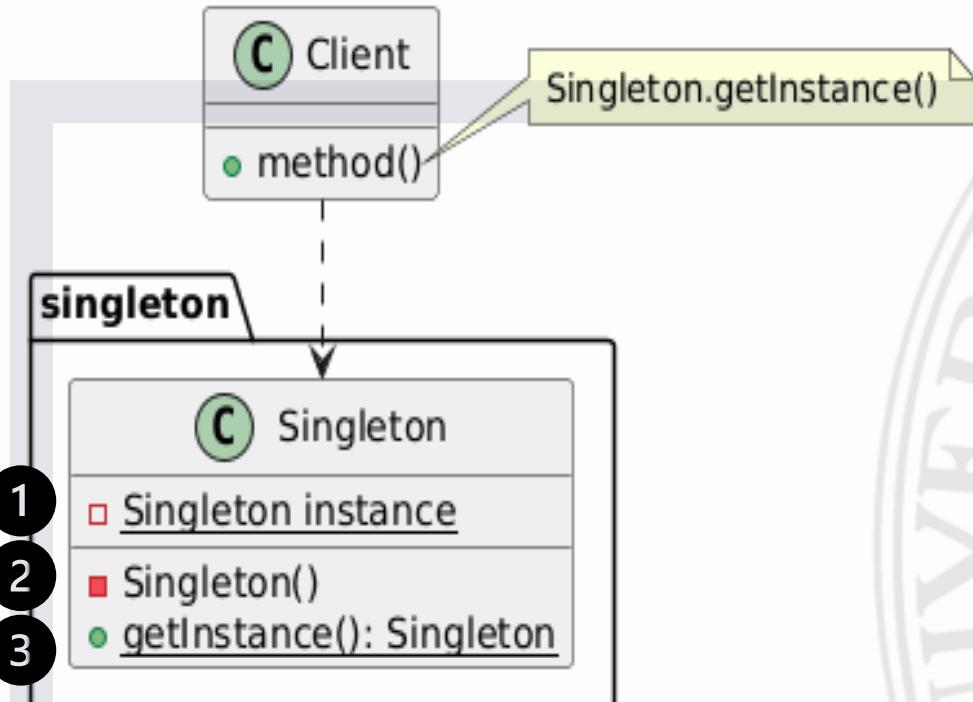
- Proporciona un modo de acceder secuencialmente a los elementos de un objeto sin exponer su representación interna.

## Memento

- Externaliza el estado interno de un objeto sin violar la encapsulación.

# Singleton

Propósito: Creación. Ámbito: objeto



Se garantiza que una clase sólo tenga una instancia y se proporciona un acceso global a ella

① Atributo privado estático de la propia clase.

- Creación temprana (*eager*).

② Constructor privado.

③ Método estático y público para devolver el atributo.

- Creación perezosa (*lazy*)

# Singleton

## Patrón Único

### Logger



Logger

- String logs
- Logger()
- addLog(String log)
- getLogs():String
- clear()

## Logger

### solution



Logger

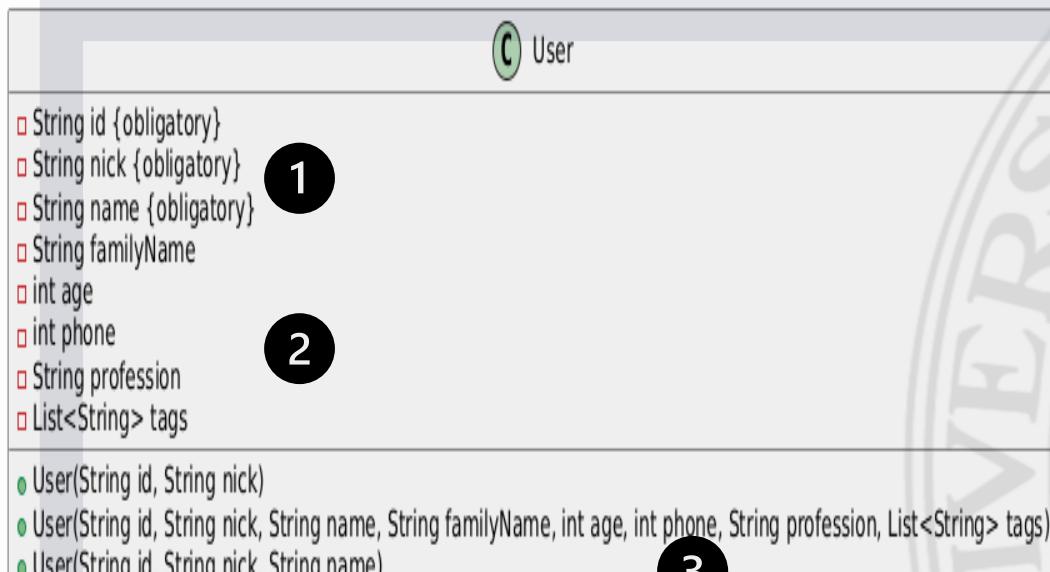
- Logger logger
- String logs
- Logger()
- getLogger(): Logger
- addLog(String log)
- getLogs():String
- clear()

## Logger & Singleton

# Builder

Propósito: Creación. Ámbito: objeto

builder



## Builder

```
User user = new User("id1", "Paco", "Jose", "De Miguel", 25, 6666666666, "Profesor",  
    Arrays.asList("Director", "Socio", "Consejo"));
```

① Atributos obligatorios.

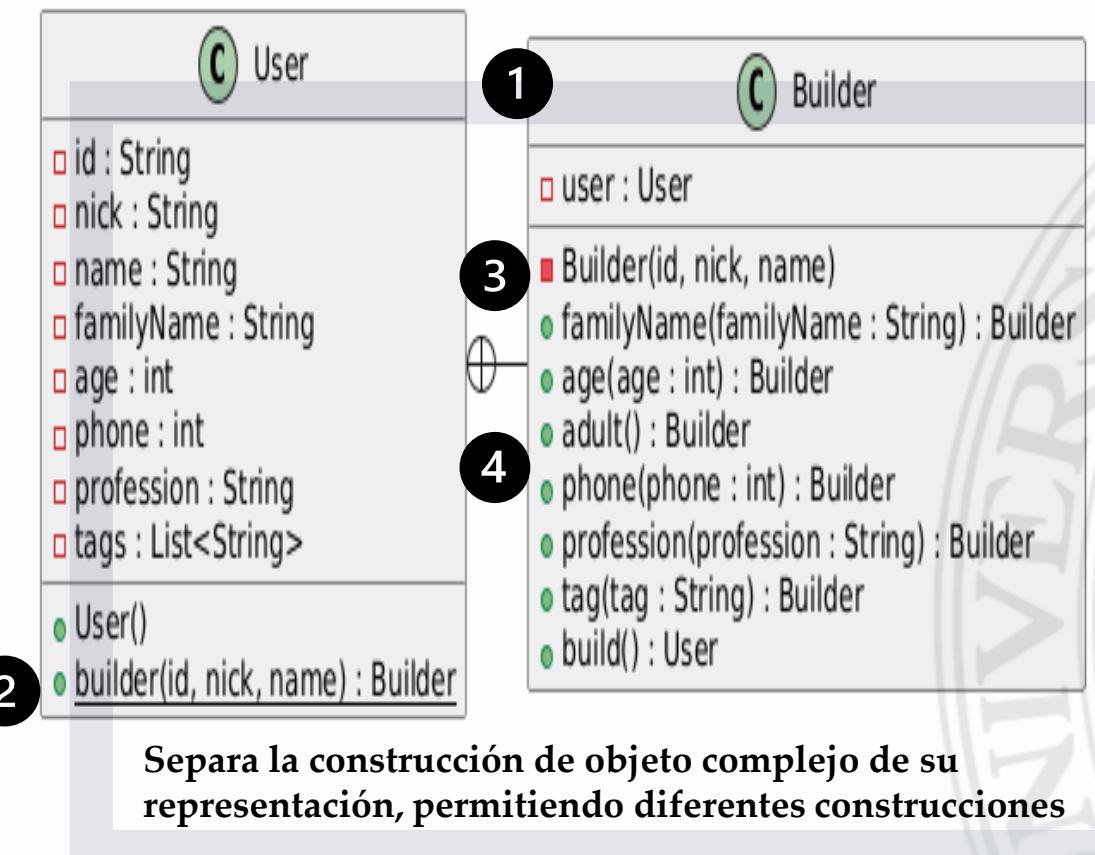
② Atributos opcionales.

③ *Code Smell*

- Constructores con demasiados parámetros
- Muchos constructores
- Poco usables.

# Builder (@Builder de Lombok)

## Propósito: Creación. Ámbito: objeto



```
User user = User.builder("1", "Paco", "Jose").familyName("De Miguel").phone(66666666).adult()
.profession("Profesor").tag("Director").tag("socio").tag("Consejo").build();
```

```
User user = User.builder("1", "Paco", "Jose").phone(66666666).familyName("De Miguel").build();
```

① Clase Interna *Builder*.

② Método estático que crea una instancia del *Builder*.

③ Constructor con los parámetros obligatorios.

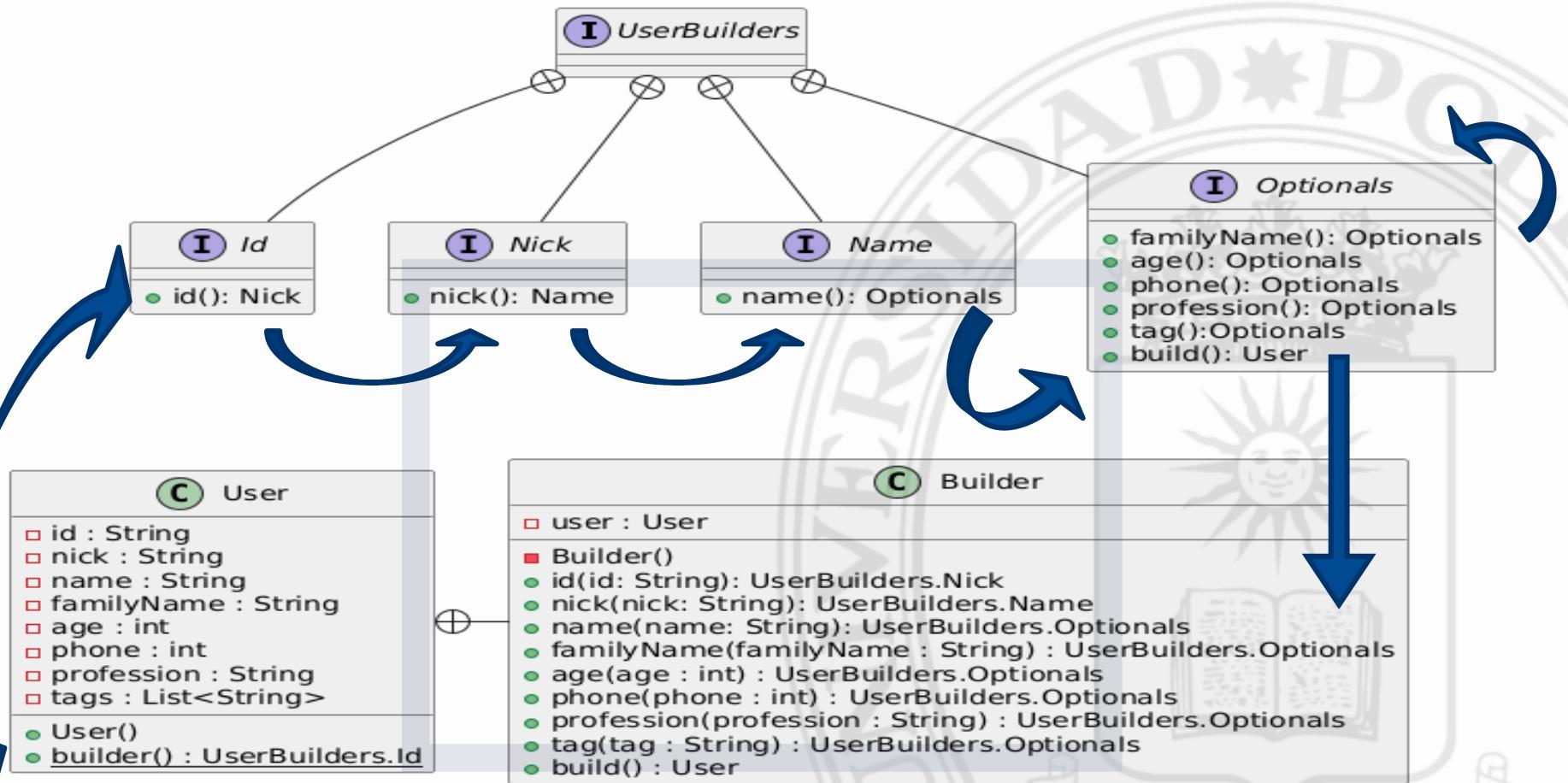
④ Métodos devuelven *this*

- ¿Qué pasa si son muchos parámetros obligatorios?

- ¿Qué pasa si tenemos

# Builder

# Propósito: Creación. Ámbito: objeto



**Separa la construcción de objeto complejo de su representación, permitiendo diferentes construcciones**

```
User user = User.builder().id("1").nick("Paco").name("Jose").tag("Director").age(18).build();
```



<https://github.com/miw-upm/apaw>

- Package: es.upm.miw.pd.builder

## Builder in action

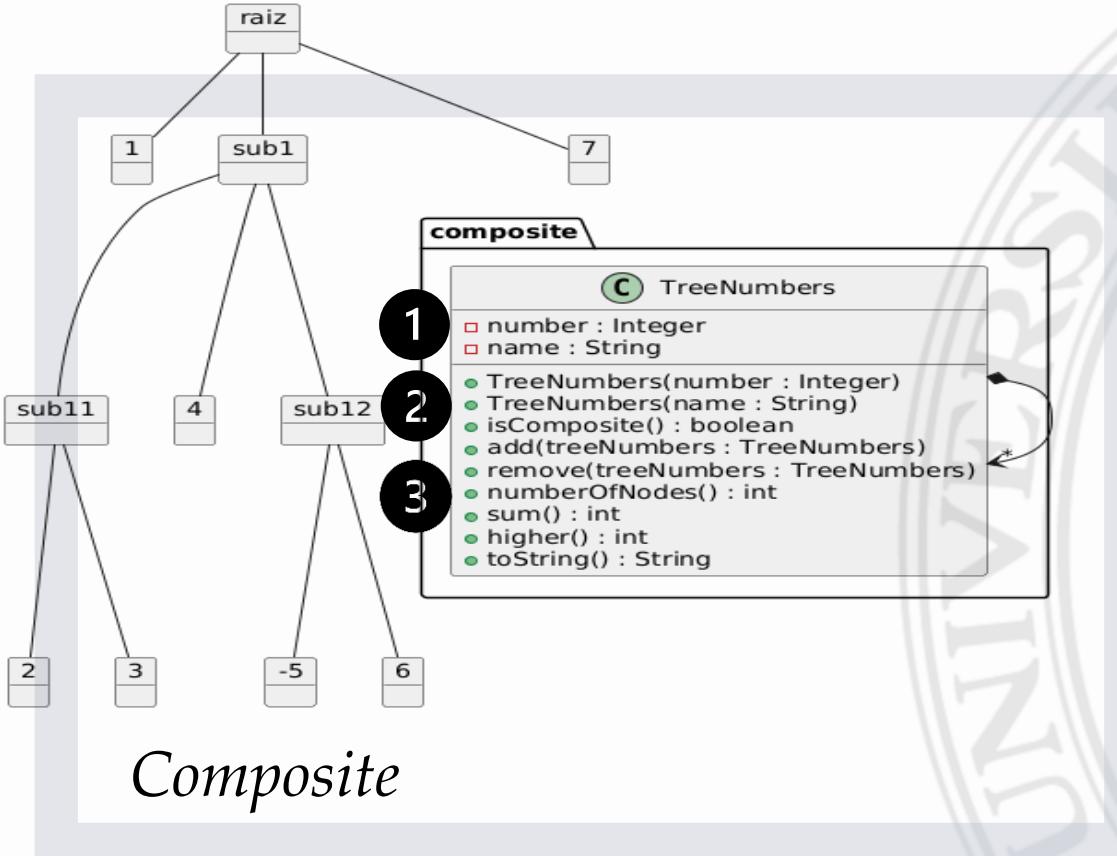
- User
- Soluciones

## 📝 Ejercicios

- Aplicar el patrón *builder* a la clase *Article*, atributos obligatorios: *id*, opcionales: *resto*.
- Aplicar el patrón *builder* a la clase *Article*, atributos obligatorios: *id, reference, description & retailPrice*, opcionales: *resto*.
- Aplicar solución con interface para secuenciar los obligatorios.
- Añadir a la clase *Article* el atributo *Provider*, debería lanzarse el *builder* de *Provider*, con los atributos *id, company* **obligatorios**.

# Composite

## Motivación

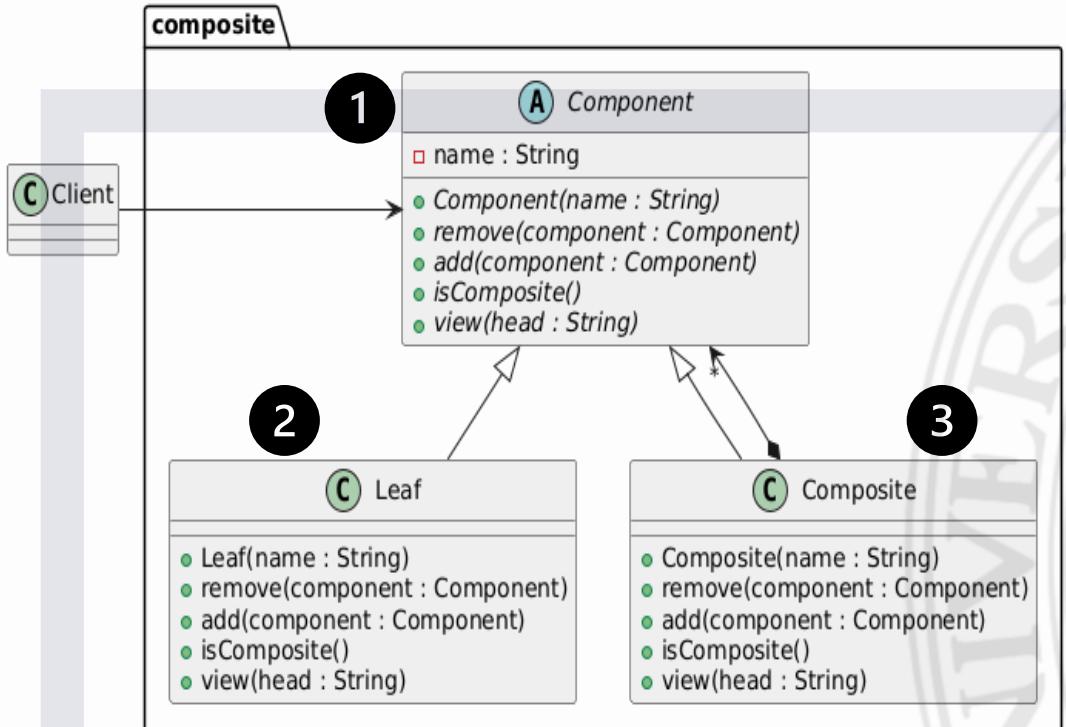


## ¿Cumple DOO?

- ① ¿Cohesión?
- ② ¿SOLID única responsabilidad?
- ③ Anti patrón:  
*Código espagueti*
- Añadir un nuevo tipo de hoja (*double*)!!!

# Composite

Propósito: Estructural. Ámbito: objeto



Permite estructuras en árbol tratando por igual a las hojas que a los elementos compuestos

## ① Rol Componente

- Clase padre abstracta, es la única visible al cliente.
- Los posibles métodos se definen aquí.

## ② Rol Hoja

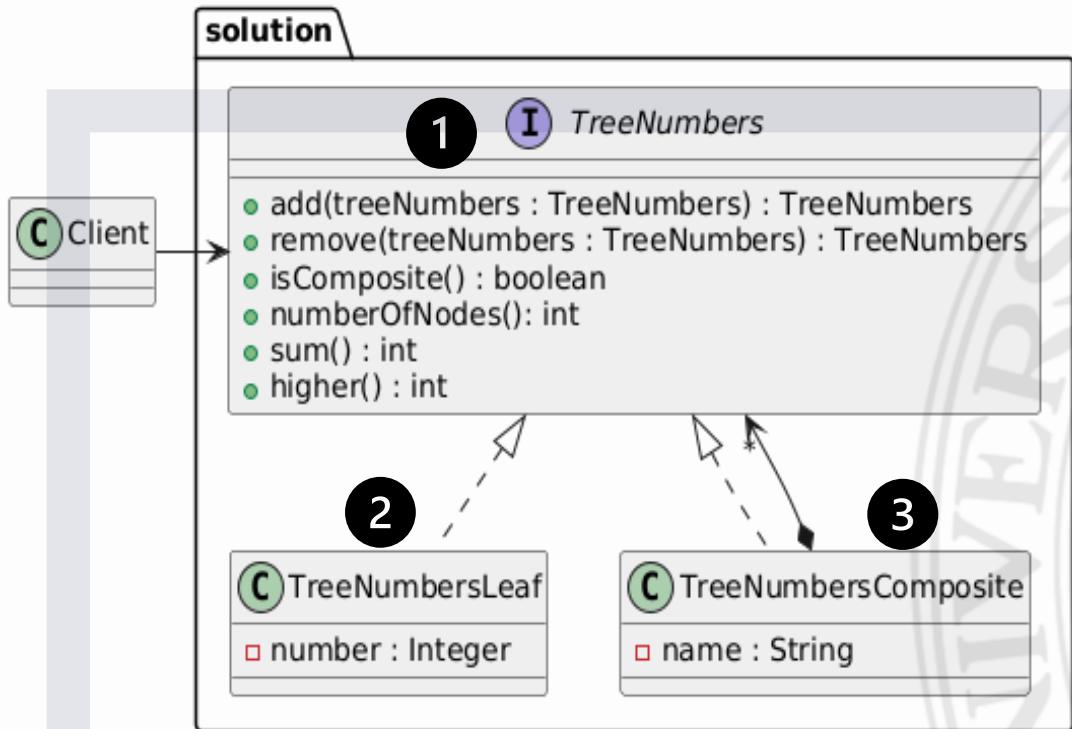
- Hereda de Componente.

## ③ Rol compuesto

- Hereda de *componente*.
- Contiene una lista de *Componente*.

# Composite

Propósito: Estructural. Ámbito: objeto



Permite estructuras en árbol tratando por igual a las hojas que a los elementos compuestos

## ① Rol Componente

- Clase padre abstracta, es la única visible al cliente.
- Los posibles métodos se definen aquí.

## ② Rol Hoja

- Hereda de Componente.

## ③ Rol compuesto

- Hereda de *componente*.
- Contiene una lista de *Componente*.

# Composite

 {→}

## Editor de Expresiones Matemáticas

- Se quiere construir un editor de expresiones matemáticas con valores enteros.
- Especificar el diagrama de clases del modelo que permita representar las expresiones.

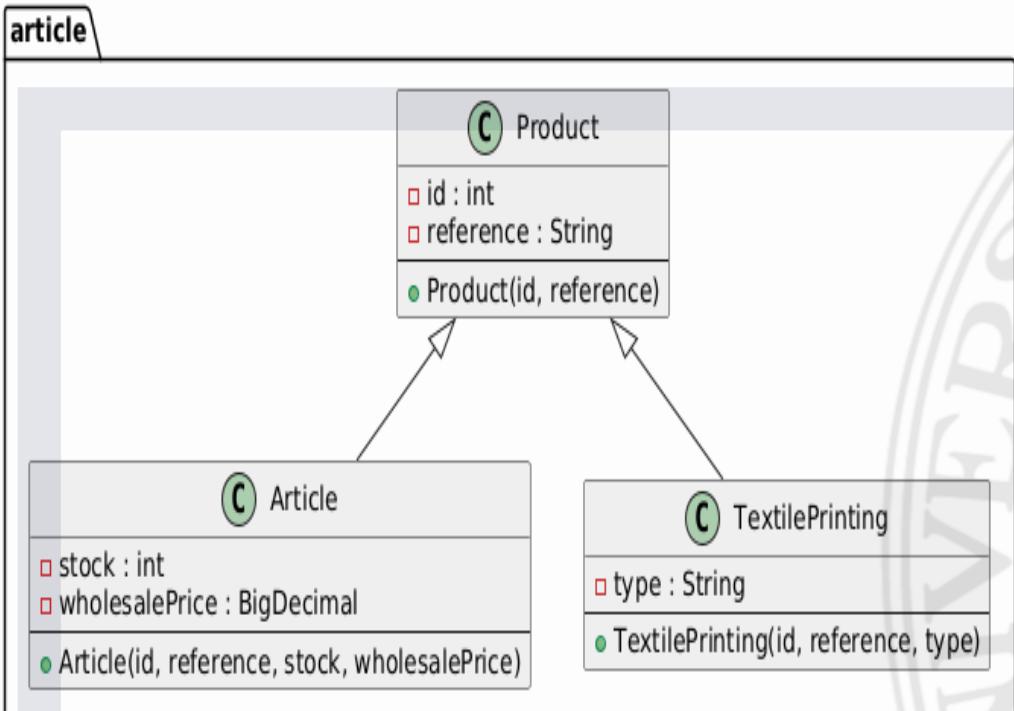
## Expresión

- Una expresión válida estará formada o bien por un número, o bien por la suma, resta, división o multiplicación de dos expresiones.

## Ejemplos de expresiones válidas:

- 5
- $(1+(8*3))$
- $((7+3) * (1+5))$

# Composite



Permite estructuras en árbol tratando por igual a las hojas que a los elementos compuestos

Se parte de un código en producción, y **no se quieren alterar las clases existentes.**

Aplicar el patrón compuesto para realizar agrupaciones en árbol de solo *artículos*.

La agrupación de artículos se identifican con un *nombre* y los artículos por su *reference*

# 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*)

## Arquitectura spring

- Basado en los principios de desarrollo limpio, desacoplado y escalable.

## Inversión de Control (IoC)

- Un objeto que necesita de otro objeto (por dependencia) **NO** lo busca **NI** lo crea, se lo dan.

## Inyección de Dependencias (DI)

- Una clase externa (*de Spring*), se encarga de crear el objeto e injectarlos a través del **constructor** o un método **set\***.

# 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 Authorization Server

- Proporciona Oauth 2.1 y OpenID Connect 1.0 y otras.

## Spring Cloud

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

Spring Cloud  
Data Flow

Spring for  
GraphQL

Spring AI

Spring REST  
Docs

Spring  
HATEOAS

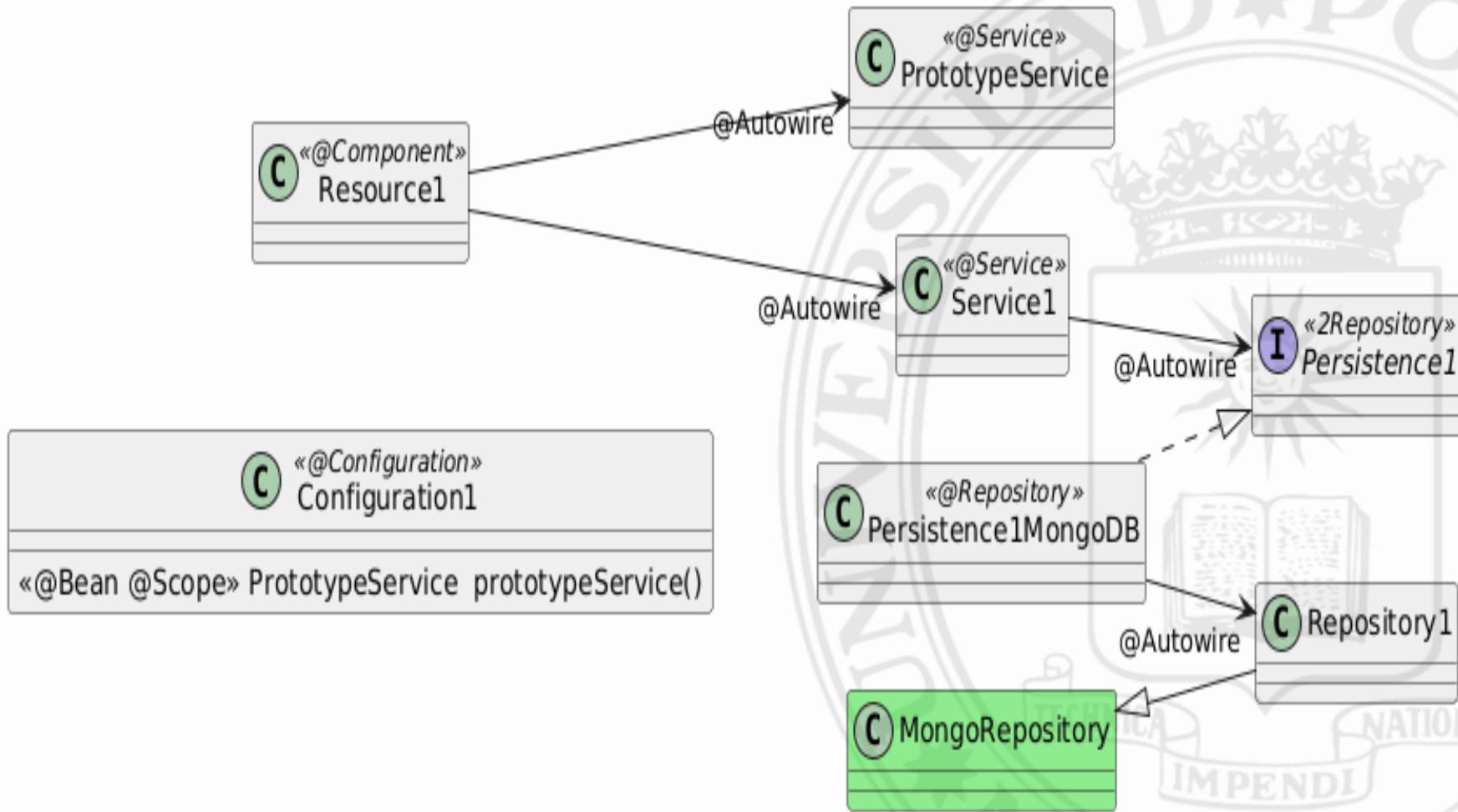
Spring Shell

...

# Spring

## Gestión de dependencias

### injection



# Spring

## Gestión de dependencias

```
@RestController
```

```
public class ArticleResource {  
    private final ArticleService articleService;  
    @Autowired  
    public ArticleResource(ArticleService articleService) {  
        this.articleService = articleService;  
    }
```

```
@Service
```

```
public class ArticleService {  
    private final ArticlePersistence articlePersistence;  
    @Autowired  
    public ArticleService(ArticlePersistence articlePersistence) {  
        this.articlePersistence = articlePersistence;  
    }
```

```
@Repository
```

```
public interface ArticlePersistence {  
}
```

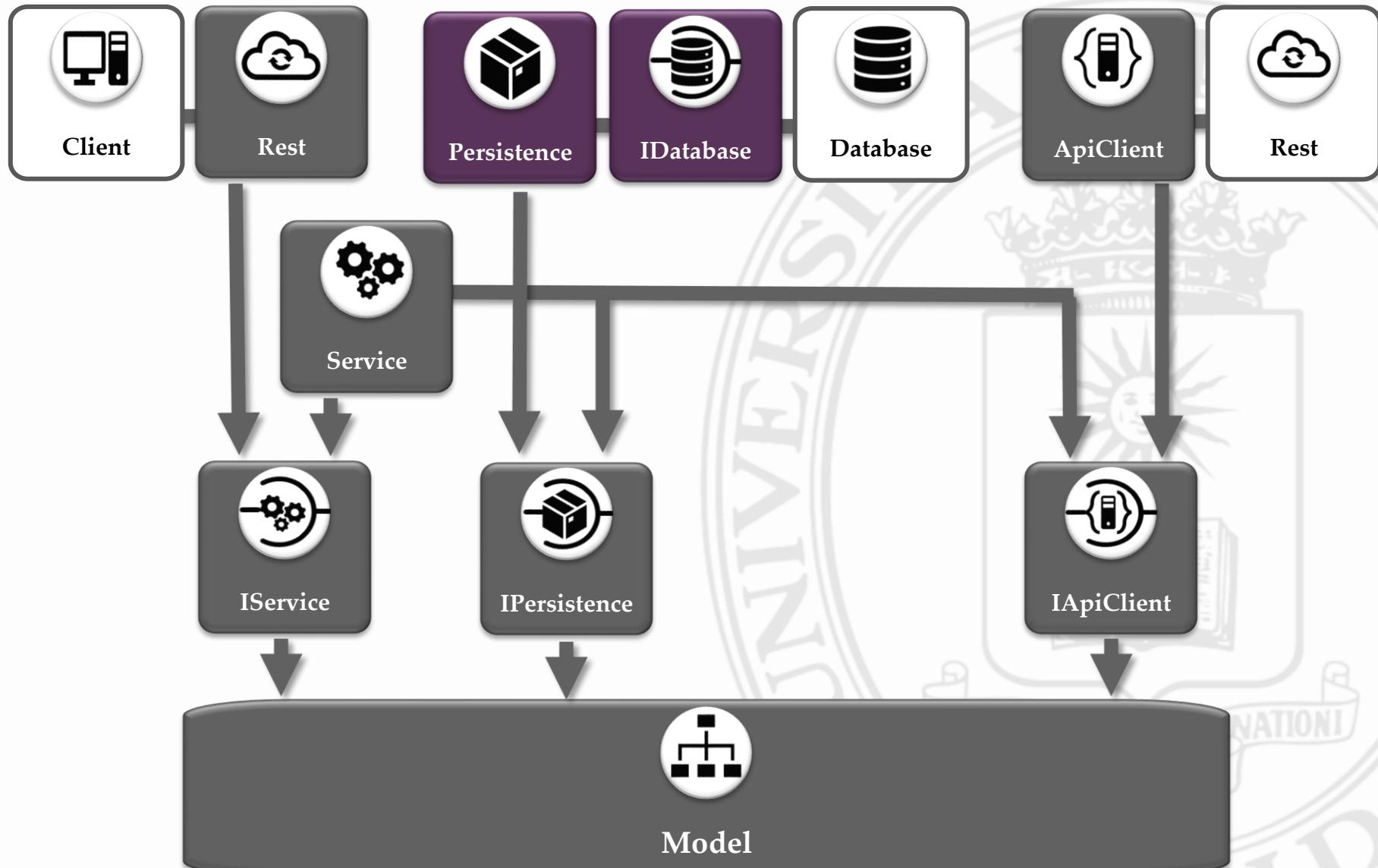
```
@Repository("articlePersistence")
```

```
public class ArticlePersistenceMongodb implements ArticlePersistence {  
    private final ArticleRepository articleRepository;  
    @Autowired  
    public ArticlePersistenceMongodb(ArticleRepository articleRepository) {  
        this.articleRepository = articleRepository;  
    }
```

```
public interface ArticleRepository extends MongoRepository<ArticleEntity, String> {  
    Optional<ArticleEntity> findByBarcode(String barcode);  
}
```

# Persistencia

## Adaptador de Bases de Datos



# Data Patrones

## Method Factory

- Define una abstracción para crear objetos, y son las subclases que deciden la clase concreta a instanciar.

## Abstract Factory

- Proporciona un interface para crear familias de objetos relacionadas.

## Inversion of Control

- Desacoplar mediante la inyección de dependencias.

## Data Access Object - DAO

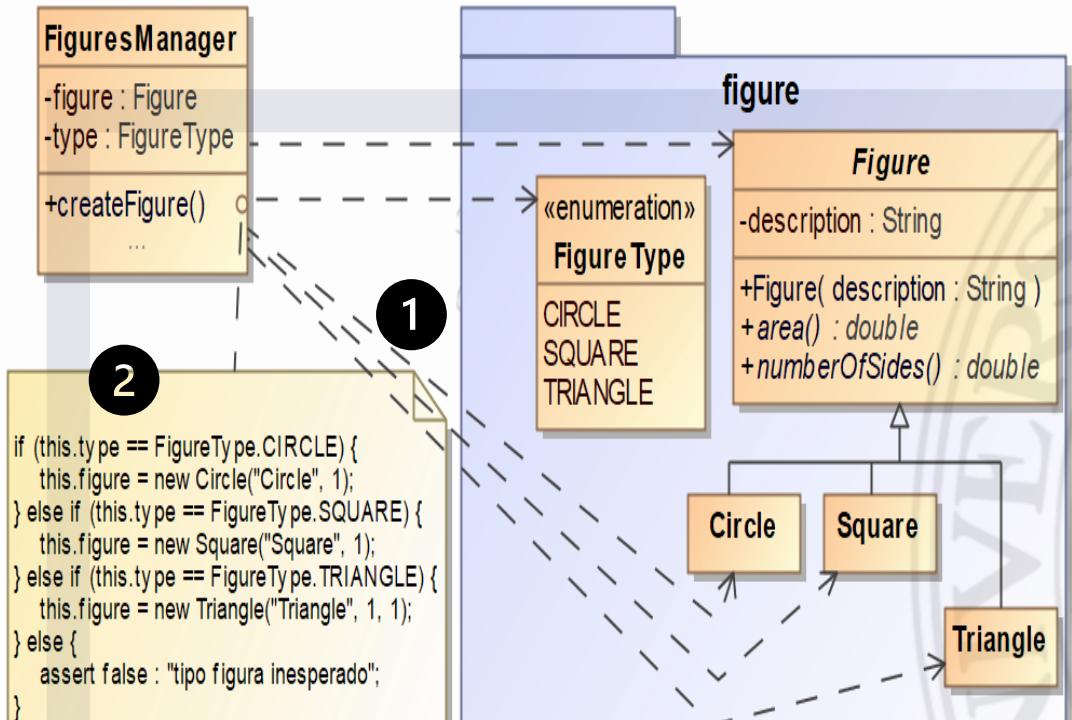
- Desacoplar el Dominio de la aplicación del Adaptador de las Bases de Datos.

## *Spring-Data*

- Framework que implementa el patrón DAO

# Factory Method

## Motivación



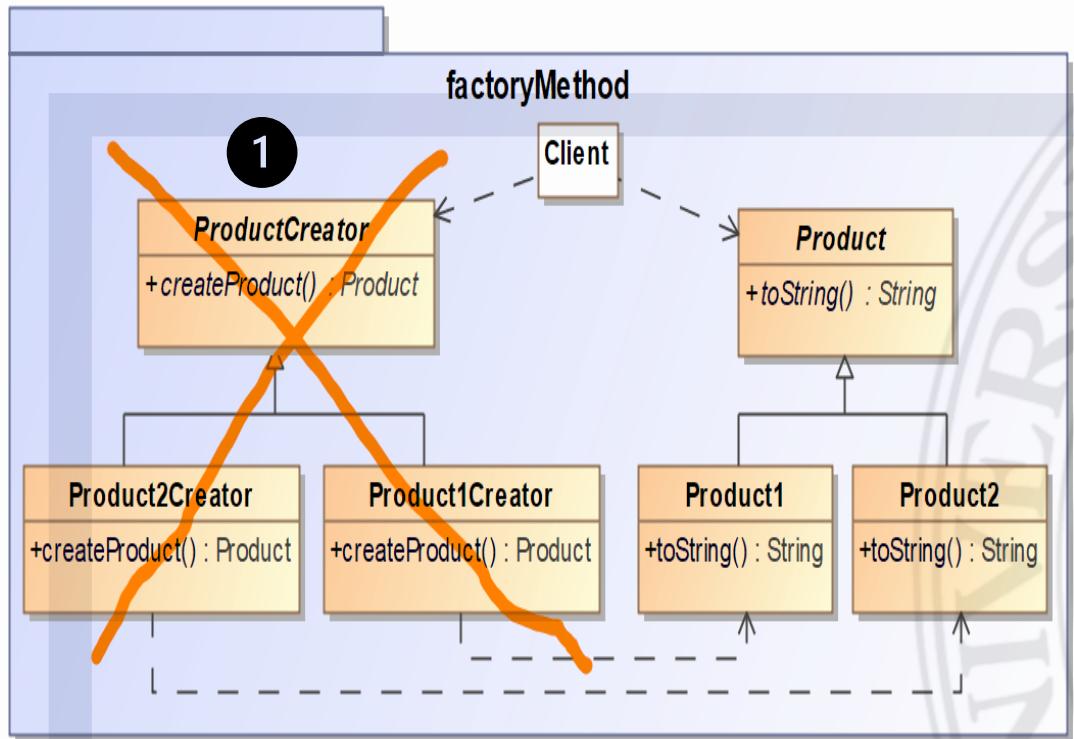
*Factory Method*

El cliente:  
*FiguresManager*,  
debe tener la  
responsabilidad  
de crear figuras

- **1** Dependencias inadecuadas.
- **2** Anti patrón:  
*código espagueti*.

# Factory Method

Propósito: Creación. Ámbito: clase



*Factory Method*

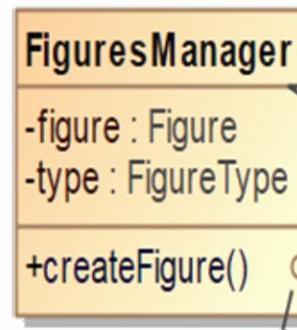
## Propósito

- Define una abstracción para crear objetos, y son las subclases que deciden la clase concreta a instanciar

### ① Función lambda

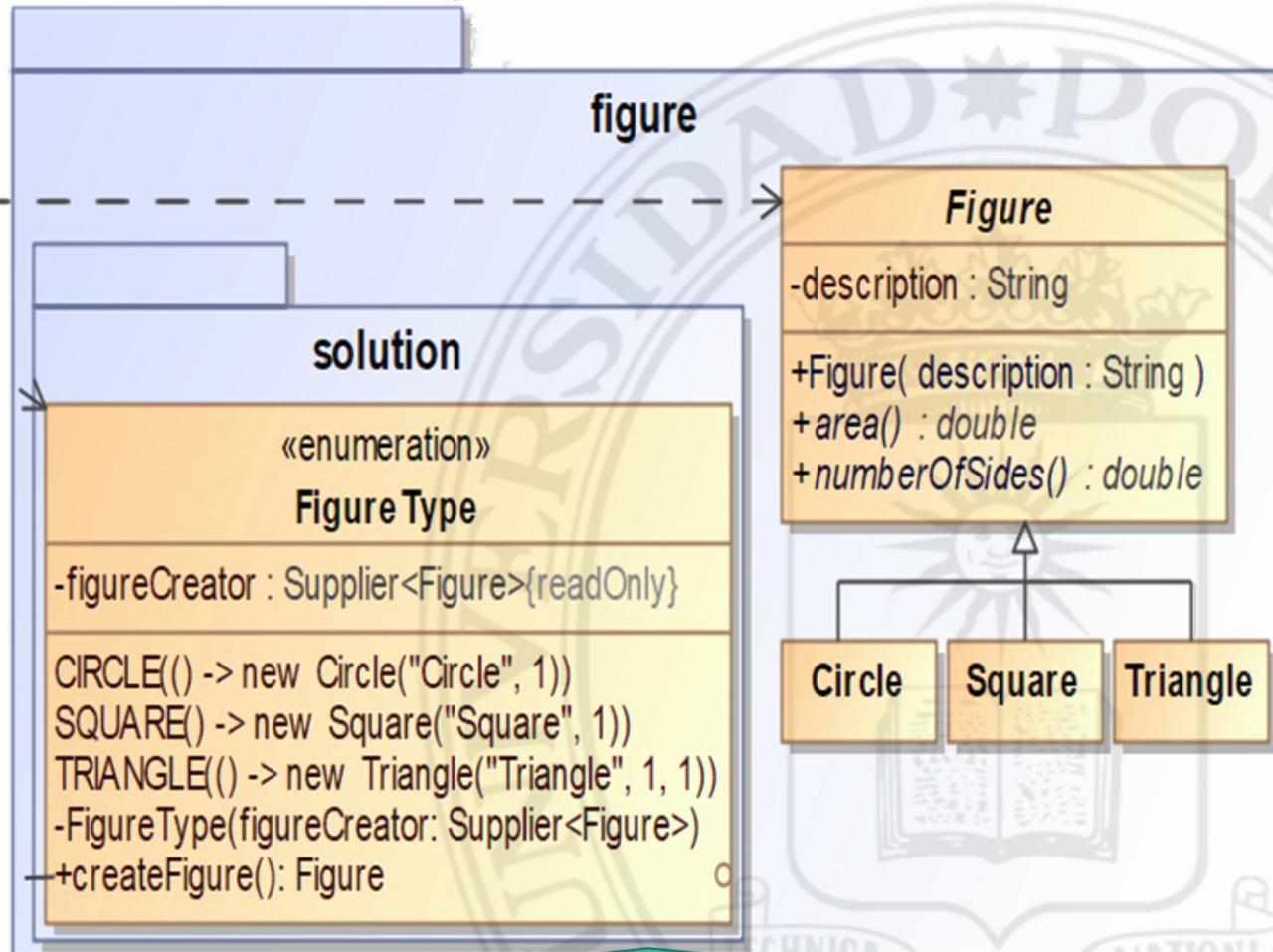
- `Supplier<T>`

# Method Factory



```
this.figure=type.createFigure();
```

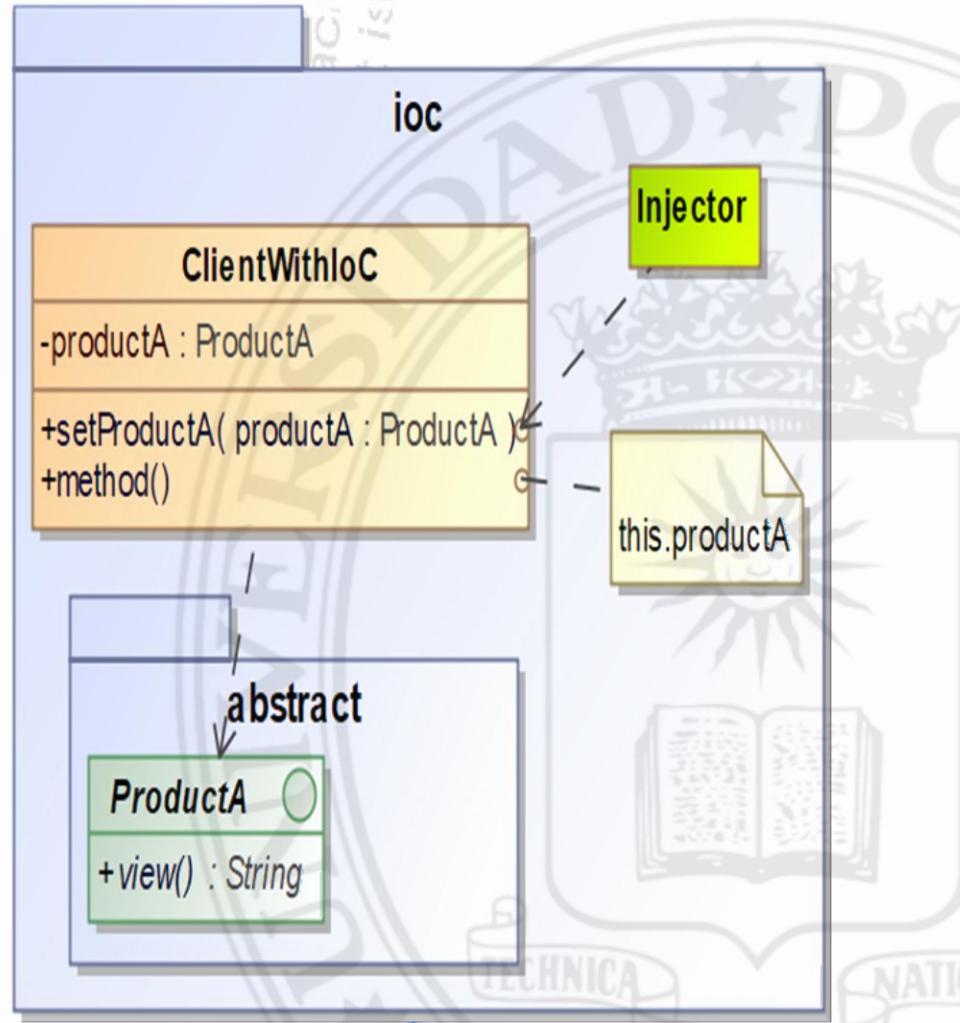
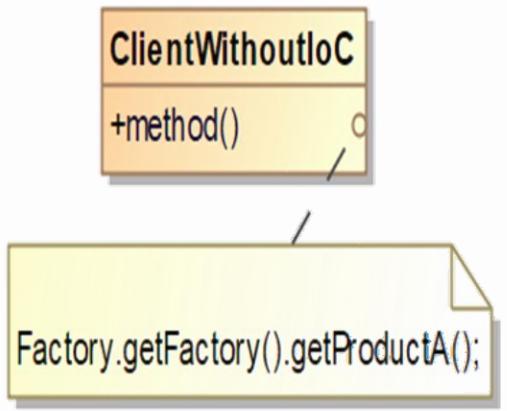
```
return this.figureCreator.get();
```



Solución mediante función Lambda

# Inversion Of Control

## Inversión de Control



Mínima dependencia, quitar dependencia con *Factory*

# Mapeo de objetos. POO

## Mapeo Objeto-Relacional - ORM

- Es una técnica para convertir objetos a una base de datos relacional.

## Mapeo Objeto-Dокументo - ODM

- Convierte objetos a una base de datos tipo JSON.

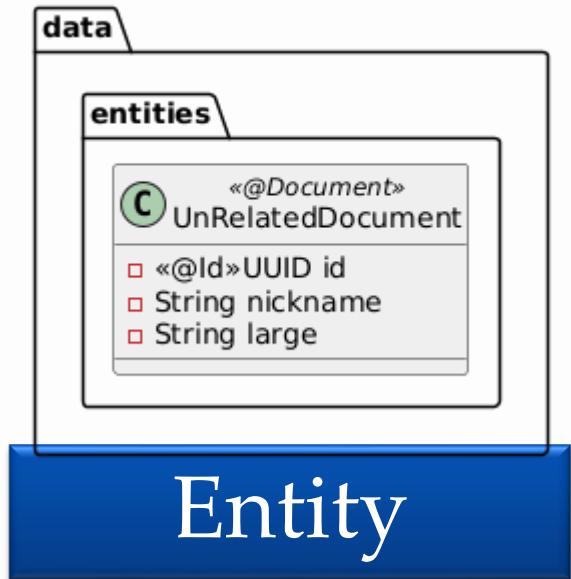
## Mapeo

- Diferentes opciones...
- Rendimiento de memoria.
- Rendimiento de ejecución.

## Dependencias

- No se debe depender del tipo de Bases de Datos

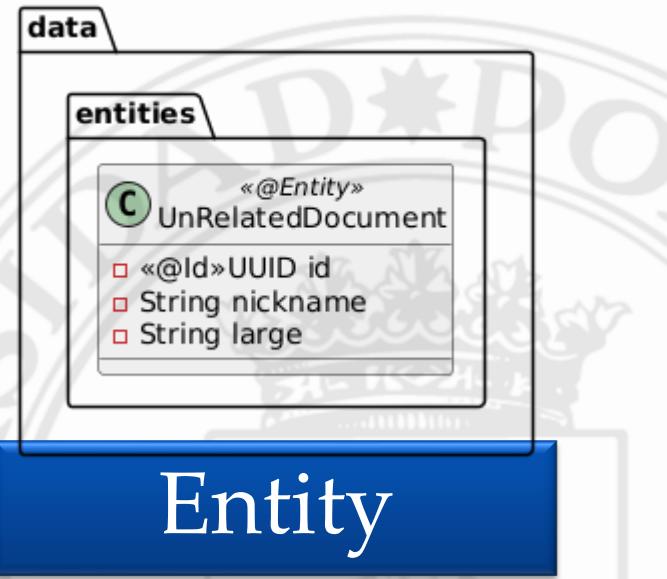
## Mapeo de objetos: sin relación



Entity

```
[{  
  "_id" : "ea5bbd06-9157-4fe8-adb3-2f533ced8ac2",  
  "nickname" : "nick1"  
}, {  
  "_id" : "07b33bd8-5da0-48d2-9c50-33b04f16b8b1",  
  "nickname" : "nick1",  
  "large" : "large"  
}]
```

MongoDB



Entity

un_related_document		
id	nickname	large
...	...	...
...	...	...
...	...	...

SQL

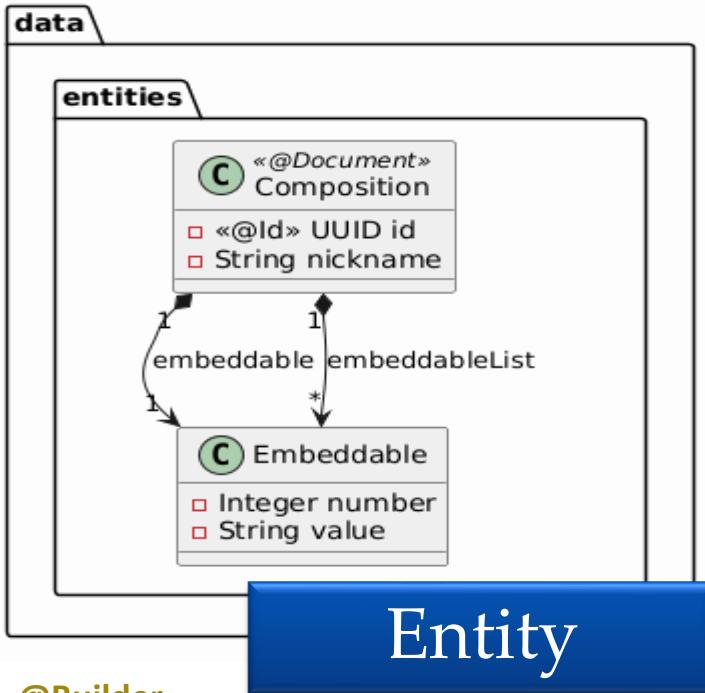
# Mapeo de objetos: sin relación

```
@Builder  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
@Document  
public class UnRelatedDocument {  
    public static final String TRANSIENT = "no persistent";  
    // id: El valor no nos importa.  
    // id: No hay razón para cambiarlo.  
    @Id  
    private UUID id;  
    @Indexed(unique = true)  
    private String nickname;  
    private Gender gender; // String  
    private LocalDateTime dateOfBirth;  
    private String[] tags;  
    private String large;  
    private Boolean active;  
    private Integer integer;  
    private Long longer;  
    private Double decimal;  
    @Transient  
    private String noPersistent;  
}
```

```
@Builder  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
@Entity  
public class UnRelatedEntity {  
    public static final String TRANSIENT = "no persistent";  
    @Id  
    private UUID id;  
    @Column(unique = true, nullable = false)  
    private String nick;  
    private Long longer;  
    @Enumerated(EnumType.STRING)  
    @Column(length = 20)  
    private Gender gender;  
    private LocalDateTime bornDate;  
    @ElementCollection(fetch = FetchType.EAGER)  
    @Singular  
    private List<String> tags;  
    @Transient  
    private String noPersistent;  
}
```

# Data

## Mapeo de objetos: composición



```

@Builder
@Data @NoArgsConstructor @AllArgsConstructor
@Document
public class CompositionDocument {
    @Id
    private UUID id;
    private String nickname;
    private EmbeddableDocument embeddableDocument;
    private List<EmbeddableDocument> embeddableDocumentList;
}
  
```

[

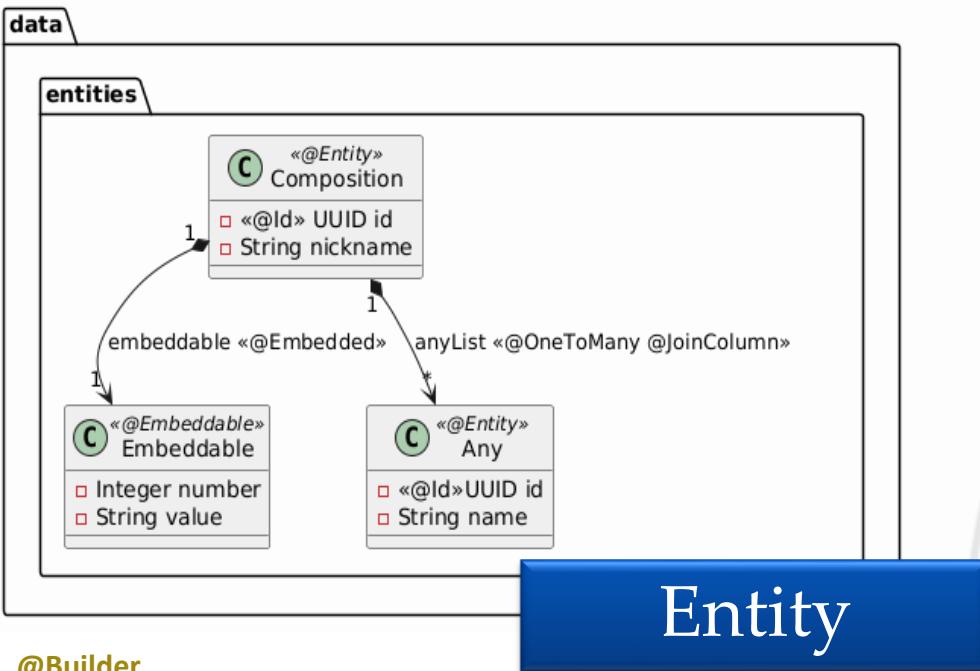
{

  "\_id" : "ed7d3f5c-ab11-43f7-9398-957828892b89",  
"nickname" : "nick",  
"embeddable" :{  
    "number" : 1,  
    "value" : "1"  
},  
"embeddableList" :[  
  { "number" : 2 , "value" : "2"},  
  { "number" : 3} ]  
}, {  
  "\_id" : "62c63b7f-34d5-44de-863d-8d5c223ec6bc",  
"nickname" : "nick"  
}

]

**MongoDB**

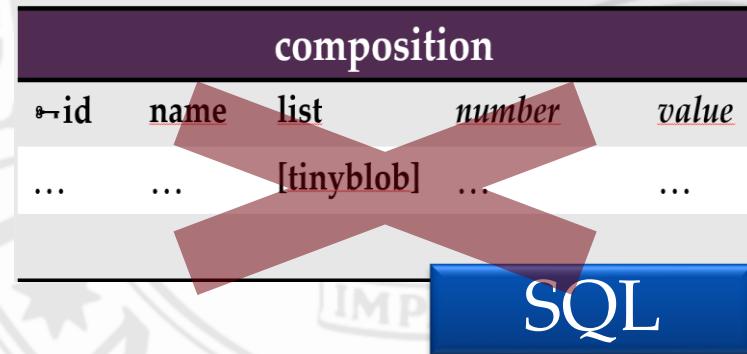
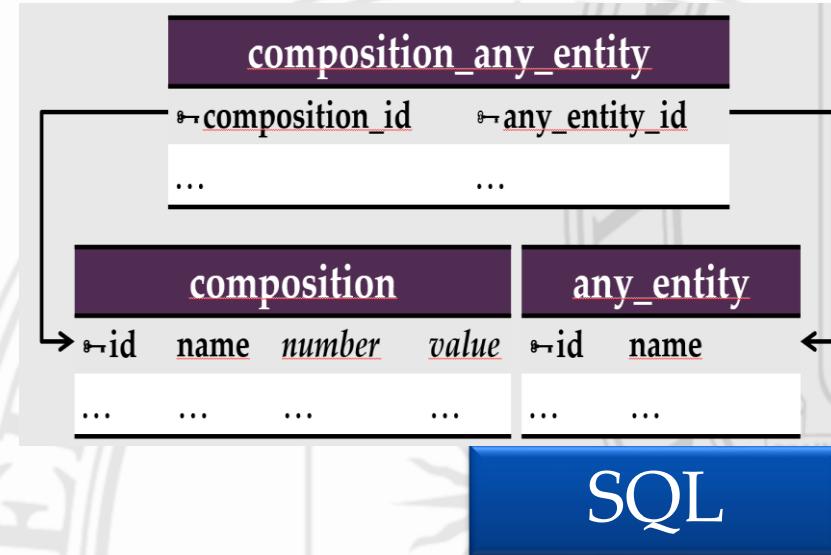
## Mapeo de objetos: composición



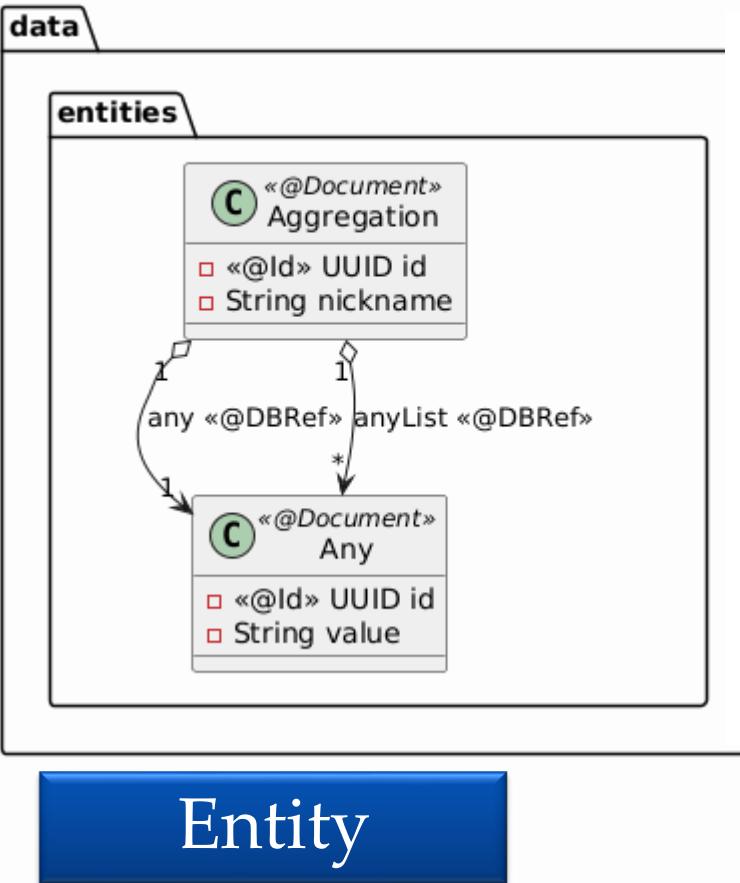
@Builder  
@Data @NoArgsConstructor @AllArgsConstructor  
@Entity

```

public class CompositionEntity {
    @Id @GeneratedValue
    private UUID id;
    private String nick;
    @Embedded
    private EmbeddableEntity embeddableEntity;
    @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER)
    @JoinColumn(name = "anyEntity_id", nullable = false)
    @Singular("anyEntity")
    private List<AnyEntity> anyEntityList;
}
    
```



# Mapeo de objetos: agregación



Entity

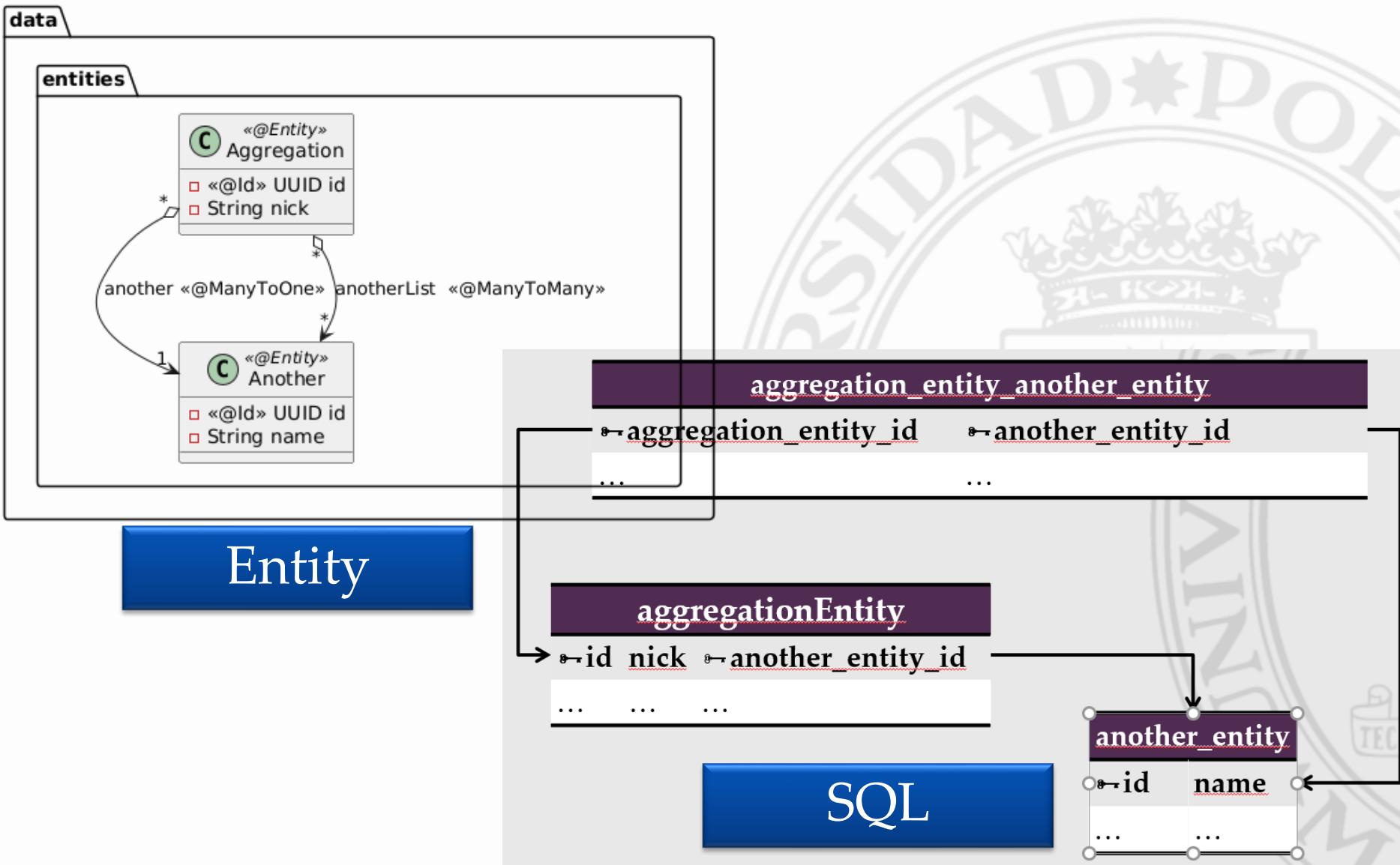
```
[
  {
    "id" : "089a8fdf-7905-46ac-8d39-eaea6eac578a",
    "nickname" : "nick",
    "anyDocument" : DBRef("any","68644e3f-83a2-42a0-b404-9cef5cf8eedb"),
    "anyDocumentList" : [
      DBRef("any","fa3e5a48-3948-44ed-8084-41ec99653575"),
      DBRef("any","769069c2-9929-4a82-a10d-ad29abc8b28e")
    ]
  },
  {
    "id" : "2b224c91-9e2b-4686-84e8-fda8cbe5aaff",
    "nickname" : "nick",
    "anyDocument" : DBRef("any","68644e3f-83a2-42a0-b404-9cef5cf8eedb"),
    "anyDocumentList" : [
      DBRef("any","fa3e5a48-3948-44ed-8084-41ec99653575"),
      DBRef("any","0654d0eb-88ad-4b8c-adcb-c8e79bfd2ecc")
    ]
  }
]
```

MongoDB (aggregation)

```
[
  {"id" : "68644e3f-83a2-42a0-b404-9cef5cf8eedb", "value" : "any"},
  {"id" : "fa3e5a48-3948-44ed-8084-41ec99653575", "value" : "any1"},
  {"id" : "769069c2-9929-4a82-a10d-ad29abc8b28e", "value" : "any2"},
  {"id" : "0654d0eb-88ad-4b8c-adcb-c8e79bfd2ecc", "value" : "any3"}
]
```

MongoDB (any)

## Mapeo de objetos: agregación



# Mapeo de objetos: agregación

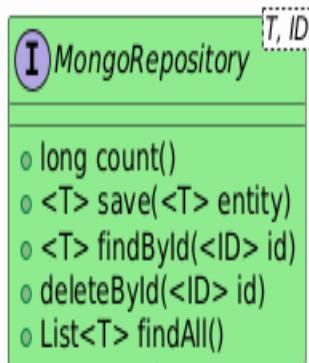
```
@Builder  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
@Entity  
public class AggregationEntity {  
    @Id  
    @GeneratedValue  
    private UUID id;  
    private String nick;  
    @ManyToOne // @JoinColumn  
    private AnotherEntity anotherEntity;  
    @ManyToMany(fetch = FetchType.EAGER) // @JoinColumn  
    private List<AnotherEntity> anotherEntityList;  
}
```

```
@Builder  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
@Entity  
public class AnotherEntity {  
    @Id  
    @GeneratedValue  
    private UUID id;  
    private String name;  
}
```

# Spring Data

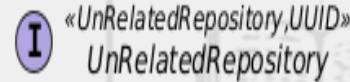
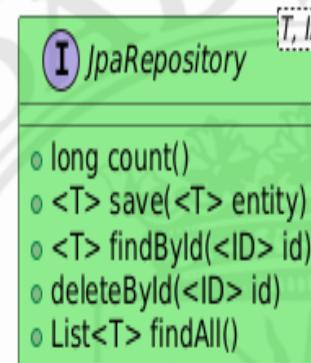
## Mongodb. Sin relación

### repositories



- List<UnRelatedRepository> findByName(String name)
- List<UnRelatedRepository> findByAgeGreaterThanOrEqual(int value)
- List<UnRelatedRepository> findByBirthdateBefore(LocalDate date)
- List<UnRelatedRepository> findByNamesIn(Collection<String> names)

### daos



- List<UnRelatedRepository> findByName(String name)
- List<UnRelatedRepository> findByAgeGreaterThanOrEqual(int value)
- List<UnRelatedRepository> findByBirthdateBefore(LocalDate date)
- List<UnRelatedRepository> findByNamesIn(Collection<String> names)

# 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 Mongodb

## Búsquedas por nombre de método VS Json Query

- **findByBirthdateAfter(Date date)** {"birthdate" : {"\$gt" : date}}
- **findByAgeGreaterThan(int age)** {"age" : {"\$gt" : age}}
- **findByAgeGreaterThanOrEqual(int age)** {"age" : {"\$gte" : age}}
- **findByBirthdateBefore(Date date)** {"birthdate" : {"\$lt" : date}}
- **findByAgeLessThan(int age)** {"age" : {"\$lt" : age}}
- **findByAgeLessThanOrEqual(int age)** {"age" : {"\$lte" : age}}
- **findByAgeBetween(int from, int to)** {"age" : {"\$gt" : from, "\$lt" : to}}
- **findByAgeIn(Collection ages)** {"age" : {"\$in" : [ages...]}}
- **findByAgeNotIn(Collection ages)** {"age" : {"\$nin" : [ages...]}}
- **findByFirstnameNotNull()** {"firstname" : {"\$ne" : null}}
- **findByFirstnameNull()** {"firstname" : null}
- **findByFirstnameLike(String name)** {"firstname" : name} (name as regex)
- **findByFirstnameNotLike(String name)** {"firstname" : { "\$not" : name }} (name as regex)
- **findByFirstnameContaining(String name)** {"firstname" : name} (name as regex)
- **findByFirstnameNotContaining(String name)** {"firstname" : { "\$not" : name}} (name as regex)
- **findByAddressesContaining(Address address)** {"addresses" : { "\$in" : address}}
- **findByAddressesNotContaining(Address address)** {"addresses" : { "\$not" : { "\$in" : address}}}
- **findByFirstnameRegex(String firstname)** {"firstname" : {"\$regex" : firstname }}
- **findByFirstname(String name)** {"firstname" : name}
- **findByFirstnameNot(String name)** {"firstname" : {"\$ne" : name}}
- **findByActiveIsTrue()** {"active" : true}
- **findByActiveIsFalse()** {"active" : false}
- **findByLocationExists(boolean exists)** {"location" : {"\$exists" : exists }}

# Spring Data Mongodb

## Búsquedas por nombre de método VS Json Query

- `@Query("{nick:{$in:?0}}") // Query NOT necessary`  
`ListUnRelatedDocument findByNickInCollection nicks`
- `@Query"?#{[0] == null ? { $where : 'true'} : { nick : {$regex:[0], $options: 'i'}} } {}"`  
`ListUnRelatedDocument findByNickLikeIgnoreCaseNullSafe(String nick)`
- `@Query"${$and:[}" // allow NULL: all elements`  
`"?#{[0] == null ? {_id : {$ne:null}} : { nick : {$regex:[0], $options: 'i'}} } ,"`  
`"?#{[1] == null ? {_id : {$ne:null}} : { large : {$regex:[1], $options: 'i'}} } "`  
`"] }"`  
`ListUnRelatedDocument findByNickLikeAndLargeLikeNullSafe(String nick,`  
`String large)`

## Search

Mobile

666000666

Username

jes

Dni

Address

 Only Customer

# Spring Data Mongodb

## Búsquedas por nombre de método VS Json Query

- `@Query("{nick:{$in:?0}}") // Query NOT necessary`  
`ListUnRelatedDocument findByNickInCollection nicks`
- `@Query"?#{[0] == null ? { $where : 'true'} : { nick : {$regex:[0], $options: 'i'}} } {}"`  
`ListUnRelatedDocument findByNickLikeIgnoreCaseNullSafe(String nick)`
- `@Query"${$and:[}" // allow NULL: all elements`  
`"?#{[0] == null ? {_id : {$ne:null}} : { nick : {$regex:[0], $options: 'i'}} } ,"`  
`"?#{[1] == null ? {_id : {$ne:null}} : { large : {$regex:[1], $options: 'i'}} } "`  
`"] }"`  
`ListUnRelatedDocument findByNickLikeAndLargeLikeNullSafe(String nick,`  
`String large)`

## Search

Mobile

666000666

Username

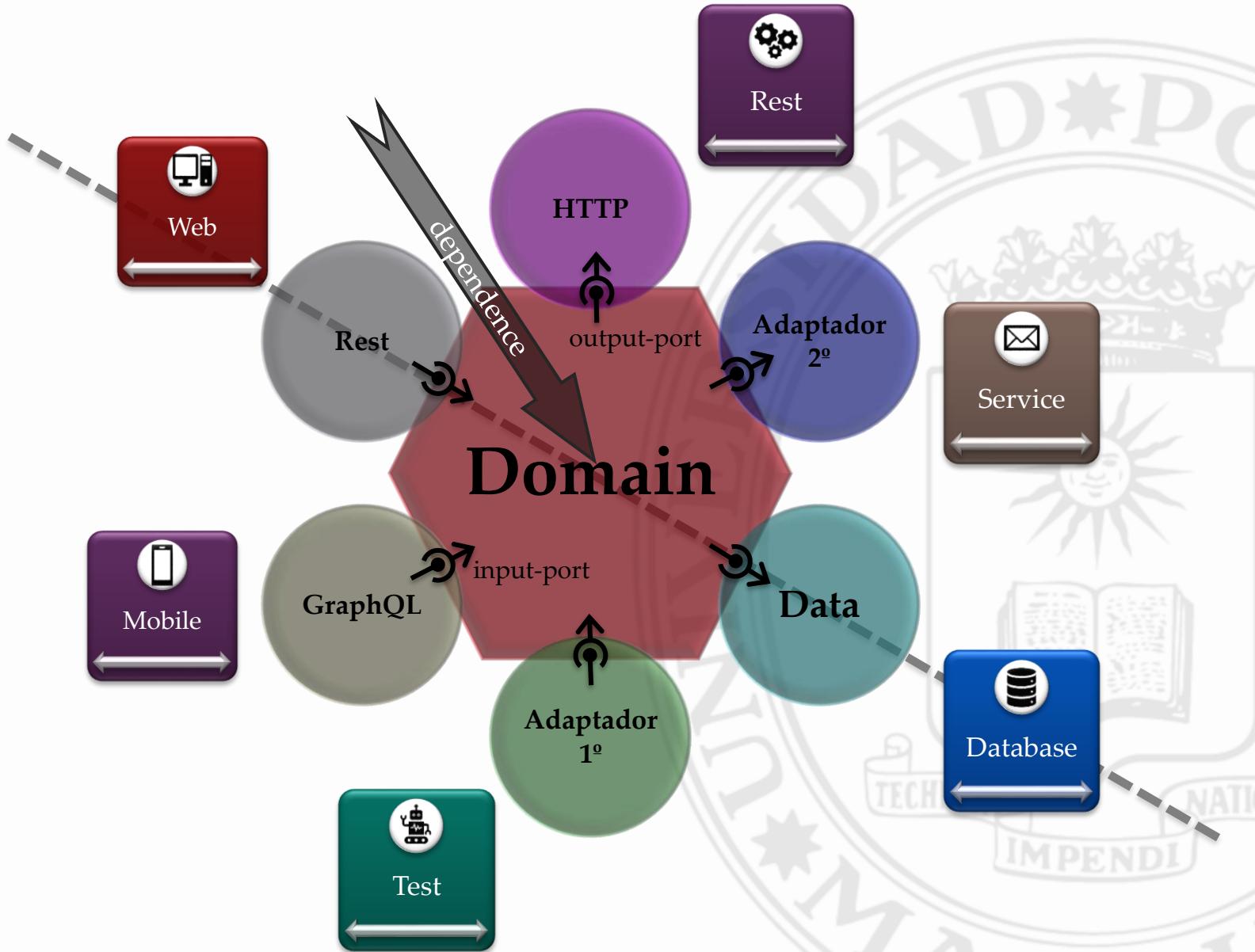
jes

Dni

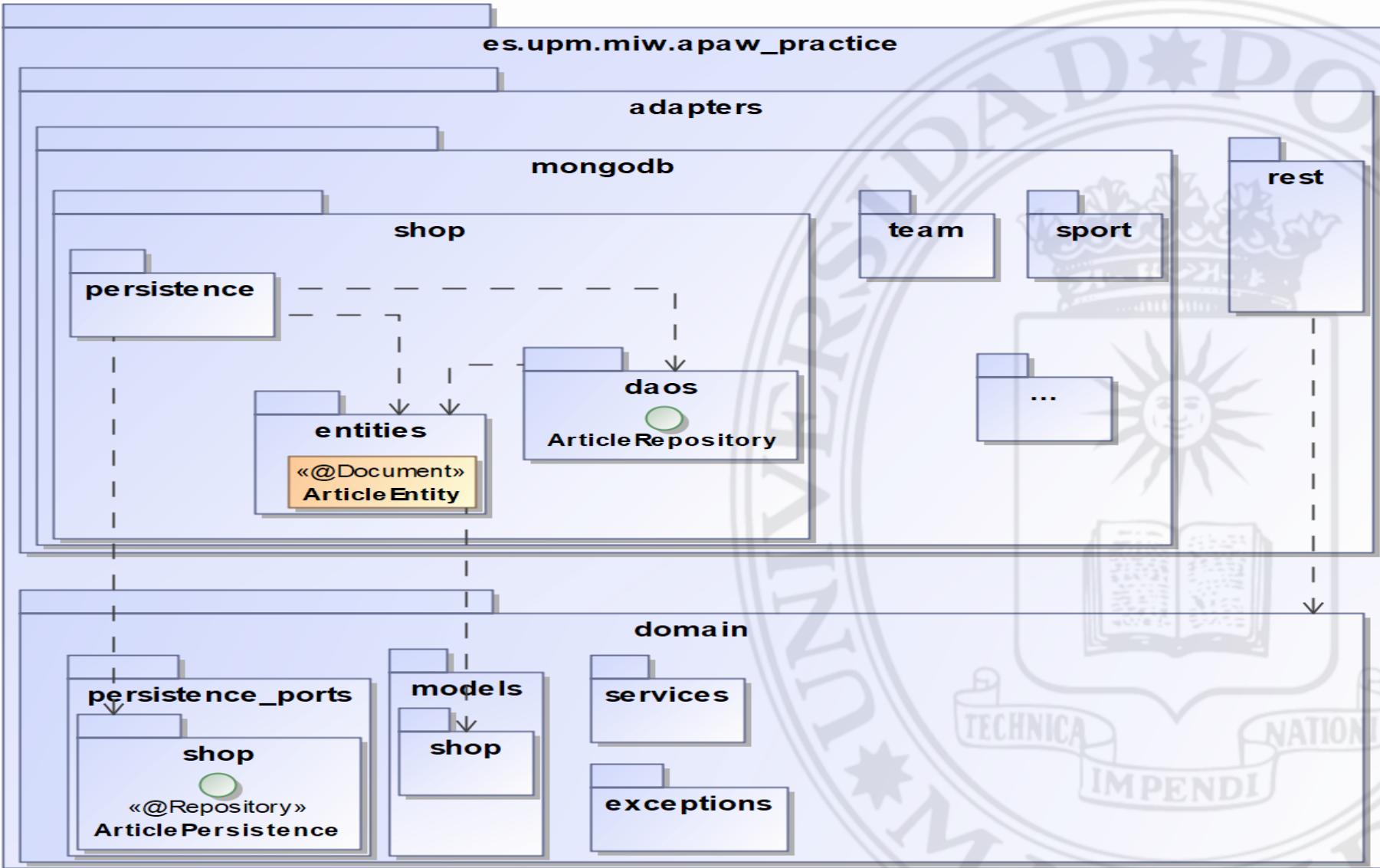
Address

 Only Customer

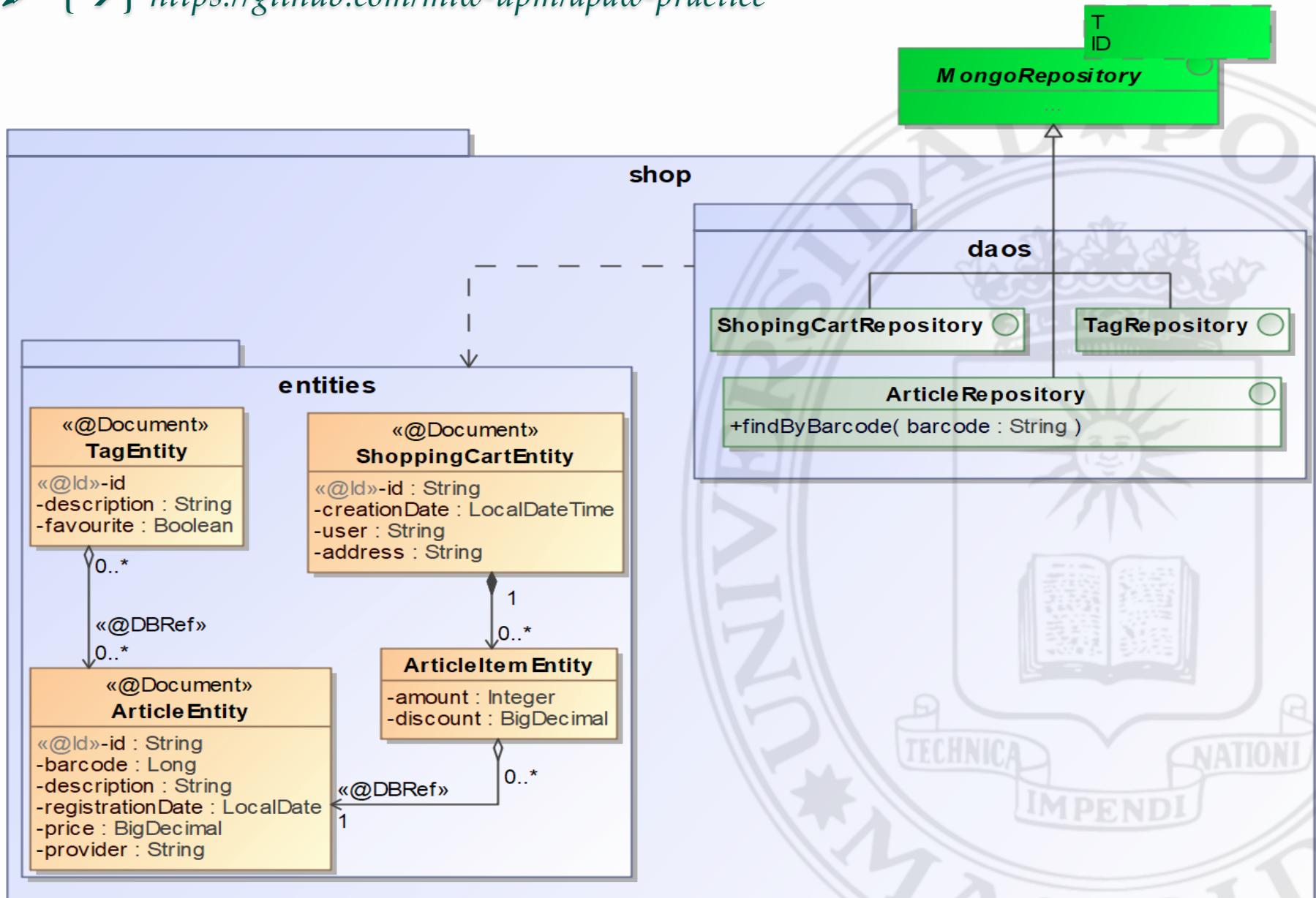
# Hexagonal Architecture Shop



📝 {→} <https://github.com/miw-upm/apaw-practice>

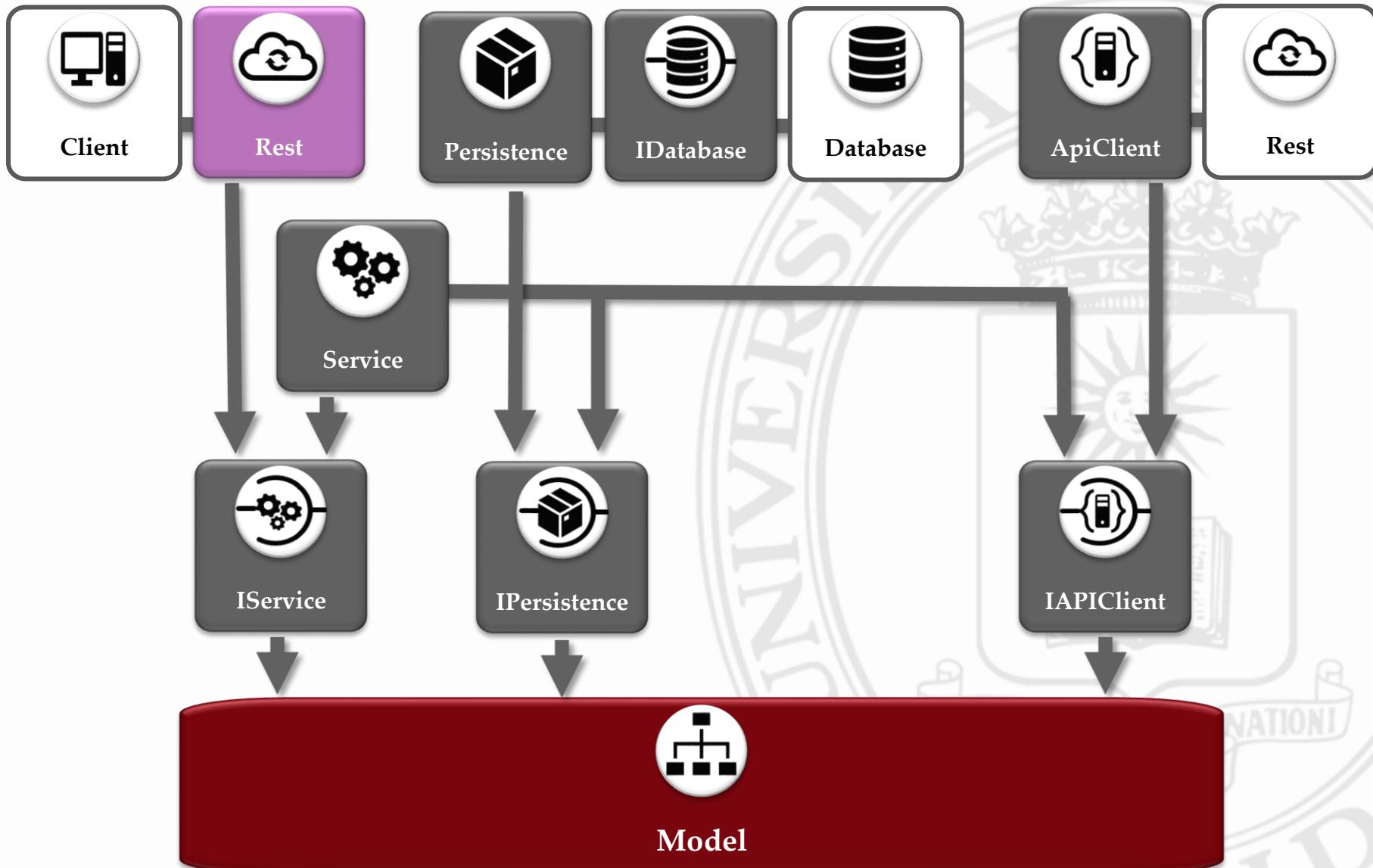


📝 {→} <https://github.com/miw-upm/apaw-practice>



# Rest

## Adaptador Rest



# 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: *UUID*

### 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 (*headers: GET*)

NO referenciar formatos /v0/orders/pdf → /v0/orders (*MIME:pdf*)

### Búsquedas:

GET /orders/opened

### GET

GET /orders?param=value&param2=value2

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

### Respuestas

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

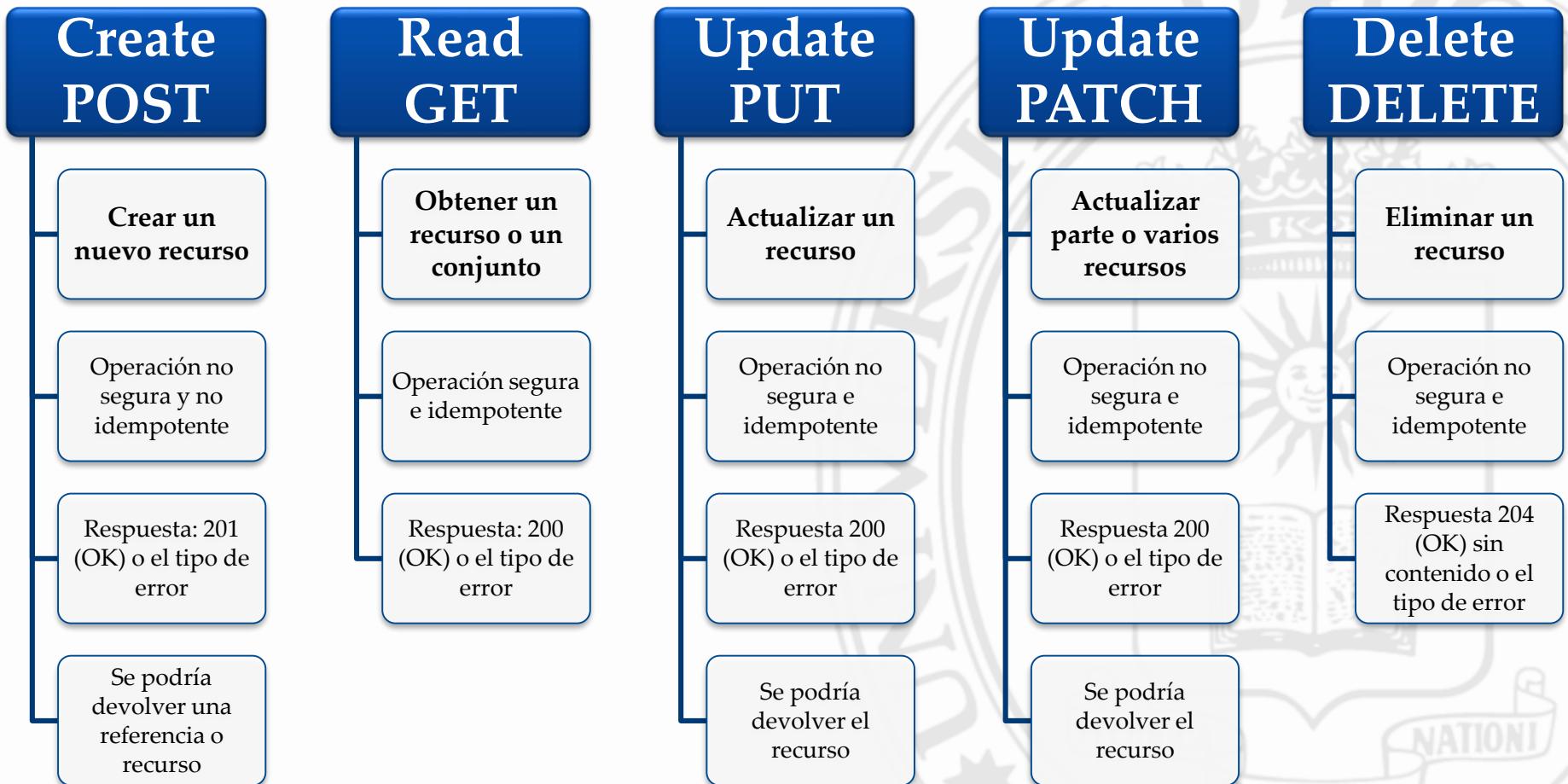
Paginación: ...?page=1&size=30

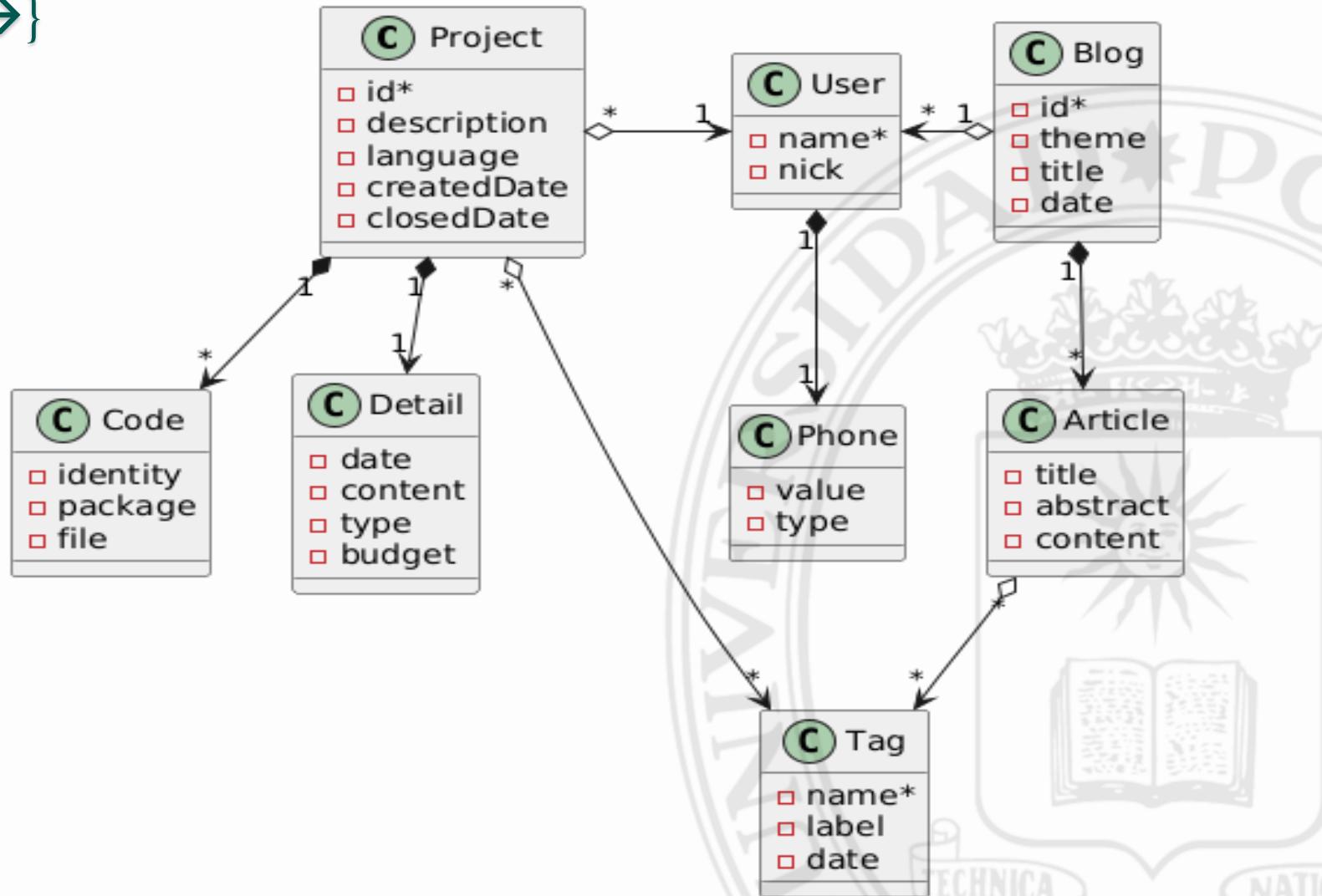
Filtros: ...?type=3,5

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

# REST

## Buenas prácticas





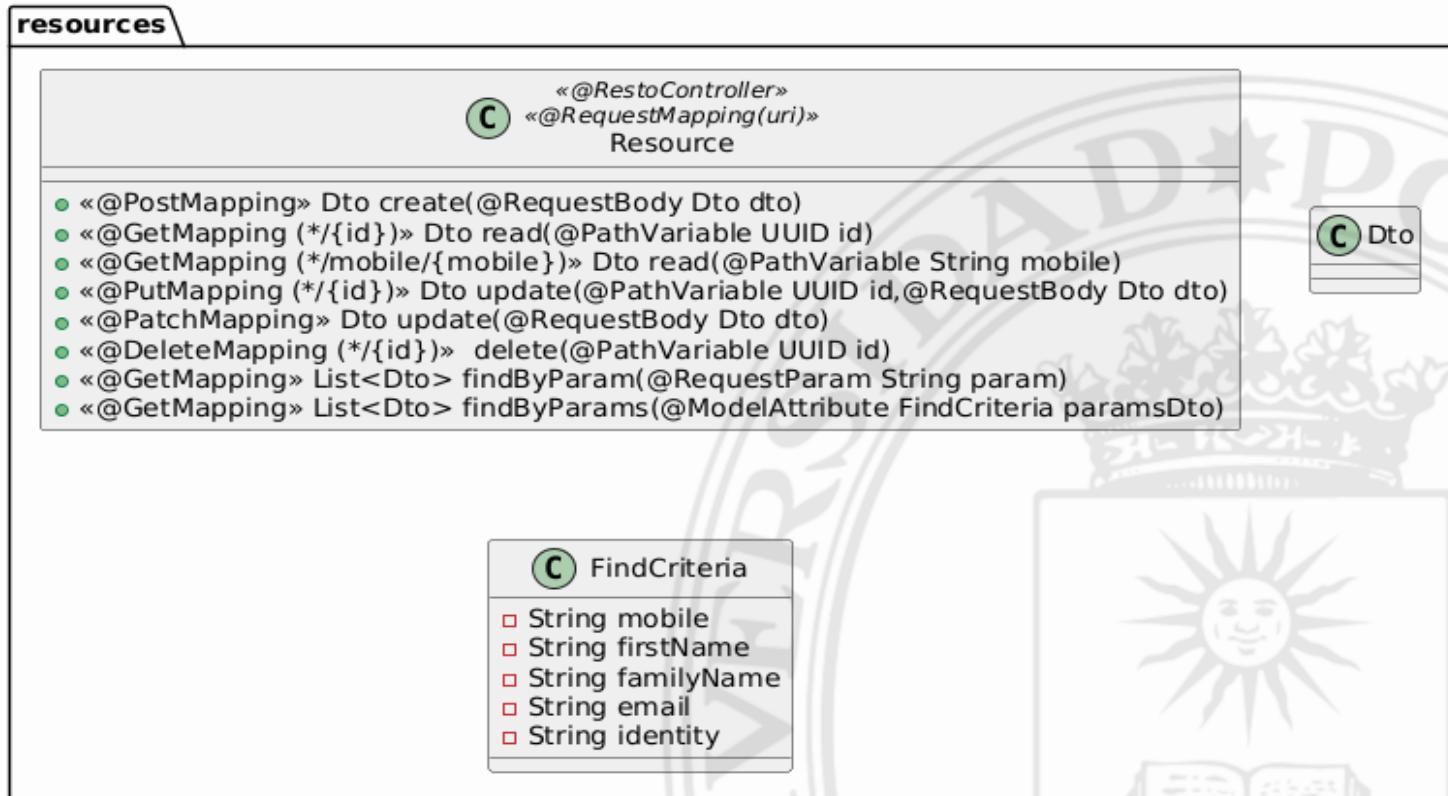
Crear un API Rest

## Server URI

- **http://server.com/api/v0/\*** (*api compartido con el frontend*)
- **http://api.server.com/v0/\*\*** (*api en servidor independiente*)

## End-points

- Obtener todos los usuarios:
- Obtener todos los artículos en formato pdf:
- Obtener todos los proyectos de un usuario, “id:a39bec10”:
- Obtener todos los proyectos de un usuario, “name:Jesús”:
- Obtener todos los ficheros de un proyecto:
- Borrar una etiqueta (tag):
- Borrar un artículo de un usuario:
- Se quieren cerrar todos los proyectos de un usuario:
- Se quiere cambiar el resumen de un artículo concreto:



**POST /resource {dto}**

**GET /resource/666**

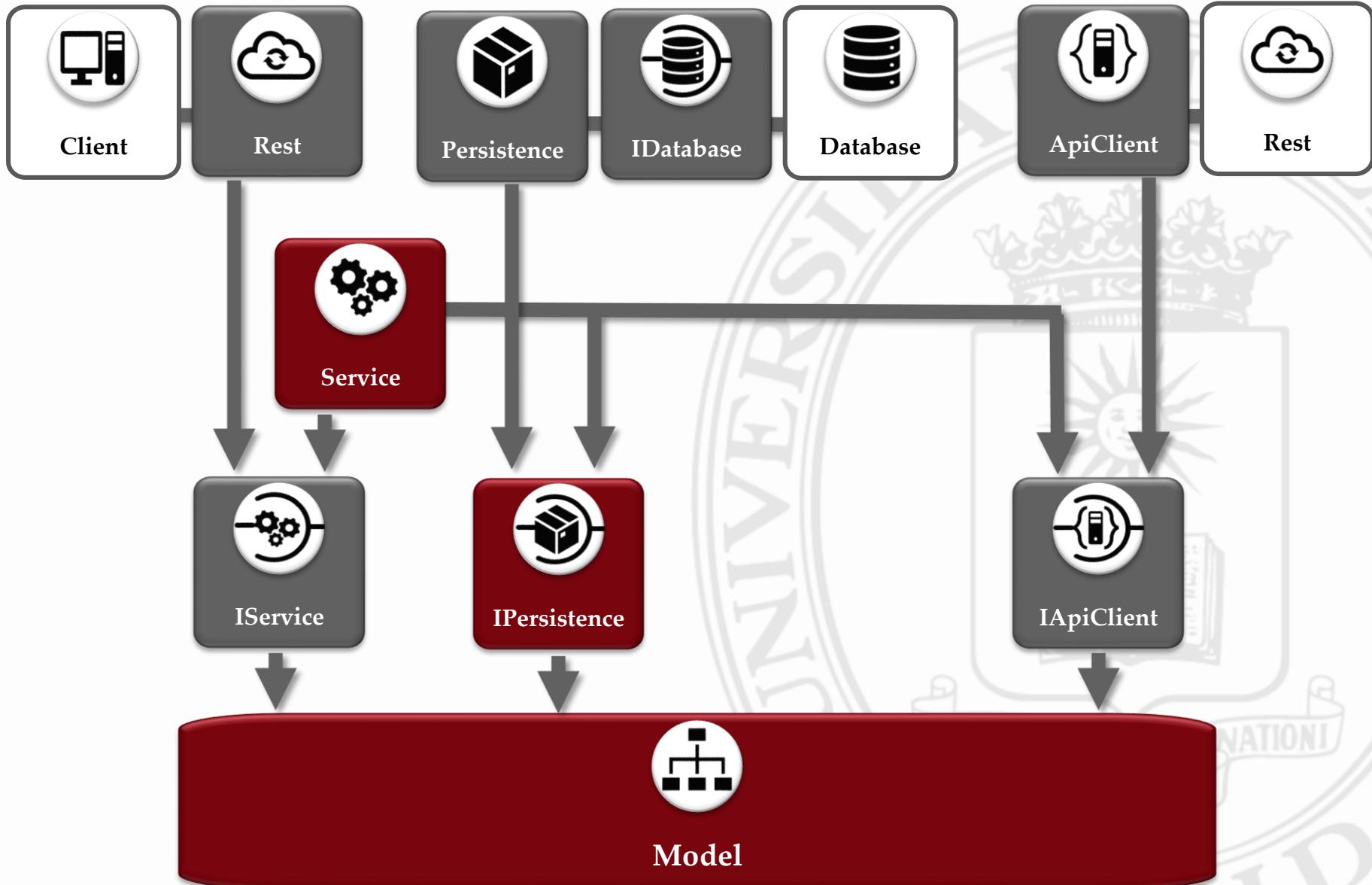
**PUT /resource/666 {dto}**

**PATCH /resource {dto}**

**DELETE /resource/666**

**GET /resource?p1=uno&p2=two&p3=three**

# Domain



# Programación Funcional

## Programación Funcional & flujos

- Programación declarativa: ¿Qué?
- Basado en Funciones: funciones Lambda.
- Funciones de orden superior. Manejan funciones como parámetros de entrada y salida.
- Flujos de datos.
- Sin estado, sin orden y sin efectos colaterales.
- Valores inmutables: paso de parámetros por valor.

# Domain Patrones

## Facade

- Proporciona un interface unificado para un conjunto de interfaces de un subsistema.

## Strategy

- Define un conjunto de algoritmo haciéndolos intercambiables dinámicamente.

## Observer: OOP

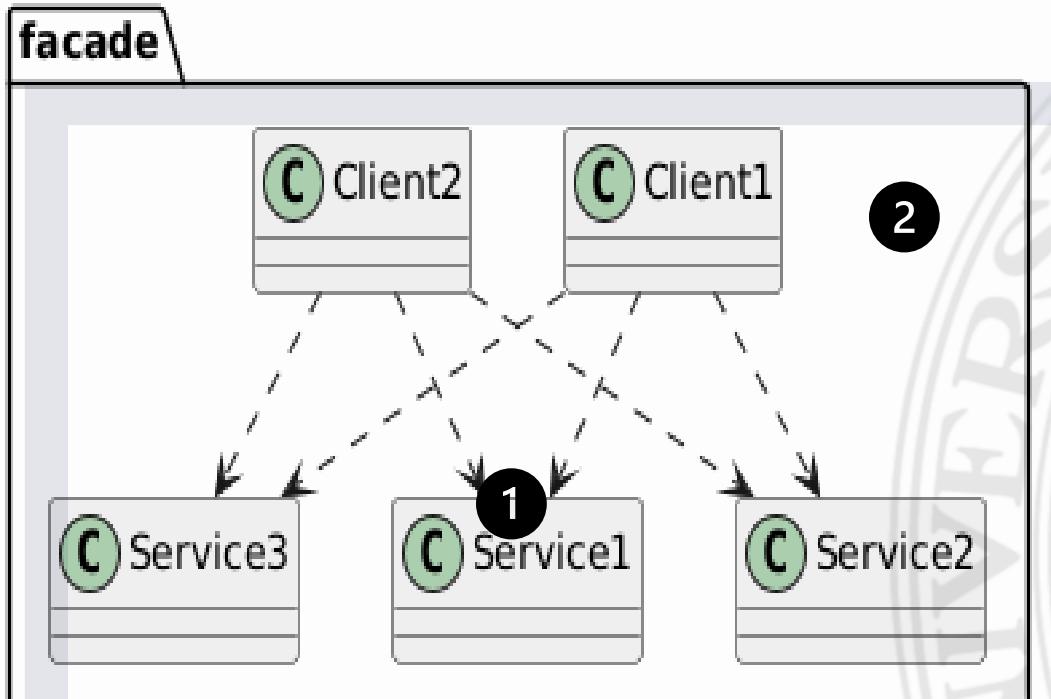
- Se define una dependencia entre uno a muchos, del tal manera, que cuando cambie el sujeto avise a todos los objetos dependientes.

## Publisher: FP

- Se permite que un publicador envíe mensajes de forma asincrónica a varios subscriptores interesados, sin crear dependencias.

# Facade

Propósito: Estructural. Ámbito: objeto



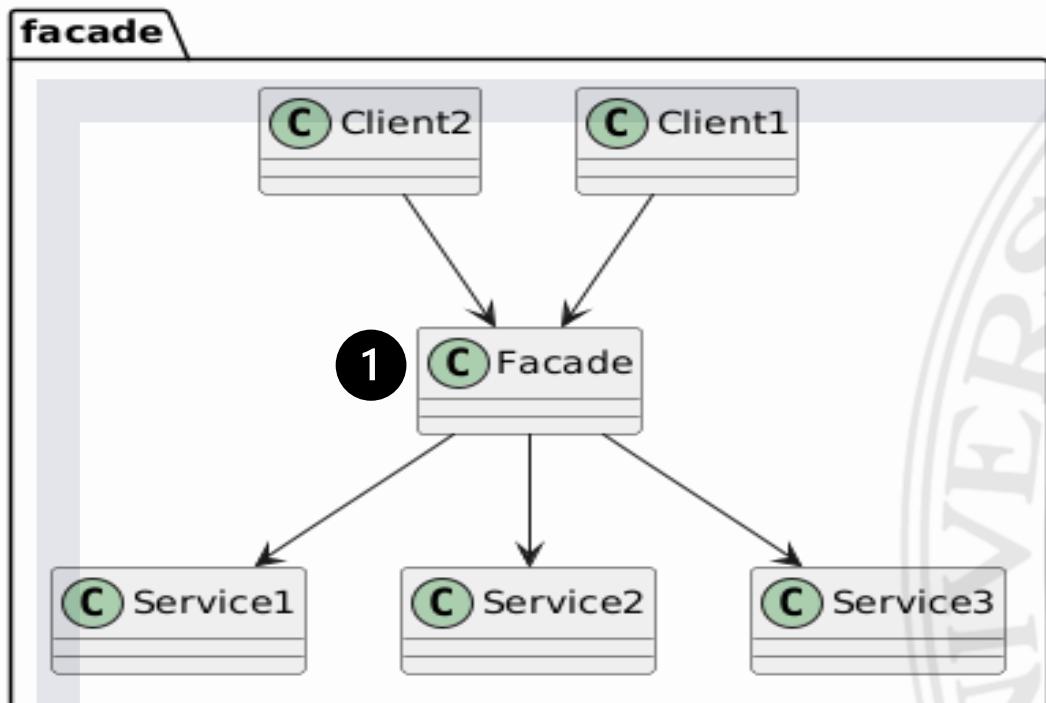
Facade

**1** Subsistema complejo, difícil de utilizar. Librerías de terceros.

**2** En el desarrollo del Cliente2 no se puede reutilizar el código del Cliente1.

# Facade

Propósito: Estructural. Ámbito: objeto



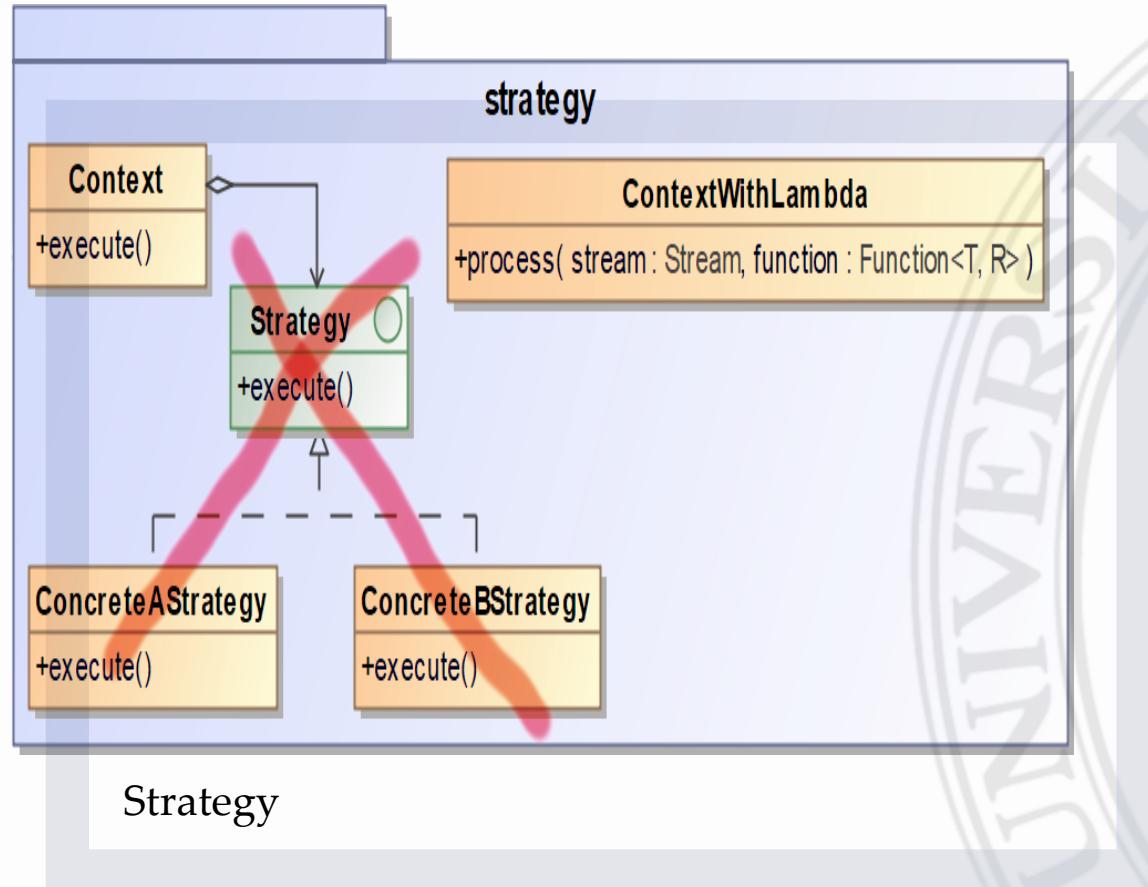
Proporciona un interface unificado para un conjunto de interfaces de un subsistema

① El código aportado en el desarrollo del *Client1* se reutiliza con facilidad para el *Client2*

Podría ser necesario ampliar la fachada para el desarrollo de Cliente2

# Strategy

Propósito: Comportamiento. Ámbito: objeto



## Propósito

- Define un conjunto de algoritmo haciéndolos intercambiables dinámicamente

# Data Acces Object

## *Spring - Test*

Functional Test:  
\*FT

API Rest  
Spring-Test  
JUnit5

IntegrationTest:  
\*IT

Spring-Test  
JUnit5

*SeedDatabaseService*

seedDatabase()  
deleteAll()

# Spring Data

## Tests. Unitarios y de Integración

```
class BadgeTest {  
    @Test  
    void testGenerateBadge() {  
        String badge = new Badge().generateBadge("Render", "v2.2.0-SNAPSHOT");  
        assertNotNull(badge);  
        assertEquals("<svg", badge.substring(0, 4));  
    }  
}  
  
@SpringBootTest  
@ActiveProfiles("test")  
class ArticleServiceIT {  
    @Autowired  
    private ArticleService articleService;  
    @Autowired  
    private ArticlePersistence articlePersistence;  
    @Test  
    void testUpdatePrices() {  
        List<ArticlePriceUpdating> articlePriceUpdatingList = List.of(  
            new ArticlePriceUpdating("84002", BigDecimal.ONE),  
            new ArticlePriceUpdating("84003", BigDecimal.TEN)  
        );  
        this.articleService.updatePrices(articlePriceUpdatingList.stream());  
        assertEquals(0, BigDecimal.ONE.compareTo(this.articlePersistence.read("84002").getPrice()));  
        assertEquals(0, BigDecimal.TEN.compareTo(this.articlePersistence.read("84003").getPrice()));  
        articlePriceUpdatingList = List.of(  
            new ArticlePriceUpdating("84002", new BigDecimal("0.27")),  
            new ArticlePriceUpdating("84003", new BigDecimal("12.13"))  
        );  
        this.articleService.updatePrices(articlePriceUpdatingList.stream());  
    }  
}
```



# Spring Data Tests. Funcionales

```
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
@ActiveProfiles("test")
class ArticleResourceFunctionalTest {
    @LocalServerPort private int port;
    @Autowired private TestRestTemplate restTemplate;
    private String baseUrl;
    private HttpHeaders headers;
    @BeforeEach
    void setUp() {
        baseUrl = "http://localhost:" + port + ArticleResource.ARTICLES;
        headers = new HttpHeaders();
        headers.setContentType(MediaType.APPLICATION_JSON);
    }
    @Test
    void testCreate() {
        Article article = new Article("666004", "art rest", new BigDecimal("3.00"), null);
        HttpEntity<Article> request = new HttpEntity<>(article, this.headers);
        ResponseEntity<Article> response = restTemplate.exchange(this.baseUrl, HttpMethod.POST, request, Article.class);
        assertThat(response.getStatusCode()).isEqualTo(HttpStatus.OK);
        assertThat(response.getBody()).isNotNull();
    }
    @Test
    void testCreateConflict() {
        Article article = new Article("84001", "repeated", new BigDecimal("3.00"), null);
        HttpEntity<Article> request = new HttpEntity<>(article, this.headers);
        ResponseEntity<String> response = restTemplate.exchange(this.baseUrl, HttpMethod.POST, request, String.class);
        assertThat(response.getStatusCode()).isEqualTo(HttpStatus.CONFLICT);
    }
}
```