



Arquitectura y Patrones para Aplicaciones Web -APAW-

WebArchitecture

Jesús Bernal Bermúdez

Arquitectura Web

¿Qué es la Arquitectura software?

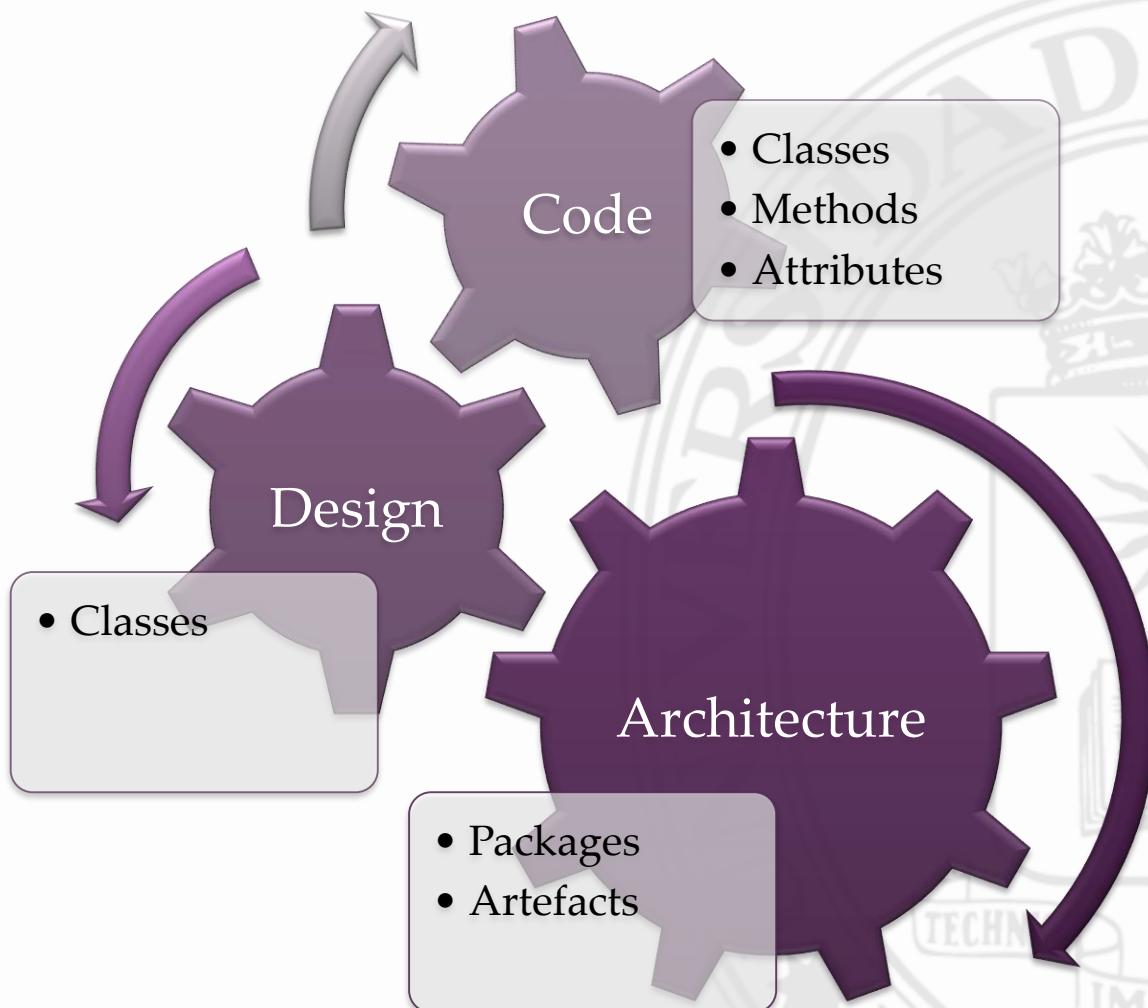


IEEE

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

Arquitectura Web

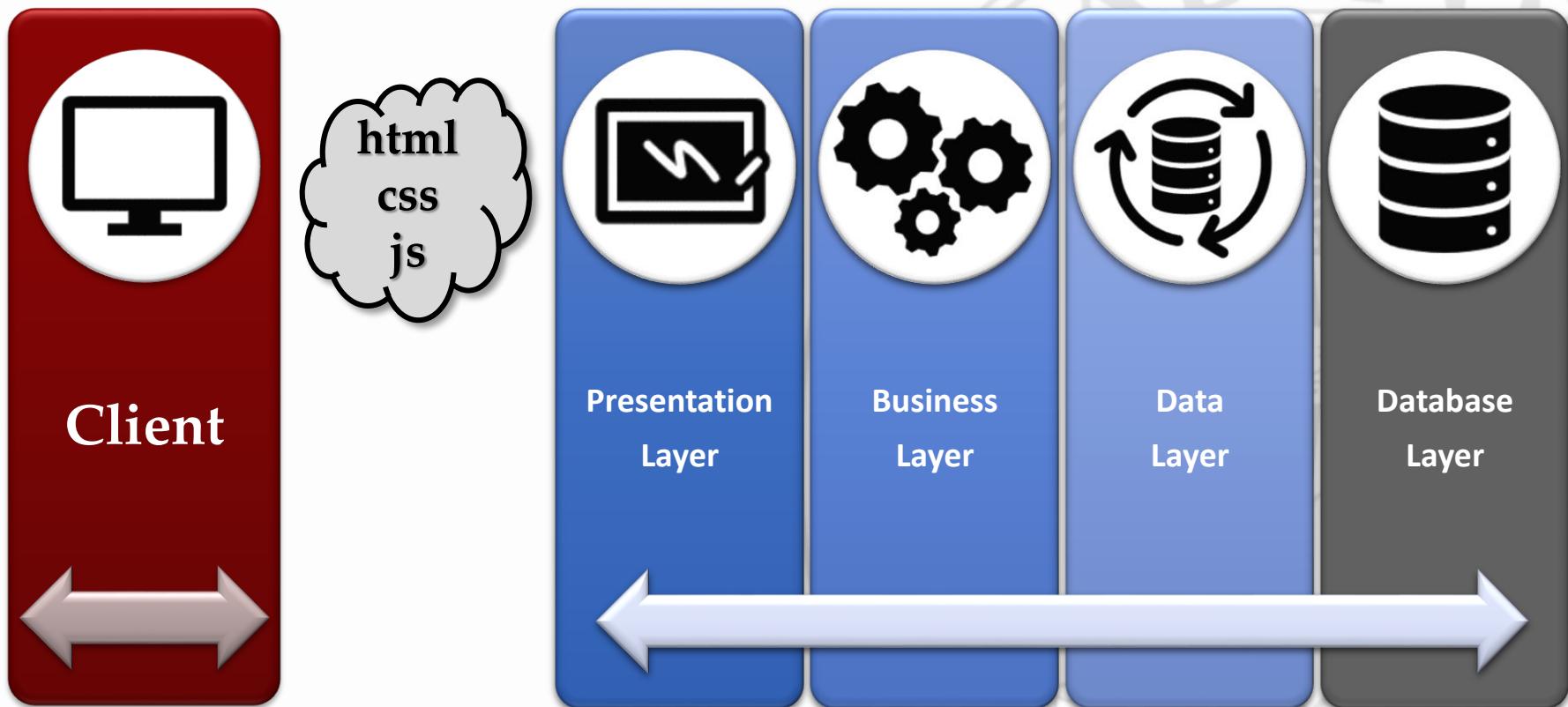
Unidad de gestión





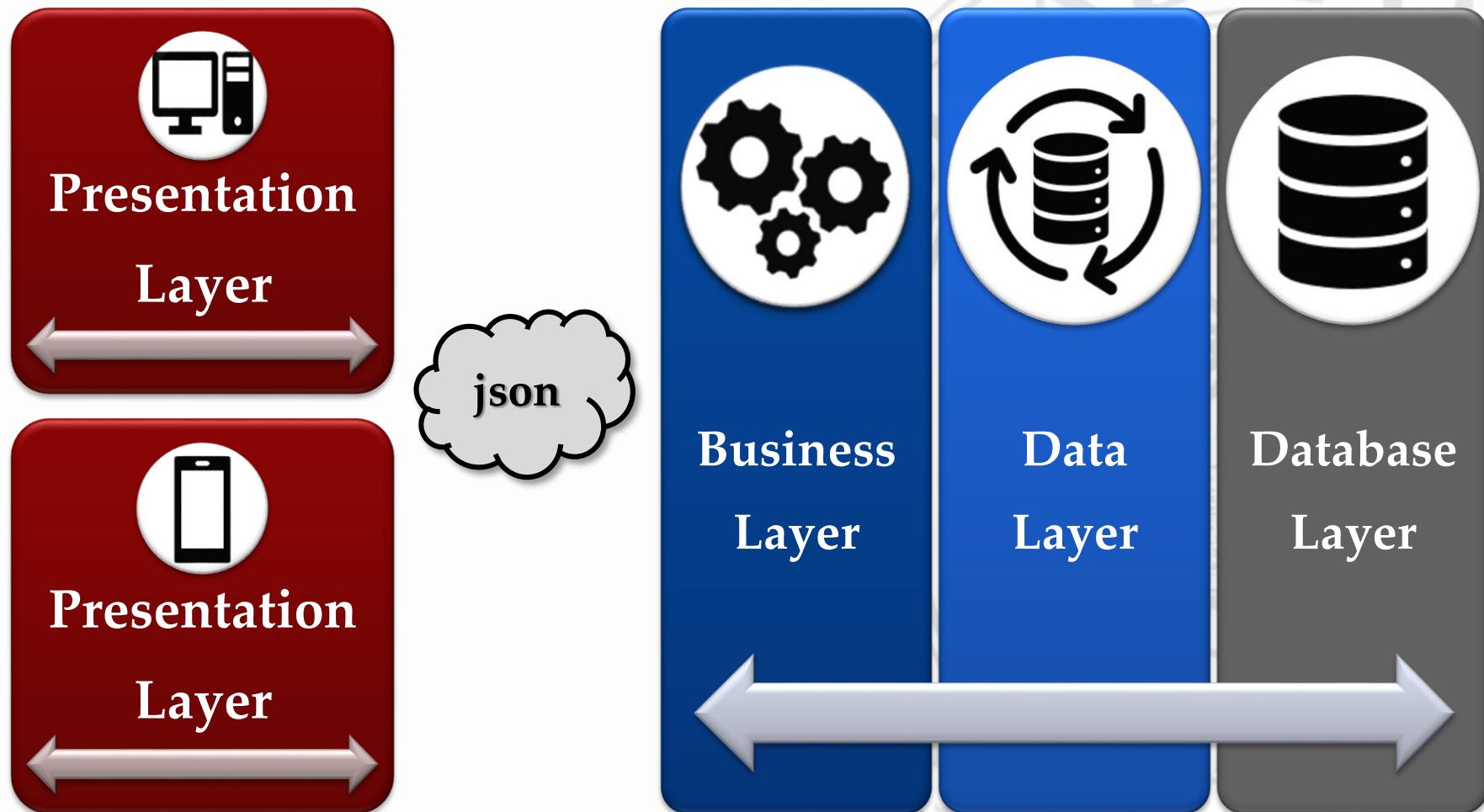
Arquitectura por Capas

Aplicación de Múltiples Páginas



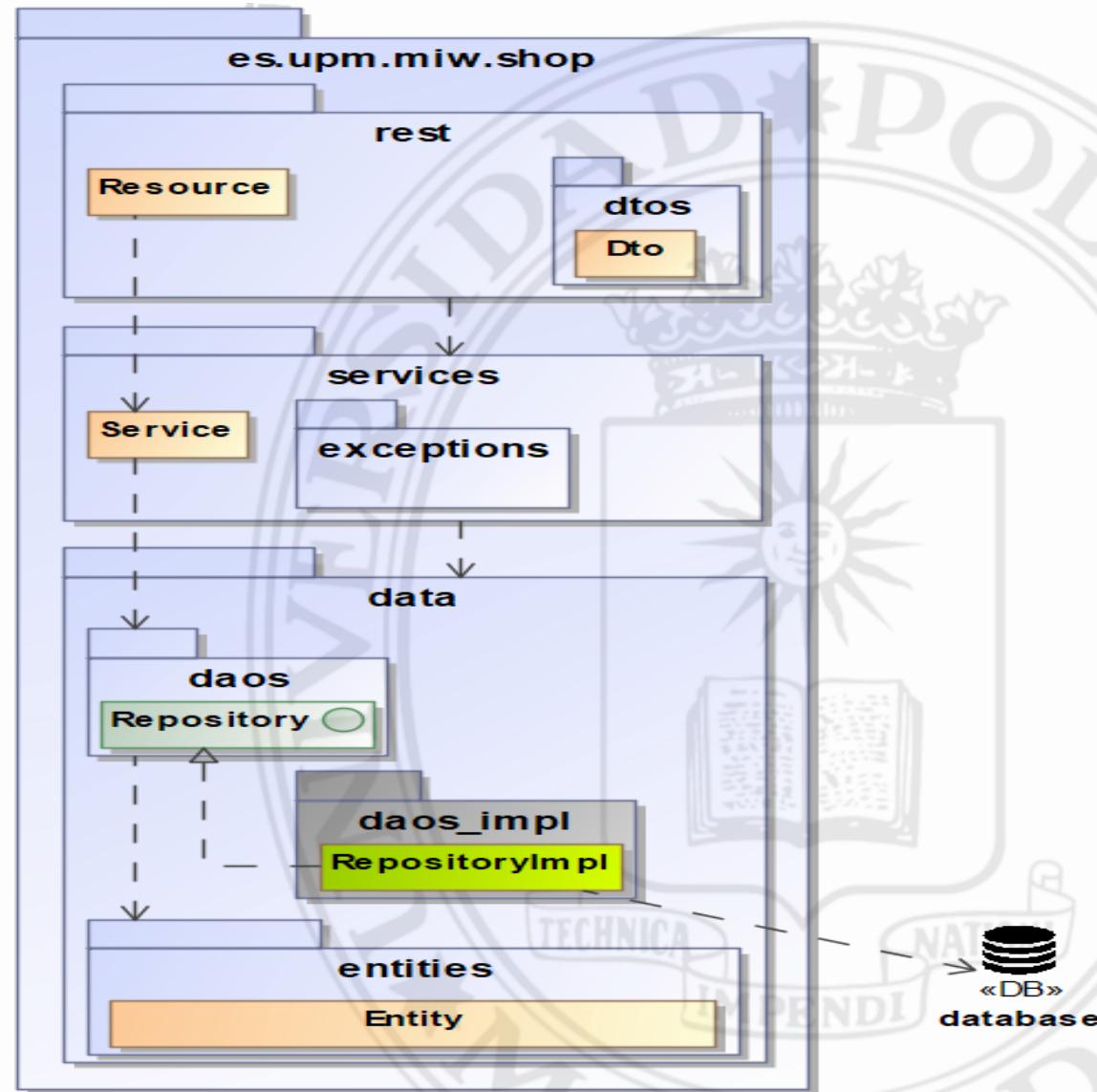
Arquitectura por Capas

Aplicación de Página Única (*SPA*)



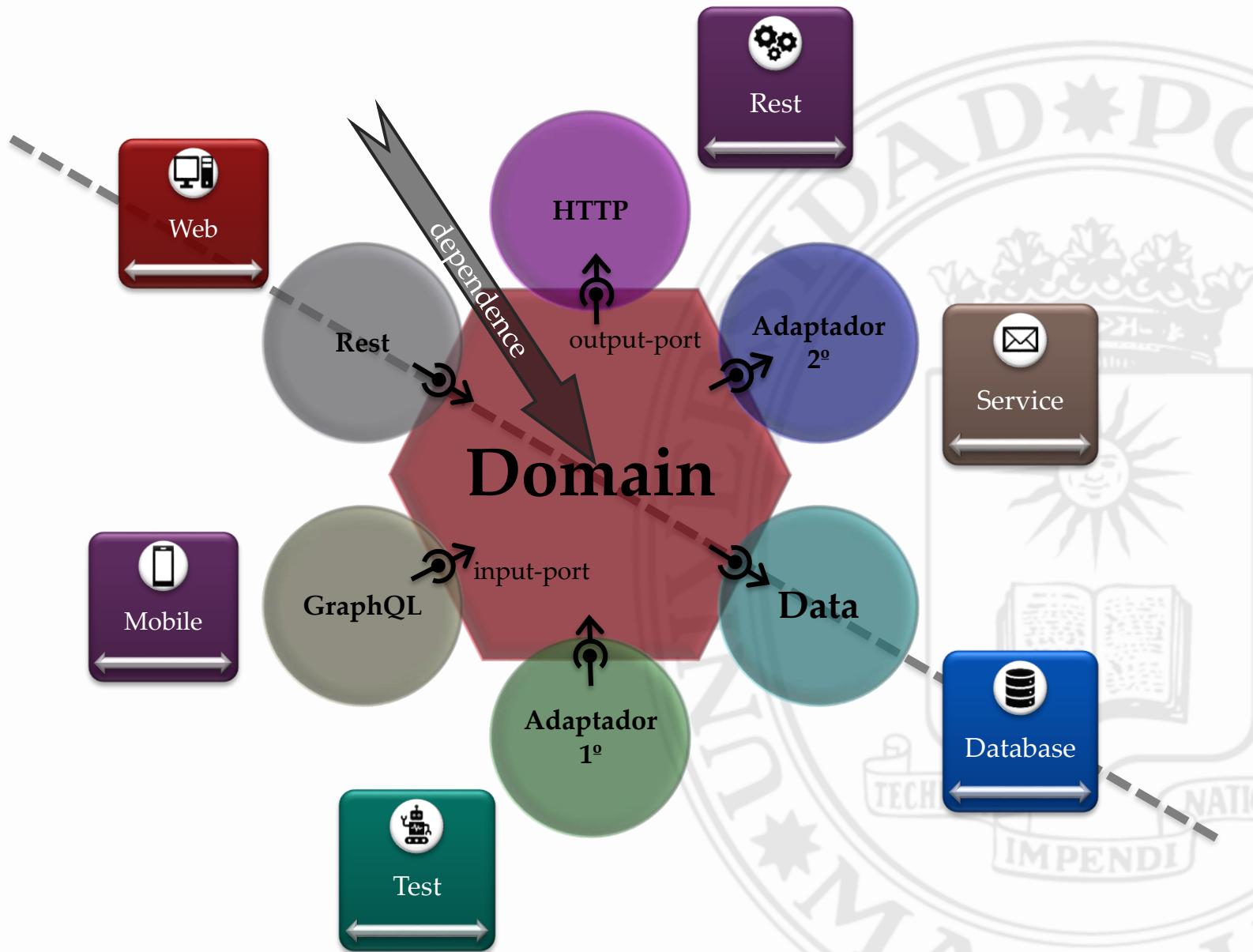
Arquitectura por Capas

Aplicación de Página Única (*SPA*)



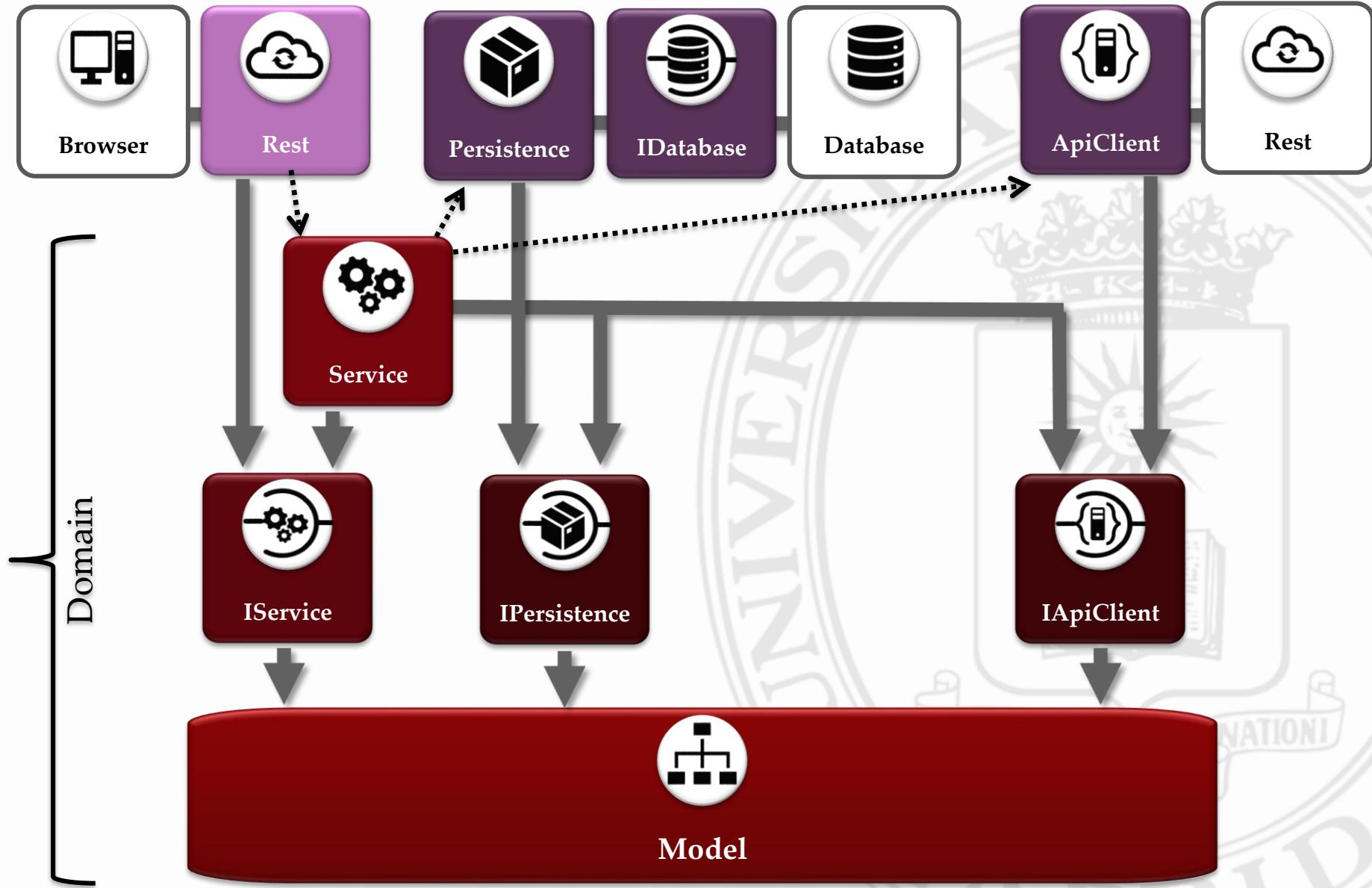
Hexagonal Architecture

Alistair Cockburn

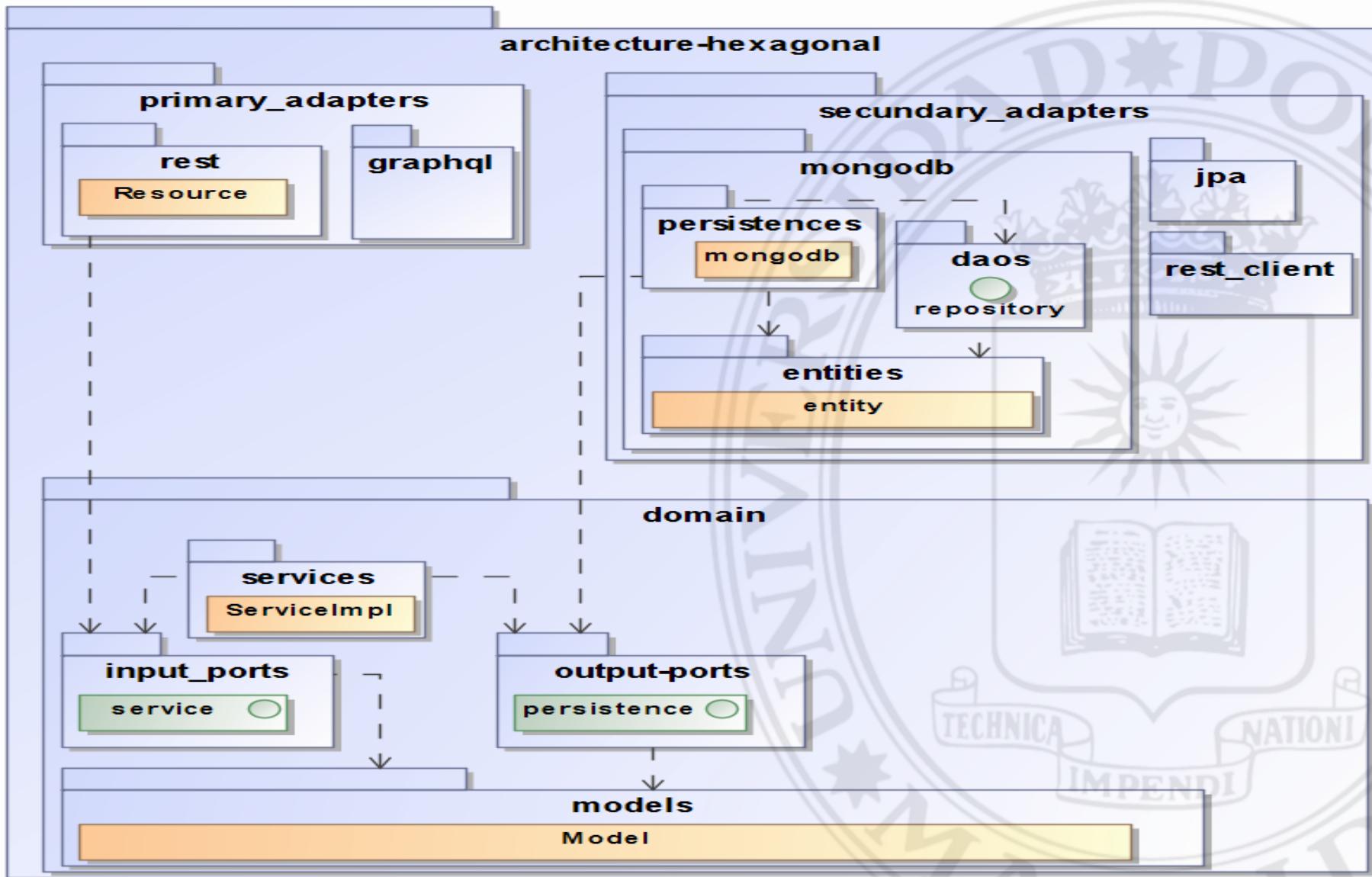


Hexagonal Architecture

Dependencias

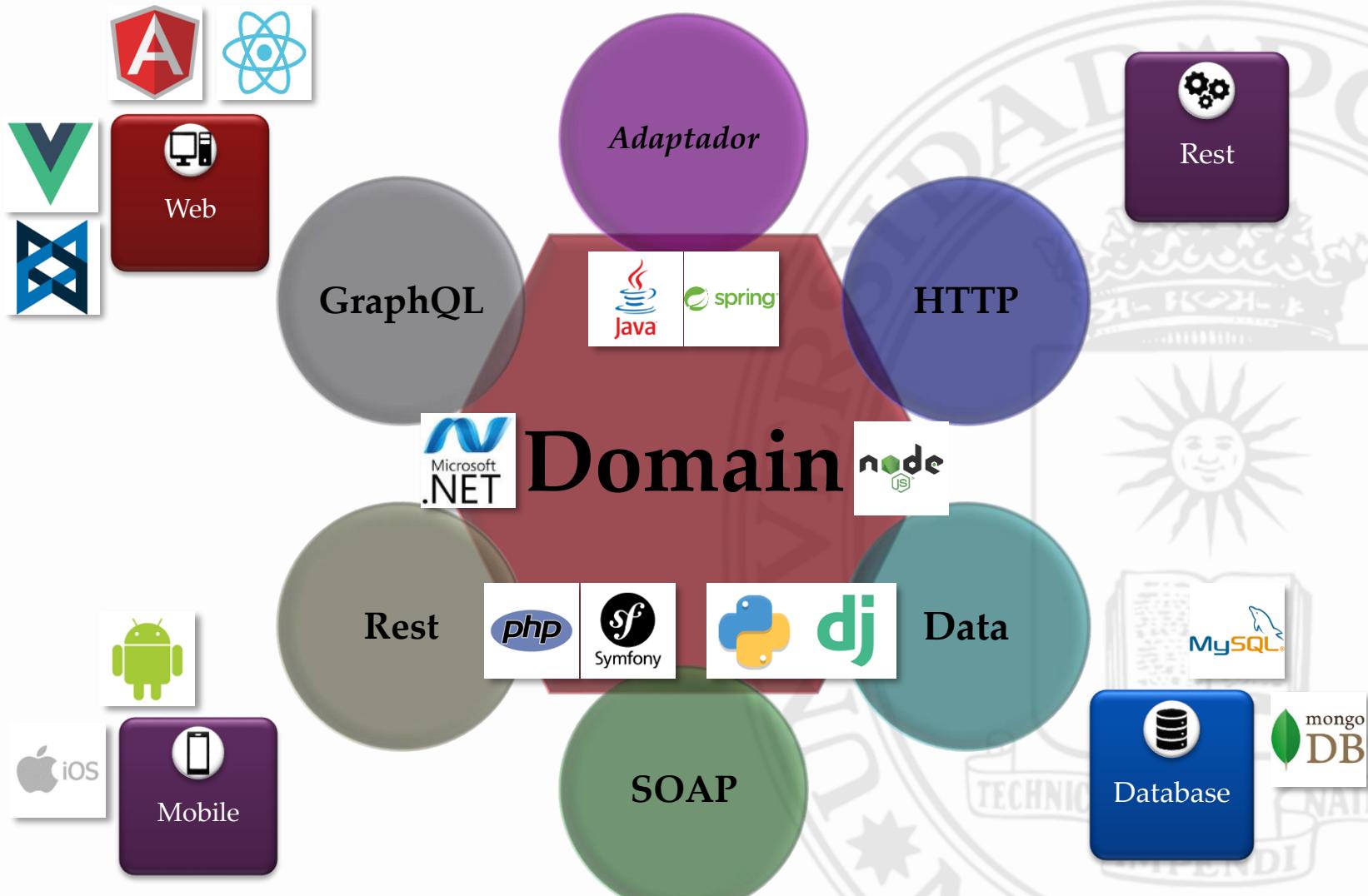


Arquitectura Paquetes



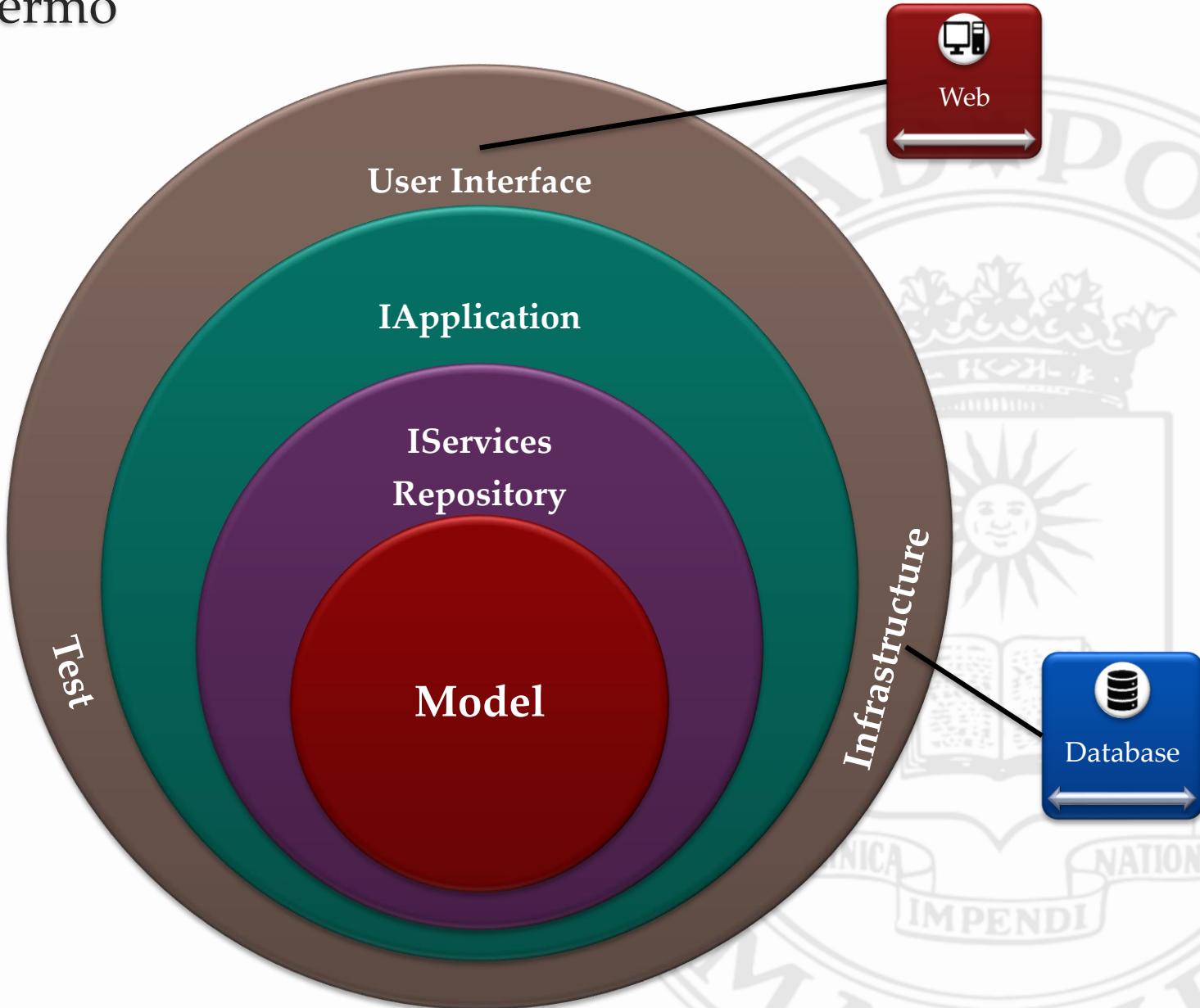
Arquitectura Hexagonal

Tecnologías



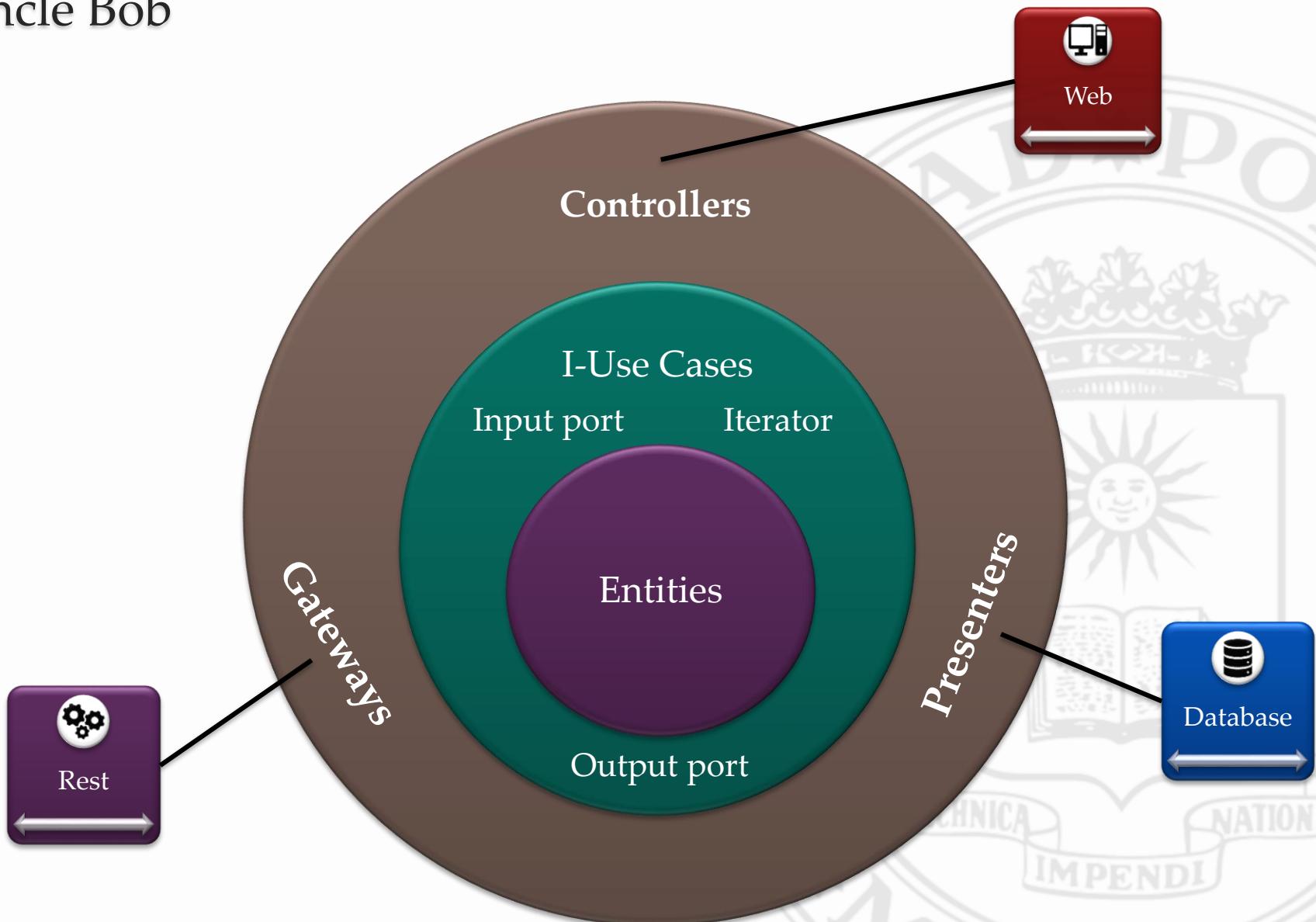
Onion Architecture

Jeffrey Palermo



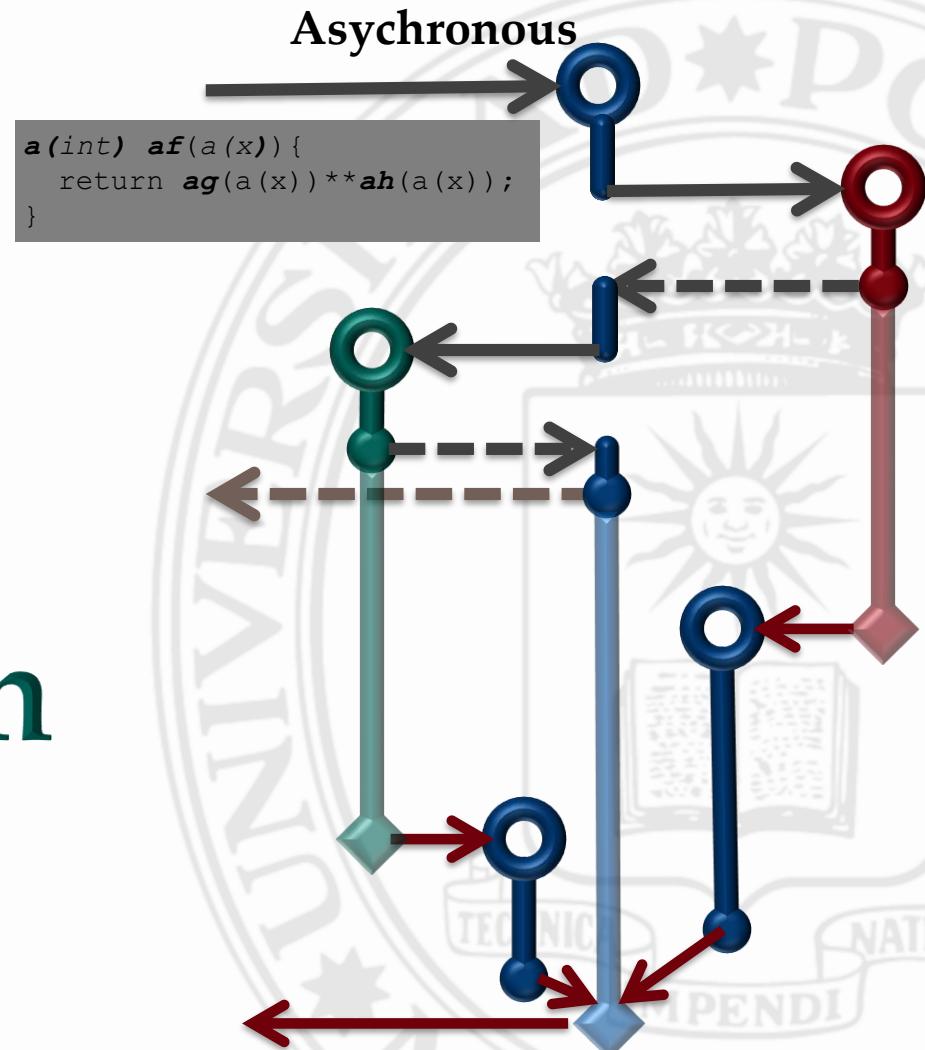
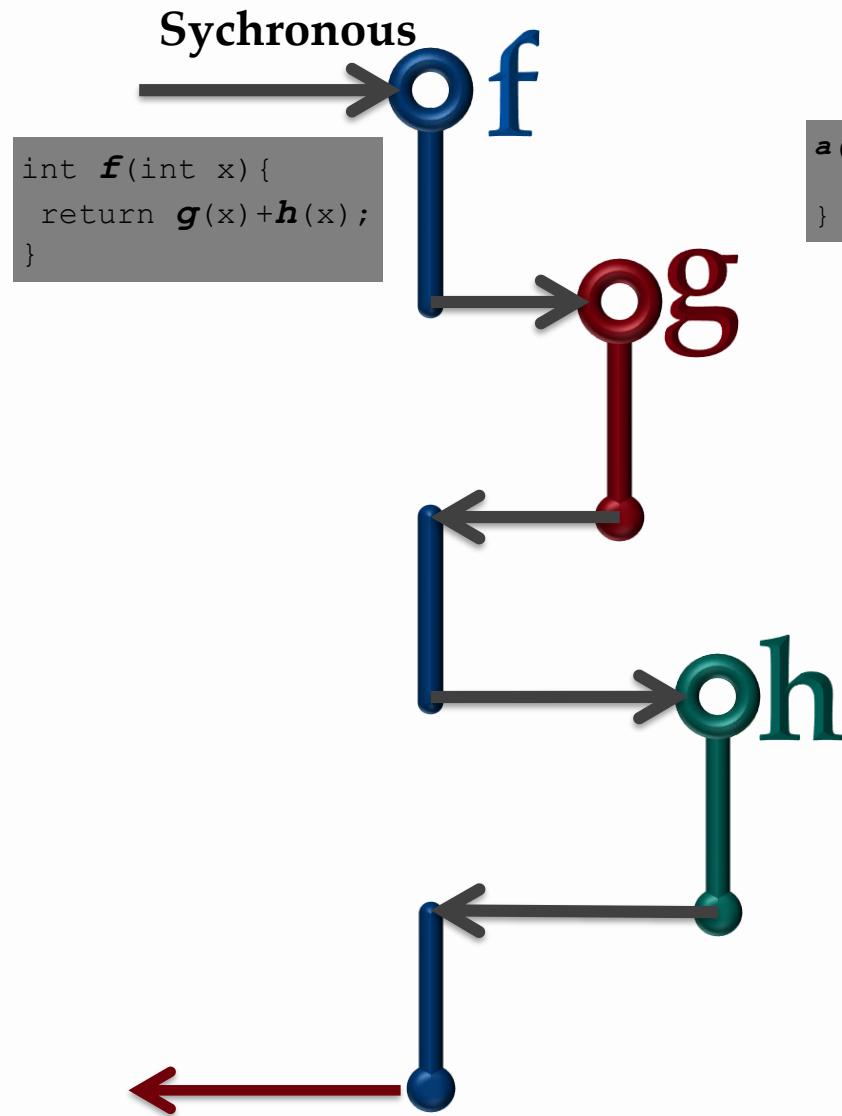
Clean Architecture

Uncle Bob



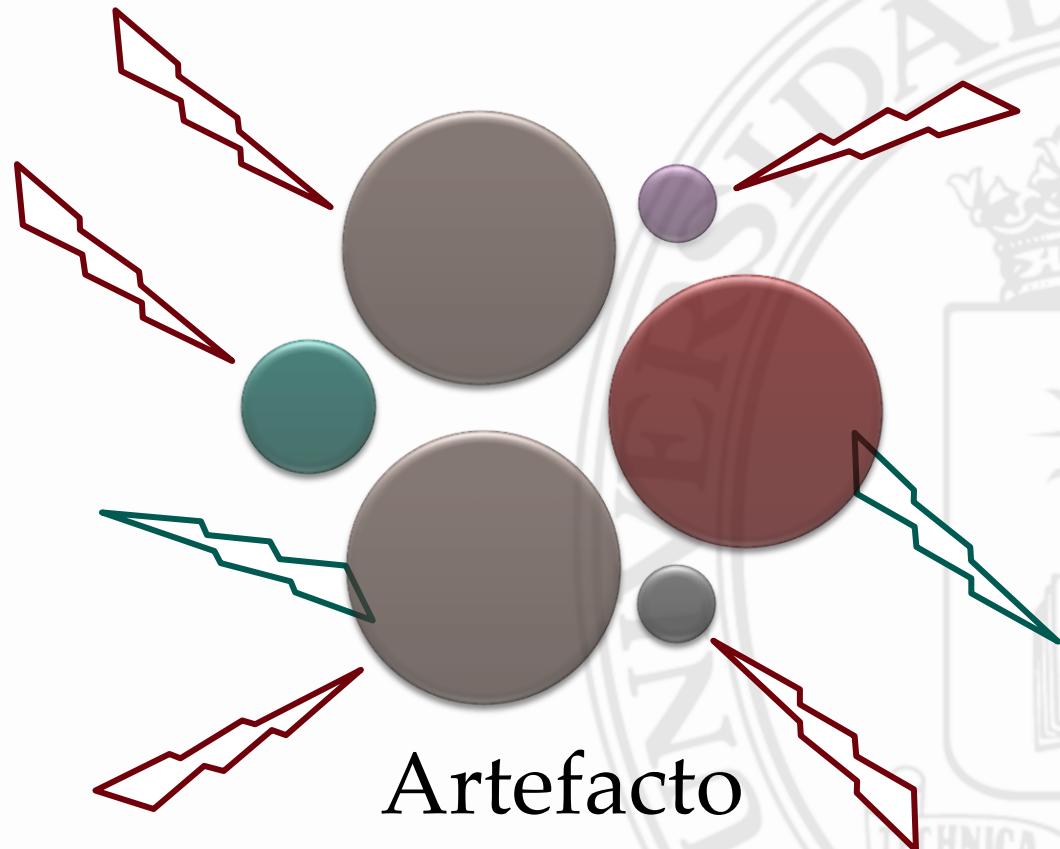
Arquitectura dirigida por Eventos

Síncrono - Asíncrono

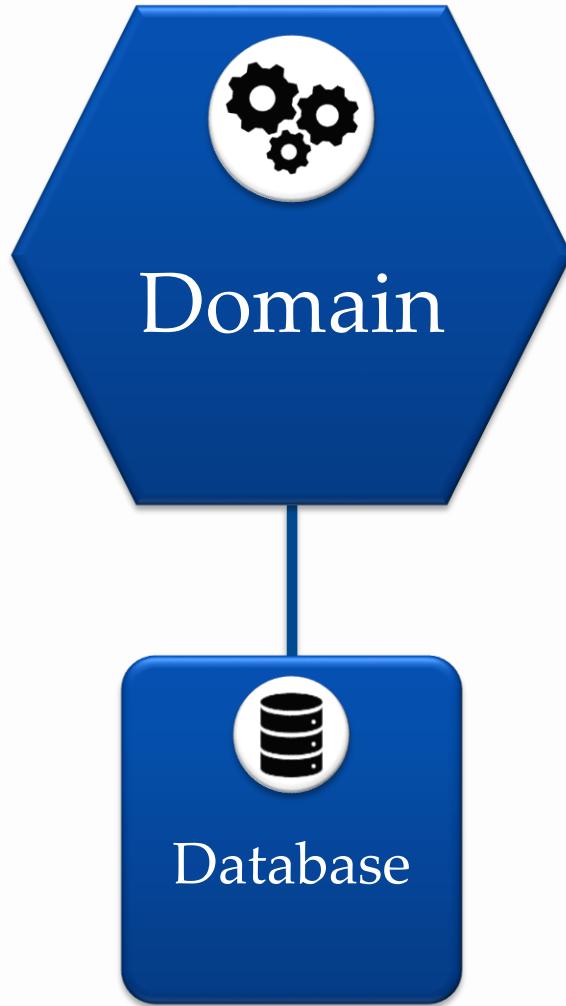


Arquitectura dirigida por Eventos

Event Driven Architecture



Aplicación Monolítica



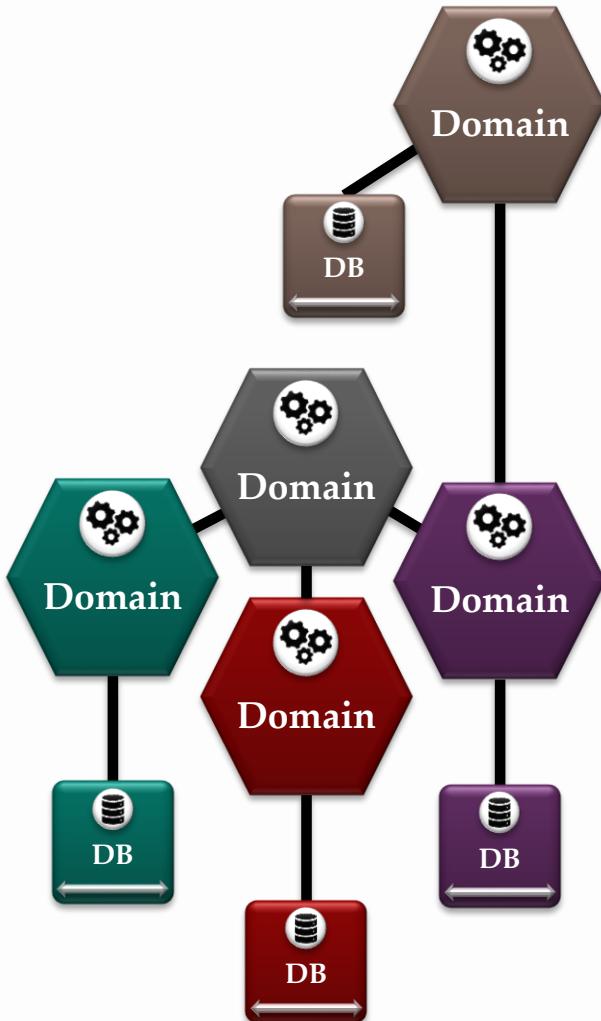
Ventajas

- 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.

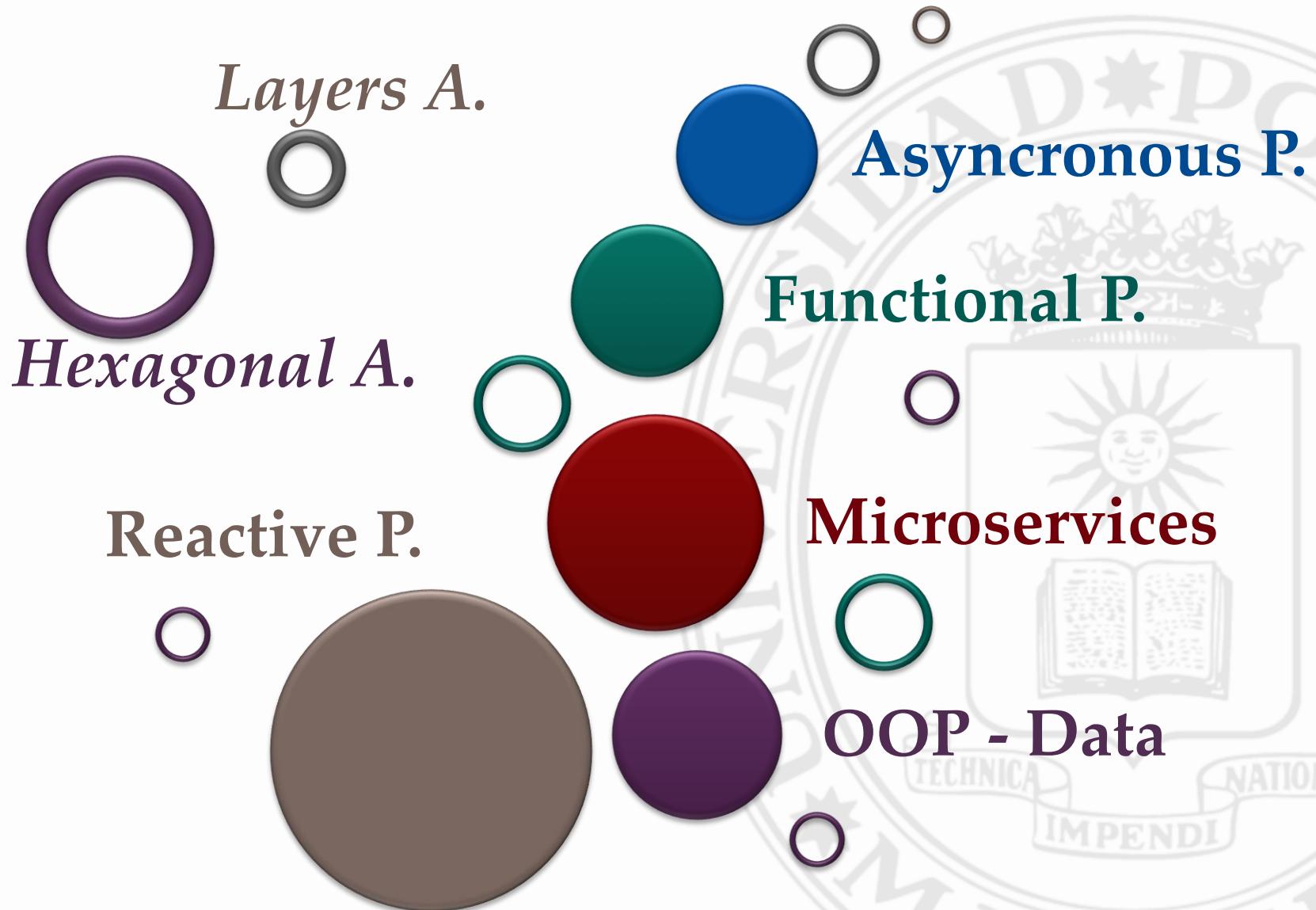
Inconvenientes

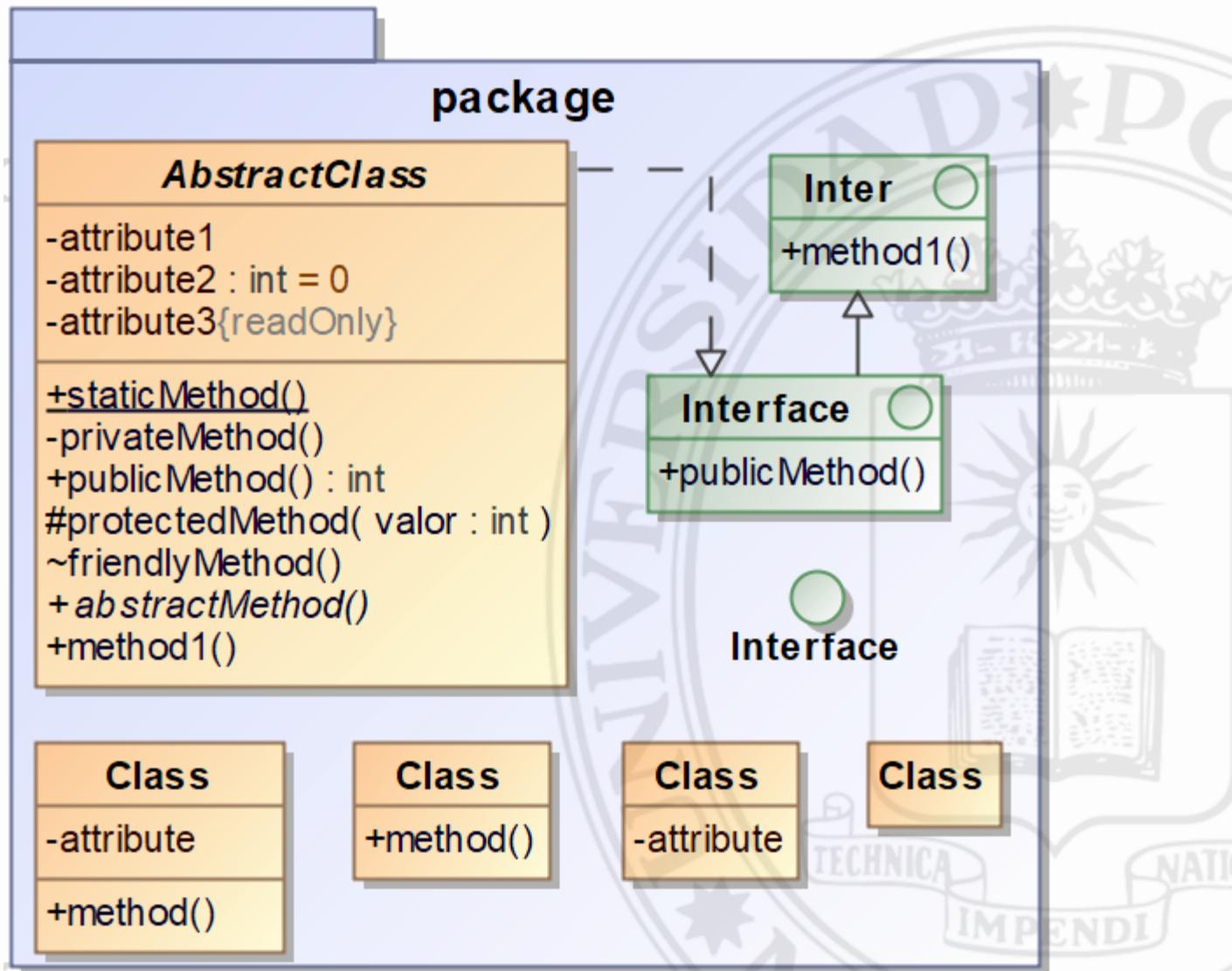
- **Distribución.** Al ser un sistema distribuido, es más complejo su despliegue

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

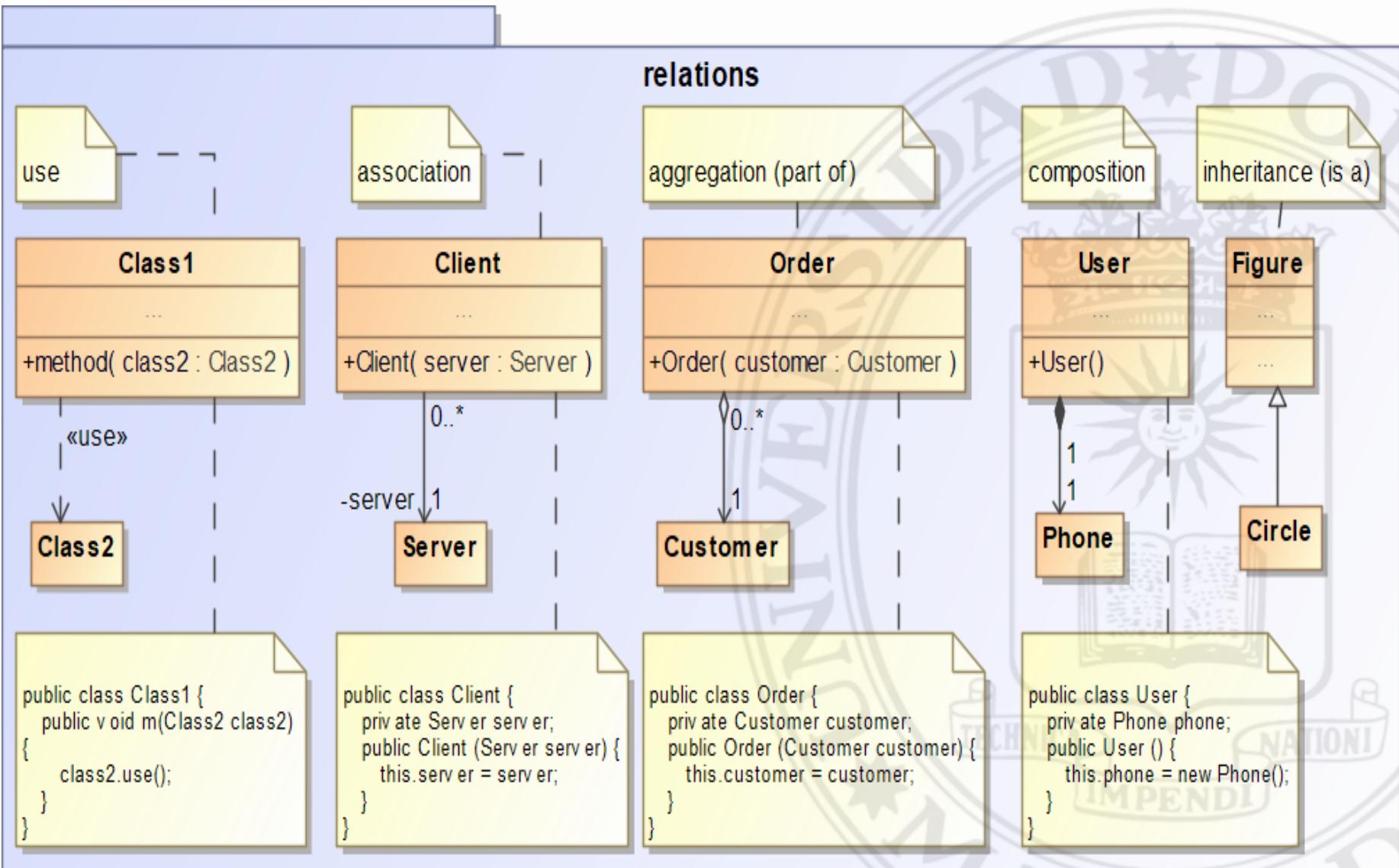
- **Complejidad operacional.** Se necesita acceder a varios servicios para realizar una operación

Tendencias...



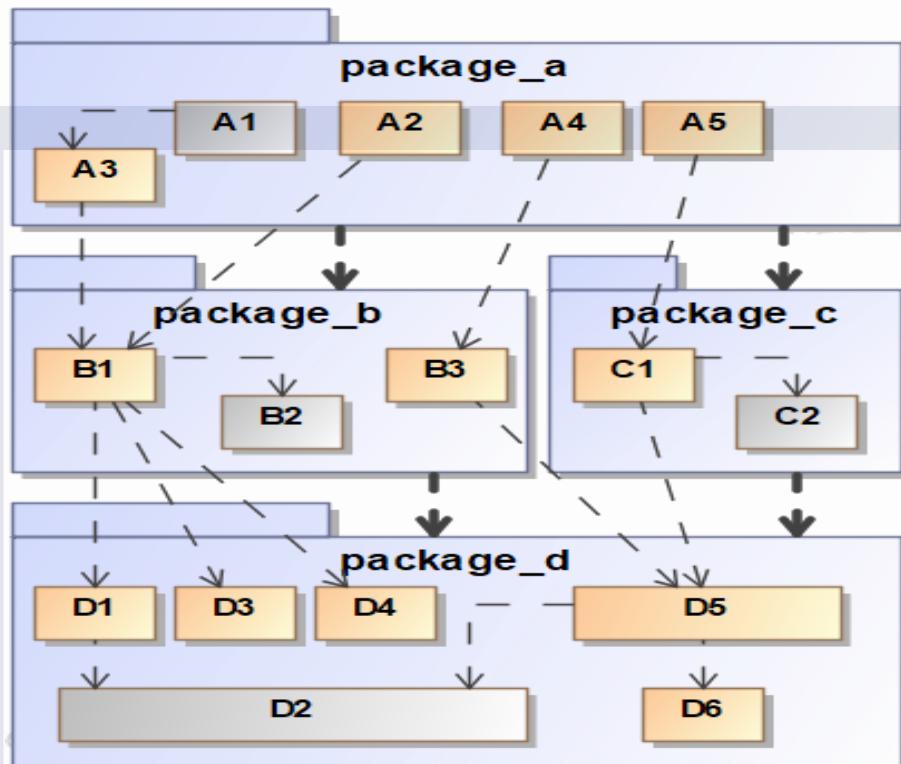


Arquitectura UML



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

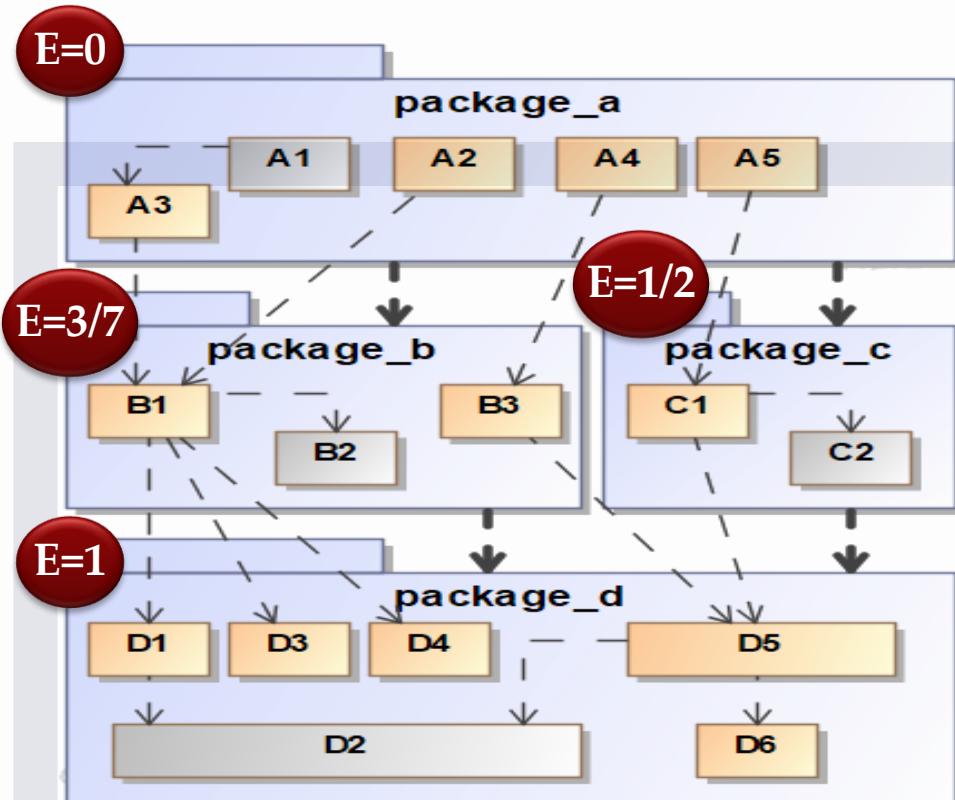
- Un cambio que afecte a una clase, afecta a todo el paquete.

Principio de dependencias acíclicas

- Ciclo de dependencias sin bucle

Arquitectura

Principios de Robert Martin



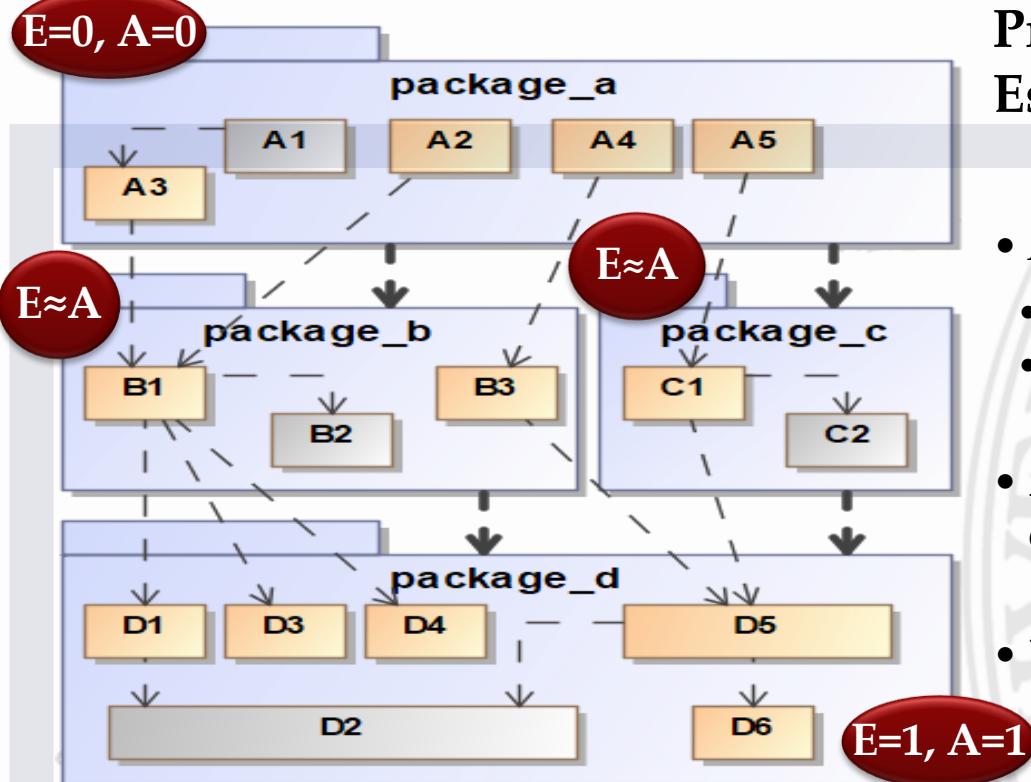
Principio de Dependencias Estables

- 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.

$$\bullet \text{Inestabilidad} = 1 - \text{Estabilidad}$$

Arquitectura

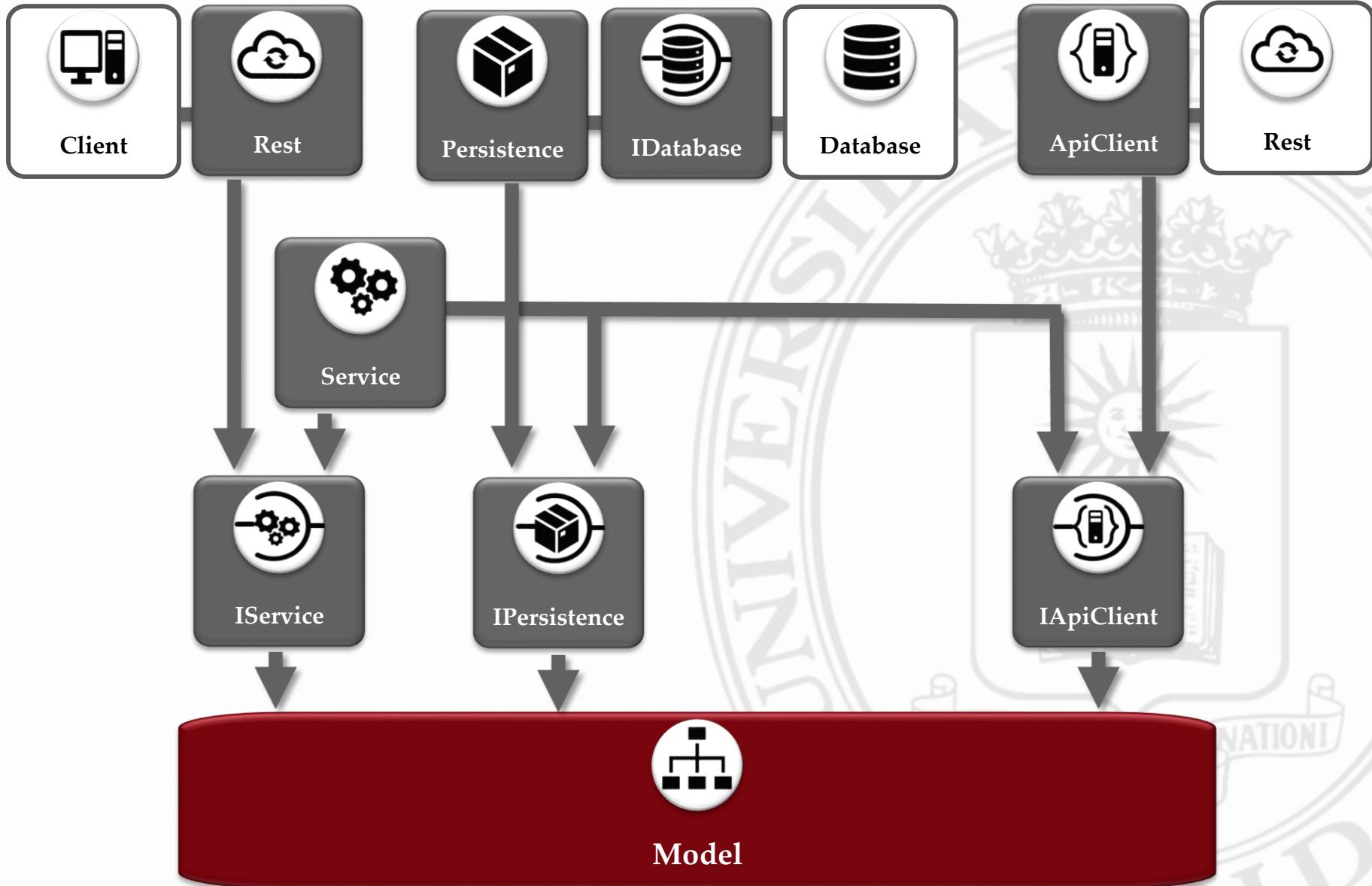
Principios de Robert Martin



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

Modelización

Clases

Relaciones

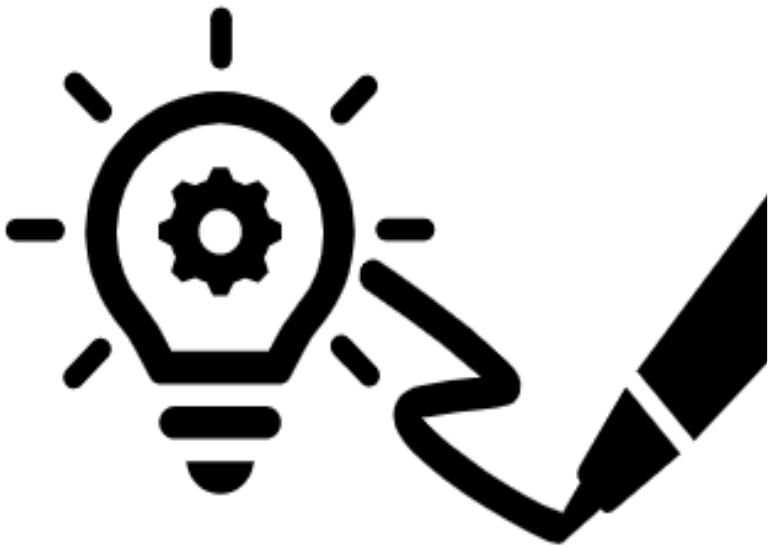
- Dirección
- Tipo
- Multiplicidad

Atributos

Métodos

- `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.

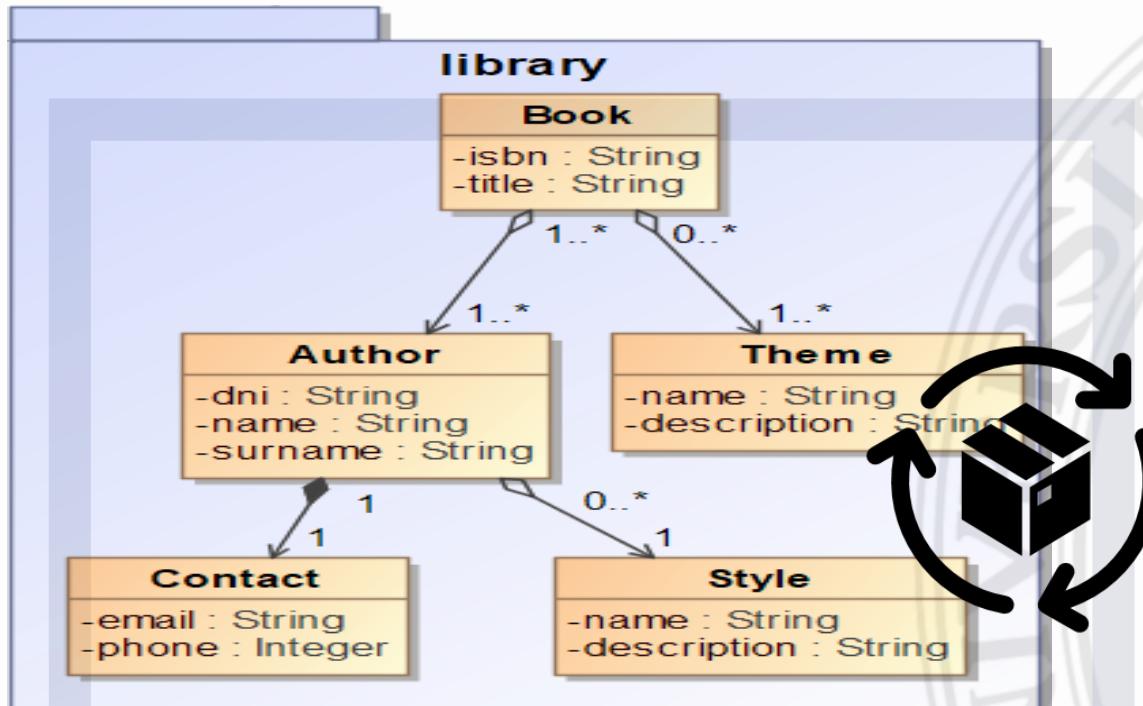
Un tema tiene un nombre y un tema puede estar asociado a muchos libros.

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



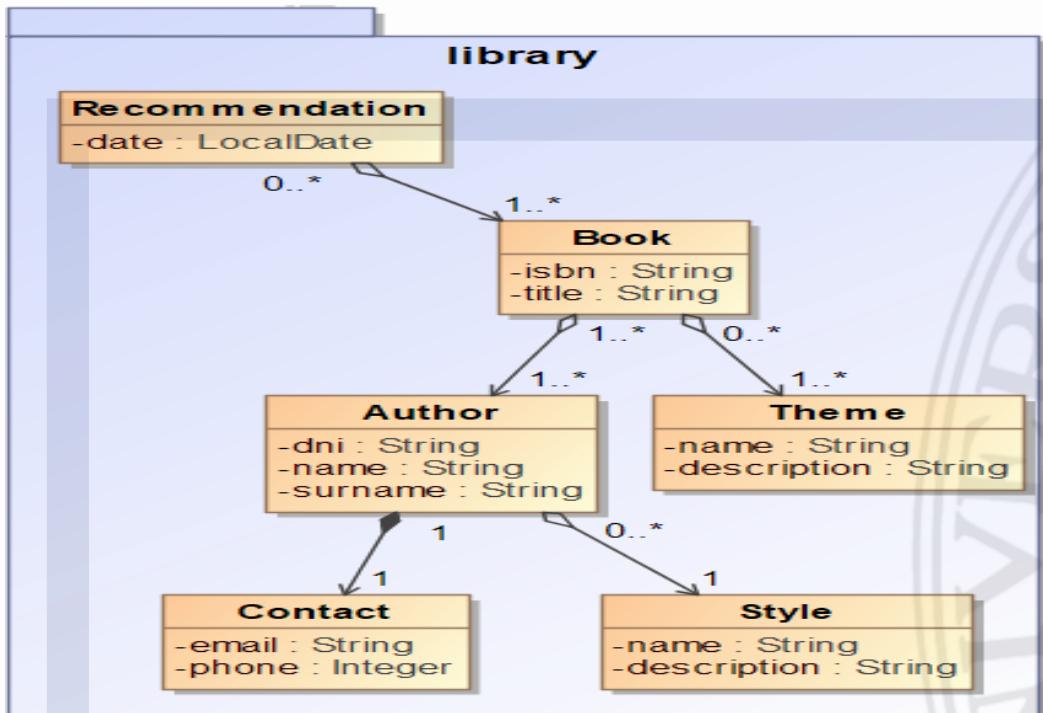
Biblioteca

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

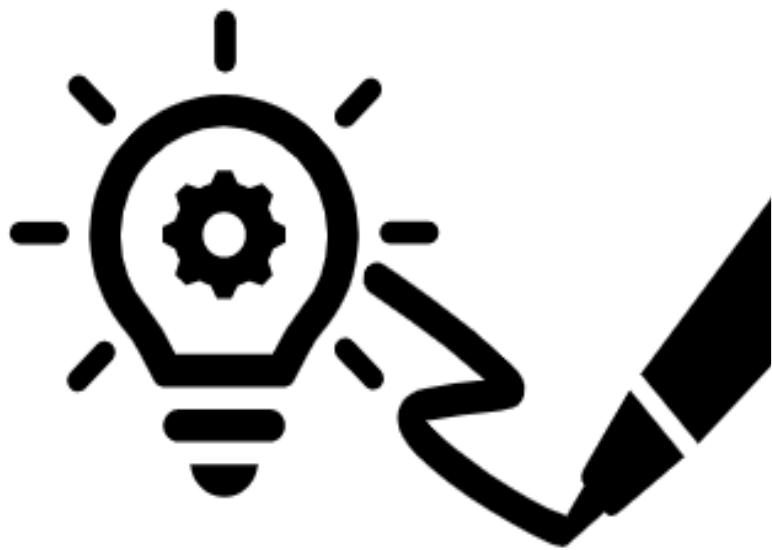


Biblioteca

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.



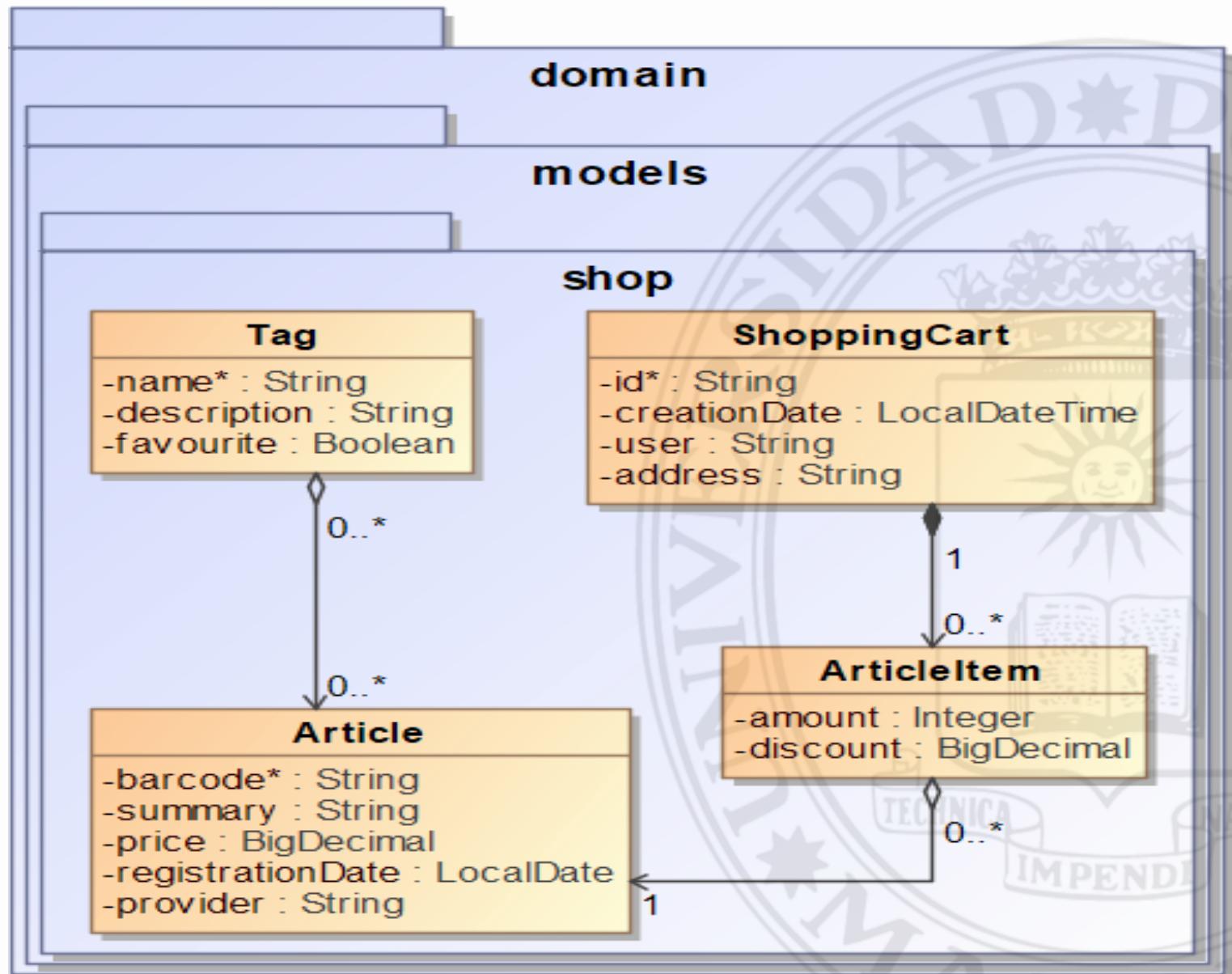
Ejercicio

Nos piden modelizar:

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

Con 4 clases, cada una con al menos 3 atributos y un total de al menos de 12 atributos. Relaciones 1..n, n..1 y n..n. Proyecto y clases únicas para todos los grupos.

Shop



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

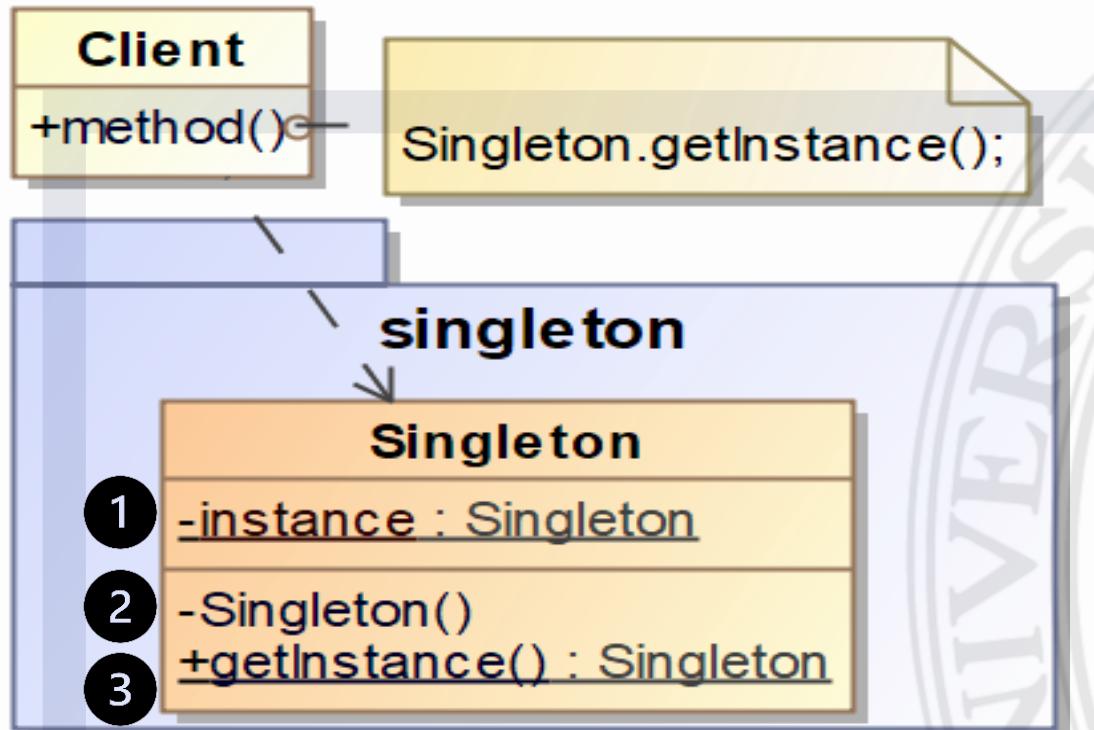
- Externaliza el estado interno de un objeto sin violar la encapsulación.

Decorator

- Asigna responsabilidades de forma dinámica a objetos, proporcionando una alternativa flexible a la herencia.

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

-logs : String

+Logger()

+addLog(log : String)

+getLogs() : String

+clear()

Logger

solution

Logger

-logger : Logger

-logs : String

-Logger()

+getLogger() : Logger

+addLog(log : String)

+getLogs() : String

+clear()

Logger & Singleton

Singleton

📝 {→}

factory

ReferencesFactory

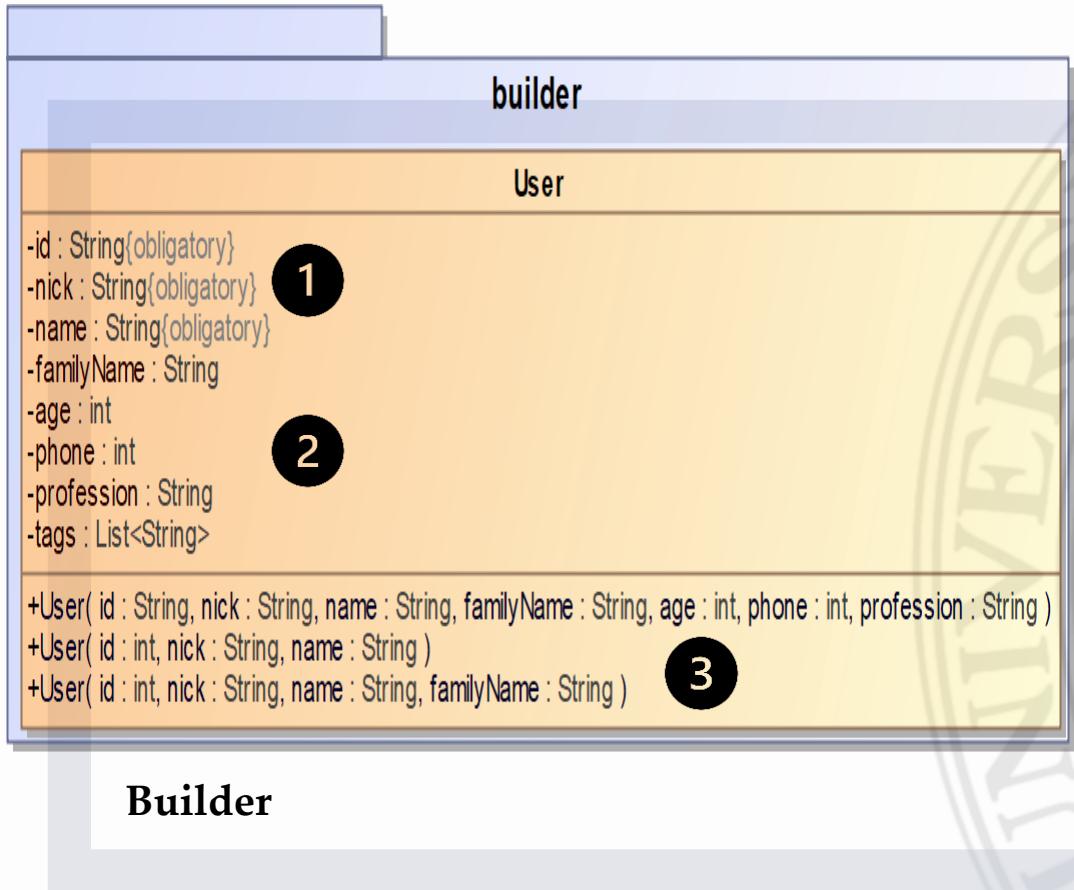
-references : Map<String, Integer>
-reference : int

+ReferencesFactory()
+getReference(key : String) : int
+removeReference(key : String)

Aplicar el patrón *Singleton* a *ReferencesFactory*, con creación temprana

Builder

Propósito: Creación. Ámbito: objeto



```
User user = new User("id1","Paco","Jose","De Miguel",25,666666666,"Profesor",
    Arrays.asList("Director", "Socio", "Consejo"));
```

① Atributos obligatorios.

② Atributos opcionales.

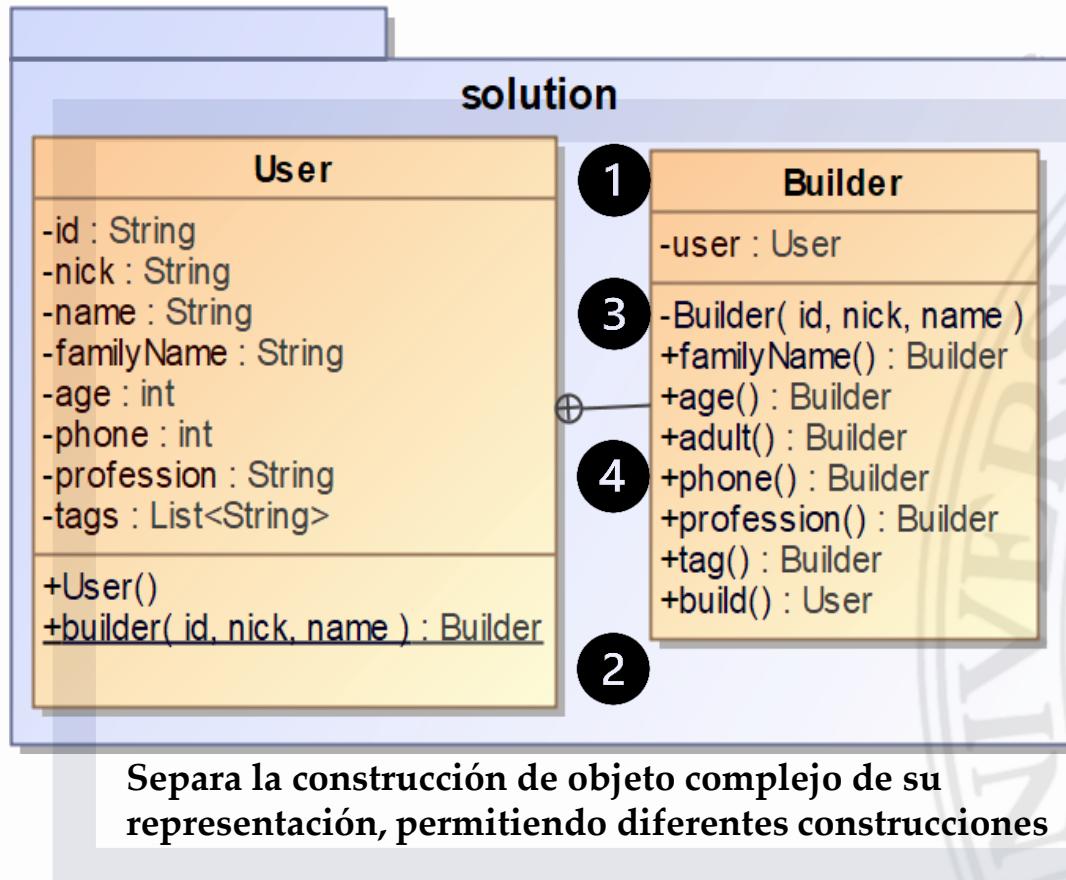
③ *Code Smell*

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



Builder

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*.

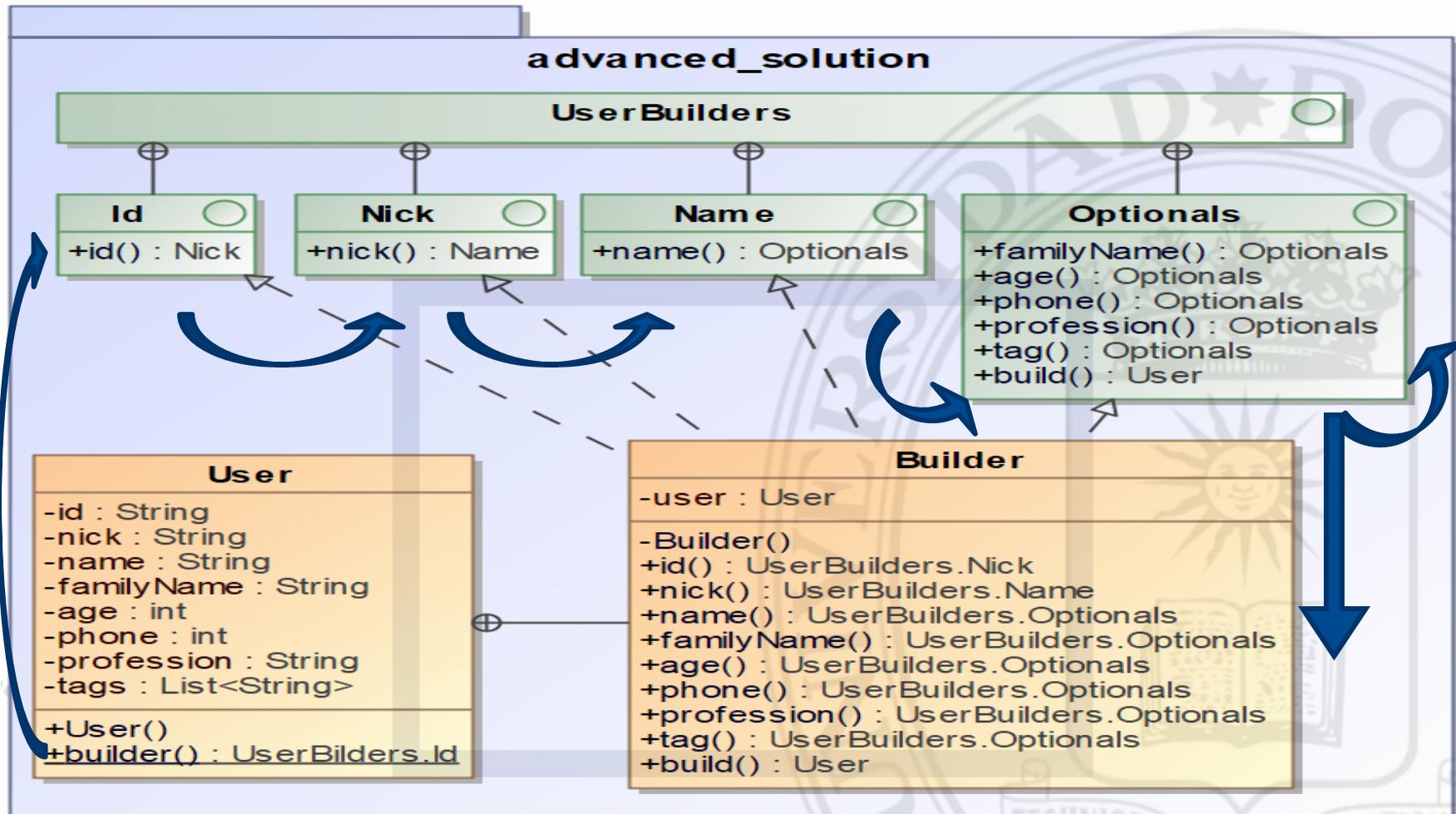
③ Constructor con los parámetros obligatorios.

④ Métodos devuelven *this*

- ¿Qué pasa si son muchos parámetros obligatorios?
- ¿Qué pasa si tenemos un orden de construcción?

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

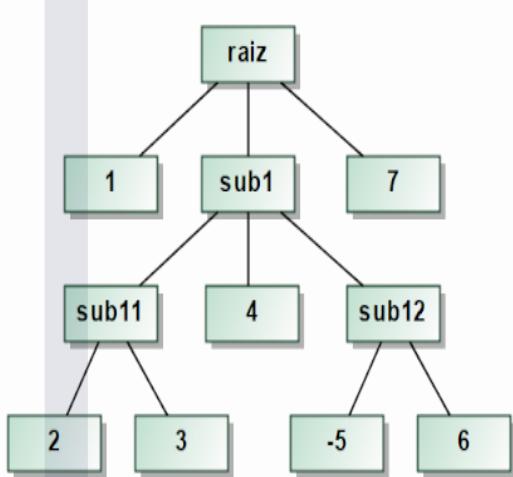
- User
- Soluciones

📝 Ejercicios

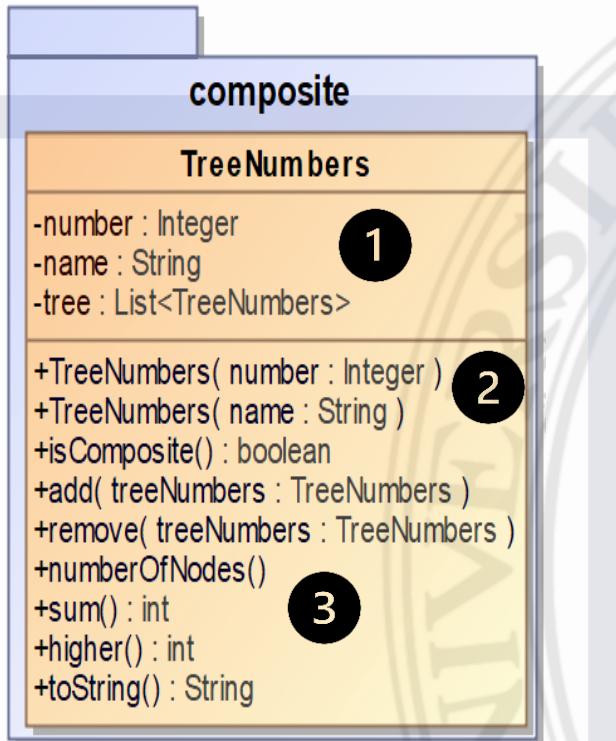
- Aplicar el patrón *builder* a la clase *Article*, atributos obligatorios: *id*, opcionales: *resto*.
- Aplicar el patrón *builder* a la clase *Article*, atributos obligatorios: *id, reference, description & retailPrice*, opcionales: *resto*.
- Aplicar solución con interface para secuenciar los obligatorios.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

Composite

Motivación



Composite

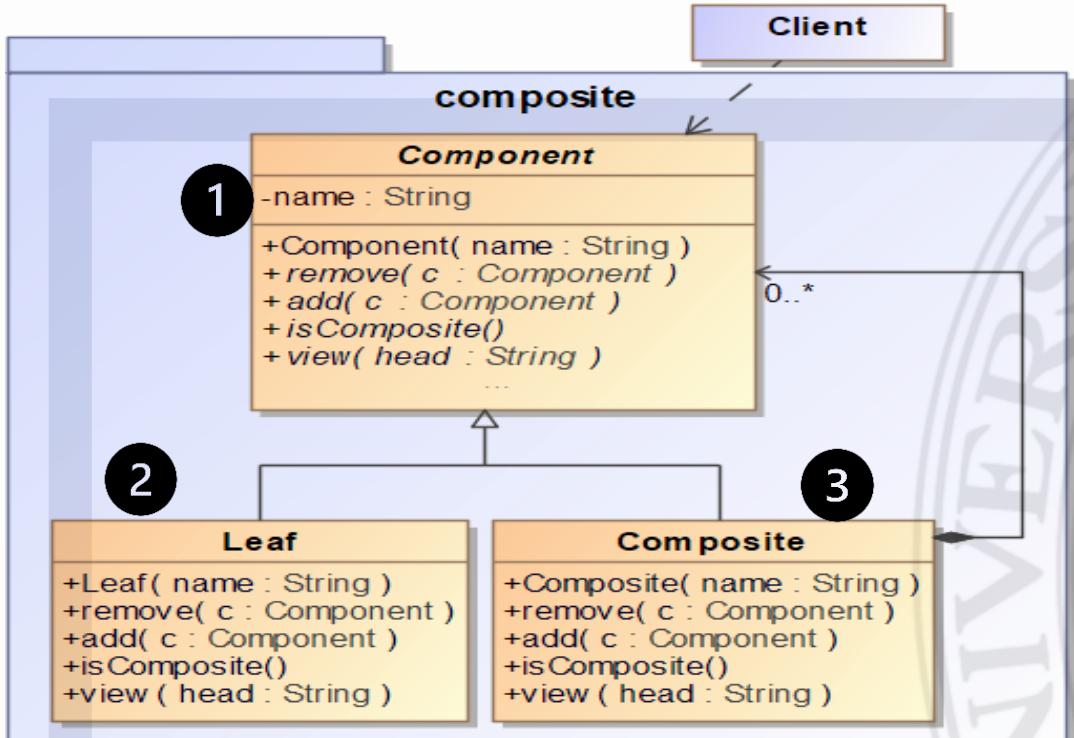


¿Cumple DOO?

- 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

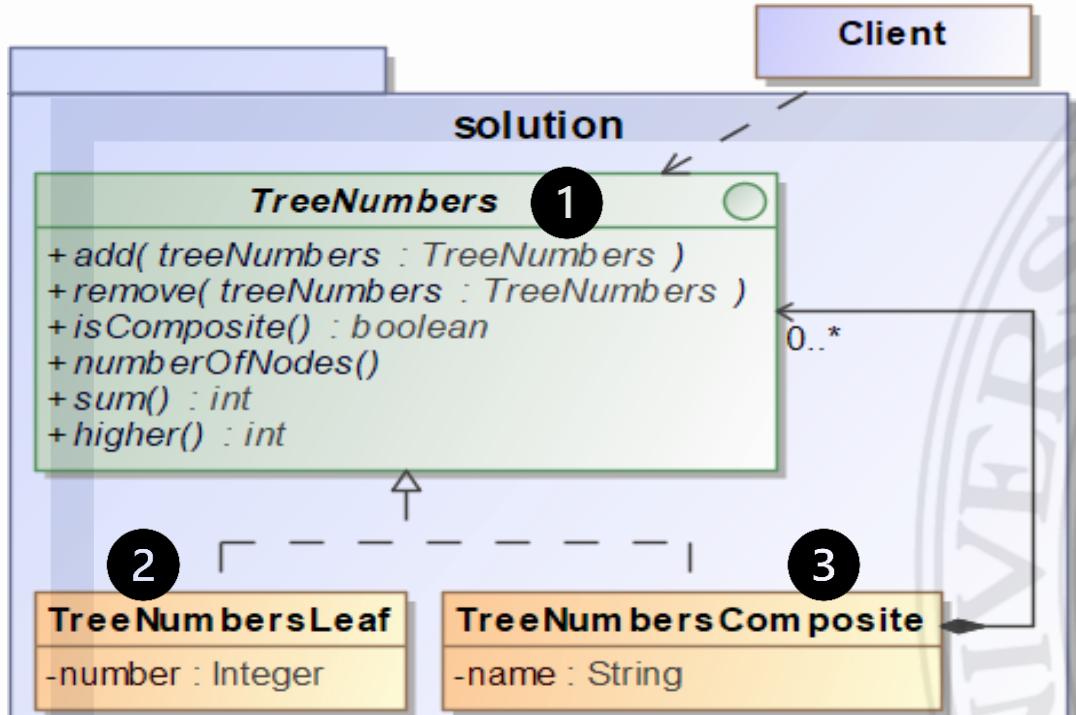
- Hereda de Componente.

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.

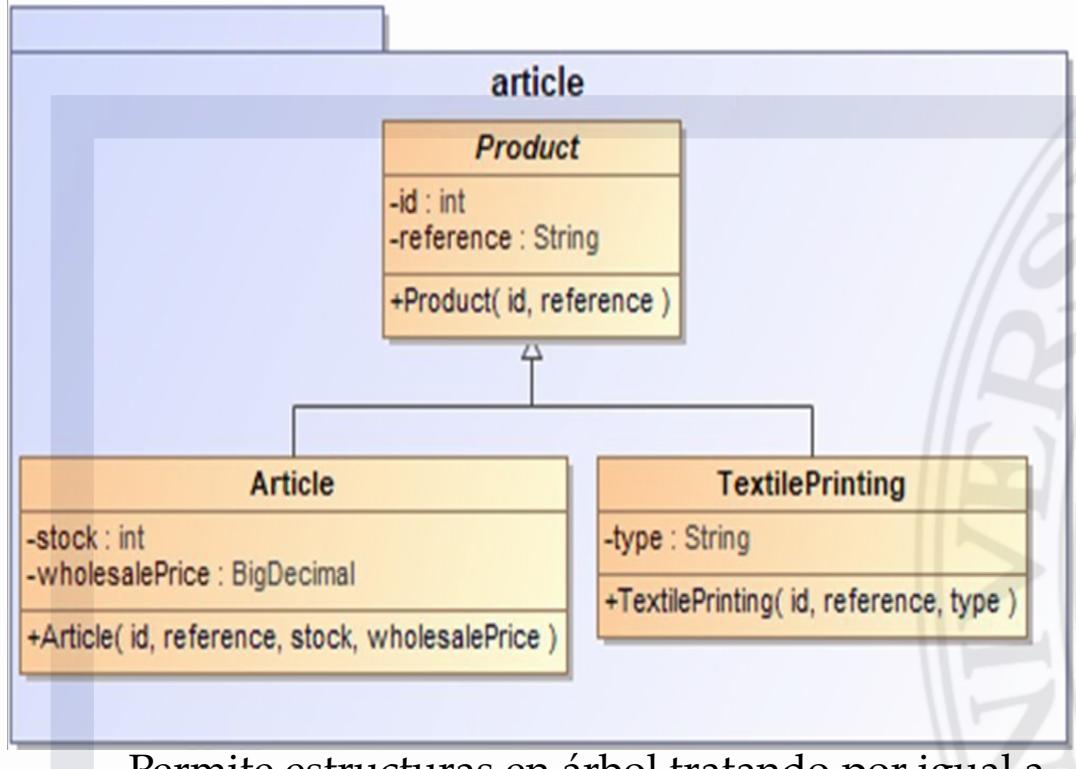
Expresión

- 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

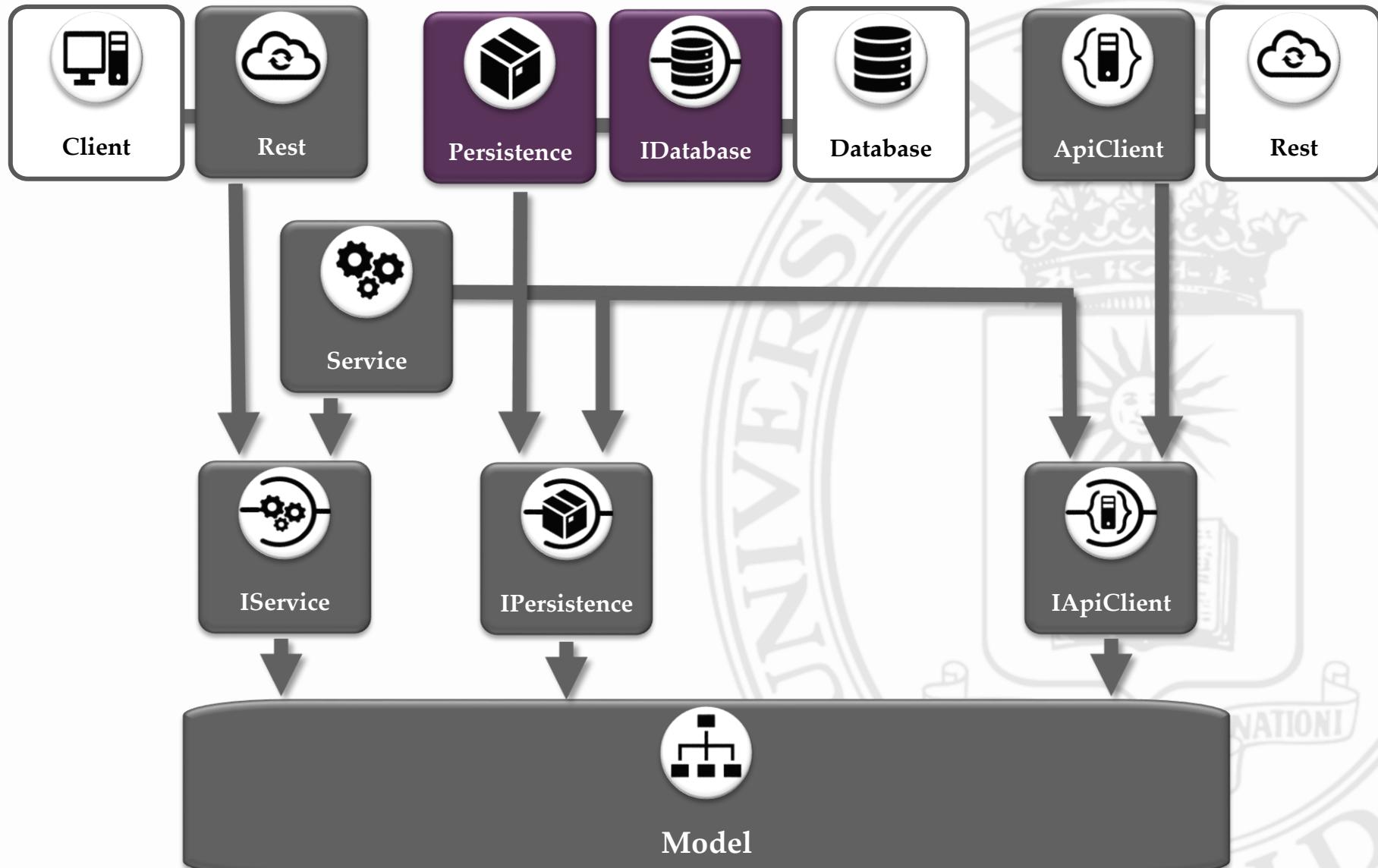
Se parte de un código en producción, y **no se quieren alterar las clases existentes.**

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*

Persistencia

Adaptador de Bases de Datos



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.

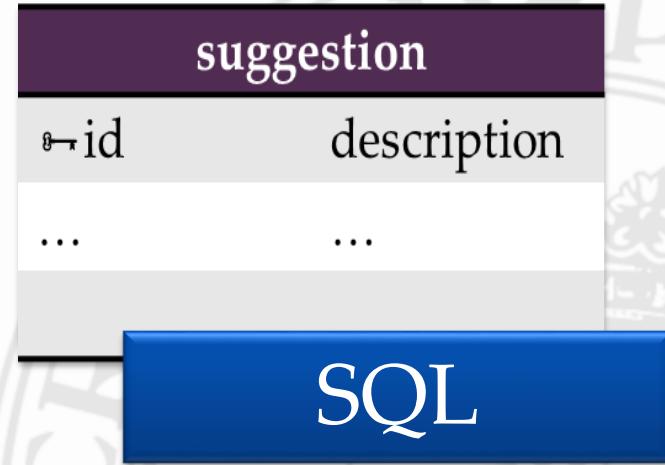
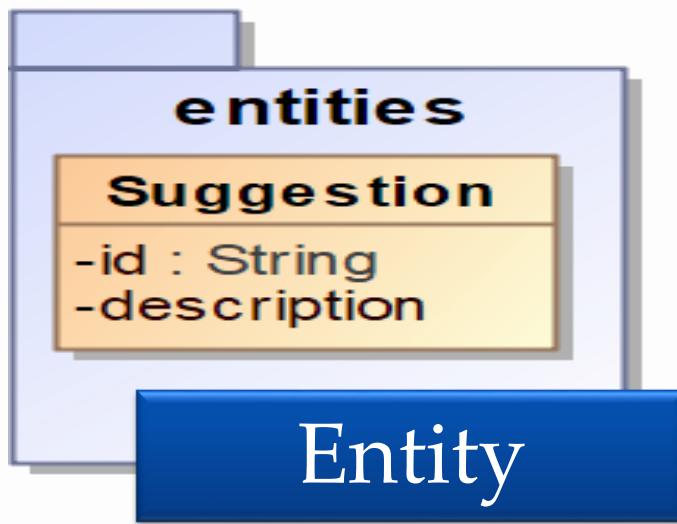
Mapeo

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

Dependencias

- No se debe depender del tipo de Bases de Datos

Mapeo de objetos: sin relación

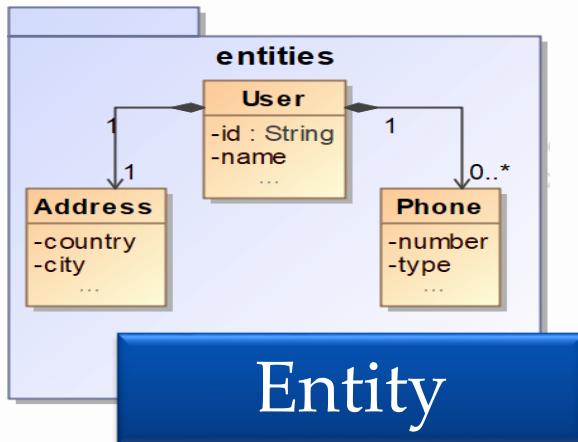


A screenshot of a code editor window titled "suggestion.json" displays the following JSON code:

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

Below the code editor is a large blue button labeled "MongoDB".

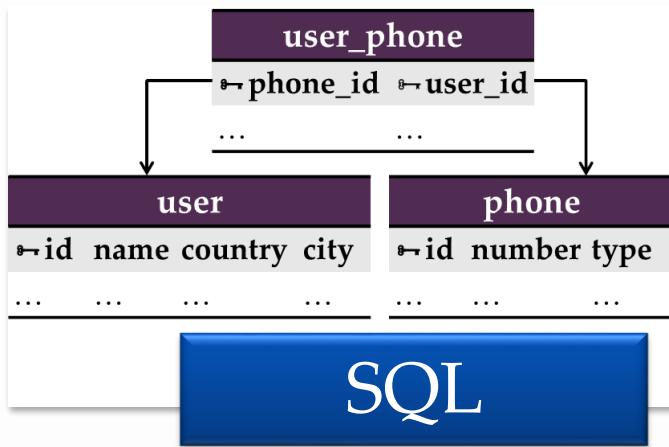
Mapeo de objetos: composición



Entity

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

SQL



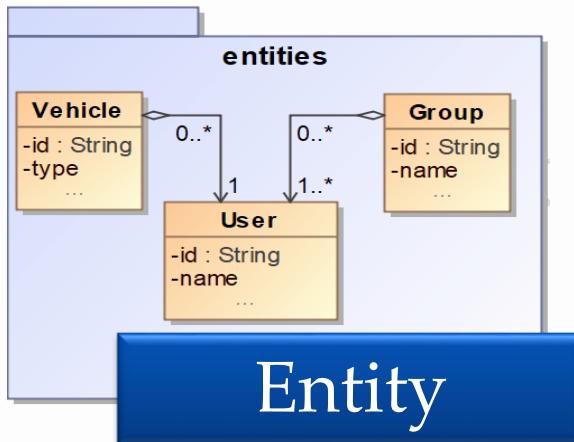
SQL

user.json

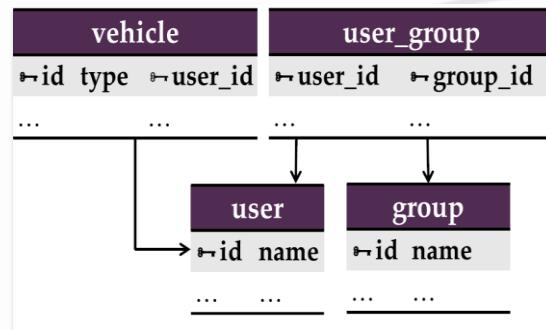
```
1 {  
2   "_id": "key-1",  
3   "name": "nombre",  
4   "address": {  
5     "country": "país", "city": "ciudad"  
6   },  
7   "phones": [  
8     { "number": "6666666666", "type": "móvil" },  
9     { "number": "999666999", "type": "casa" }  
10  ],  
11 },  
12 {  
13   "_id": "key-2",  
14   "address": {  
15     "cou  
16   }  
17 }  
18 ]  
19  
20  
21  
22
```

MongoDB

Mapeo de objetos: agregación



Entity



SQL

MongoDB JSON Document illustrating Entity aggregation:

```
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 }
```

The document represents a User entity with fields: _id, name, address, and phones. The address field contains country and city. The phones field is an array of objects with number and type.

MongoDB

MongoDB JSON Documents illustrating SQL aggregation:

```
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 }
```

The vehicle document represents a vehicle entity with fields: _id, type, and user (a DBRef to a user document). The group document represents a group entity with fields: _id, name, and users (an array of DBRefs to user documents).

MongoDB

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

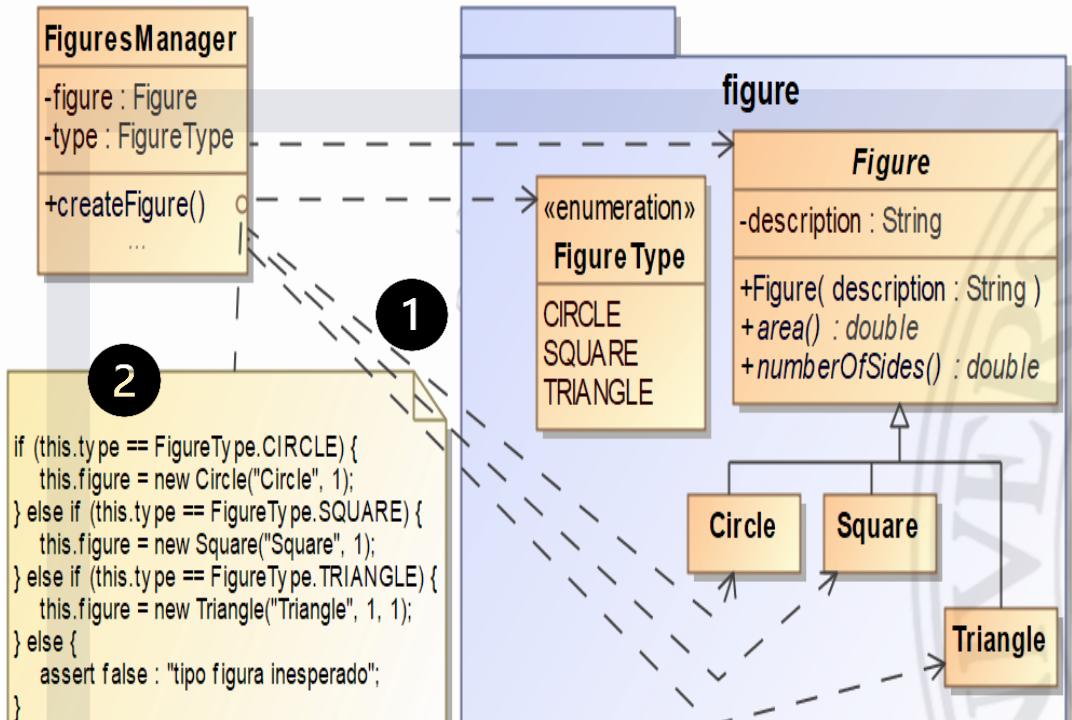
- Desacoplar el Dominio de la aplicación del Adaptador de las Bases de Datos.

Spring-Data

- Framework que implementa el patrón DAO

Factory Method

Motivación



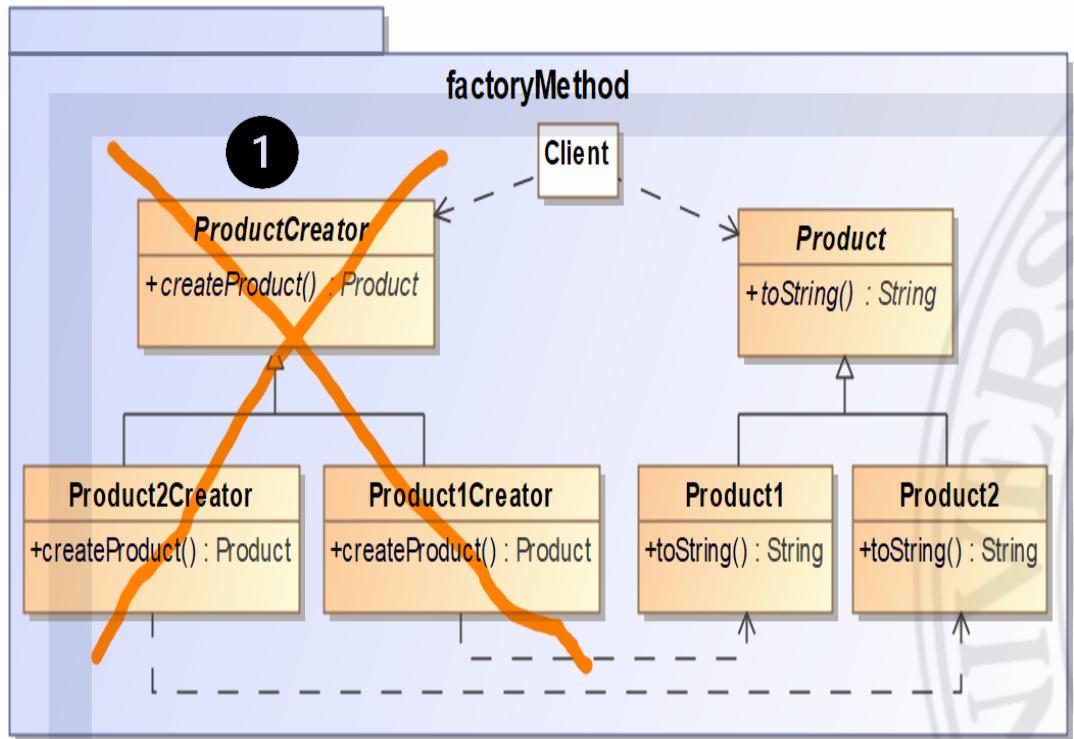
Factory Method

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

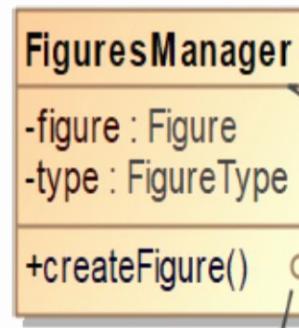
Propósito

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

① Función lambda

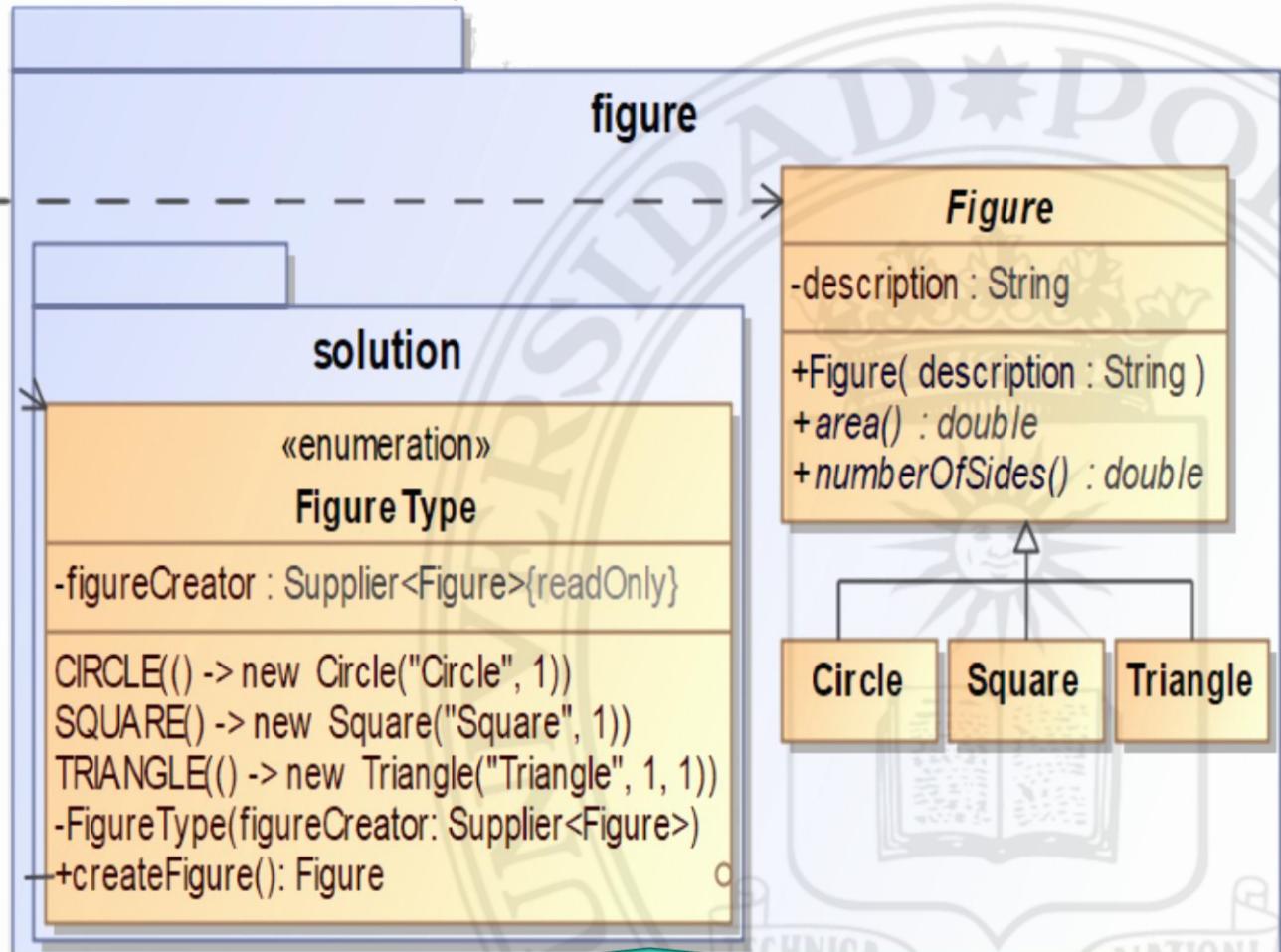
- `Supplier<T>`

Method Factory



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

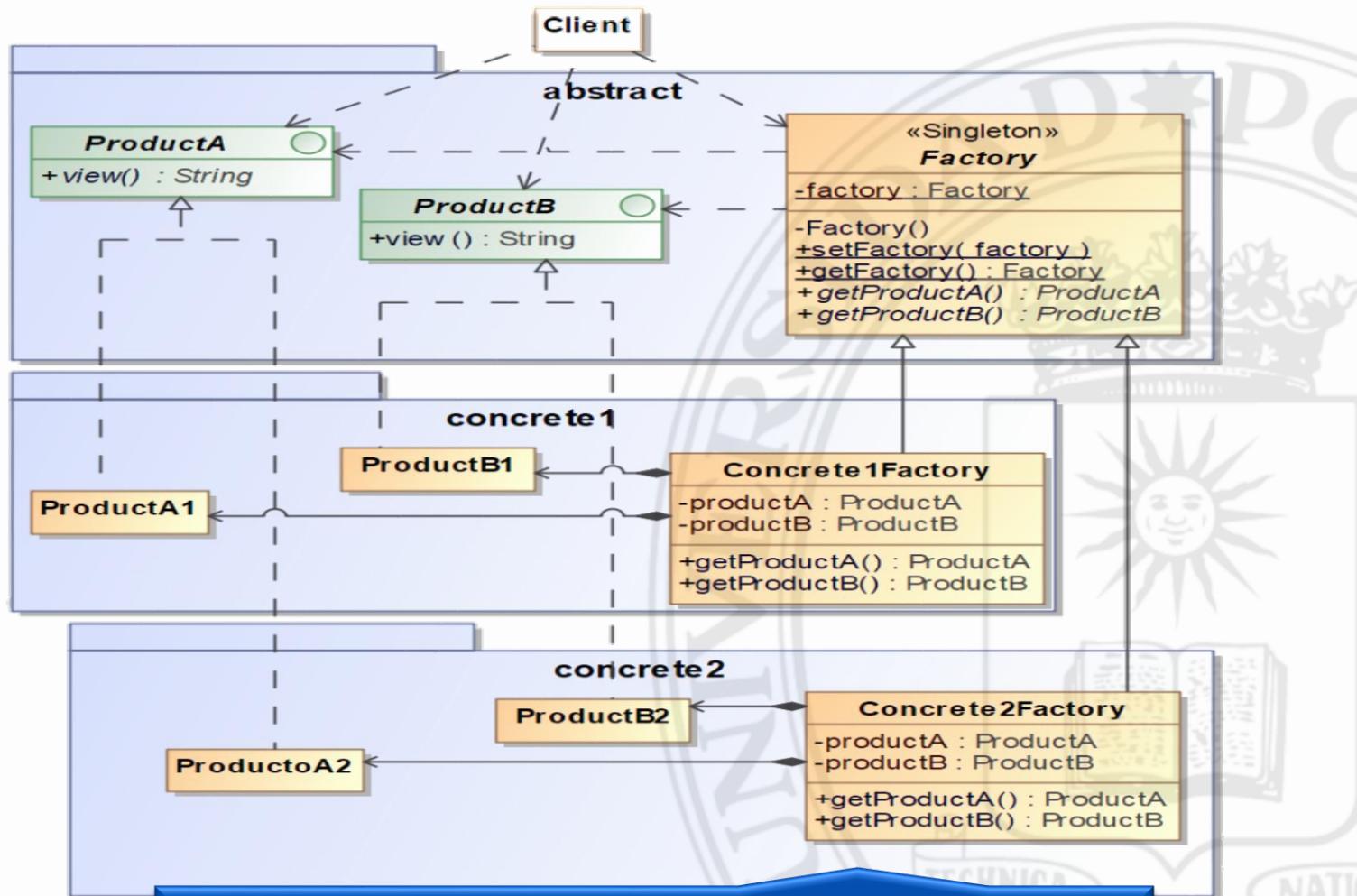
```
return this.figureCreator.get();
```



Solución mediante función Lambda

Abstract Factory

Propósito: Creación. Ámbito: objeto



Proporciona un interface para crear familias de objetos relacionadas

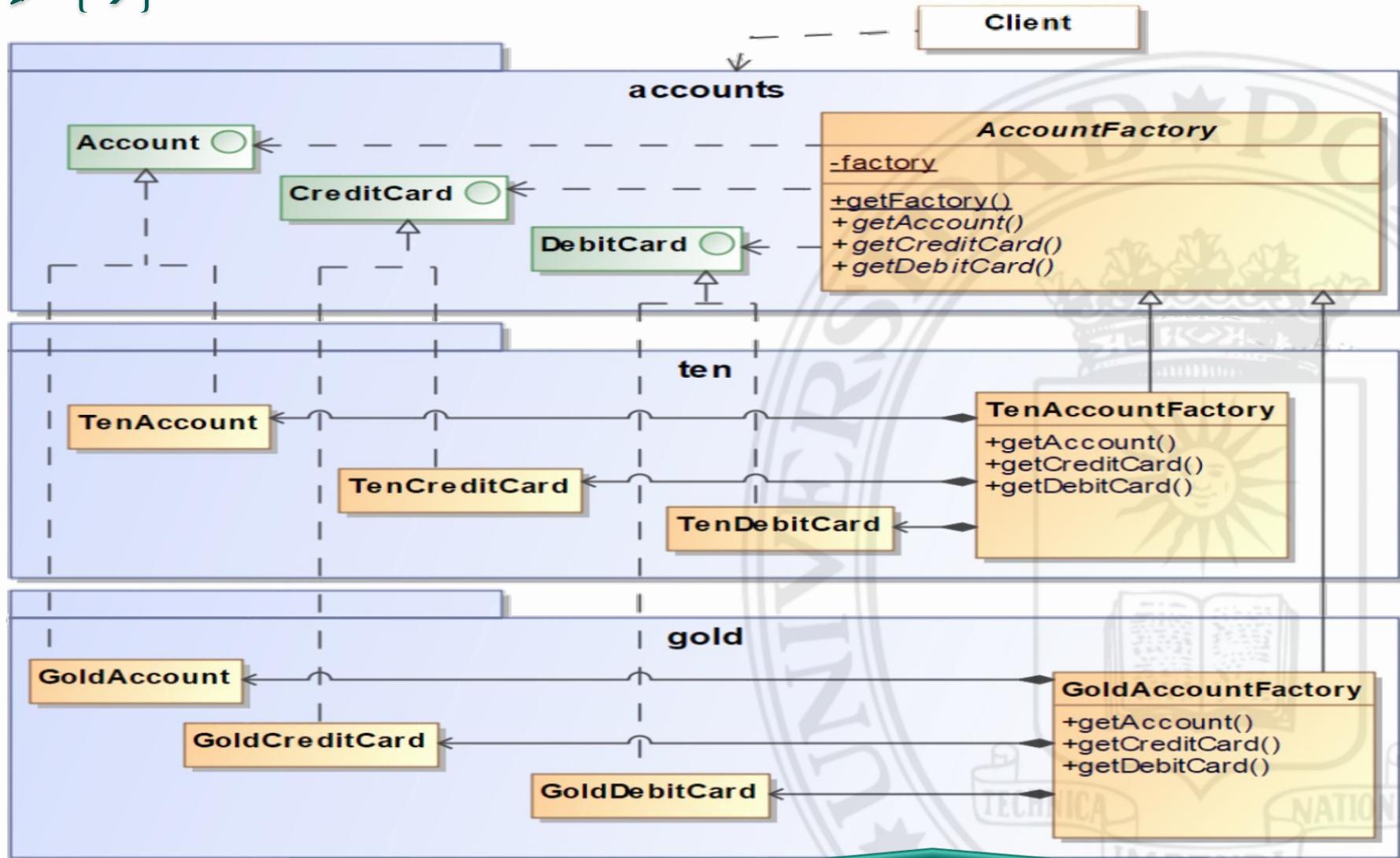
Abstract Factory



Tipo Cuenta	Cuenta	Tarjeta débito	Tarjeta crédito
Joven	1%	Gratuita	Gratuita Max.600
10	1'5%	Gratuita	10 € Max. 2000€
Oro	2%	5 €	20€ Max. 4000€

Aplicar el patrón *Abstract Factory*,
solo con nombres de clases

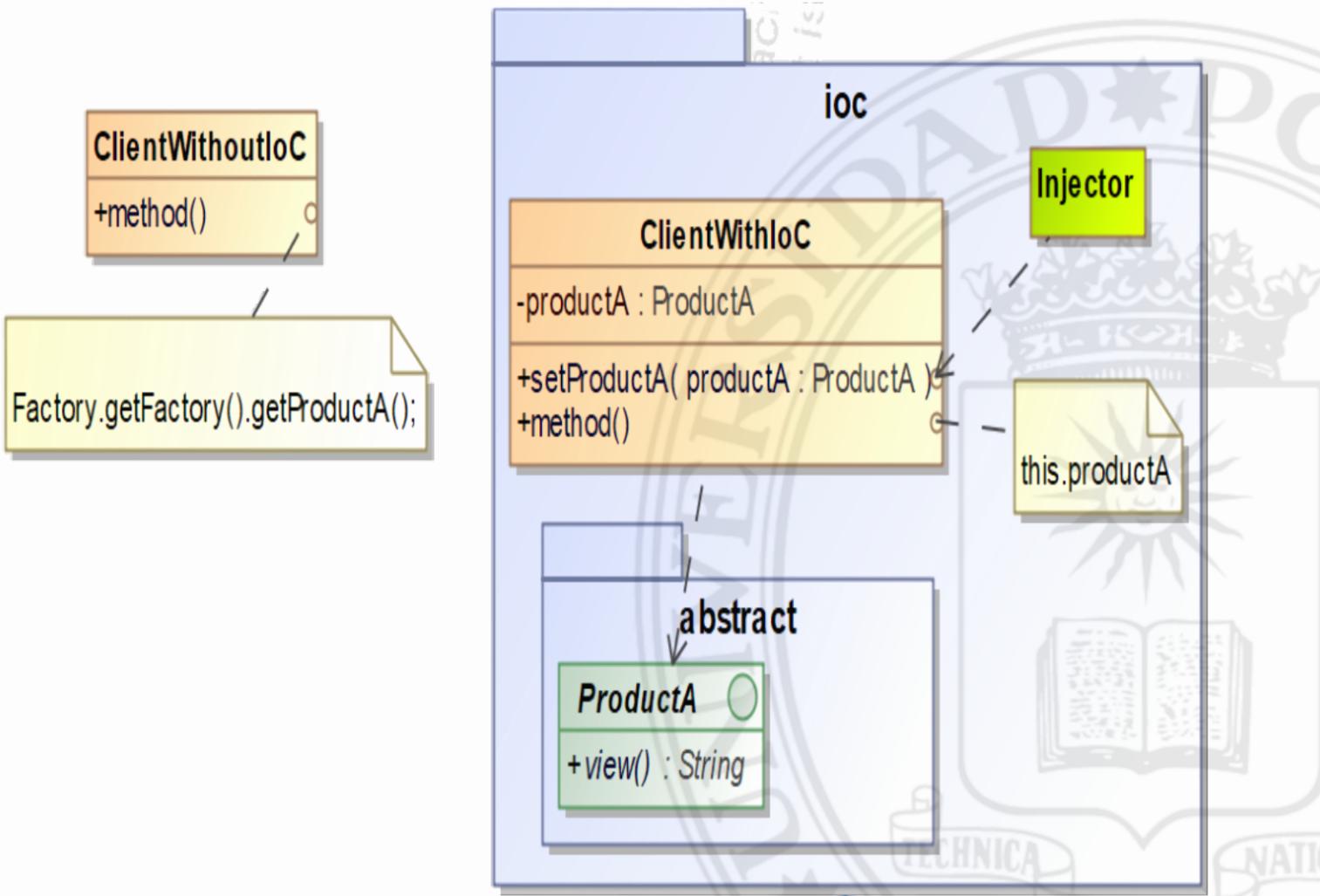
Abstract Factory



Solución

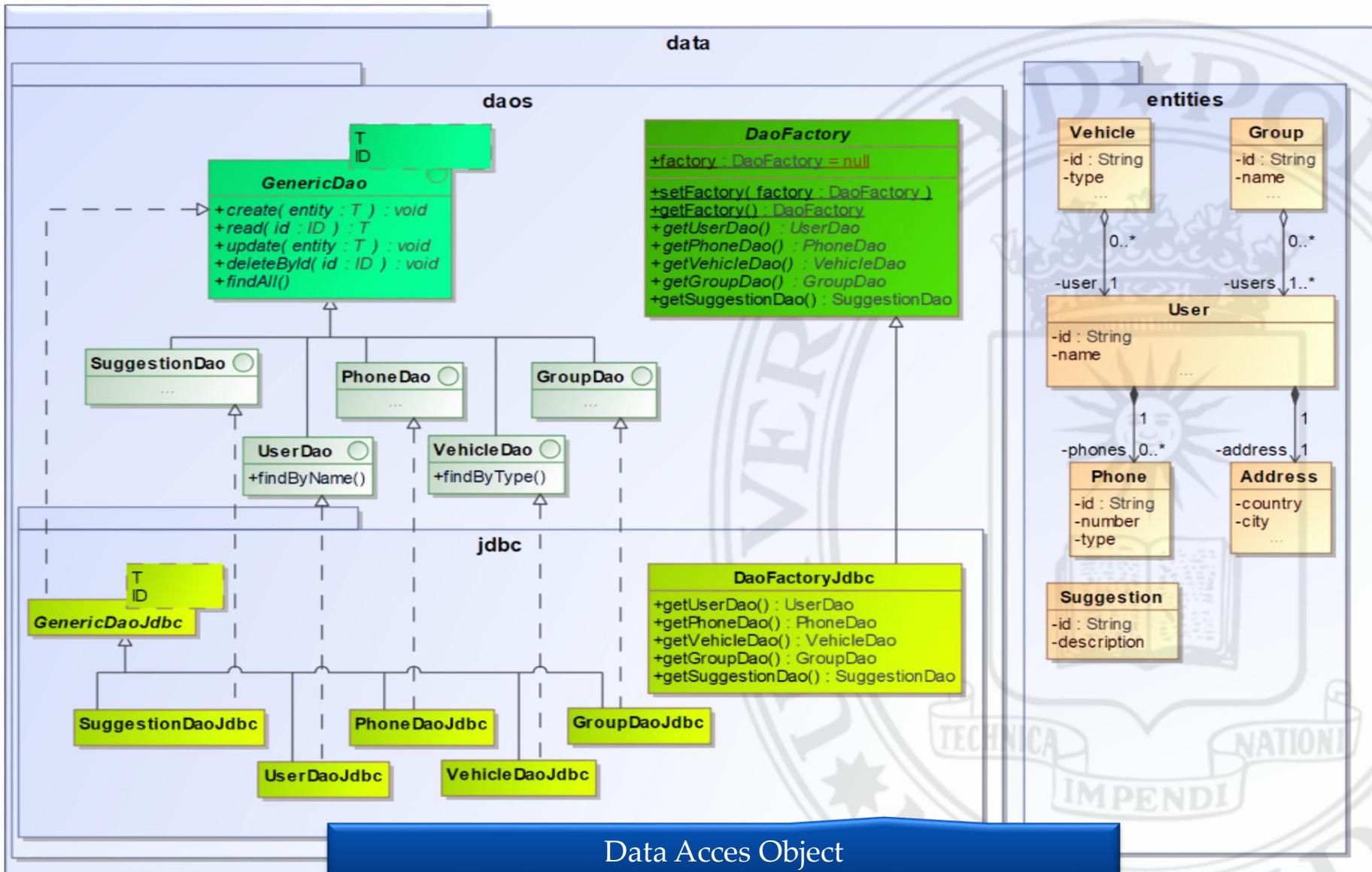
Inversion Of Control

Inversión de Control



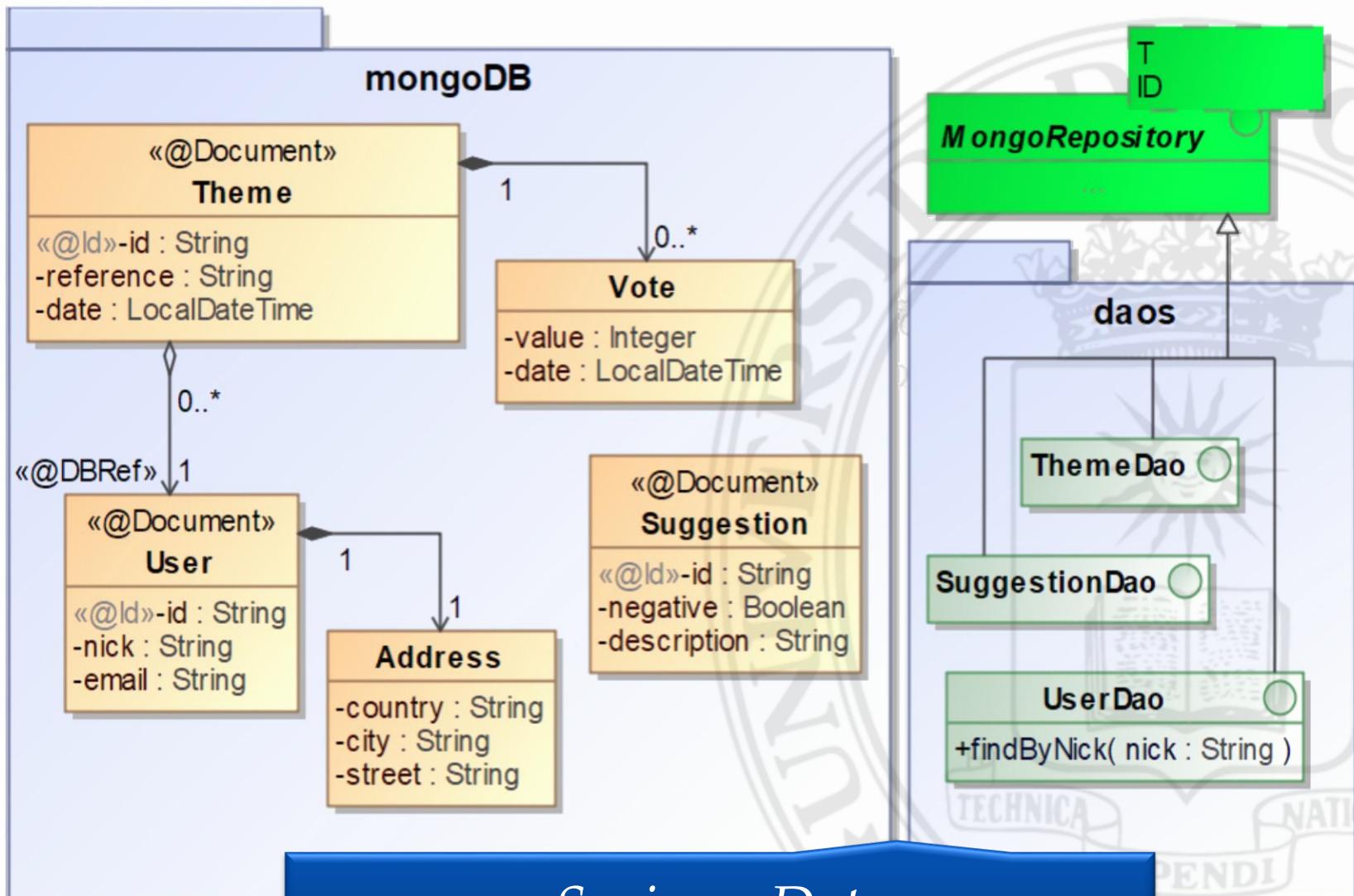
Mínima dependencia, quitar dependencia con *Factory*

Data Access Object DAO



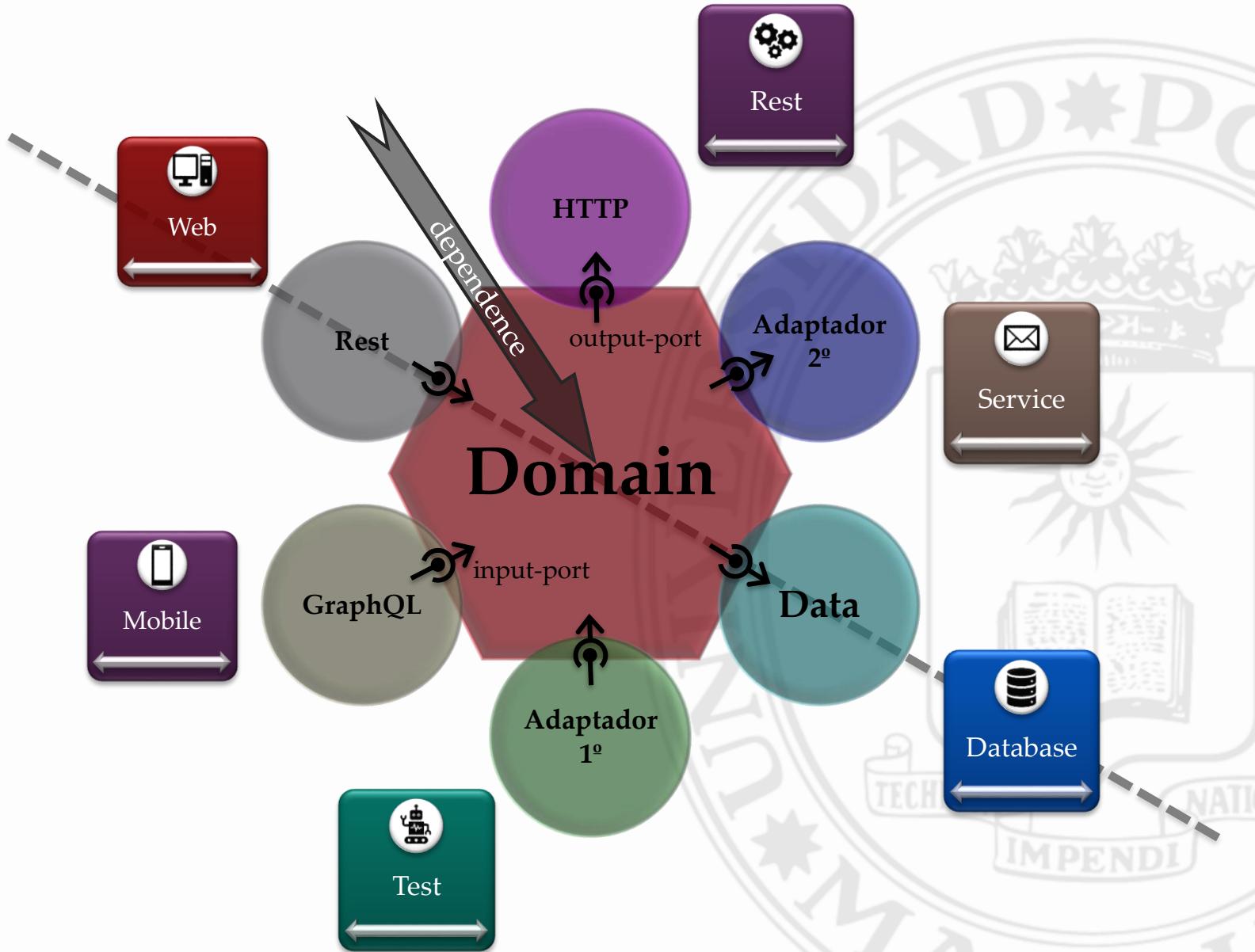
Data Acces Object

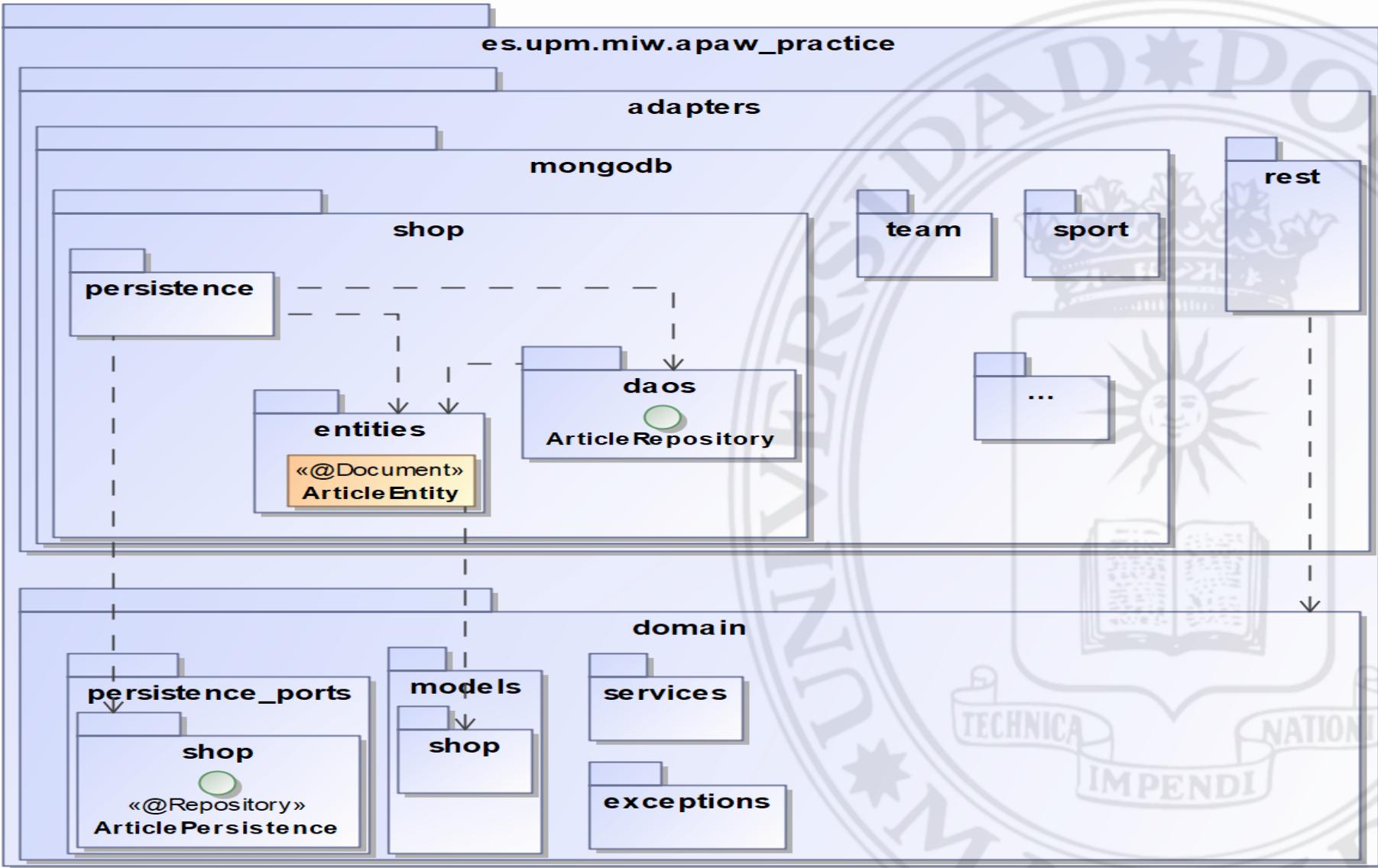
Spring - Data



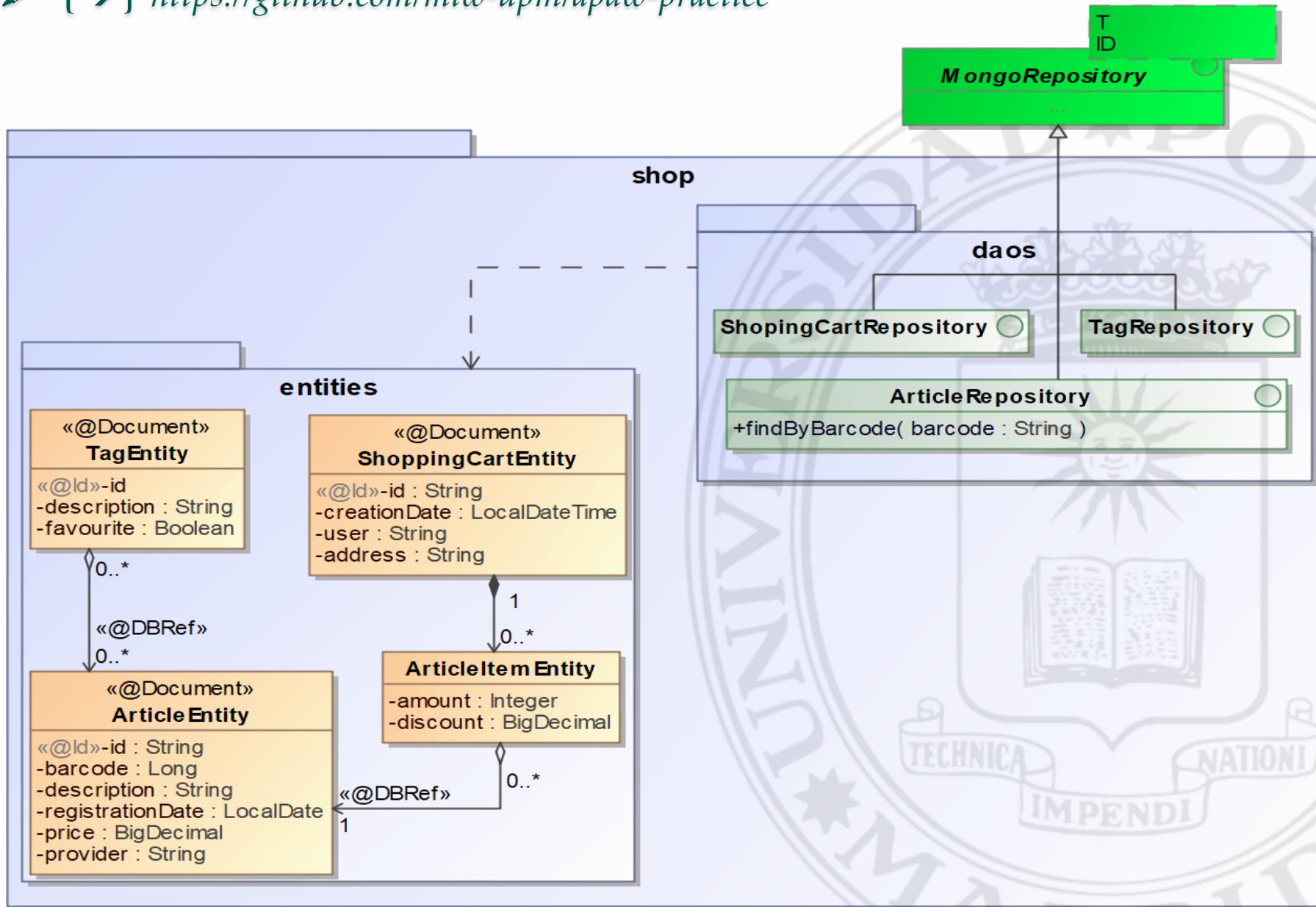
Spring - Data

Hexagonal Architecture Shop



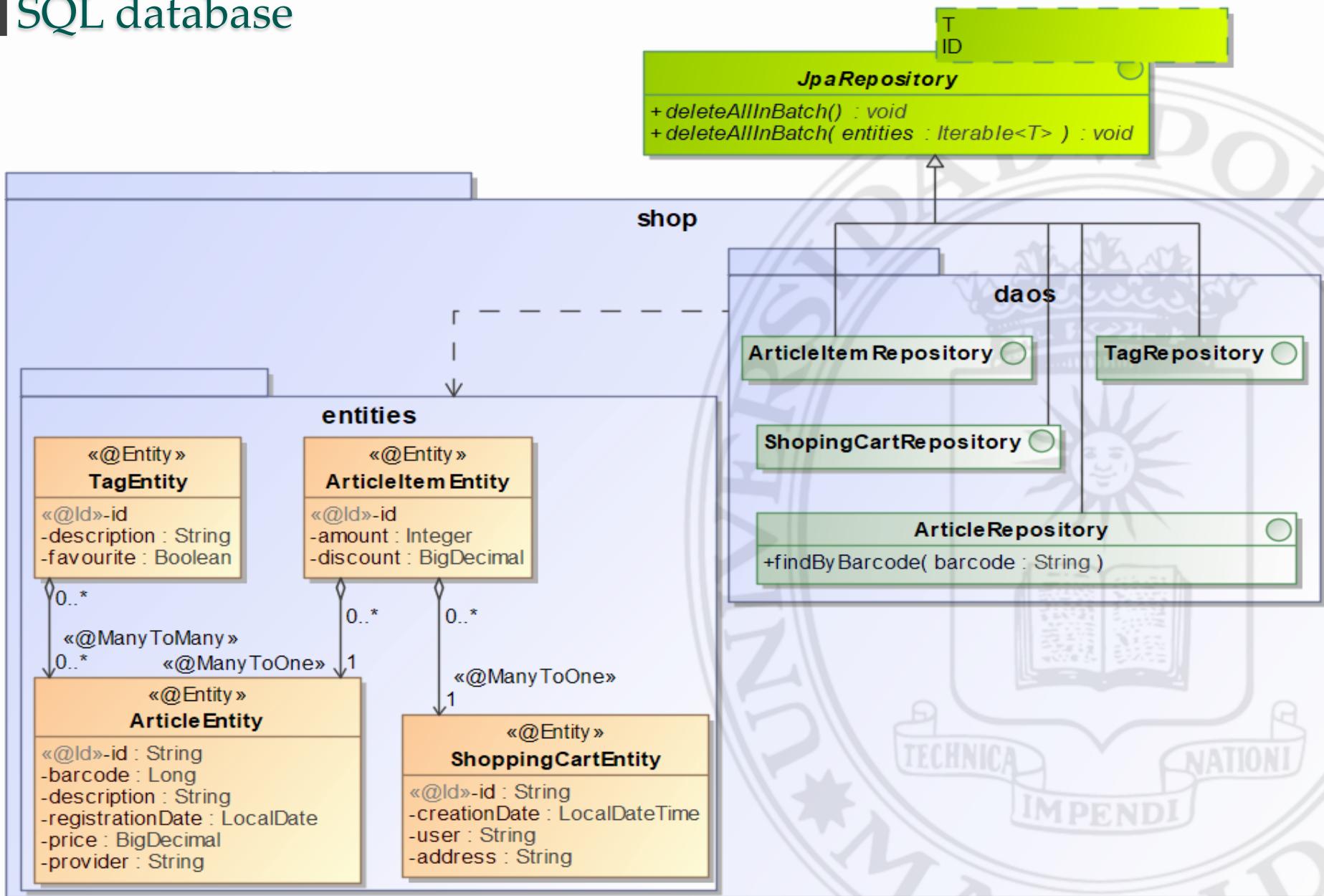


📝 {→} <https://github.com/miw-upm/apaw-practice>



Shop

SQL database



java.util.Optional

Ejemplo: code

```
public class OptionalDemo {  
    public Optional<String> readById(String id){  
        if("0".equals(id)){ return Optional.empty(); }  
        else { return Optional.of("value"); }  
    }  
    public String readByIdAssured(String id){  
        return this.readById(id).orElse("default");  
    }  
    public String readByIdAssuredWithException(String id){  
        return this.readById(id).orElseThrow(()->new RuntimeException("Not Found"));  
    } }
```

```
@Test void readByIdIsPresent() {  
    assertTrue(new OptionalDemo().readById("test").isPresent());  
    assertEquals("value", new OptionalDemo().readById("test").get());  
}  
@Test void readByIdIsEmpty() {  
    assertTrue(new OptionalDemo().readById("0").isEmpty());  
}  
@Test void readByIdAssured() {  
    assertEquals("default", new OptionalDemo().readByIdAssured("0"));  
}  
@Test void readByIdAssuredWithExceptionThrow() {  
    assertThrows(RuntimeException.class, () -> new OptionalDemo().readByIdAssuredWithException("0"));  
}  
@Test void readByIdAssuredWithExceptionNoThrow() {  
    assertEquals("value", new OptionalDemo().readByIdAssuredWithException("test"));  
}
```

Data Acces Object

Spring - Test

IntegrationTest: IT

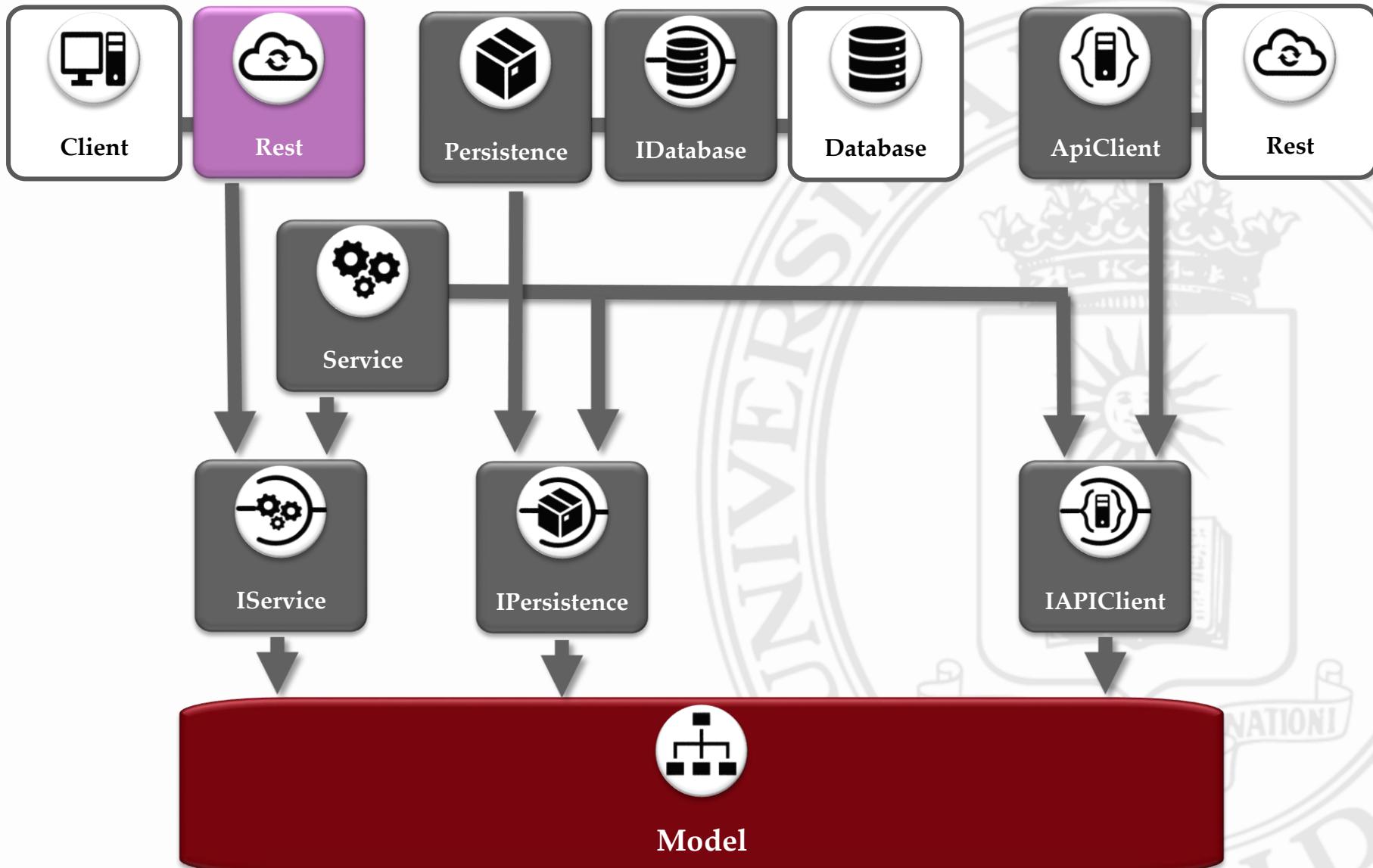
Spring-Test
JUnit5

SeedDatabaseService

seedDatabase()
deleteAll()

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

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~~

NO referenciar formatos /v0/orders/~~pdf~~

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

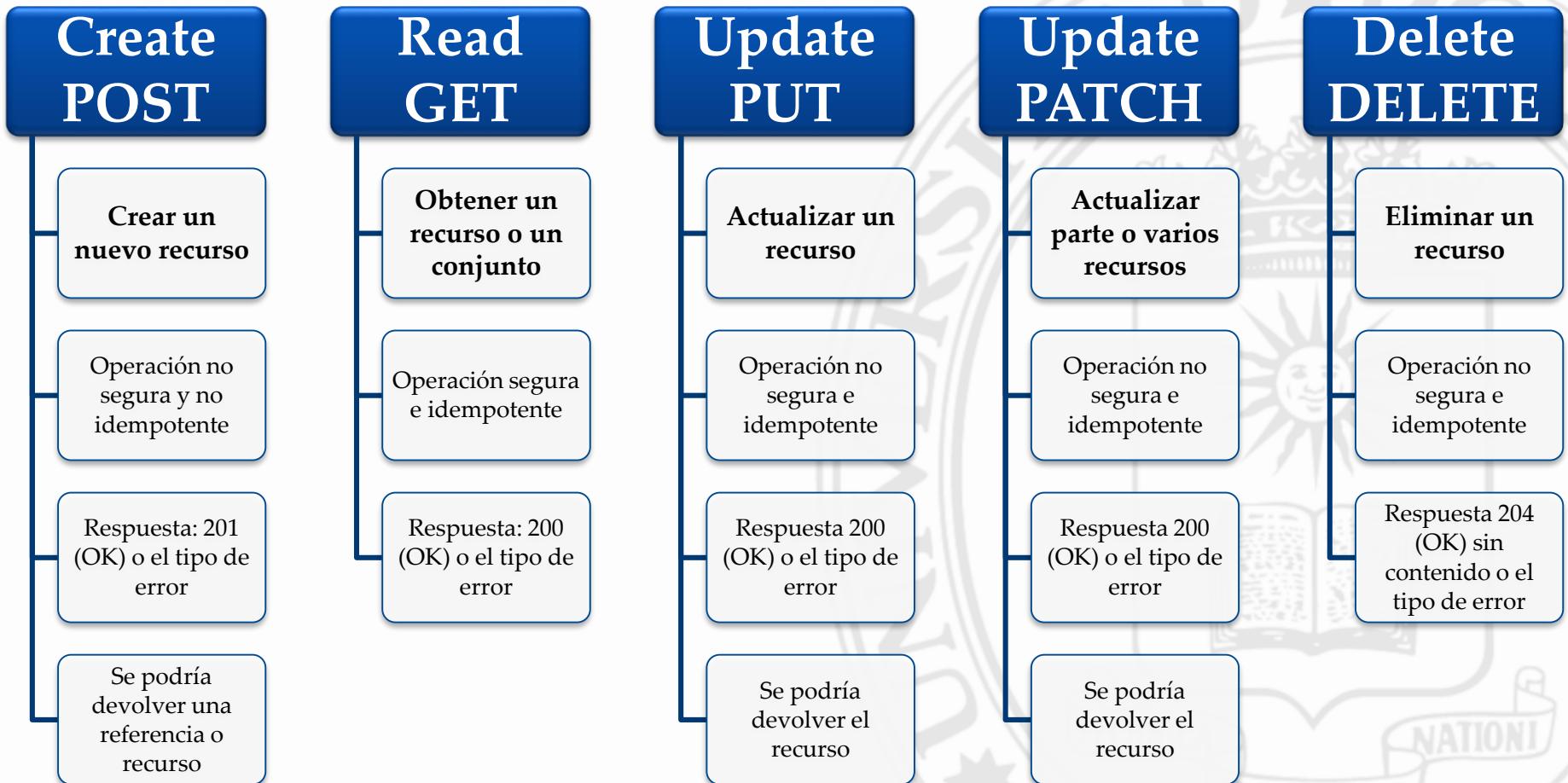
Paginación: ...?page=1&size=30

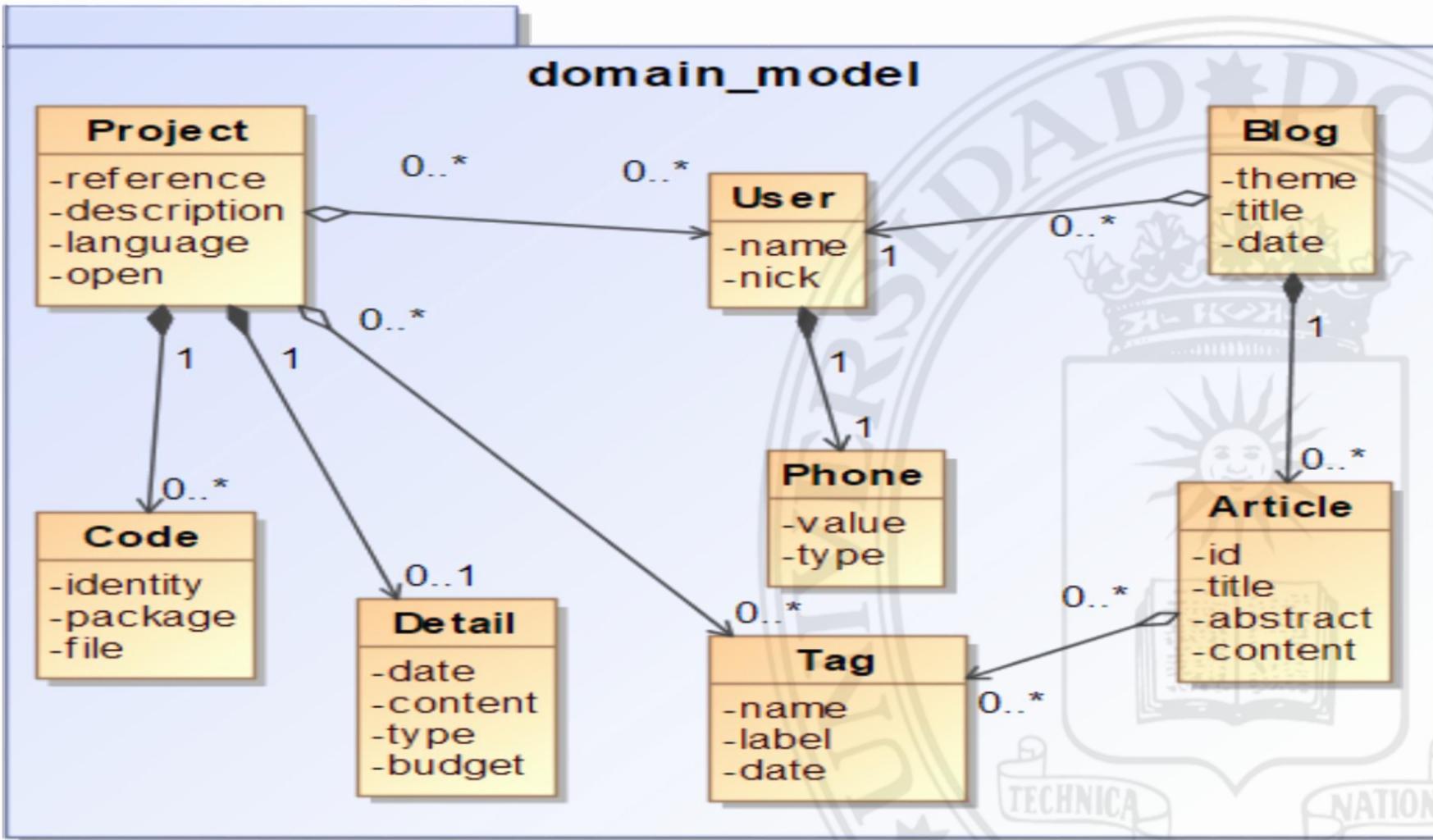
Filtros: ...?type=3,5

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

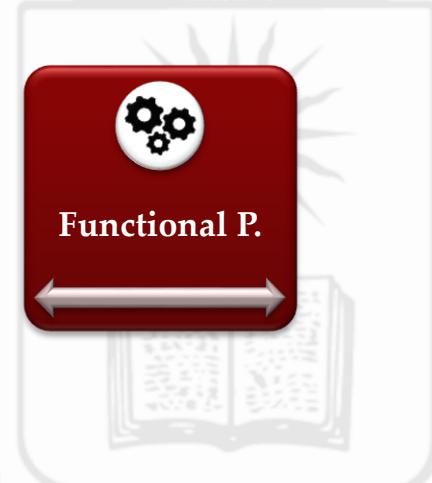
Responsabilidades

Define el *path* y el
método del recurso

Gestionar la validación de
los *dtos*

Organizar la recepción de
datos de la petición

Delega en el *Servicio* la
ejecución de la petición



Rest Spring

rest_adapter

```
«@RestController»  
«@RequestMapping("/users")»  
UserResource
```

-userService : UserService

```
«@Autowired»+User( userService : UserService )  
«@PostMapping»+create( @RequestBody userCreation : UserCreation )  
«@GetMapping»+readAll() : List<User>  
«@GetMapping("/{id}")»+read( @PathVariable id : String ) : User  
«@PutMapping("/{id}")»+update( @PathVariable id : String, @RequestBody user : User )  
«@PutMapping("/{id}/name")»+updateName( @PathVariable id : String, @RequestBody user : User )  
«@PatchMapping("/{id}")»+updateAll( @PathVariable id : String, @RequestBody updateUsers : UpdateUsers )  
«@DeleteMapping("/{id}")»+delete( @PathVariable id : String )  
«@GetMapping("/search")»+find( @RequestParam q : String ) : List<User>  
«@GetMapping("/{id}/name")»+readName( @PathVariable id : String ) : User
```

GET /users/666/name

GET /users/search?q=name:j&order=asc

Rest

Spring - Test

Spring-Test: WebTestClient

URI & params?

Method: *post, get...*

Body?

Exchange

expect*

- expectStatus, expectBody...

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

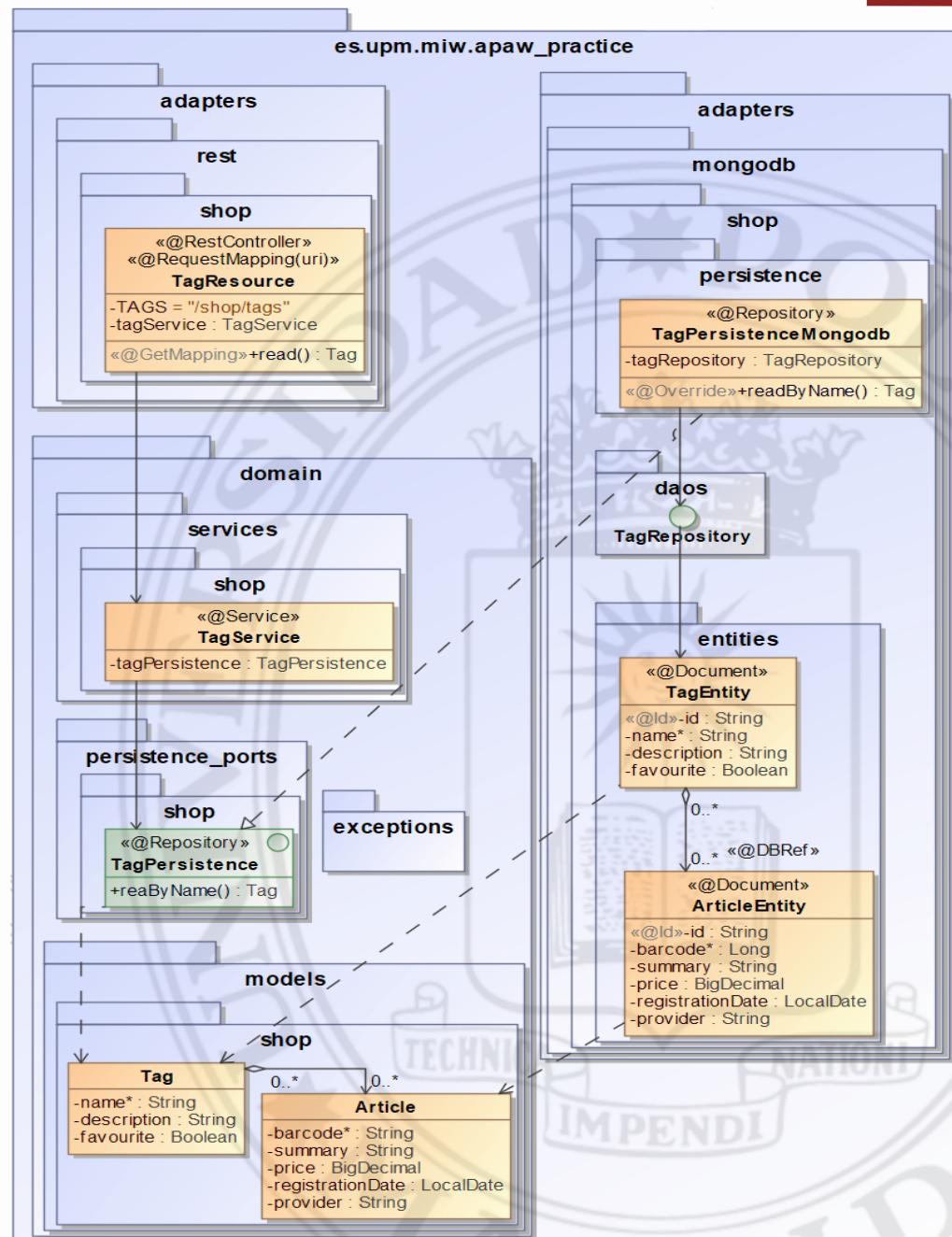
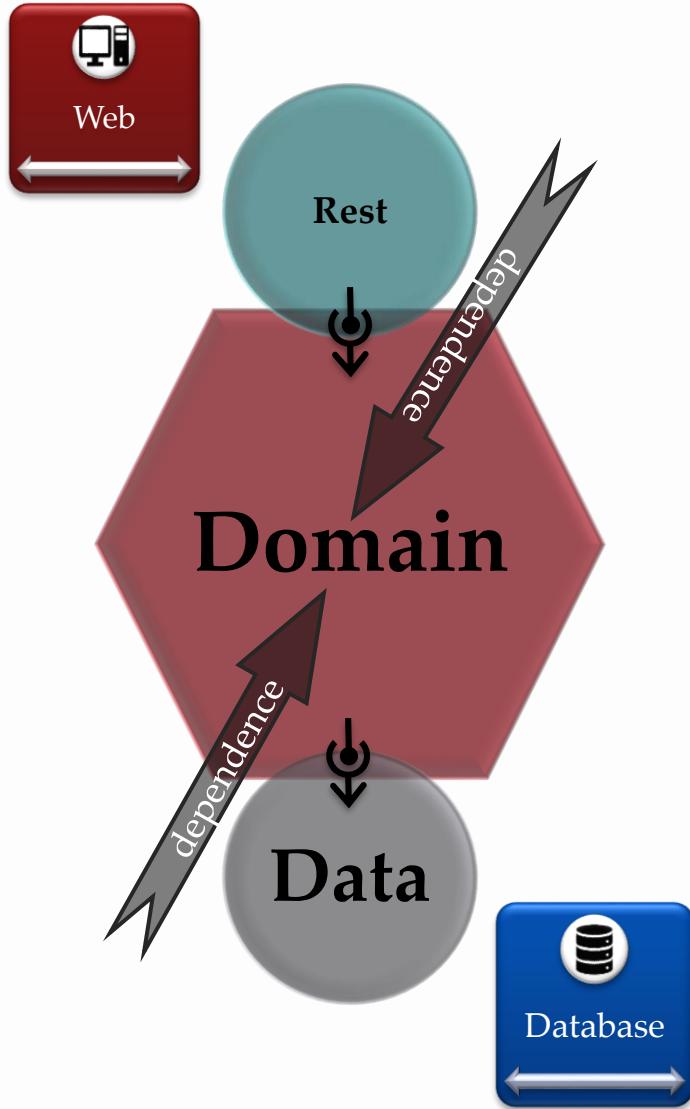
- **main** es.upm.miw.apaw_practice.adapters.rest.shop
- **test** es.upm.miw.apaw_practice.adapters.rest.shop

End-point en práctica

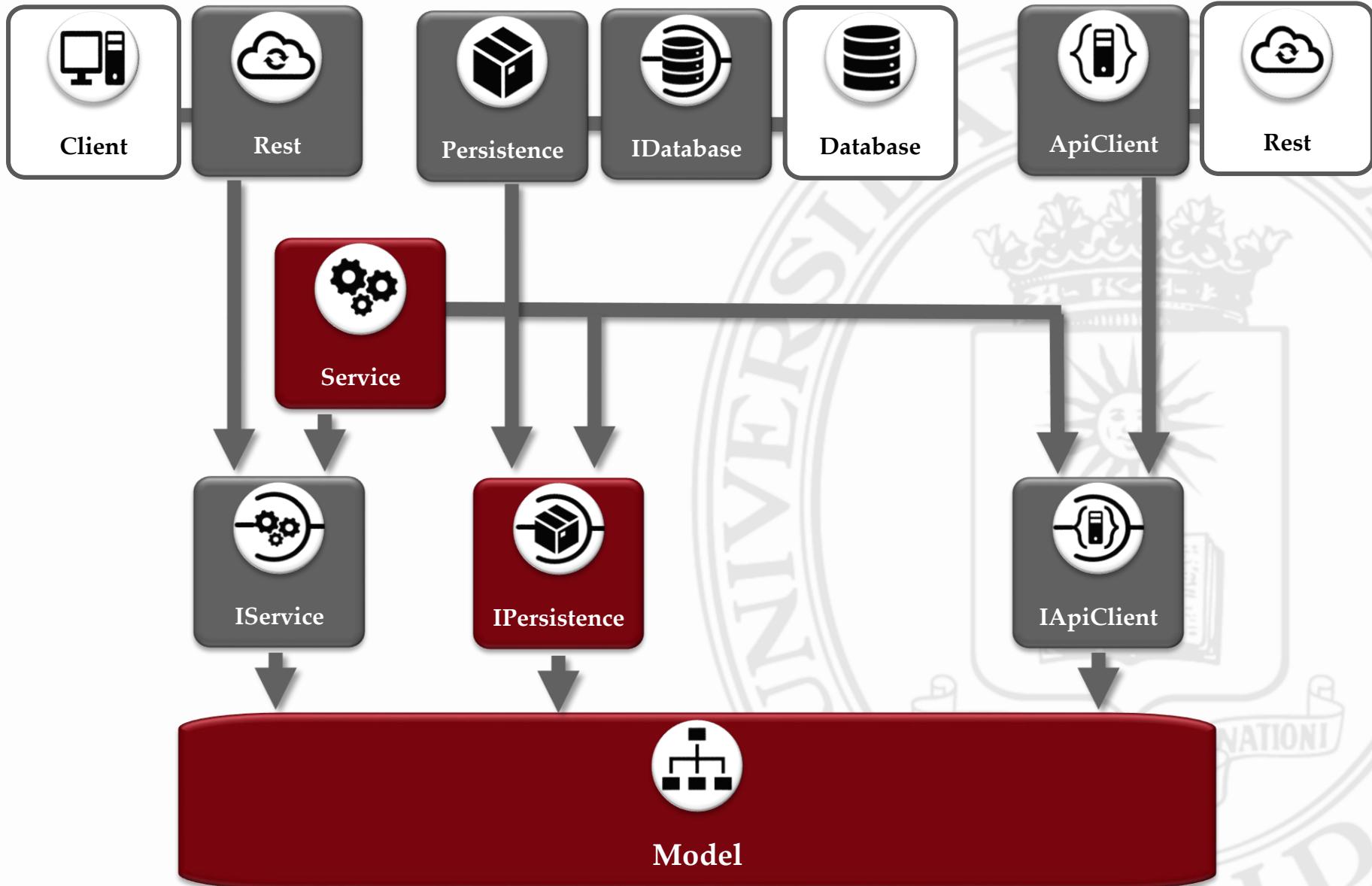
- Clases, métodos y modelos bajo demanda, respetando la arquitectura, solo se programa lo estricto necesario para resolver la petición.
- EXCEPTIONS
- GET **/shop/tags/{name}
- POST **/shop/articles
- DELETE **/shop/tags/{name}
- PUT **/shop/shopping-carts/{id}/article-items
- PATCH **/shop/articles {**}

Shop

📝 {→}



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

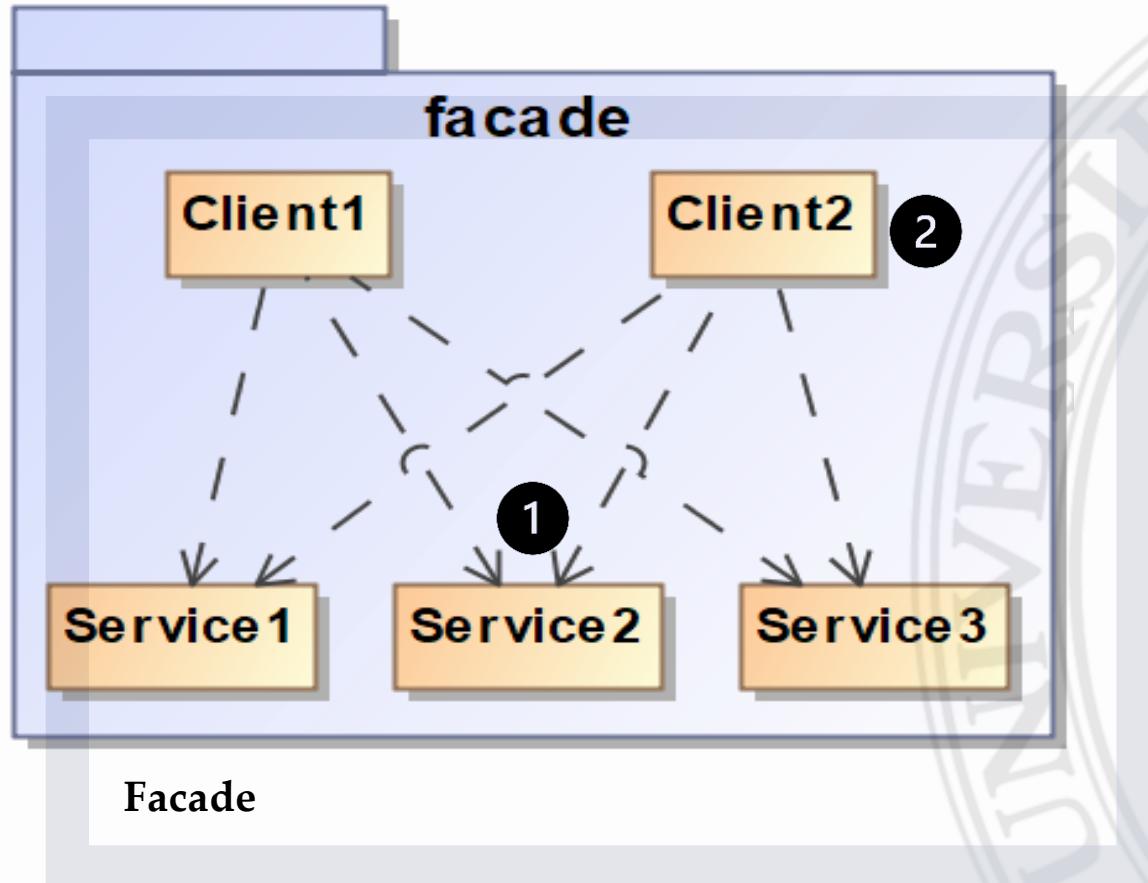
- Se define una dependencia entre uno a muchos, del tal manera, que cuando cambie el sujeto avise a todos los objetos dependientes.

Publisher: FP

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

Facade

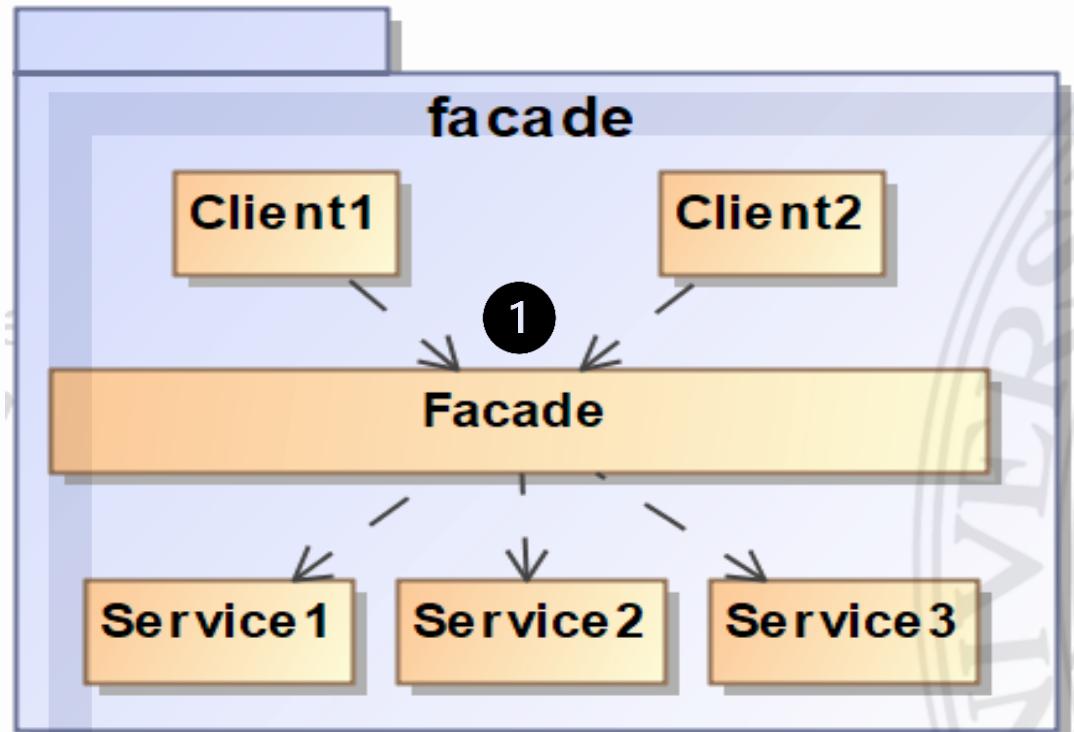
Propósito: Estructural. Ámbito: objeto



- 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



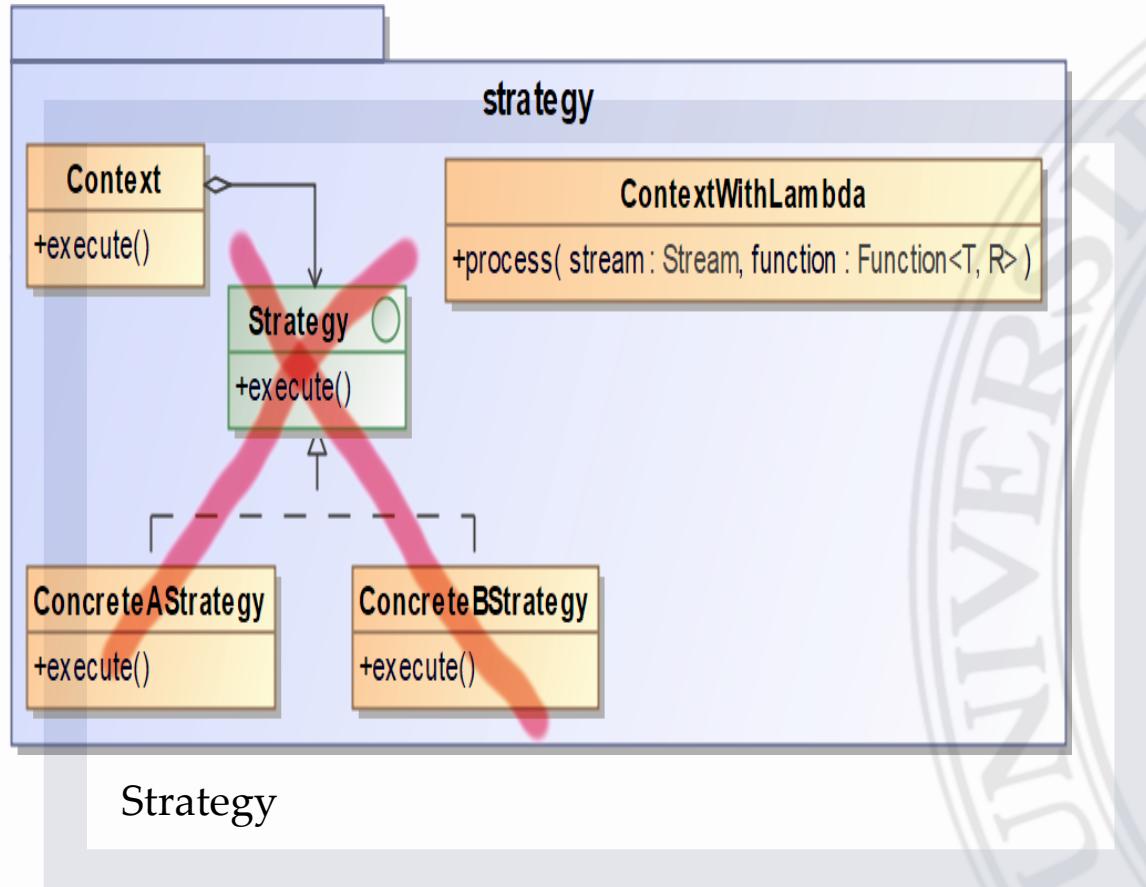
Proporciona un interface unificado para un conjunto de interfaces de un subsistema

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

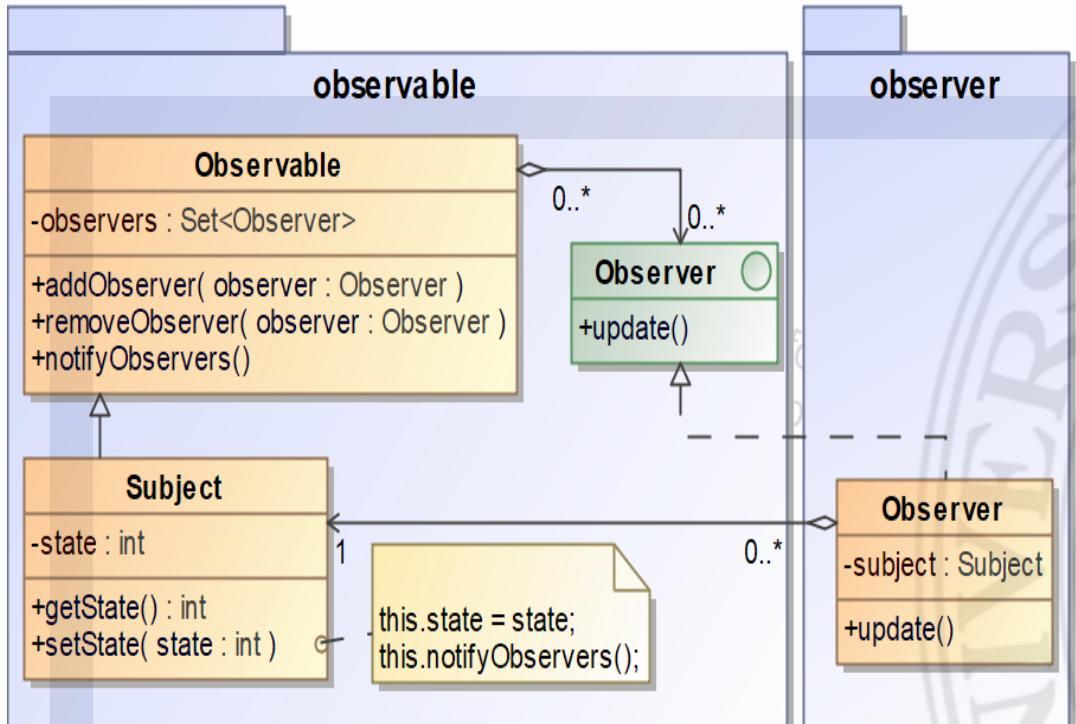


Propósito

- Define un conjunto de algoritmo haciéndolos intercambiables dinámicamente

Observer

Propósito: Comportamiento. Ámbito: objeto



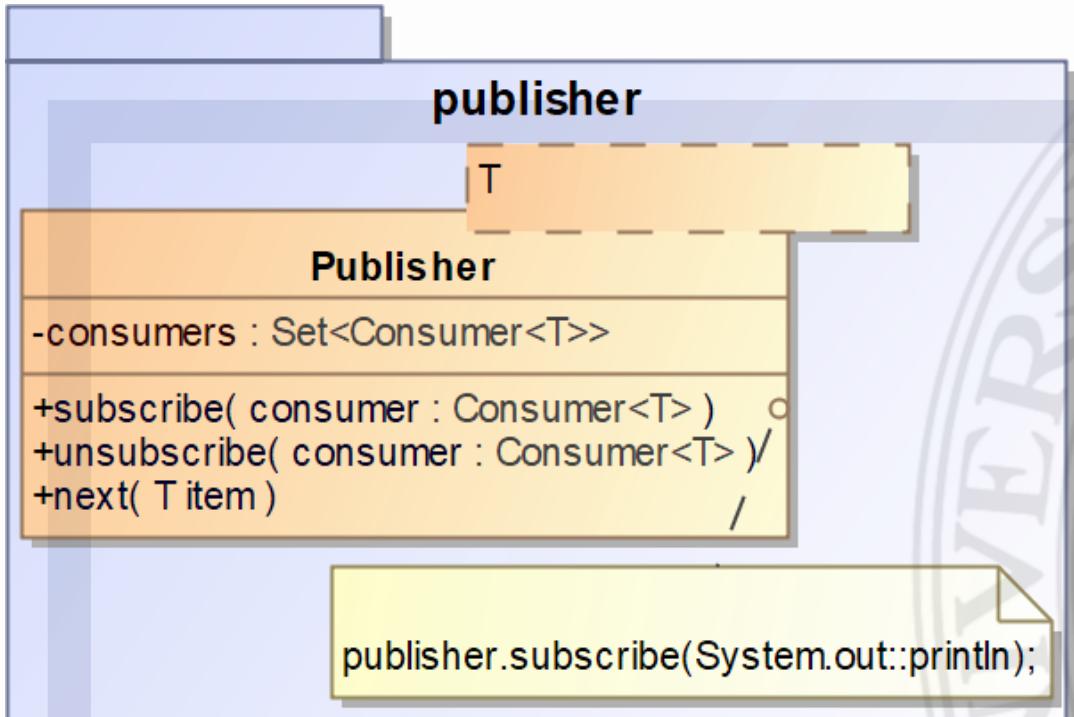
Observer: OOP

Propósito

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

Publisher

Propósito: Comportamiento. Ámbito: objeto



Publisher: FP

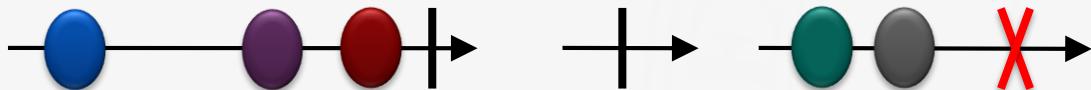
Propósito

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

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



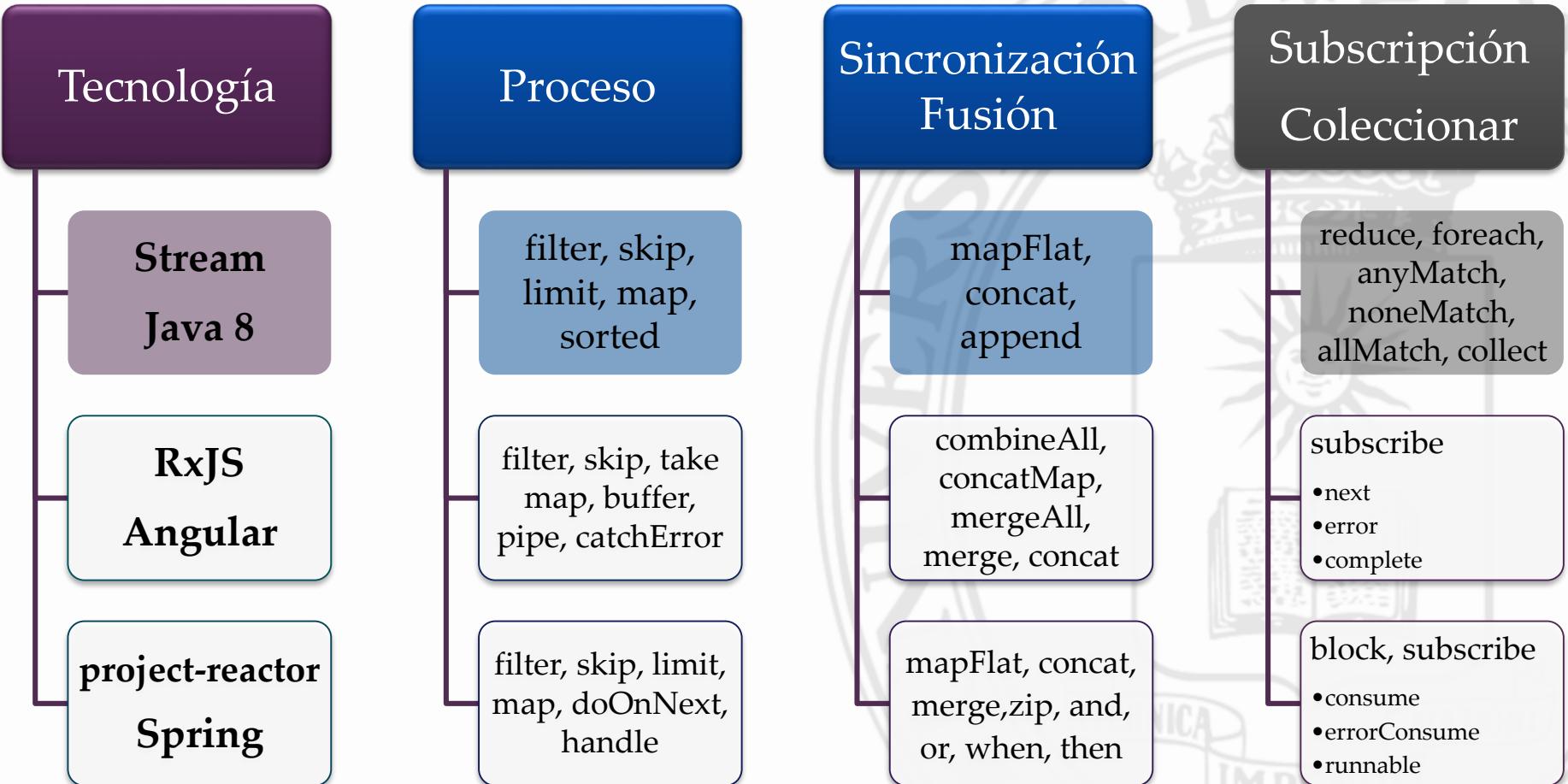
Project
Reactor

Project-reactor - Java

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

Programación Reactiva

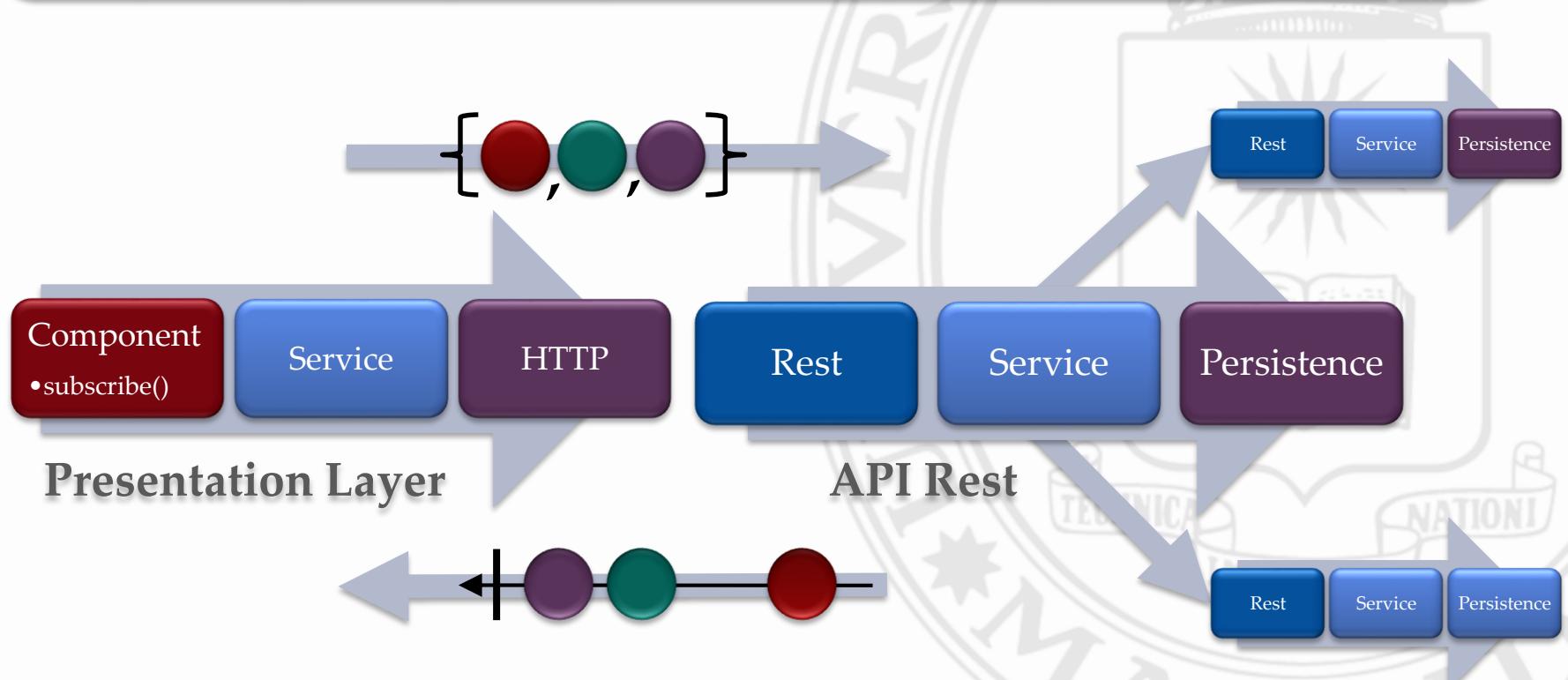
Proyectos



Programación Reactiva

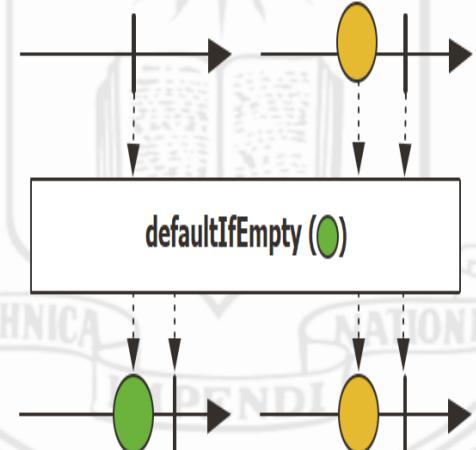
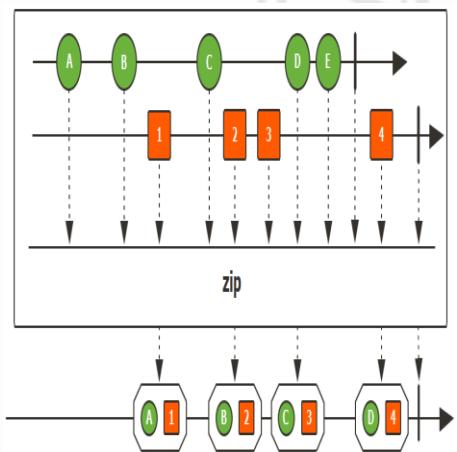
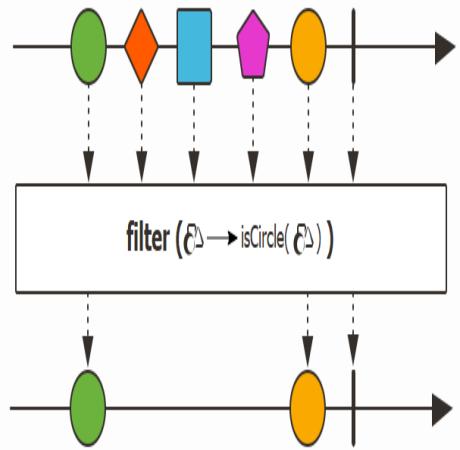
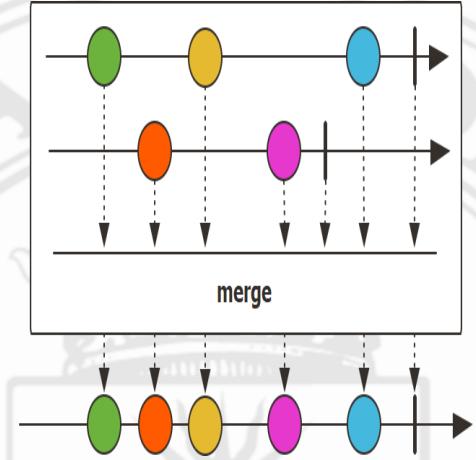
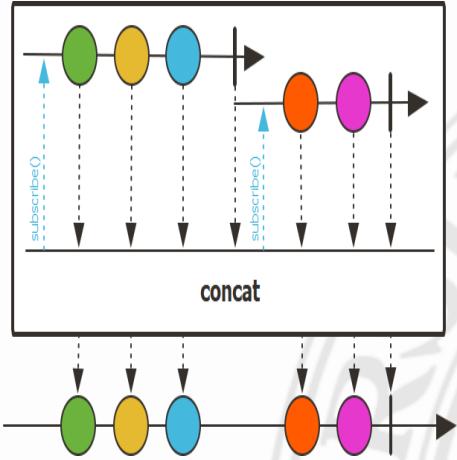
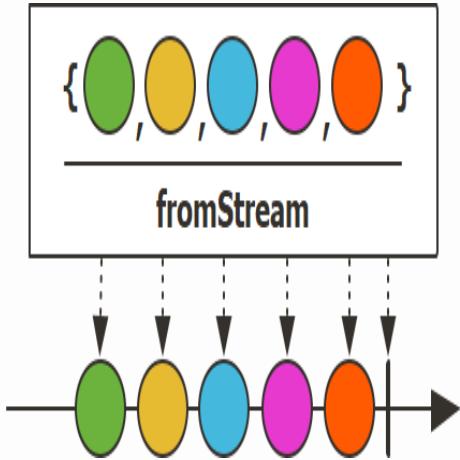
Flujo

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



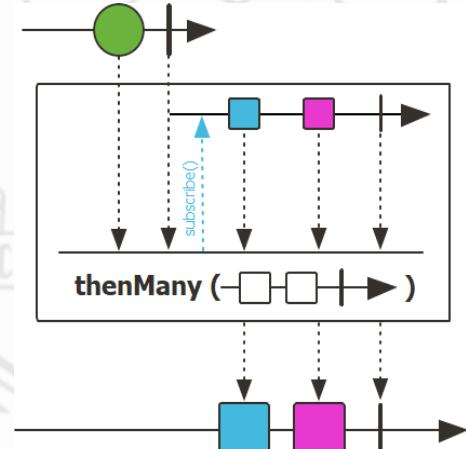
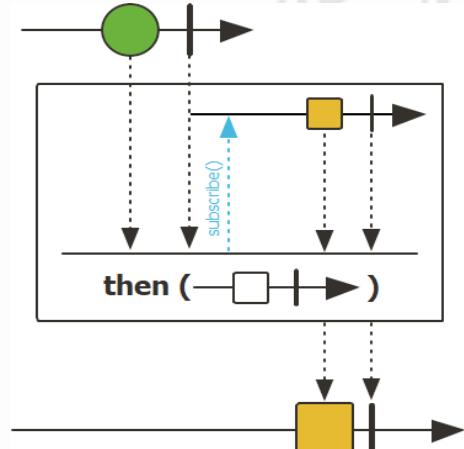
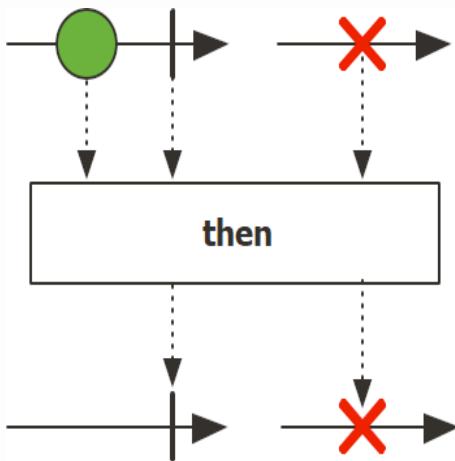
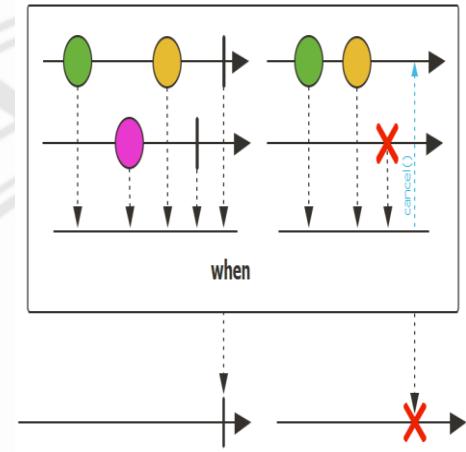
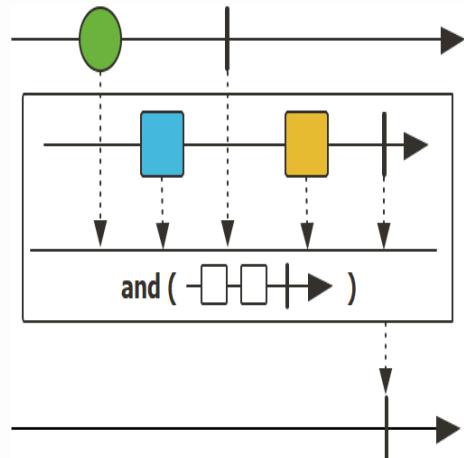
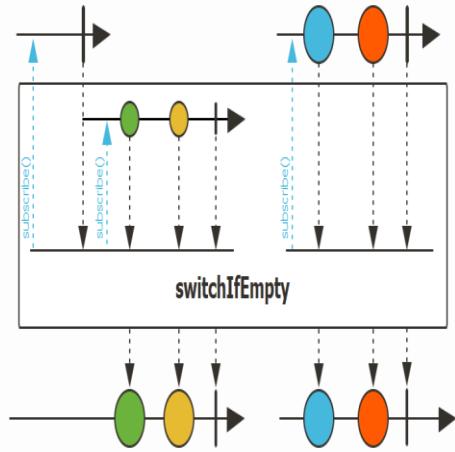
Programación Reactiva

Project Reactor. Funciones



Programación Reactiva

Project Reactor. Funciones

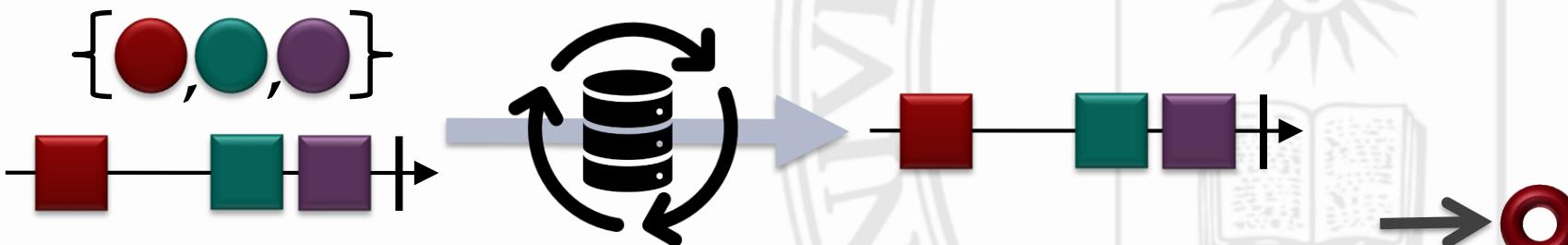


Programación Reactiva

Project Reactor. BD



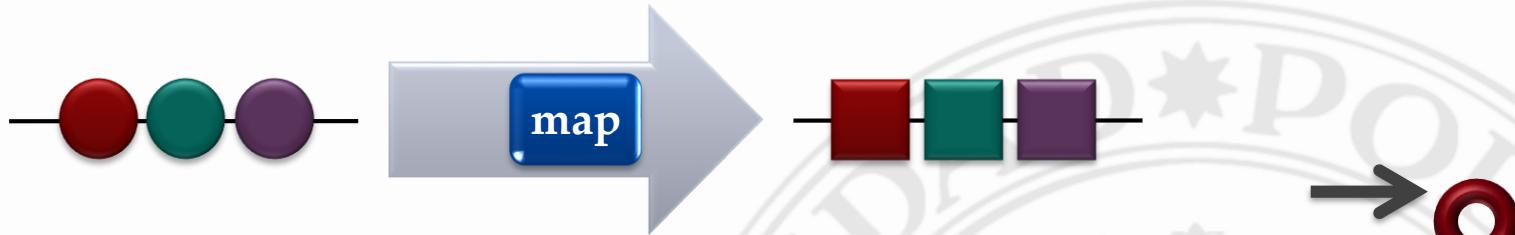
```
public ArticleDto readArticleSynchronous(String code) {  
    return new ArticleDto(this.articleRepository.findById(code) //Optional  
        .orElseThrow(() -> new NotFoundException("Article code (" + code + ")"))  
    );  
}
```



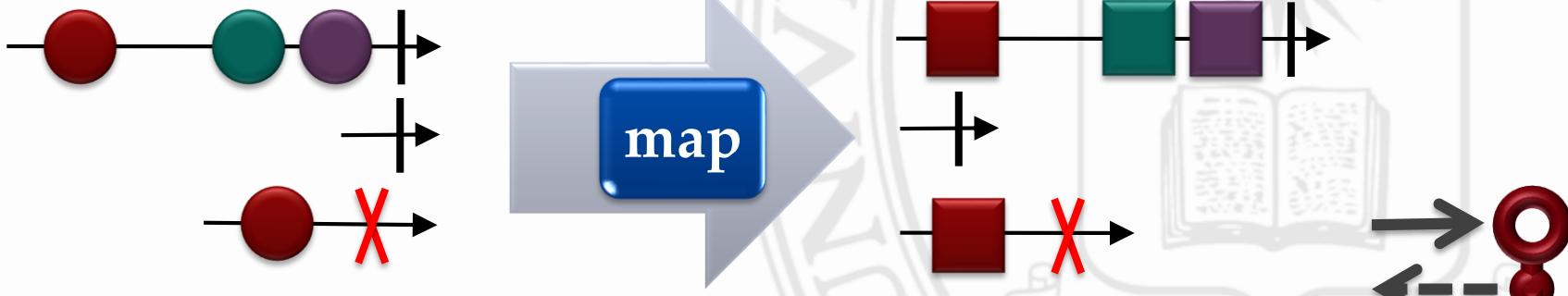
```
public Mono<ArticleDto> readArticle(String code) {  
    return this.articleReactRepository.findById(code)  
        .switchIfEmpty(Mono.error(new NotFoundException("Article code (" + code + ")")))  
        .map(ArticleDto::new);  
}
```

Programación Reactiva

Project Reactor. Funciones



```
public Stream<Integer> convertToIntegers(Stream<String> stream) {  
    return stream.map(Integer::valueOf);  
}
```



```
public Flux<Integer> convertToIntegers(Flux<String> flux) {  
    return flux.map(Integer::valueOf);  
}
```

Programación Reactiva



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

- Package: `es.upm.miw.reactive`

Reactive in action

- `StreamAsynchronous`
- `ReactiveService`
- `ReactiveComponent`

Test

- `StepVerifier`

Ejercicios

- Se recibe un `Flux<String>`, se devuelve un `Flux<Integer>` de número pares y positivo
- Ser reciben 3 `Flux<Integer>`, se devuelve un flujo que fusiona los dos primeros quitando los negativos y continua con el tercero quitando los ceros

Angular: RxJS. <https://rxjs.dev/guide/overview>,
<https://stackblitz.com/>

```
import { of, map, filter, tap, Observable, EMPTY, throwError, iif, merge, concat } from 'rxjs';
import { catchError, concatMap } from 'rxjs/operators';

//Services
function httpGetMock(): Observable<any> {
  return of(-1, 0, 1);
}

function httpPostMock(param: string): Observable<any> {
  if (param === "") {
    return throwError(() => new Error('Error!!!'));
  } else {
    return of(`httpPostMock=${param}`);
  }
}

function httpDeleteMock(): Observable<any> {
  return of('httpDeleteMock');
}

function httpPutMock(): Observable<any> {
  return of('httpPutMock');
}
```

Programación Reactiva

Angular: RxJS

```
//Services
function service1(): Observable<string> {
  return httpGetMock().pipe(
    filter((item) => item >= 0), //first, skip, take...
    tap((item) => console.log('debug-s1 >>> ' + item)), //logs
    map((item) => 'S1: ' + item)
  );
}
function service2(): Observable<string> {
  return httpPostMock('').pipe(
    tap((item) => console.log('debug-s2 >>> ' + item)),
    catchError((error) => {
      console.log('debug-s2 >>> processed ERROR');
      return EMPTY;
    })
  );
}
function service3(): Observable<string> {
  return httpGetMock().pipe(
    concatMap((item) => of(item)), //flatMap
    map((item) => 'S3: ' + item) //map
  );
}
function concatService(): Observable<string> {
  return concat(httpPostMock('666'), httpGetMock()) //merge, zip ...
  .pipe(map((item) => 'ConcatS: ' + item));
}
function ifService(param: boolean): Observable<string> {
  return iif( () => param, merge(httpPutMock(), httpDeleteMock()), httpPutMock())
  .pipe(map((item) => 'IfS: ' + item));
}
```

Programación Reactiva

Angular: RxJS.

```
//Component
```

```
service1().subscribe(console.log); //Se debe subscribir para que se ejecute!!!
```

```
service2().subscribe();
```

```
service3().subscribe({
  next: (item) => console.log('next:', item),
  error: (err) => console.log('error:', err),
  complete: () => console.log('the end'),
});
```

```
concatService().subscribe({
  next: (item) => console.log('next:', item),
  error: (err) => console.log('error:', err),
  complete: () => console.log('the end'),
});
```

```
ifService(true).subscribe(console.log);
```

Programación Reactiva

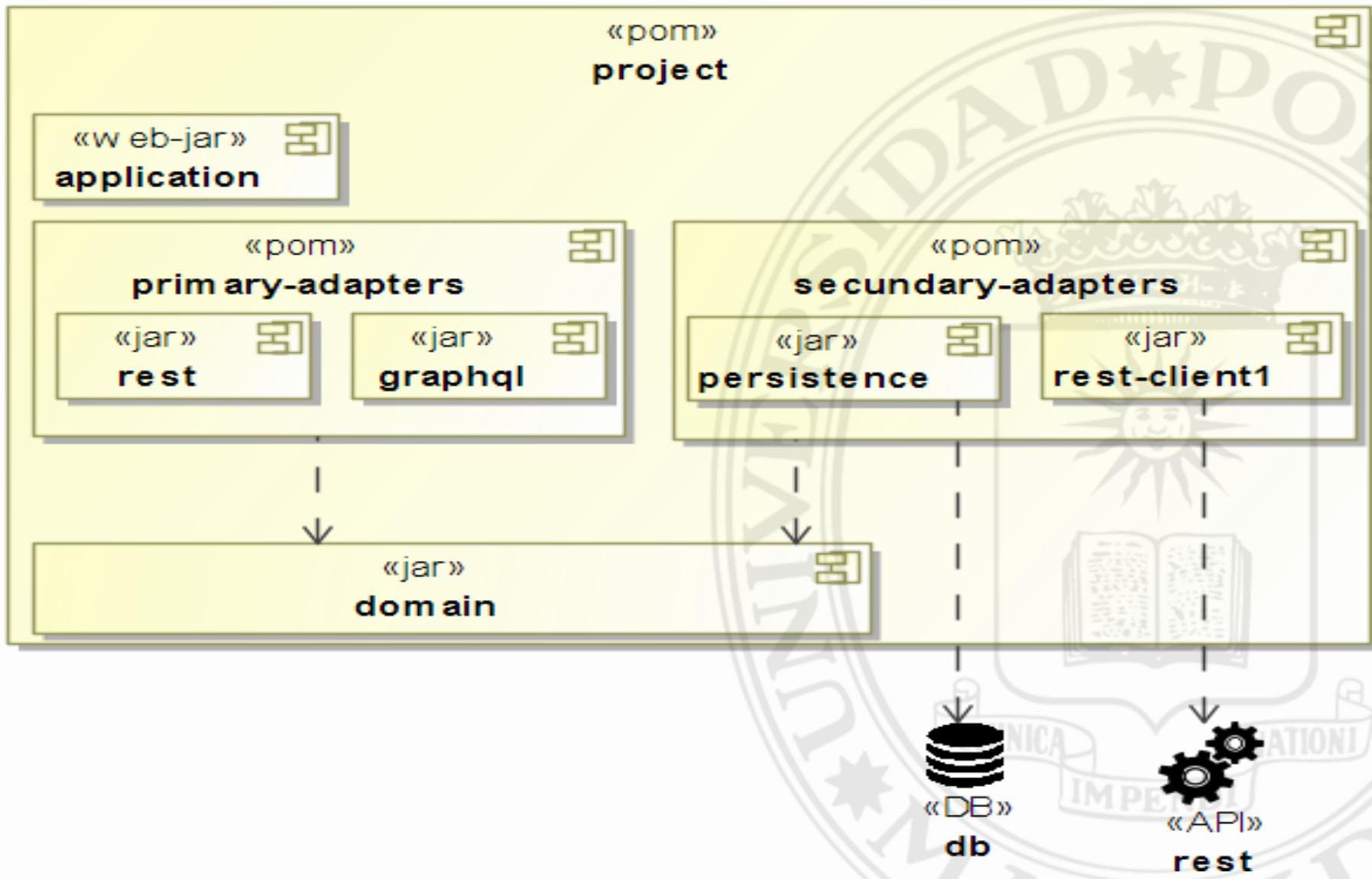


<https://stackblitz.com>

<https://rxjs.dev/guide/overview>

Arquitectura

Múltiples Artefactos



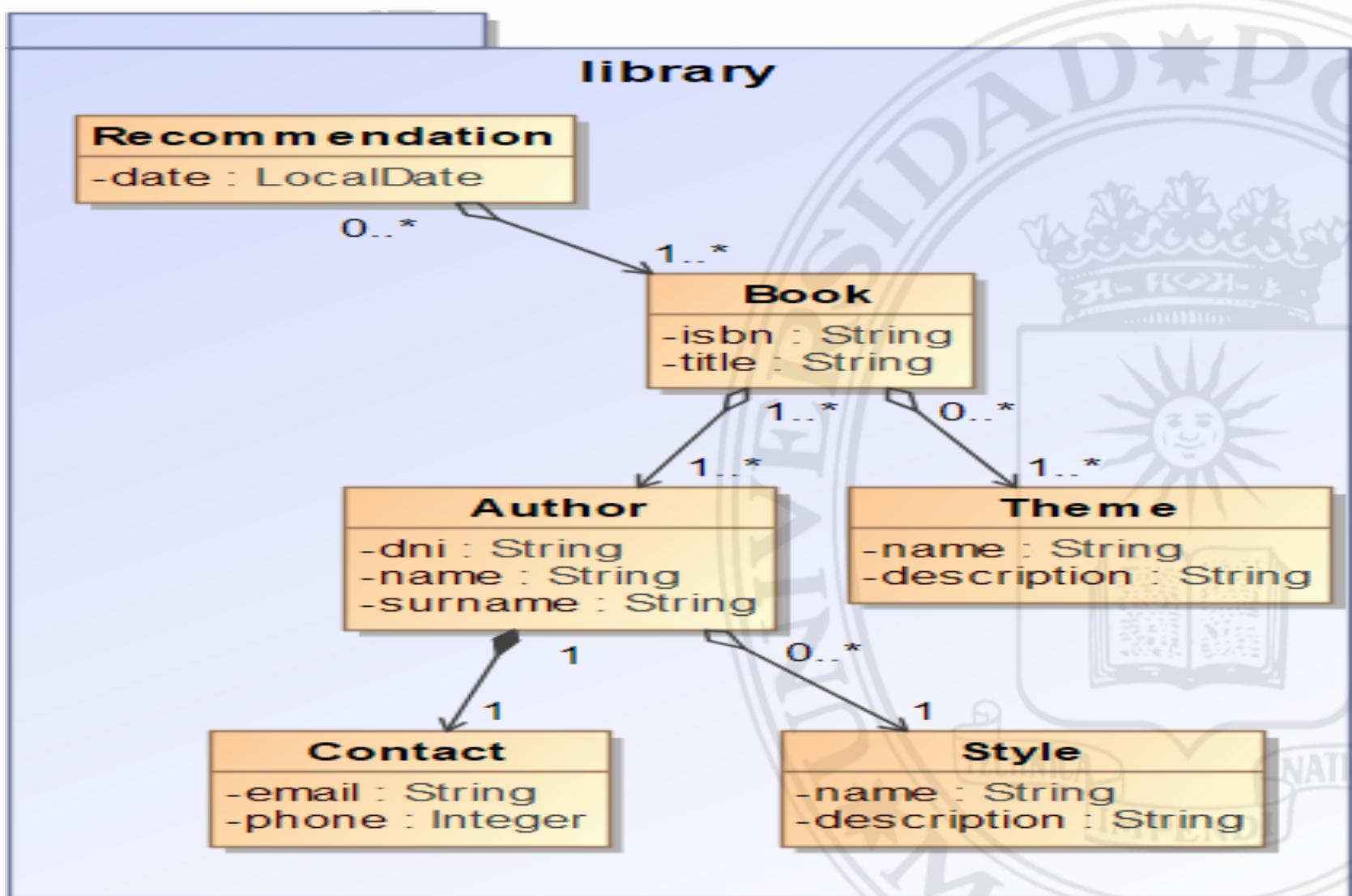
Arquitecturas. Shop



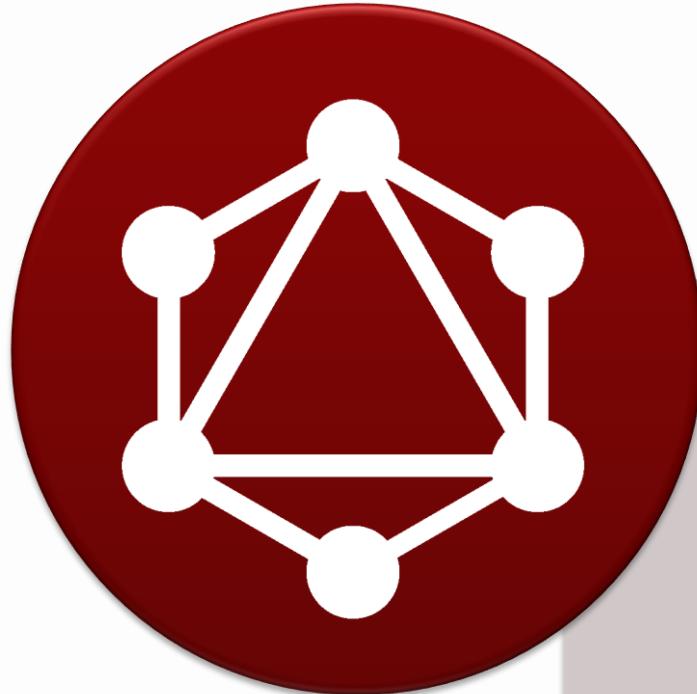
<https://github.com/miw-upm/apaw-shop-architectures>

- shop-three-layers-rest
- shop-hexagonal

GraphQL End-points



¿Qué es GraphQL?



Es un lenguaje de consulta y modificación de un API.

<https://graphql.org/>

GraphQL

Características

GraphQL

- Desarrollado por Facebook y liberado en 2015
- Puede funcionar sobre HTTP y en un end-point: `*/graphql`
 - GET `?query=***`
 - POST `{"query":....}`
- El servidor describe un esquema (*schema.graphqls*) del API.
- El cliente especificar los detalles de los datos solicitados.
- Cliente de prueba: **GraphiQL**.
- Existen librerías para diferentes lenguajes.
 - JavaScript, Java, Python, Go...

GraphQL

Características

```
#article.graphqls x
```

```
18   input ArticlePriceUpdating {
19     barcode:Int!
20     price: String!
21   }
22
23   type Article {
24     id: String!
25     registrationDate: String!
26     barcode: Int!
27     description: String
28
29
30 }
```

Datos

```
#Article
extend type Query {
  articles:[Article]
}
extend type Mutation {
  createArticle(articleCreation:ArticleCreation):Article
}
```

Operaciones

GraphiQL  Prettify

```
1 query {
2   articles {
3     barcode
4     registrationDate
5   }
6 }
```

Petición

```
{
  "data": {
    "articles": [
      {
        "barcode": 84001,
        "registrationDate": "2020-09-25"
      },
      {
        "barcode": 84002,
        "registrationDate": "2020-09-25"
      },
      {
        "barcode": 84003,
        "registrationDate": "2020-09-25"
      }
    ]
  }
}
```

Respuesta

GraphQL

Implementaciones



graphql-java-kickstart

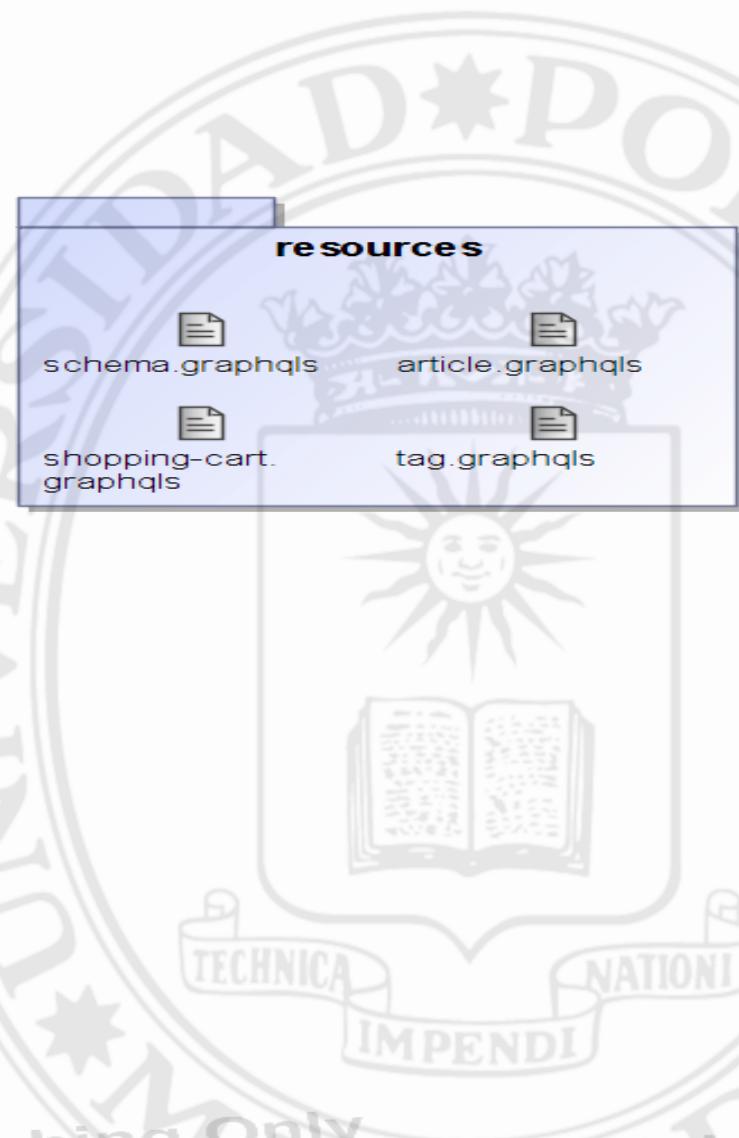
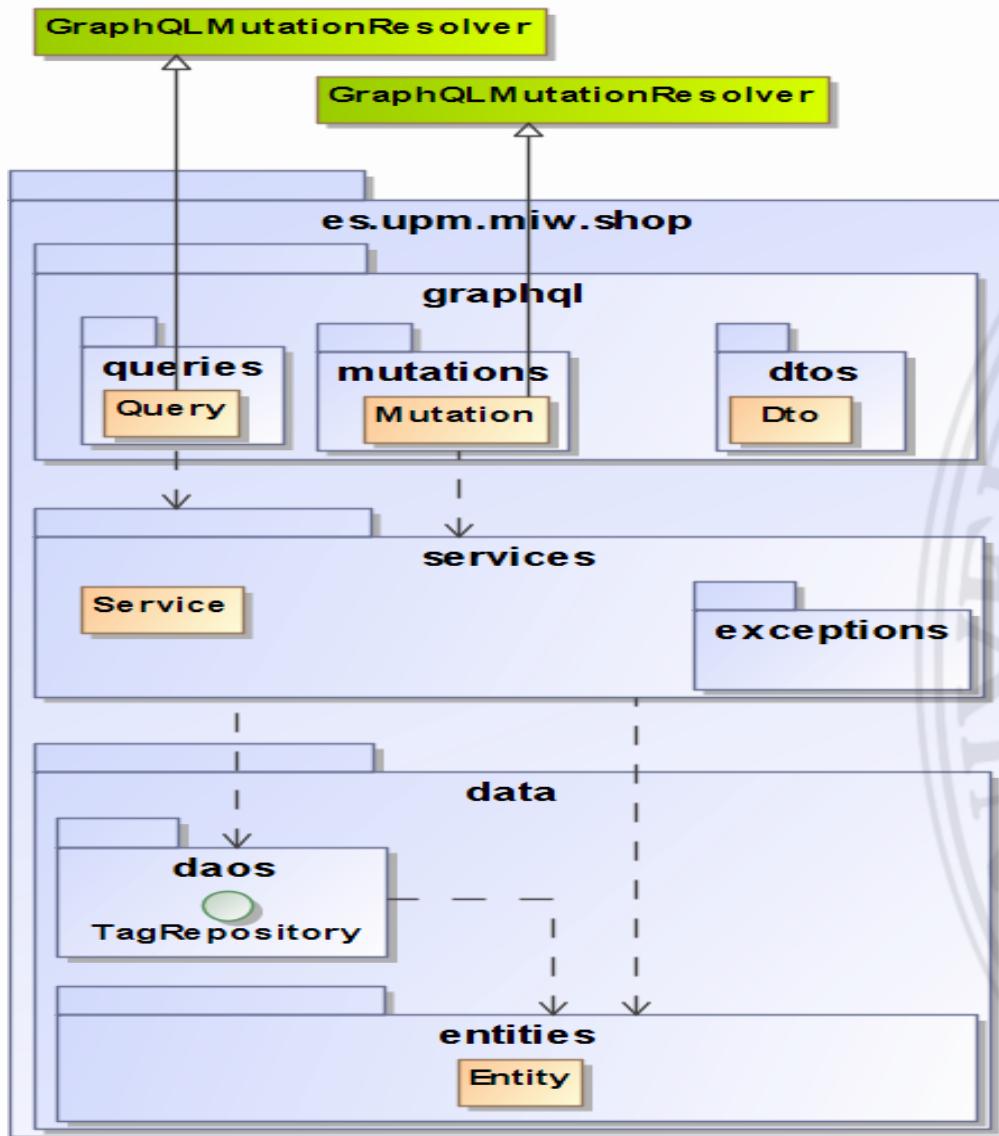
- <https://www.graphql-java-kickstart.com>
- <https://github.com/graphql-java-kickstart/samples>



graphql-java

- <https://www.graphql-java.com/>

Shop. Capas - Graphql



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

- Proyecto: *shop-three-layers-graphql*

Reactive in action

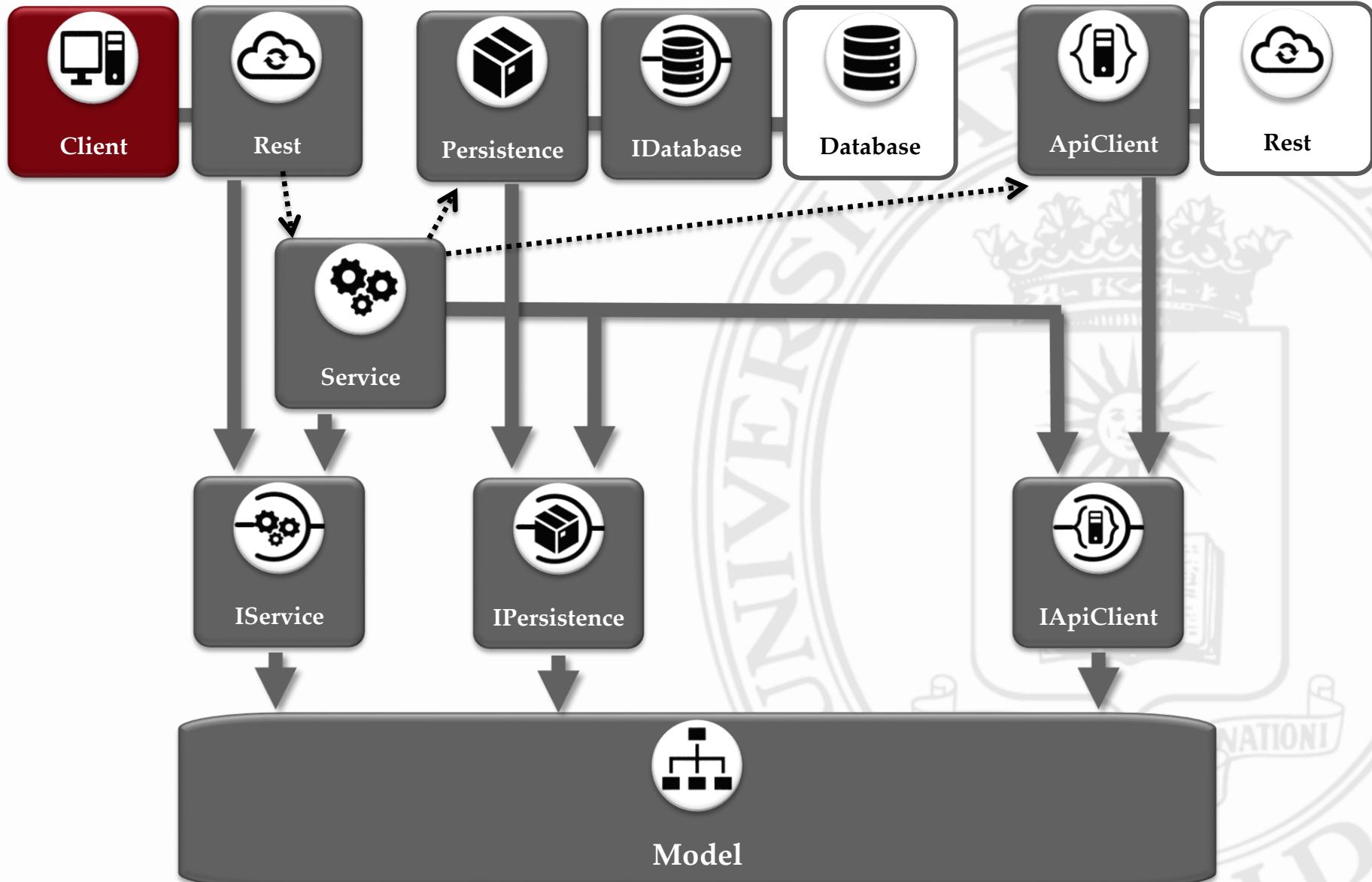
- *schema.graphqls*
- *Query & *Mutation

📝 Ejercicios

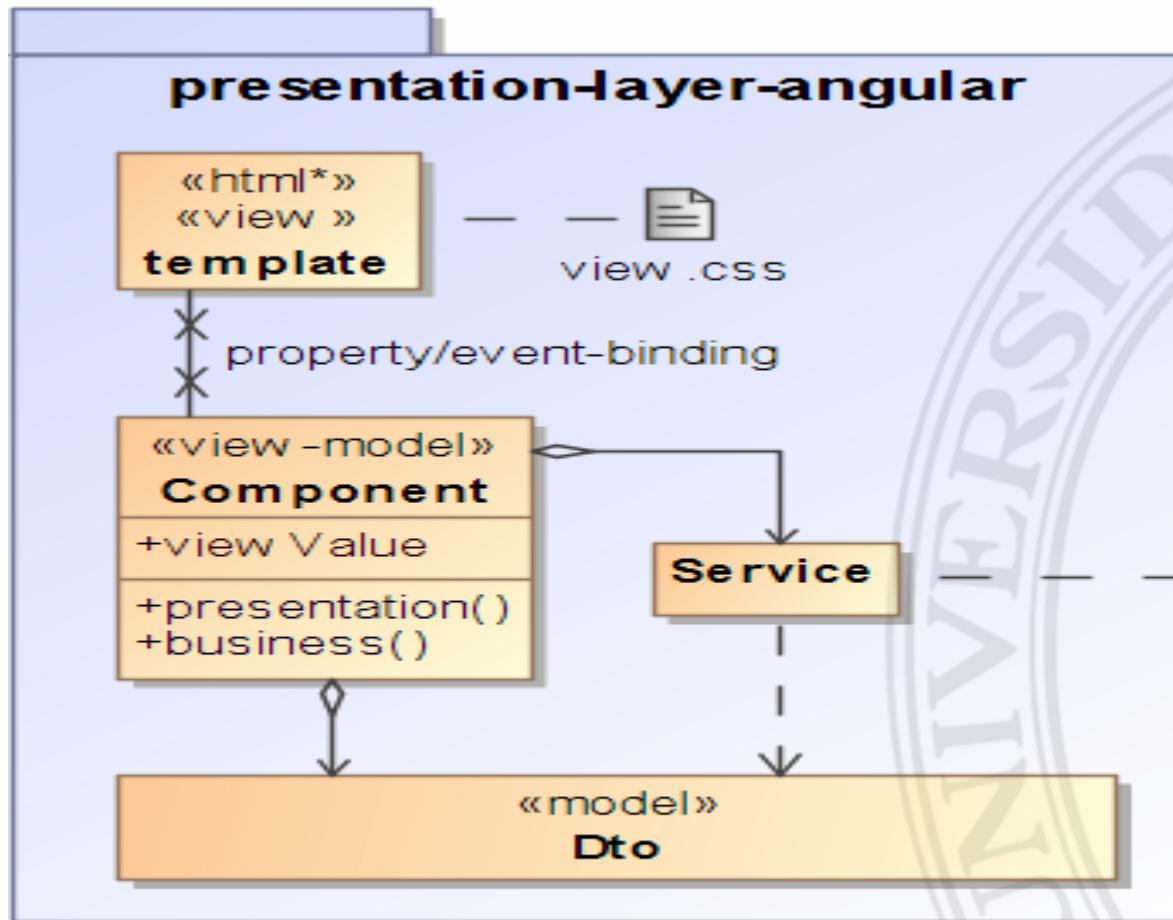
- Probar varias peticiones: */docs/requests.txt*
- Arrancar el servidor: *mvn spring-boot:run*
- Cliente web: *http://localhost:8080/graphiql*

Presentación

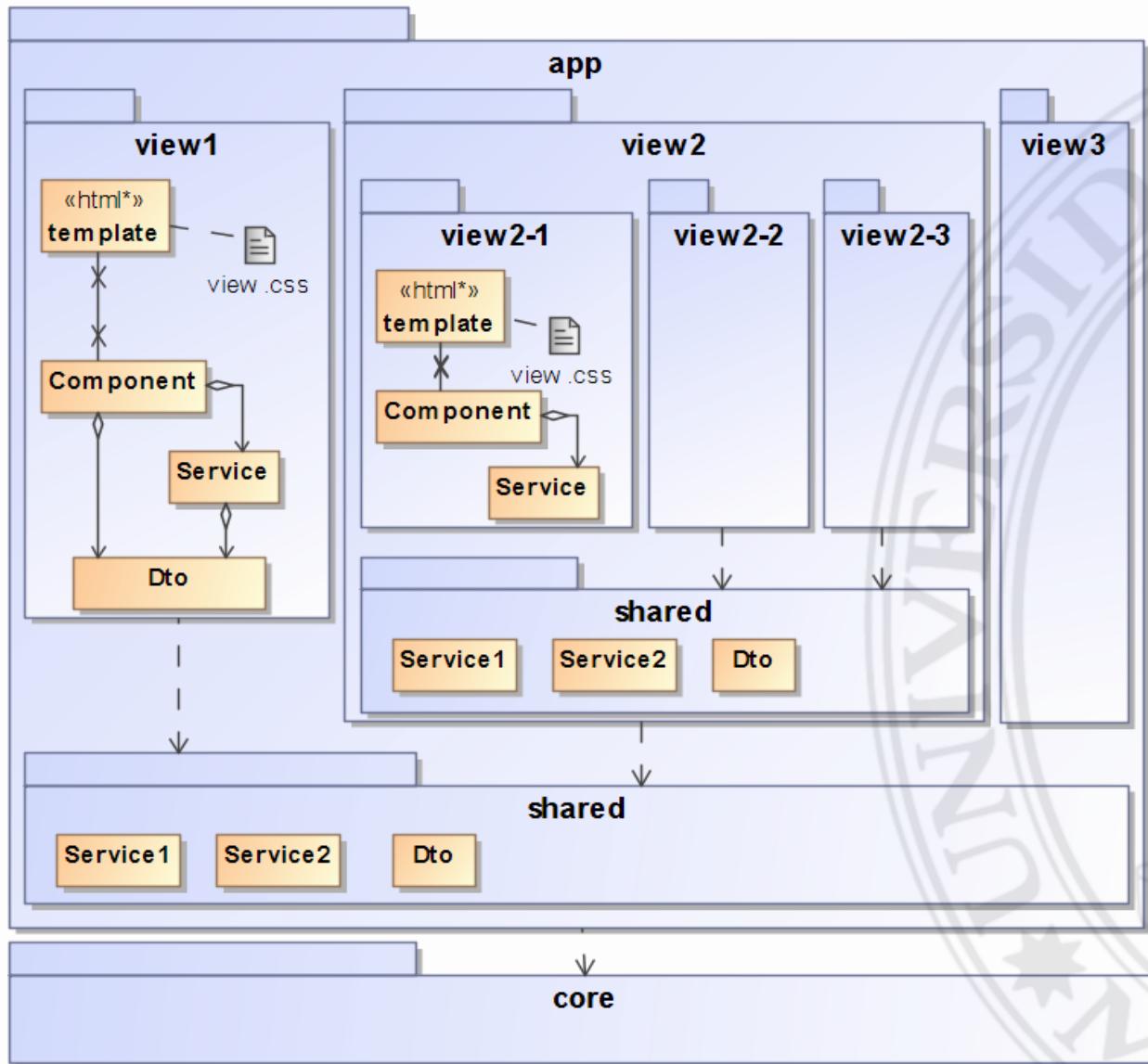
Cliente



Angular Presentación



Angular Presentación



Android Presentación

