Arquitectura y Patrones para Aplicaciones Web

Arquitecturas y Patrones Web

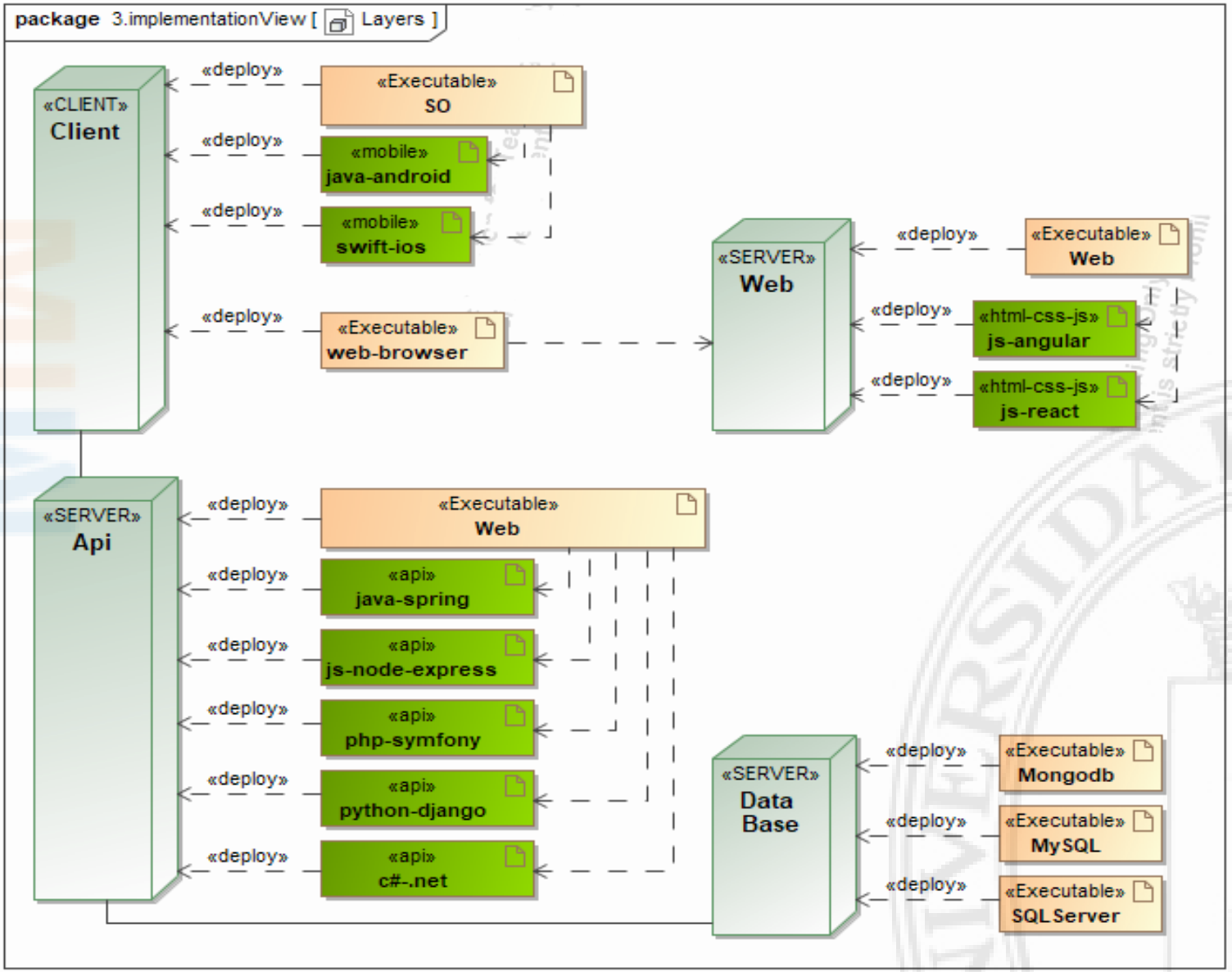
Jesús Bernal Bermúdez

Modelo de Aplicación Web

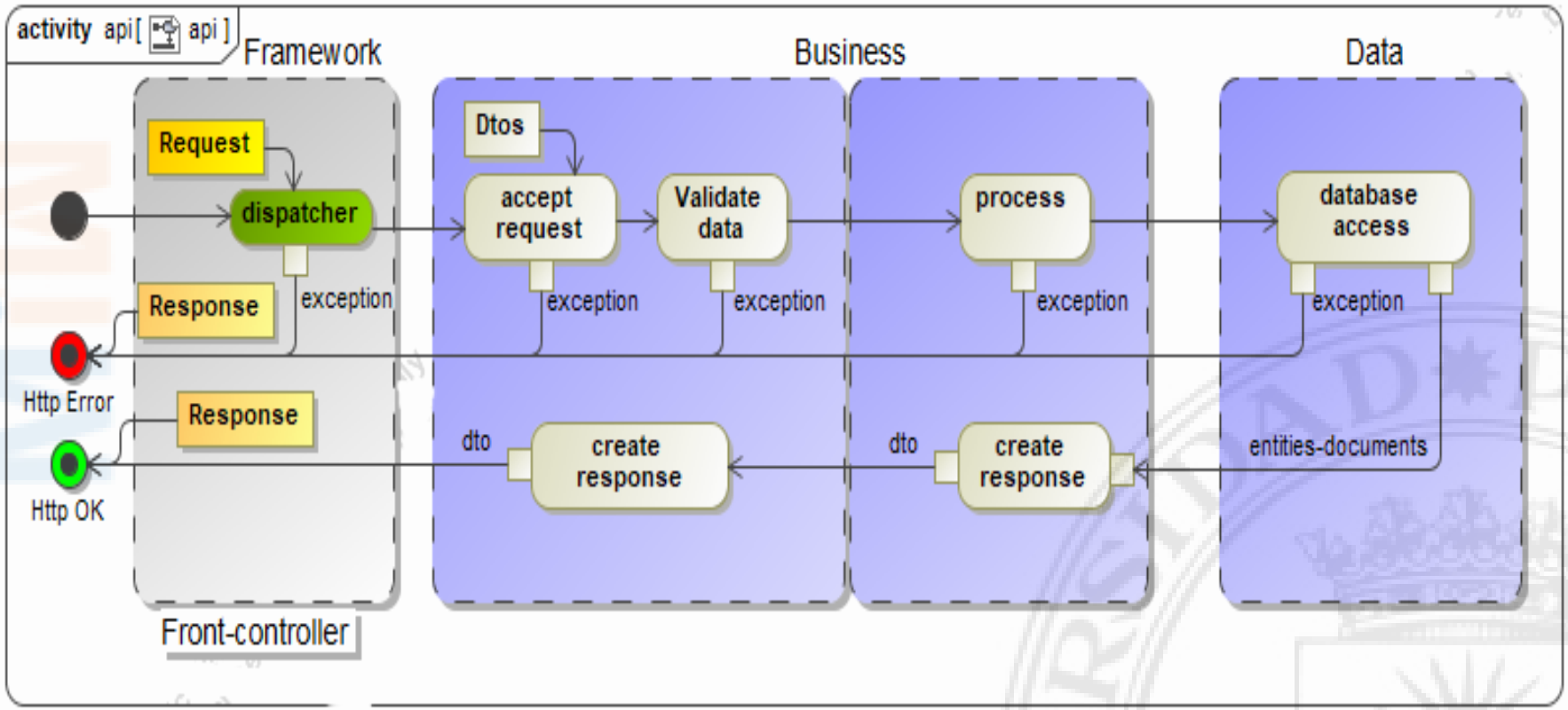
- API (*Application Programming Interfaces*)
- REST (*Representational State Transfer):* HTTP
- Aplicación WEB: API REST
- API REST EndPoint

- Modelo distribuido multitarea. Esta dividido en componentes dependiendo de su función e instalado en diferentes máquinas
- Layers (*logic*) vs Tiers (*deployment*)
- Estructurada por Capas. Existen varias organizaciones, dependiendo de los autores o de las tecnologías
  - Presentation Layer
    - Tendencia actual a ejecutarse en cliente
      - Framework JS (Angular, React, backbone…)
      - Mobile
  - Business Layer
    - Tendencia actual a ser un API rest
  - Data Layer

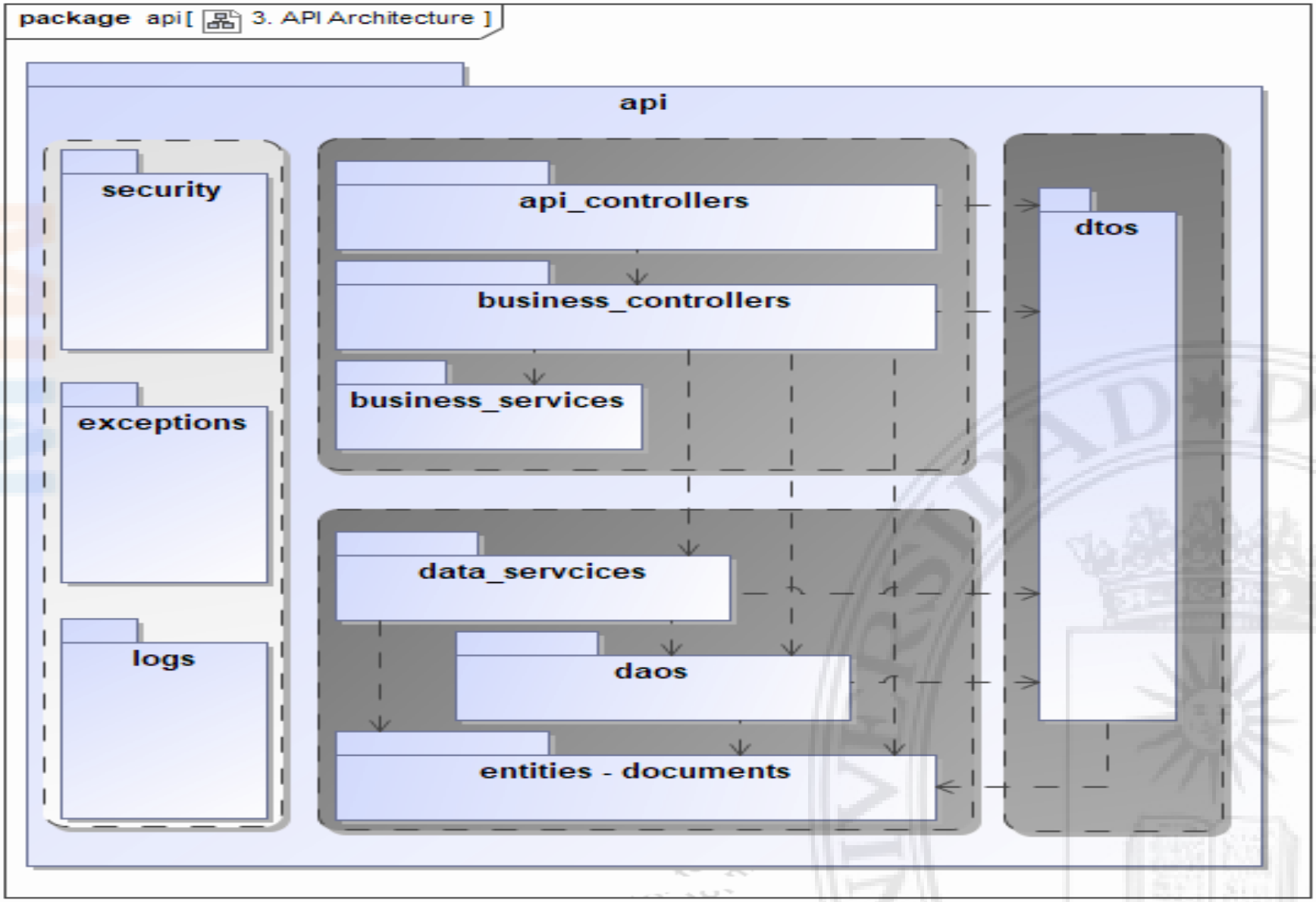
Aplicación Web



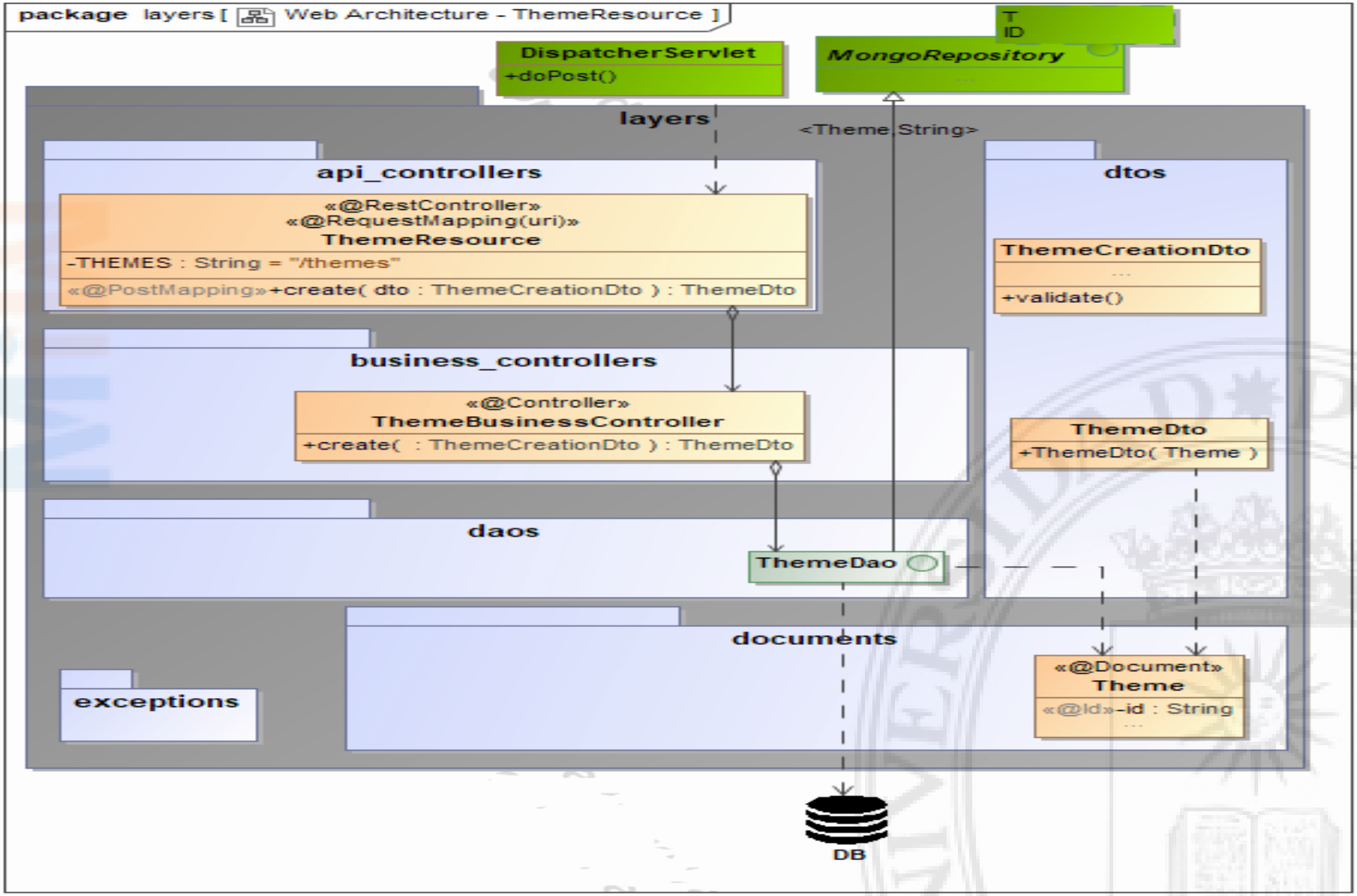
API Rest. Flujo de datos



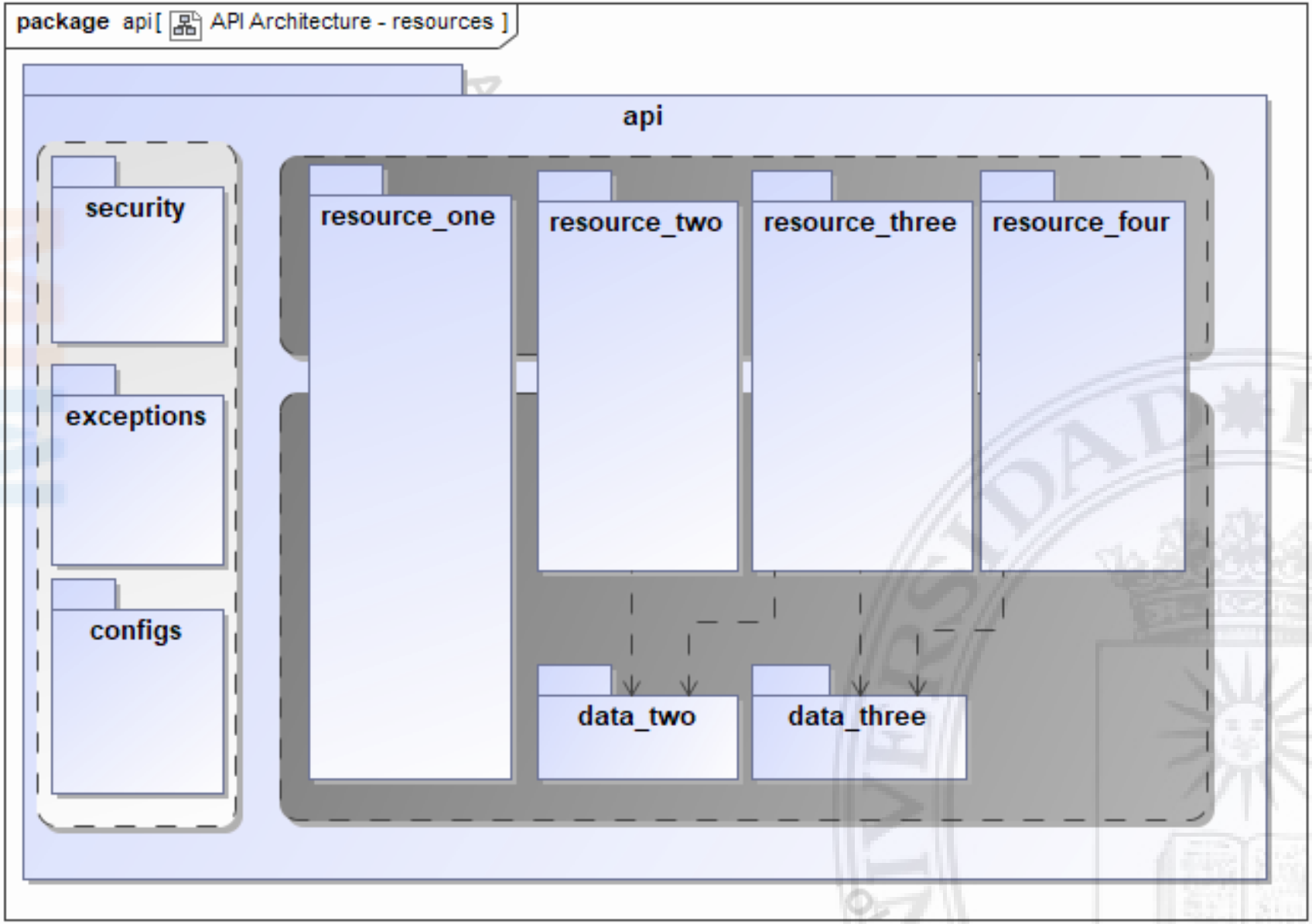
Arquitectura Web. Layers



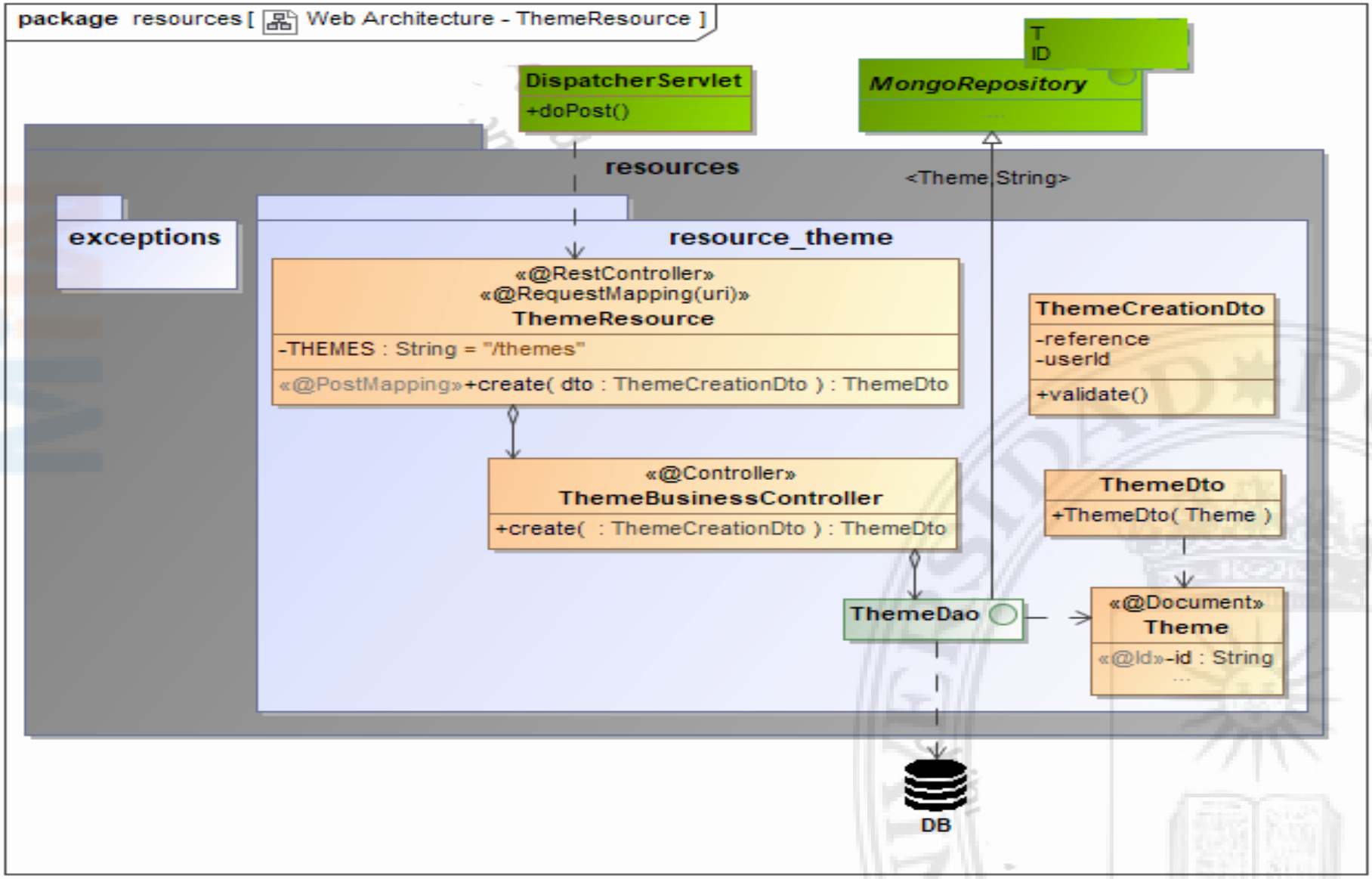
Arquitectura Web. Layers



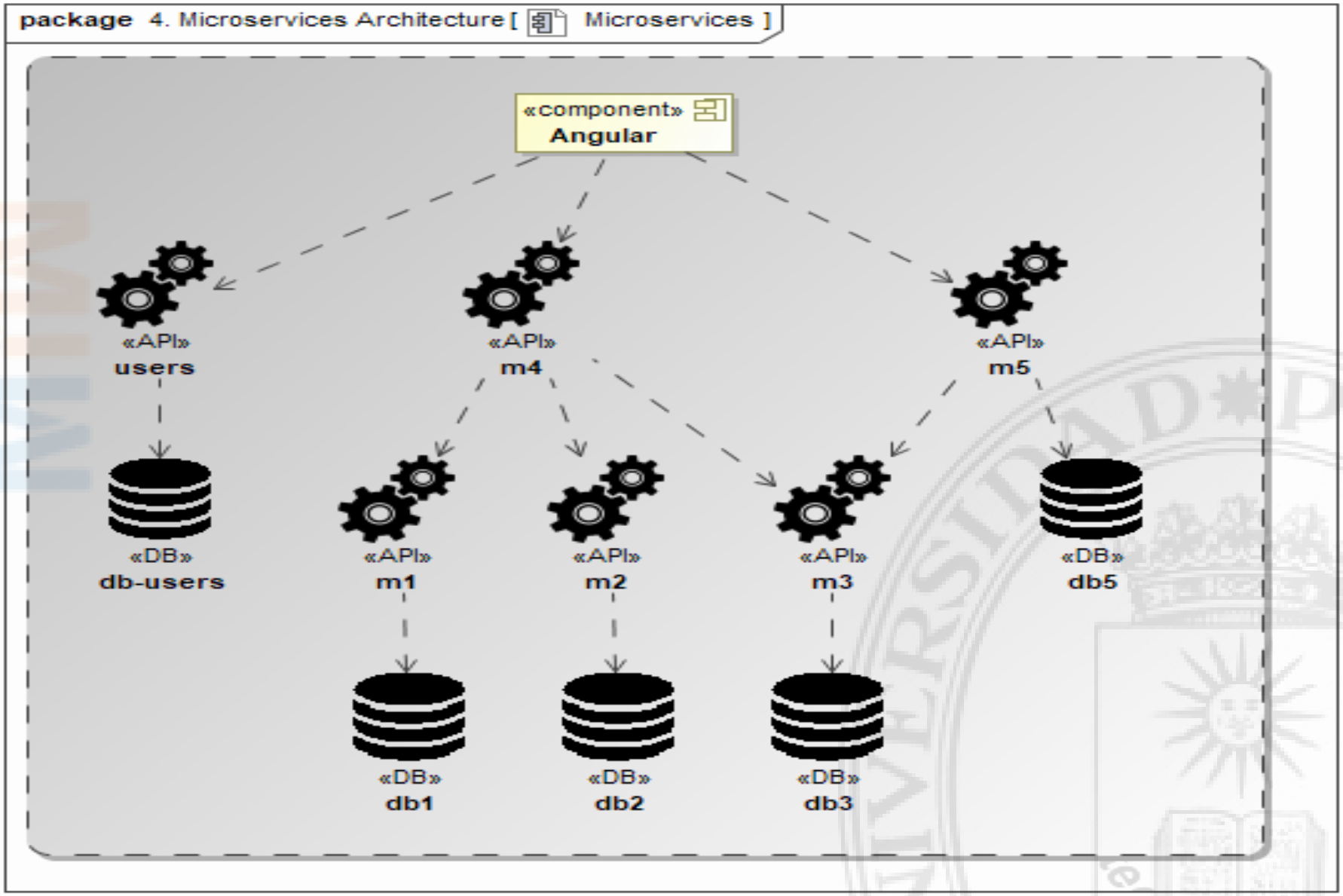
Arquitectura Web. Resources



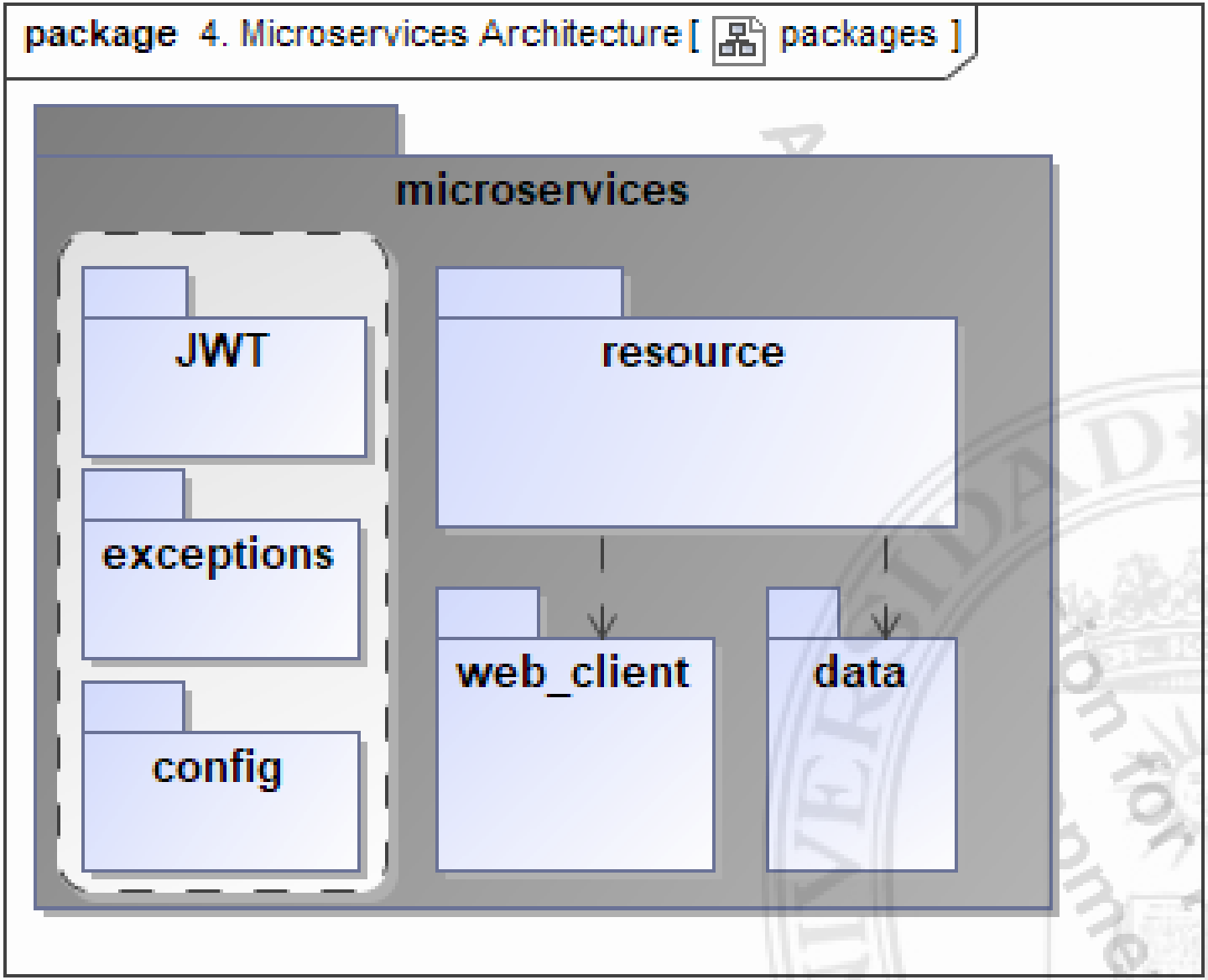
Arquitectura Web. Layers



Arquitectura Web. Micro Servicios



Arquitectura Web. Micro Servicios



Data Transfer Object (DTO)

- Motivación
  - En la capa de presentación se necesita obtener diferentes datos de diferentes partes de la capa de negocio. La aplicación se puede ver degradada por el uso intensivo de red
- Propósito
  - Permitir el intercambio de datos eficiente entre la capa de presentación y la capa de negocio
- Ventajas
  - Al utilizar un *DTO* para encapsular los datos de negocio, se realiza una única llamada a un método para enviar y recuperar el *DTO*, optimizando los recursos de red y de servidor

- Api Rest: los objetos devueltos serán DTOs, normalmente en JSON

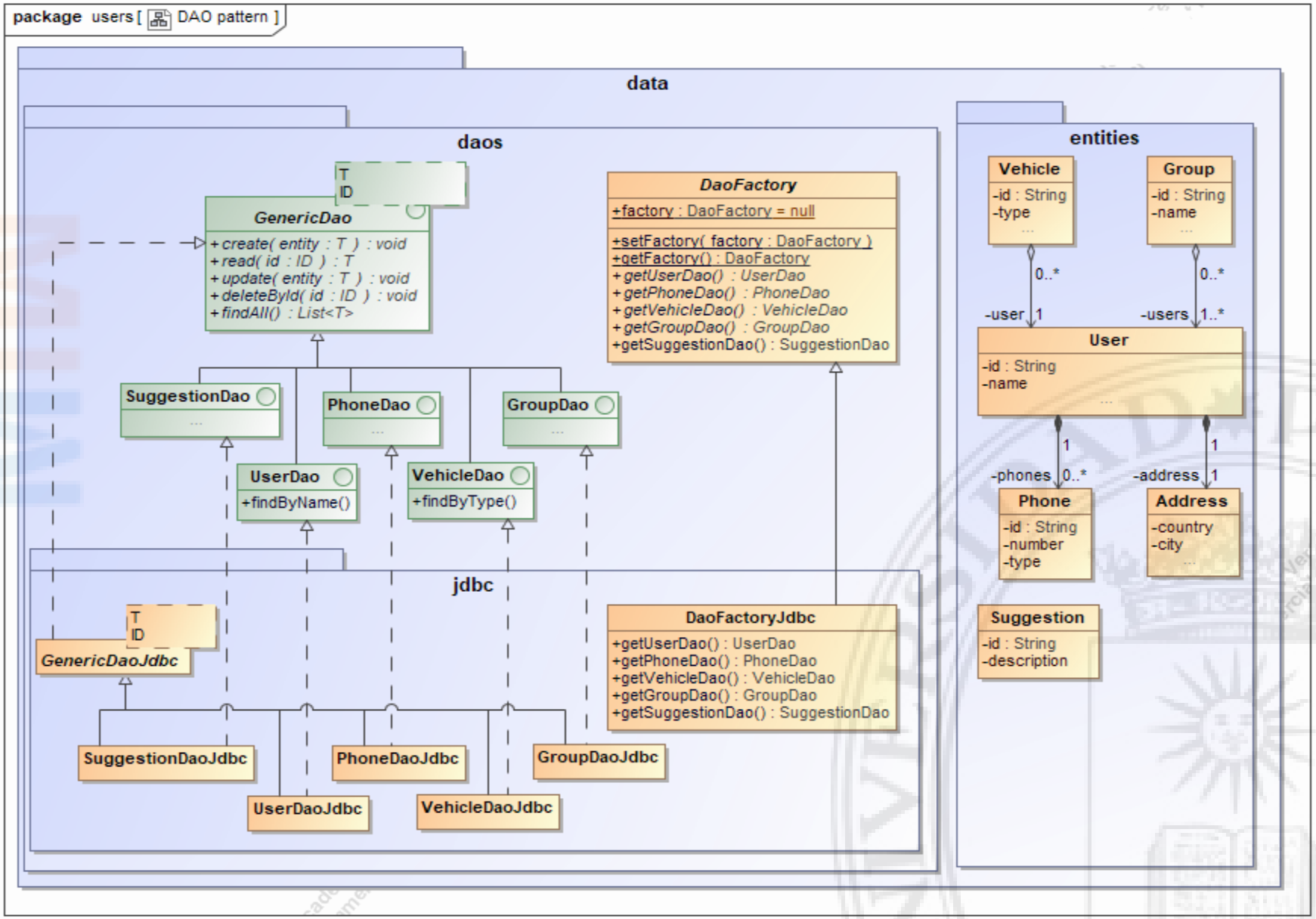
DAO (Data Access Object)

- La persistencia consiste en guardar los datos de forma permanente y de una manera ordenada, normalmente en Bases de Datos (relacionales, NoSQL...), ficheros XML/JSON...
- Data Transfer Object (DTO), normalmente llamado ***entidad*** *(Entity-SQL)*, ***documento*** *(Document-MongoDB)*. Son los objetos en donde se guardan los datos que vienen o van a la capa de persistencia
- DAOs (*Repositories*). Son las clases que se encargan de convertir los datos almacenados de forma persistente en BD en *objetos* o viceversa. Estas clases están conectadas con las BD
- Propósito
  - Abstraer y encapsular el acceso a la capa de persistencia, permitiendo desacoplar la capa de negocio con la capa de persistencia
  - Adapta el modelo de persistencia con el modelo de objetos
  - Permite el intercambio de implementaciones de persistencia sin afectar a la capa de negocio

Diseño: DAO

- Los DAOs transforman los datos del modelo de persistencia al modelo de Objetos
- En general, existirá un DAO por cada *Entidad-Documento*, componiendo un conjunto de DAOs relacionados
- Existirá un tipo de DAO por cada modelo de persistencia, y para que la *capa de negocio* esté desacoplada del modelo de persistencia, se aplicará el patrón ***Abstract Factory***
- Para que exista una sola factoría, se aplicara el patrón ***Singleton***

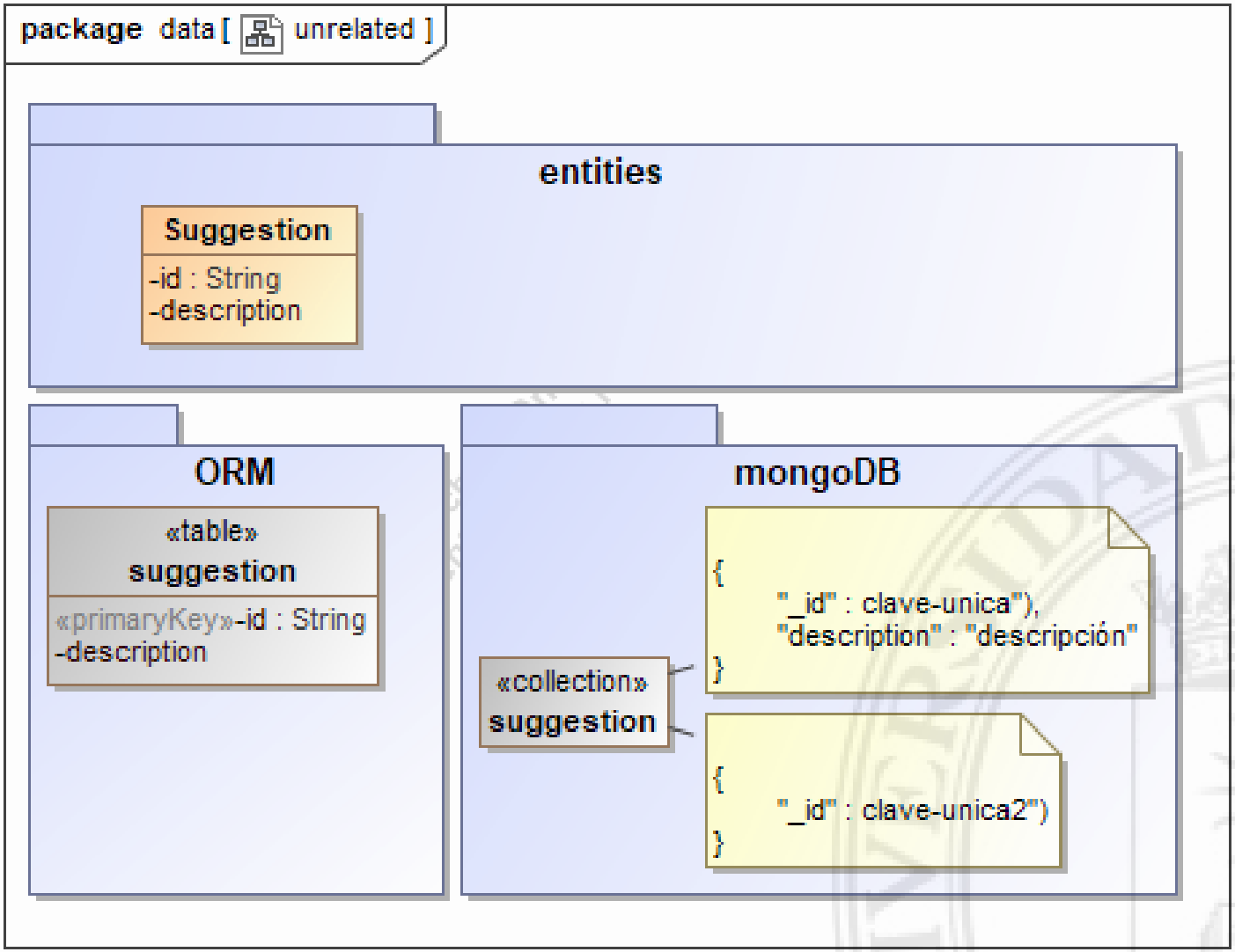
DAO


package users [DAO pattern]
data
daos
T
ID
GenericDao
+create(entity : T) : void
+read(id : ID) : T
+update(entity : T) : void
+deleteById(id : ID) : void
+findAll() : List<T>
DaoFactory
+factory : DaoFactory = null
+setFactory(factory : DaoFactory)
+getFactory() : DaoFactory
+getUserDao() : UserDao
+getPhoneDao() : PhoneDao
+getVehicleDao() : VehicleDao
+getGroupDao() : GroupDao
+getSuggestionDao() : SuggestionDao
SuggestionDao
PhoneDao
GroupDao
UserDao
+findByName()
VehicleDao
+findByType()
jdbc
T
ID
GenericDaoJdbc
DaoFactoryJdbc
+getUserDao() : UserDao
+getPhoneDao() : PhoneDao
+getVehicleDao() : VehicleDao
+getGroupDao() : GroupDao
+getSuggestionDao() : SuggestionDao
SuggestionDaoJdbc
PhoneDaoJdbc
GroupDaoJdbc
UserDaoJdbc
VehicleDaoJdbc
entities
Vehicle
-id : String
-type
Group
-id : String
-name
0..*
-user
1
0..*
-users
1..*
User
-id : String
-name
1
-phones
0..*
1
-address
1
Phone
-id : String
-number
-type
Address
-country
-city
Suggestion
-id : String
-description

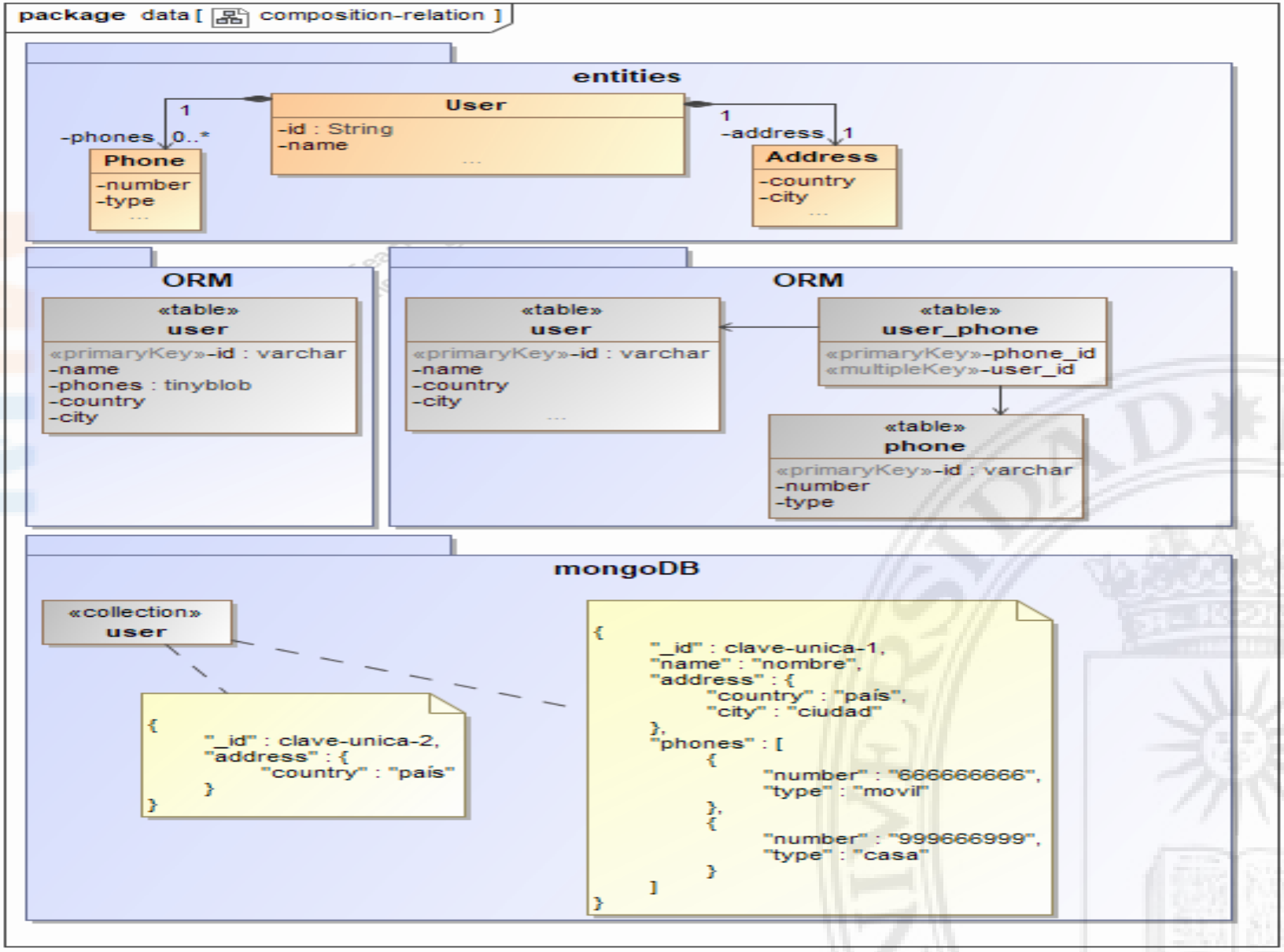
Mapeo Objeto-Relacional: ORM

- El mapeo objeto-relacional (Object-Relational Mapping ORM) es una técnica para convertir datos de un lenguaje de programación orientado a objetos a una base de datos relacional
- Existen varias alternativas para el mapeo
- Eficiencia dependiendo de cada caso
  - Ocupación de memoria
  - Velocidad de acceso

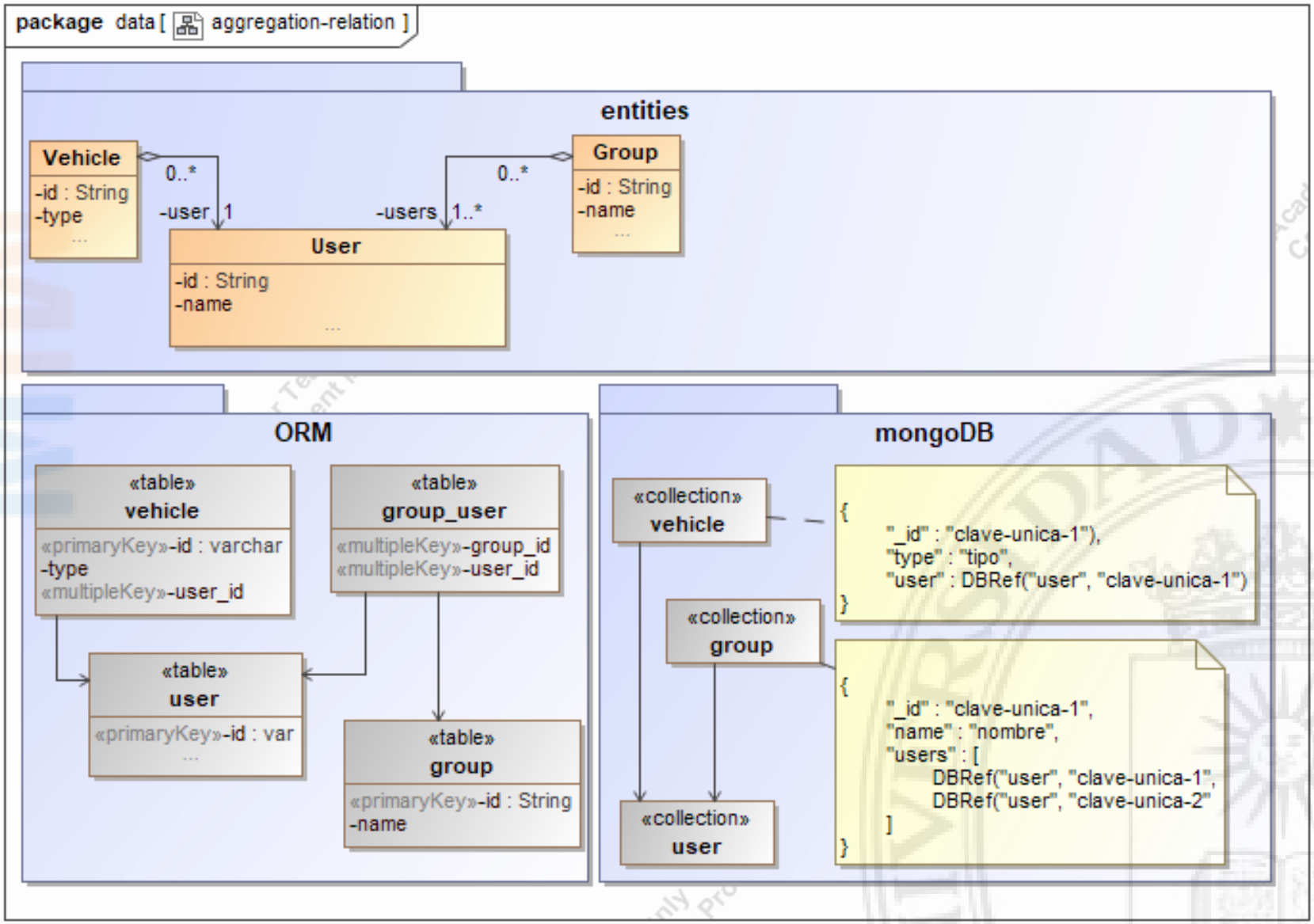
Sin relación



Relación de composición

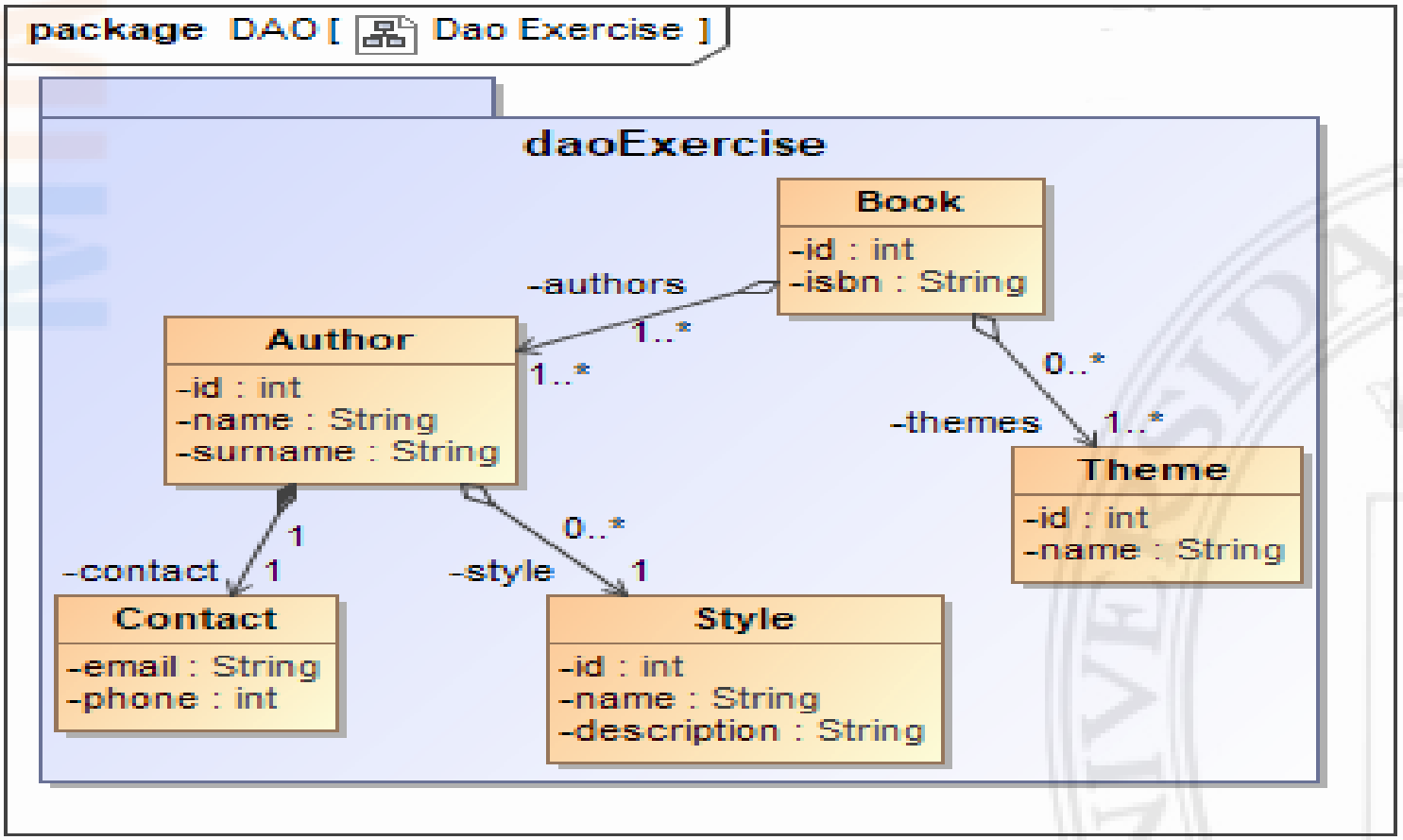


Relación de agregación & asociación



Ejercicio

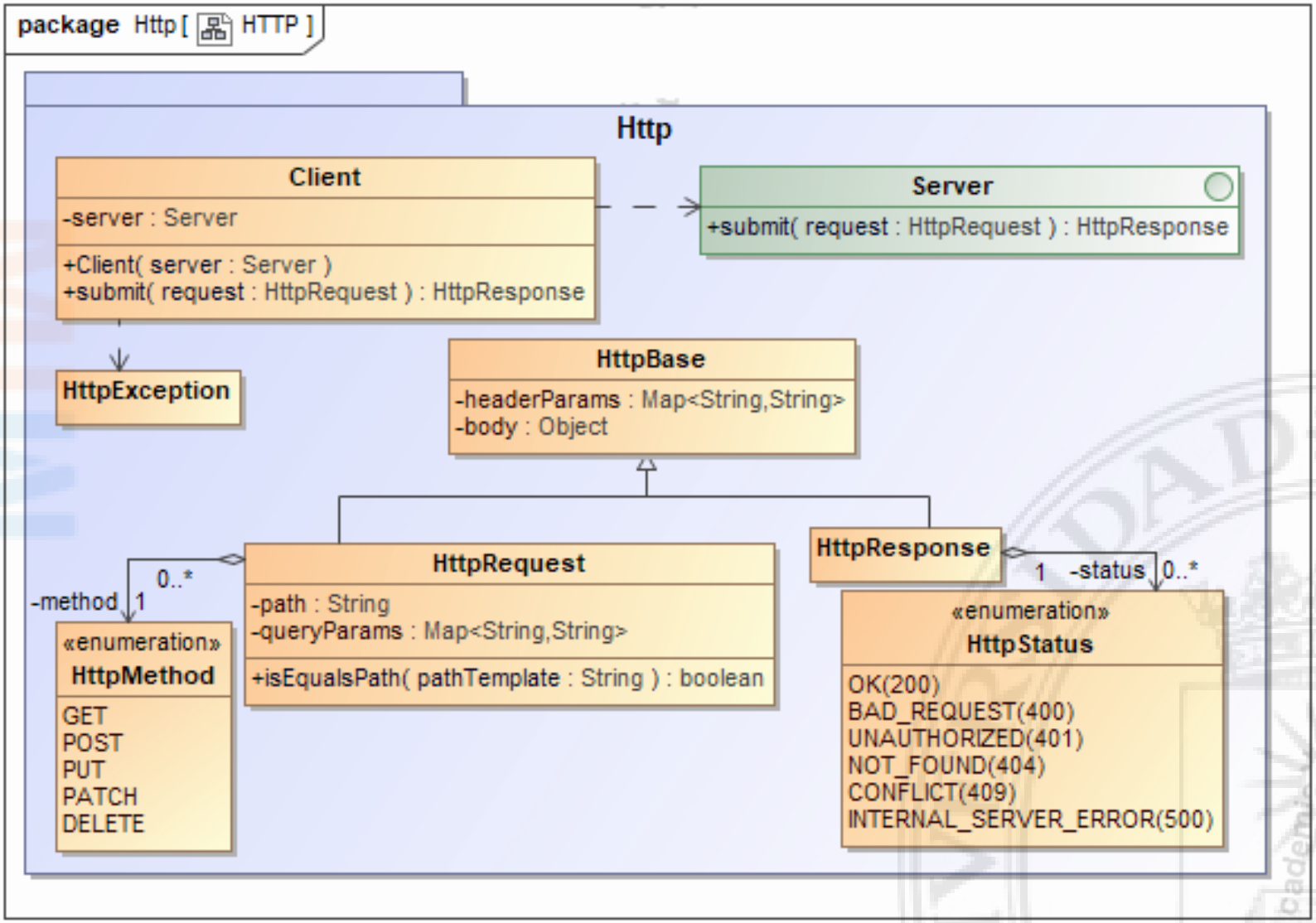
- Implementar el paquete *entities*
- Implementar el paquete *daos*
- Buscar diseños alternativos



Protocolo HTTP

- HTTP (Hyper Text Transfer Protocol) es el protocolo usado en cada transacción de la Web
- Una transacción HTTP consiste básicamente en:
  - Solicitud: envío por parte del cliente de una petición al servidor
    - URI (Uniform Resource Identifier)
      - http://server:80/resource?params=one&params=two&param=three
    - Encabezado
      - Método: **GET, POST, PUT, PATH, DELETE**, OPTIONS, HEAD, TRACE…
      - Parámetros
    - Cuerpo
  - Respuesta: envío por parte del servidor de una respuesta al cliente
    - Encabezado
      - Estado: 1xx: Respuestas informativas, 2xx: Peticiones correctas, 3xx: Redirecciones, 4xx Errores del cliente y 5xx Errores internos
      - Parámetros
    - Cuerpo
- HTTP es un protocolo sin estado, es decir, que se guarda ninguna información sobre conexiones anteriores

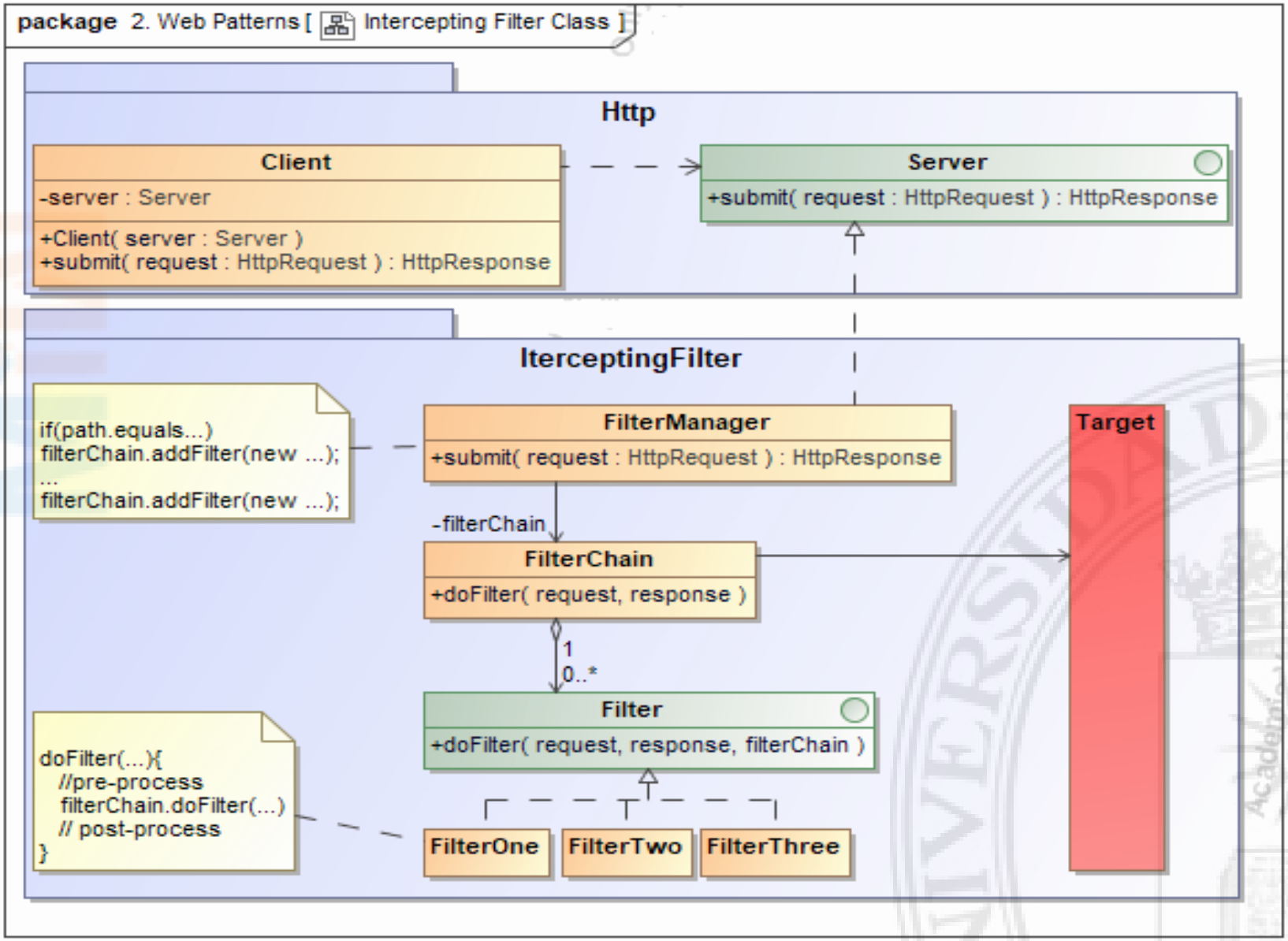
Protocolo HTTP



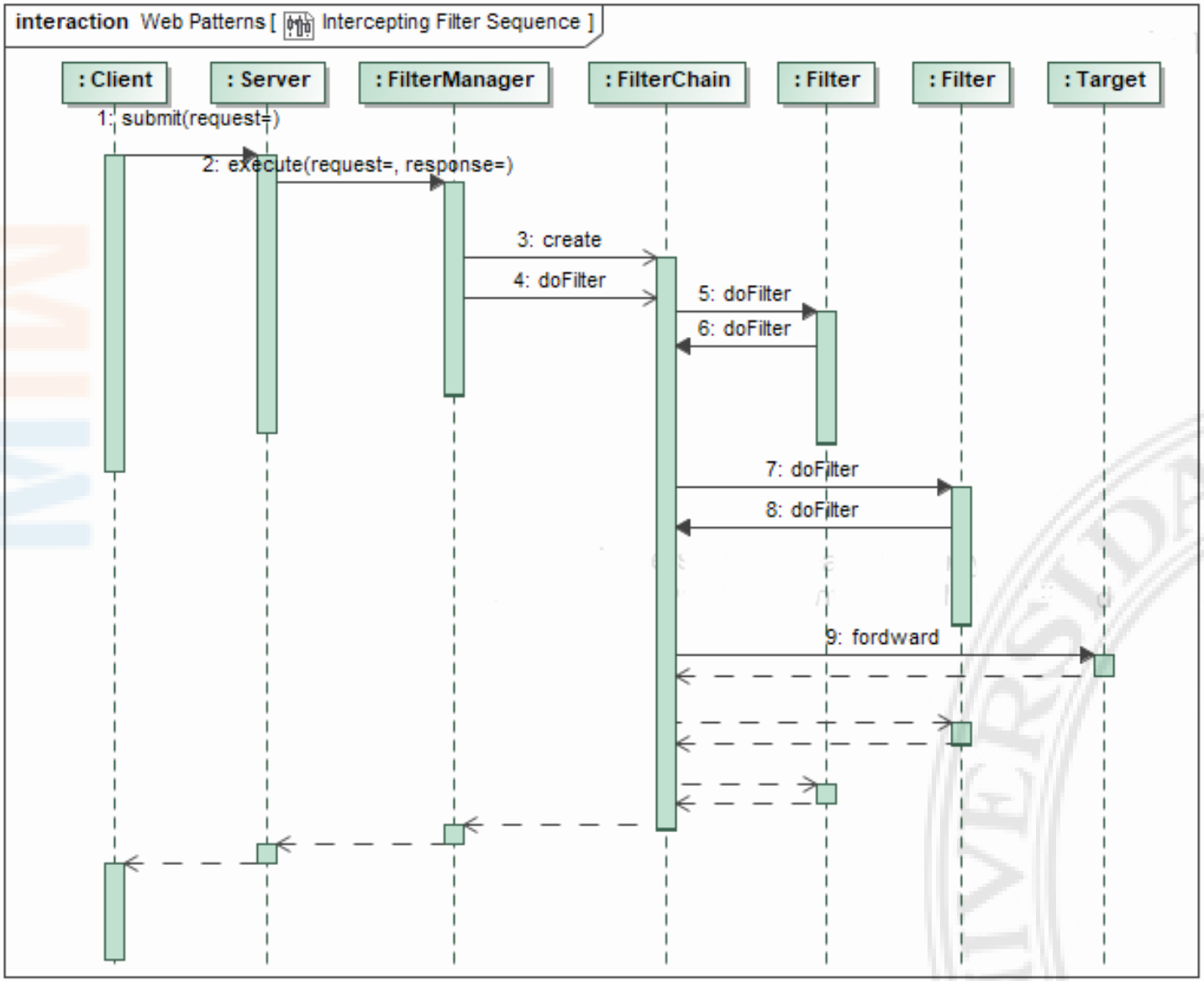
Intercepting Filter

- Motivación
  - En una gestión de peticiones, cada tipo de petición conlleva un tipo de procesamiento propio, resulta necesario identificar y estandarizar dicho proceso
  - Una petición web puede recorrer varias comprobaciones previas a su procesamiento (autentificación, validación de permisos, navegador, registrar...). En estas comprobaciones, se evalúa si se continua o no la petición
  - Se necesita una forma flexible y simple para añadir y eliminar componentes de procesamiento
- Propósito
  - Crear un conjunto de filtros conectables para procesar servicios comunes de una forma estándar sin requerir cambios en el código principal del procesamiento de la petición
- Ventajas
  - Podemos añadir y eliminar estos filtros, sin necesitar cambios en el código existente
  - Podemos, decorar nuestro procesamiento principal con una variedad de servicios comunes, como la seguridad, el logging, el depurado, etc.
  - Estos filtros son componentes independientes del código de la aplicación principal, y pueden añadirse o eliminarse de forma declarativa

Intercepting Filter



Intercepting Filter



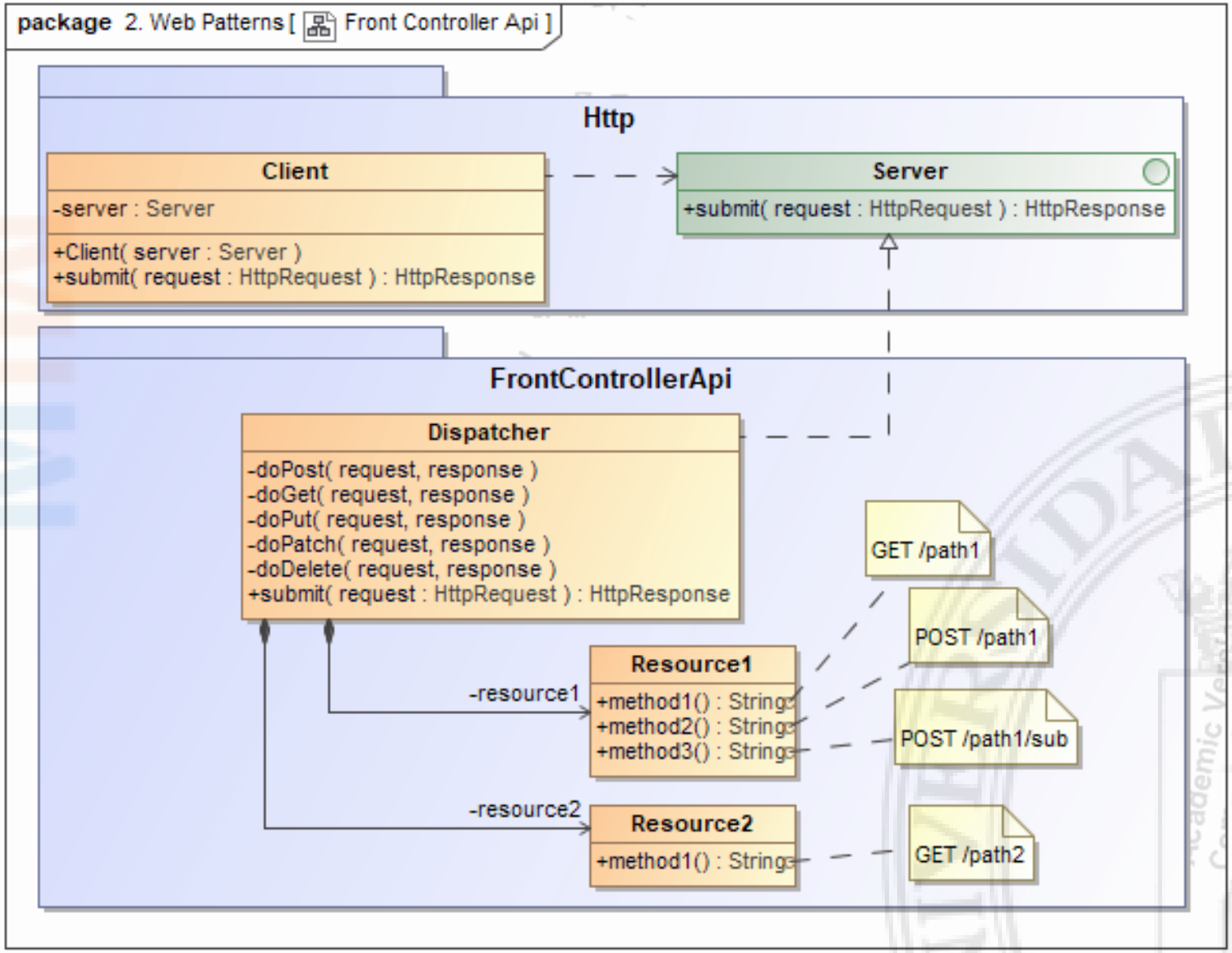
Intercepting Filter

- [INFO] 27/sep 23:20:20 http.HttpClientService --> GET /public/debug?param=value
- [INFO] 27/sep 23:20:20 ingfilter.AuthenticationFilter --> Authenticating pre-process...
- [INFO] 27/sep 23:20:20 interceptingfilter.TimeFilter --> Time pre-process: Wed Sep 27 23:20:20
- [INFO] 27/sep 23:20:20 interceptingfilter.DebugFilter --> Debuging pre-process...
- [INFO] 27/sep 23:20:20 interceptingfilter.Target --> ---------> Executing TARGET.GET /public/debug?param=value
- [INFO] 27/sep 23:20:20 interceptingfilter.DebugFilter --> Debuging post-process...
- [INFO] 27/sep 23:20:20 interceptingfilter.TimeFilter --> Time post-process: 16ms
- [INFO] 27/sep 23:20:20 ingfilter.AuthenticationFilter --> Authenticating post-process...
- [INFO] 27/sep 23:20:20 http.HttpClientService --> HTTP/1.1 200 OK headerParams={debug=DebugFilter, time=16ms}
- [INFO] 27/sep 23:20:20 http.HttpClientService --> ----------ooo----------
- [INFO] 27/sep 23:20:20 http.HttpClientService --> GET /noPublic?param=value
- [INFO] 27/sep 23:20:20 ingfilter.AuthenticationFilter --> Authenticating pre-process...
- [INFO] 27/sep 23:20:20 ingfilter.AuthenticationFilter --> Authenticating post-process...
- [INFO] 27/sep 23:20:20 http.HttpClientService --> HTTP/1.1 401 UNAUTHORIZED headerParams={auth=AuthenticationFilter}
- [INFO] 27/sep 23:20:20 http.HttpClientService --> ----------ooo----------
- [INFO] 27/sep 23:20:20 http.HttpClientService --> GET /public?param=value
- [INFO] 27/sep 23:20:20 ingfilter.AuthenticationFilter --> Authenticating pre-process...
- [INFO] 27/sep 23:20:20 interceptingfilter.TimeFilter --> Time pre-process: Wed Sep 27 23:20:20
- [INFO] 27/sep 23:20:20 interceptingfilter.Target --> ---------> Executing TARGET.GET /public?param=value
- [INFO] 27/sep 23:20:20 interceptingfilter.TimeFilter --> Time post-process: 0ms
- [INFO] 27/sep 23:20:20 ingfilter.AuthenticationFilter --> Authenticating post-process...
- [INFO] 27/sep 23:20:20 http.HttpClientService --> HTTP/1.1 200 OK headerParams={time=0ms}
- [INFO] 27/sep 23:20:20 http.HttpClientService --> ----------ooo----------

Front Controller

- Motivación
  - En una Aplicación Web se ofrece un gran variedad de peticiones en donde existen necesidades comunes de proceso, esto nos lleva a código duplicado y aumentar la dificultad de mantener robusta la aplicación
- Propósito
  - Centralizar el acceso de las peticiones web provenientes del cliente, mediante un controlador único
- Ventajas
  - El controlador maneja el control de peticiones, incluyendo la invocación de los servicios de seguridad como la autentificación y autorización, la delegación del procesamiento de negocio, el control de la elección de una vista apropiada, el manejo de errores... Permitiendo reutilizar el código entre peticiones diferentes y aumentando la mantenibilidad y reusabilidad del mismo

Front Controller API



Front Controller API

- [INFO] 27/sep 23:18:48 http.HttpClientService --> GET /path1?param=value
- [INFO] 27/sep 23:18:48 http.HttpClientService --> HTTP/1.1 200 OK, body={"name":"Resource1:method1:value"}
- [INFO] 27/sep 23:18:48 http.HttpClientService --> ----------ooo----------

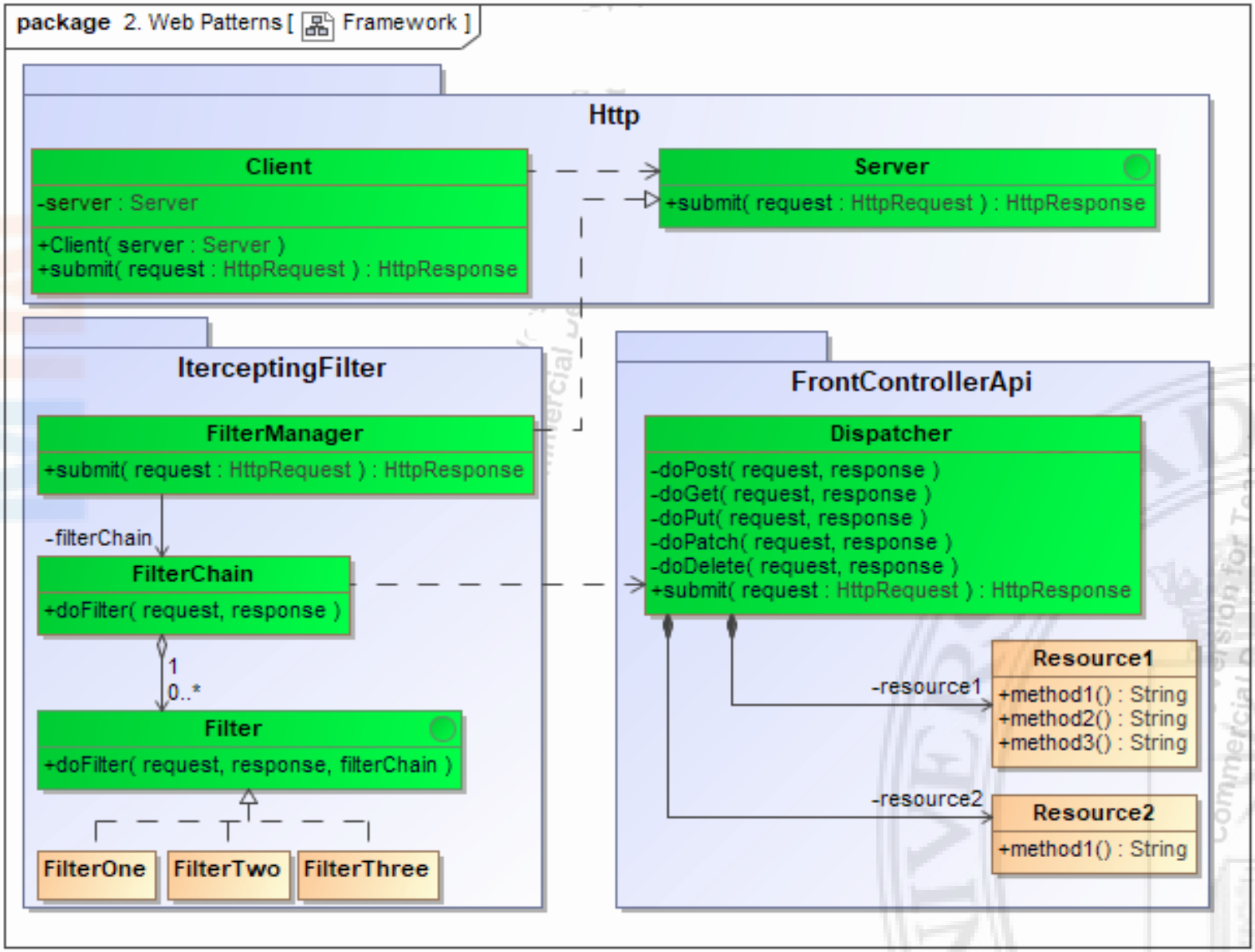
- [INFO] 27/sep 23:18:48 http.HttpClientService --> POST /path1
- [INFO] 27/sep 23:18:48 http.HttpClientService --> HTTP/1.1 200 OK, body={"name":"Resource1:method2"}
- [INFO] 27/sep 23:18:48 http.HttpClientService --> ----------ooo----------

- [INFO] 27/sep 23:18:48 http.HttpClientService --> POST /path1/sub
- [INFO] 27/sep 23:18:48 http.HttpClientService --> HTTP/1.1 200 OK, body={"name":"Resource1:method3"}
- [INFO] 27/sep 23:18:48 http.HttpClientService --> ----------ooo----------

- [INFO] 27/sep 23:18:48 http.HttpClientService --> GET /path2?param=value
- [INFO] 27/sep 23:18:48 http.HttpClientService --> HTTP/1.1 200 OK, body={"name":"Resource2:method1"}
- [INFO] 27/sep 23:18:48 http.HttpClientService --> ----------ooo----------

- [INFO] 27/sep 23:18:48 http.HttpClientService --> GET /no?param=value
- [INFO] 27/sep 23:18:48 http.HttpClientService --> HTTP/1.1 400 BAD_REQUEST, body={"error":"Path Error"}
- [INFO] 27/sep 23:18:48 http.HttpClientService --> ----------ooo----------

Framework API



Servicios Web

- Los servicios Web (WS) son aplicaciones de cliente-servidor que se comunican a través de la Web mediante el protocolo HTTP, intercambiando HTML, XML, JSON...
- Existen dos tipos:
  - Web pesadas. Sobre HTTP, se intercambia XML, bajo el protocolo SOAP
  - Web ligeras . Sobre HTTP y se intercambia principalmente JSON: **API Rest**

API rest

- ***REST*** (*Representational State Transfer*) es un estilo arquitectónico para el desarrollo de aplicaciones sobre la Web. Se basa en la Web tal como está en la actualidad: ***HTTP***. Fue introducido por *Roy T. Fielding* en el año 2000. También referenciado como RESTful. Ejemplos: *Twitter*, *Facebook*, *GitHub*...
- Relación de Cliente-Servidor
- Servidor sin estado. El servidor **no gestiona** los recursos de los clientes. En cada petición el cliente debe enviar toda la información
- Se permite la elección del tipo de representación (HTML, XML, JSON...) a través de los tipos MIME. El cliente decide el tipo MIME
- Se permite la caché. En las respuestas se indica si es cacheable, en tal caso, el cliente lo almacena para no volver a preguntarlo
- El servicio se organiza como un conjunto de recursos con cuatro operaciones (**CRUD**): ***Create*** (post), ***Read*** (get), ***Update*** (put/patch) y ***Delete*** (delete); pero ofrece un servicio por encima de la capa DAO
  - **Seguro**. No cambia su estado ni provoca efectos secundarios
  - **Idempotente**. Operación que repetida siempre da el mismo resultado

Buenas prácticas

- Usar ***SSL*** siempre
- Documenta adecuadamente, con ejemplos sencillos para ejecutar con *Curl* (*http://curl.haxx.se/download.html*)
- Recursos sencillos y autodescriptivos
- Para versiones utilizar **v*** en el nivel mas alto
- Las *Uris* deben ser estables en el tiempo
- Se organiza en **recursos** (sustantivos) y se referencia por *URIs*. Mejor utilizar plurales para los recursos frente a singulares
- Las *Uris* **NO** referencian acciones (post, get, put, patch, delete)
- Las *Uris* **NO** referencian formatos de representación
- Uris**: *spinal-case*** VS *snake_case.* Cuerpo*: **lowerCamelCase** VS snake_case*
- Identificación (id), procurar que no sea deducible, código *hash*
- "/" partes jerárquicas. "," y ";" para partes no jerárquicas, matrices de valores. Ejemplos:
  - `https://api.server.com/v0/orders. Se accede al recurso en su conjunto`
  - `https://api.server.com/v0/orders/vkg23ds5. Se accede a uno concreto`
  - `https://api.server.com/v0/orders/vkg23ds5/name. Se accede al atributo "name" del un recurso concreto`

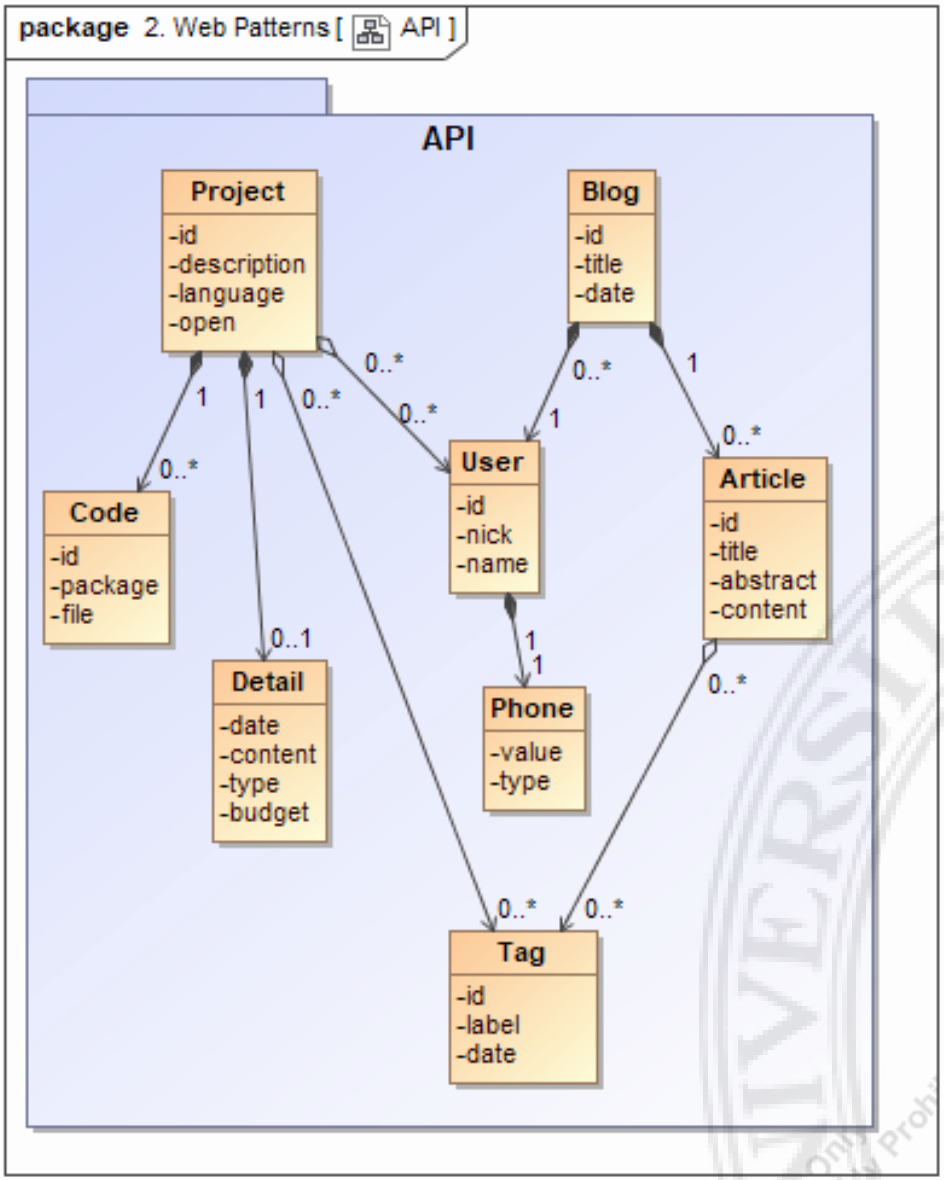
Buenas prácticas

- Controlar la granularidad. Datos que van juntos suministrarlos en una sola llamada. Evitar respuestas demasiadas voluminosas: **GET**
  - Respuestas parciales: ***...?fields=id,description***
  - Paginación: ***...?page=1&size=30***
  - Filtros: ***...?type=3,5***
  - Orden: ***...?sort=name,surname&desc=id***
- Búsquedas: **GET**
  - **`/v0/orders/busquedaCerrada`**
  - **`/v0/orders/search?q=name:jes`**
  - **`/v0/search/repositories?q=tetris+language:java&sort=stars&order=desc`**
- Atributo boolean: **post:true, delete:false**
- Utilizar el dominio cruzado: **CORS**
- Utilizar mensajes de error: mensaje corto, mensaje largo y url del api con descripción completa
- Seguridad: Header + SSL
  - nombre:clave
  - token (JWT)
- OAuth 2 debería ser usado para facilitar la transferencia del token seguro a terceros
- HATEOAS (Hypermedia As The Engine Of Application State), significa que se deben devolver las URIs completas de los recursos asociados a la petición, en forma de hipervínculos. Existen opiniones enfrentadas
- *Pretty print* por defecto y asegura que *gzip* sea soportado

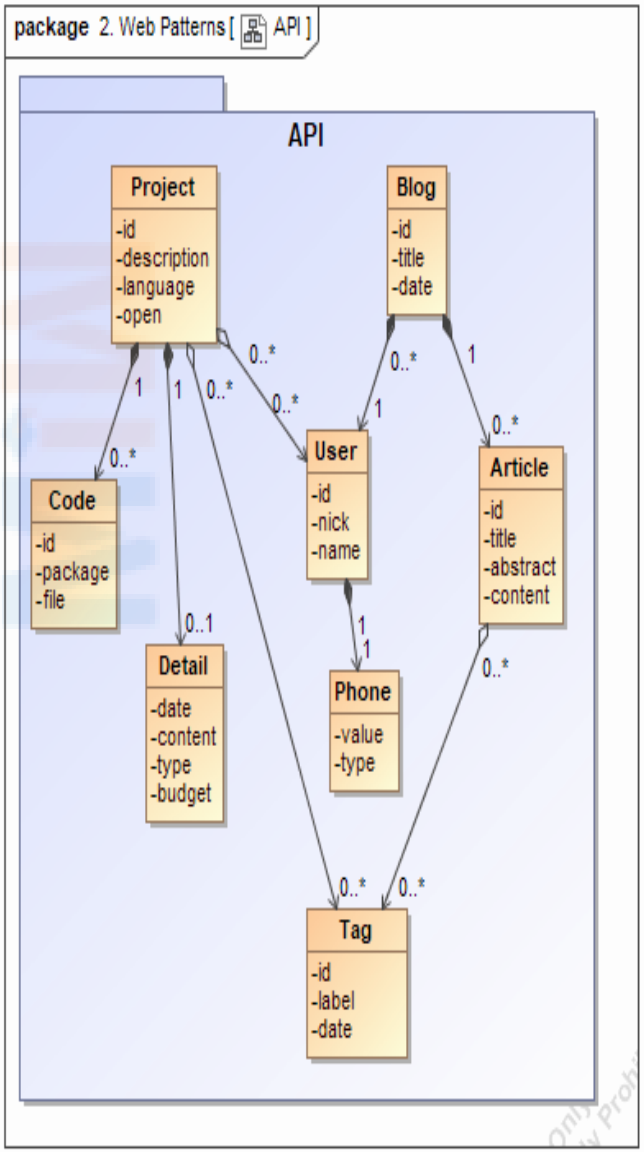
Operaciones: CRUD

- Métodos de HTTP:
  - **Create: post**. Para crear un nuevo recurso
    - Operación no segura y no idempotente
    - Respuesta: 201 (OK) o el tipo de error
    - Recomendable devolver una referencia del nuevo recurso
    - Recomendable devolver el recurso
  - **Read: get**. Para obtener un recurso o un conjunto
    - Operación segura e idempotente
    - Respuesta: 200 (OK) o el tipo de error
    - Devolver el recurso o un conjunto
  - **Update: put/patch**. Para actualizar un recurso
    - Operación no segura e idempotente
    - Respuesta 200 (OK) o el tipo de error
    - Recomendable devolver el recurso
  - **Delete: delete**. Para eliminar un recurso
    - Operación no segura e idempotente
    - Respuesta 204 (OK) sin contenido o el tipo de error

Ejercicio


package 2. Web Patterns [API]
API
Project
-id
-description
-language
-open
Blog
-id
-title
-date
Code
-id
-package
-file
Detail
-date
-content
-type
-budget
User
-id
-nick
-name
Phone
-value
-type
Article
-id
-title
-abstract
-content
Tag
-id
-label
-date
1
0..*
0..1

Ejercicio



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

- **GET /users** *response*: [{id, nick}]
- **POST /users** *request*: {nick, name…}

- **GET /users/{id}** *response*:{id, nick, name, phone{value, type}}
- **PUT /users/{id}** *request*:{…}
- **PATCH /users/{id}** *request*:{nick:new-nick, phone/value:new-value}
  Son un conjunto de operaciones atómicas
- **DELETE /users/{id}**

- **GET /blogs** *response*:{id,title}
- **DELETE /blogs/{id}**
- **GET /blogs/{id}/user**
- **POST /blogs** *request*: {…}
- **GET /blogs/{id}/articles/{id}/abstract** *response*:{abstract}
- **GET /tags** *response*:{…}

- **GET /projects/search?*language=java&sort=id***
- **GET /projects/{id}/tags?*sort=-date,label***
- **GET /projects?*fields=language,description***
- **GET /users?*page=2&size=20***
- **GET /search?*q=language:java+language:typescript&sort=id***
- **GET /projects/opened**
- **GET /projects/search?q=tag:id**
- **GET/projects/{id}/open**

Introducción a Spring

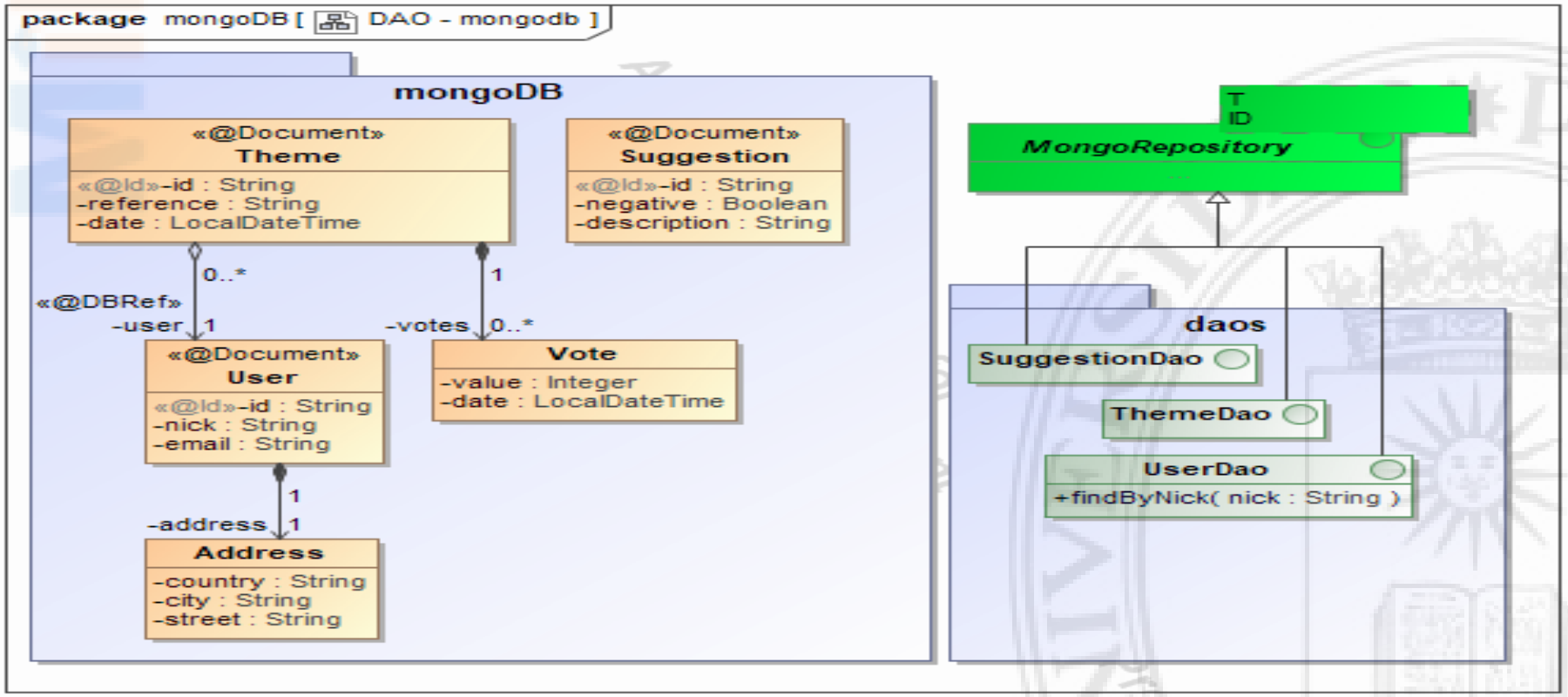
- Dependencias
  - API: *spring-boot-starter-web*
  - Logs: *spring-aspects*
  - BD monoDB: *spring-boot-starter-data-mongodb*
  - BD embebida: *de.flapdoodle.embed.mongo*
  - Programación rectiva: *spring-boot-starter-webflux*
  - Cliente Swagger: *springfox-swagger2, springfox-swagger-ui*
  - Tests: *spring-boot-starter-test*
  - Cliente http: *reactor-test*
- Configuración
  - *application.properties*
  - *SwaggerConfig*
  - *ApiLogs*

Introducción a Spring

- Arranque
  - *Application*
- *API*
  - *@RestController, @RequestMapping("/users")*
    - *@PostMapping*
    - *@GetMapping, @ GetMapping("/{id}")*
    - *@PutMapping("/{id}"), PatchMapping("/{id}")*
    - *@DeleteMapping("/{id}"), ~~@DeleteMapping~~*
      - *@PathVariable String id*
      - *@RequestBody Dto dto*
      - *@RequestParam String q*
      - *@RequestHeader Integer value*
- *Inyección*
  - *@Autowired: en el constructor*

Introducción a Spring

- Persistencia: patrón DAO
  - *Entidades, Documentos*
    - *@Document*
      - *@Id*
      - *@DBRef*
  - *Daos, repositorios*
    - *UserDao extends MongoRepository<User, String>*



Introducción a Spring

- *@ApiTestConfig* TestClass {}
  - *@Autowired* private WebTestClient **webTestClient;**
  - SuggestionDto body = this.webTestClient
    .post().uri(SuggestionResource.SUGGESTIONS)
    .body(BodyInserters.fromObject(suggestionDto))
    .exchange()
    .expectStatus().isOk()
    .expectBody(SuggestionDto.class)
    .returnResult().getResponseBody();
  - List<SuggestionDto> body = this.webTestClient
    .get().uri(SuggestionResource.SUGGESTIONS)
    .exchange()
    .expectStatus().isOk()
    .expectBodyList(SuggestionDto.class);
  - this.webTestClient
    .put().uri(ThemeResource.THEMES + ThemeResource.ID_ID, "no")
    .body(BodyInserters.fromObject(dto))
    .exchange()
    .expectStatus().isEqualTo(HttpStatus.NOT_FOUND);