



POLITÉCNICA

"Ingeniamos el futuro"

CAMPUS  
DE EXCELENCIA  
INTERNACIONAL



25/09/2019 •

# Arquitectura y Patrones para Aplicaciones Web

## Arquitecturas Web

[\*https://github.com/miw-upm/apaw\*](https://github.com/miw-upm/apaw) (Teoría)

[\*https://github.com/miw-upm/apaw-ep-themes\*](https://github.com/miw-upm/apaw-ep-themes)

[\*https://github.com/miw-upm/apaw-microservice-themes-theme\*](https://github.com/miw-upm/apaw-microservice-themes-theme)

[\*https://github.com/miw-upm/apaw-microservice-themes-user\*](https://github.com/miw-upm/apaw-microservice-themes-user)

[\*https://github.com/miw-upm/apaw-microservice-themes-suggestion\*](https://github.com/miw-upm/apaw-microservice-themes-suggestion)

# Arquitectura

## ■ Definición

- “La organización fundamental de un sistema incorporado en sus **componentes**, sus **relaciones** con otros y su **entorno** y las **principales guías** de su diseño y evolución” [IEEE]

## ■ Diseño Orientado a Objetos

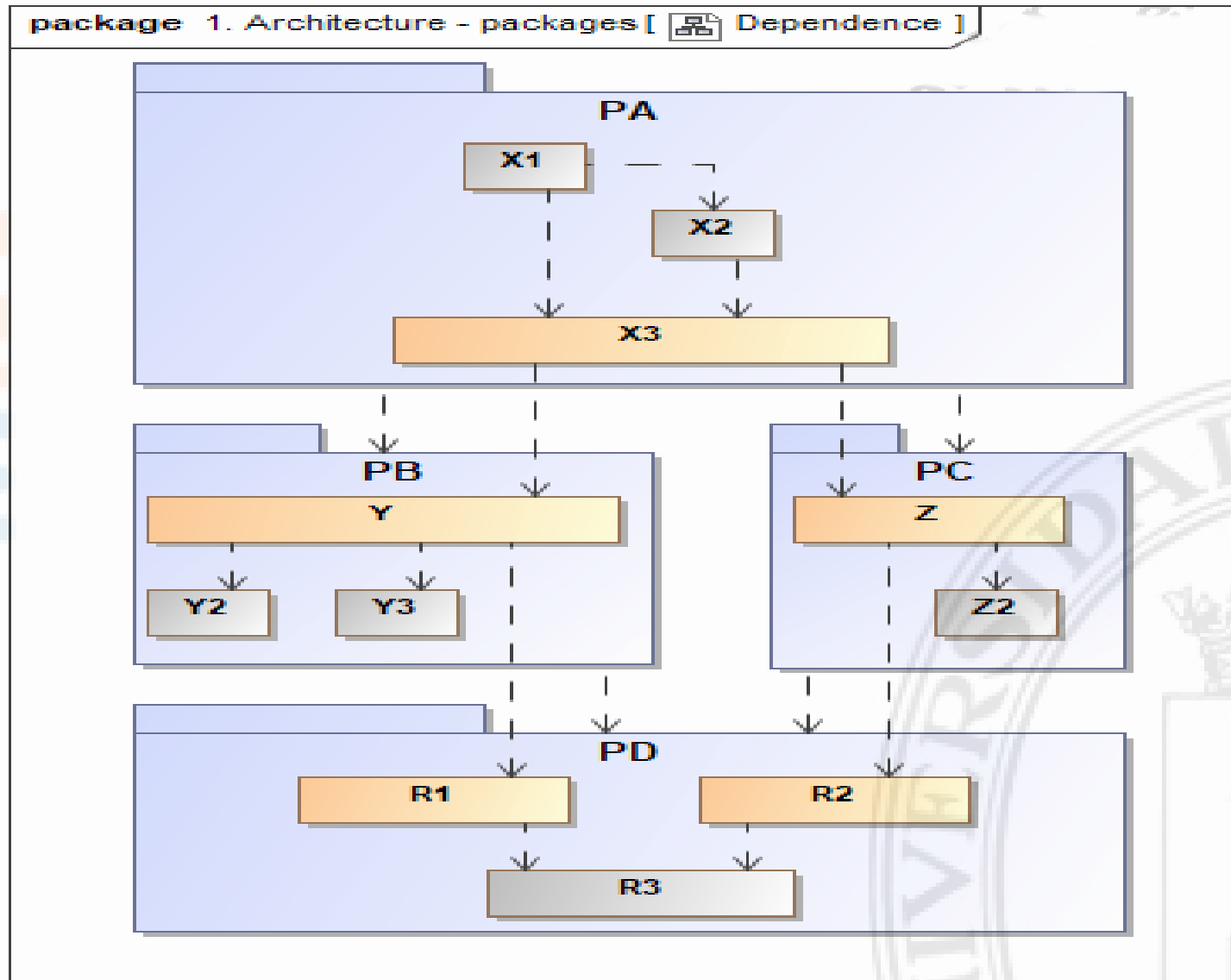
- Clases: patrones

## ■ Arquitectura

- **Paquetes**
  - Alta cohesión
  - Muy Bajo acoplamiento
- **Componentes Software: artefactos**
  - Despliegue: versionado



# Dependencias



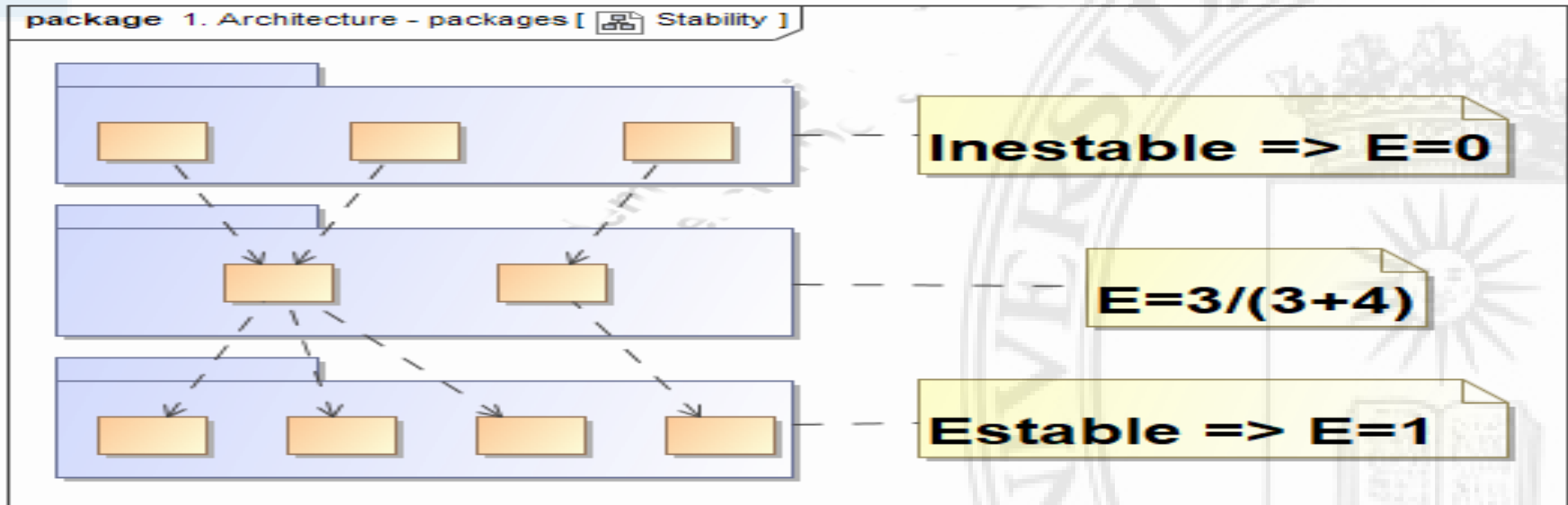
## Principios: Robert Martin

- Principio de Equivalencia entre Reusabilidad y Entregable
  - Se reusa código, si y sólo si, nunca se necesita mirar el código fuente, que no sean sus partes públicas.
  - El autor es el responsable de su mantenimiento y distribución. Se necesita control de versiones.
- Principio de Reusabilidad Común
  - Las clases de un paquete se reutilizan juntas. Si se reusa una de las clases de un paquete, se reusan todas.
- Principio de Cierre Común
  - Las clases de un paquete deberían encerrarse juntas contra la misma clase de cambios. Un cambio que afecta a un paquete, afecta a todas las clases del paquete.
- Principio de Dependencias acíclicas
  - La estructura de dependencias entre los paquetes debe ser un grafo dirigido acíclico. Es decir, no debe haber ciclos en la estructura de dependencias

## Principios: Robert Martin

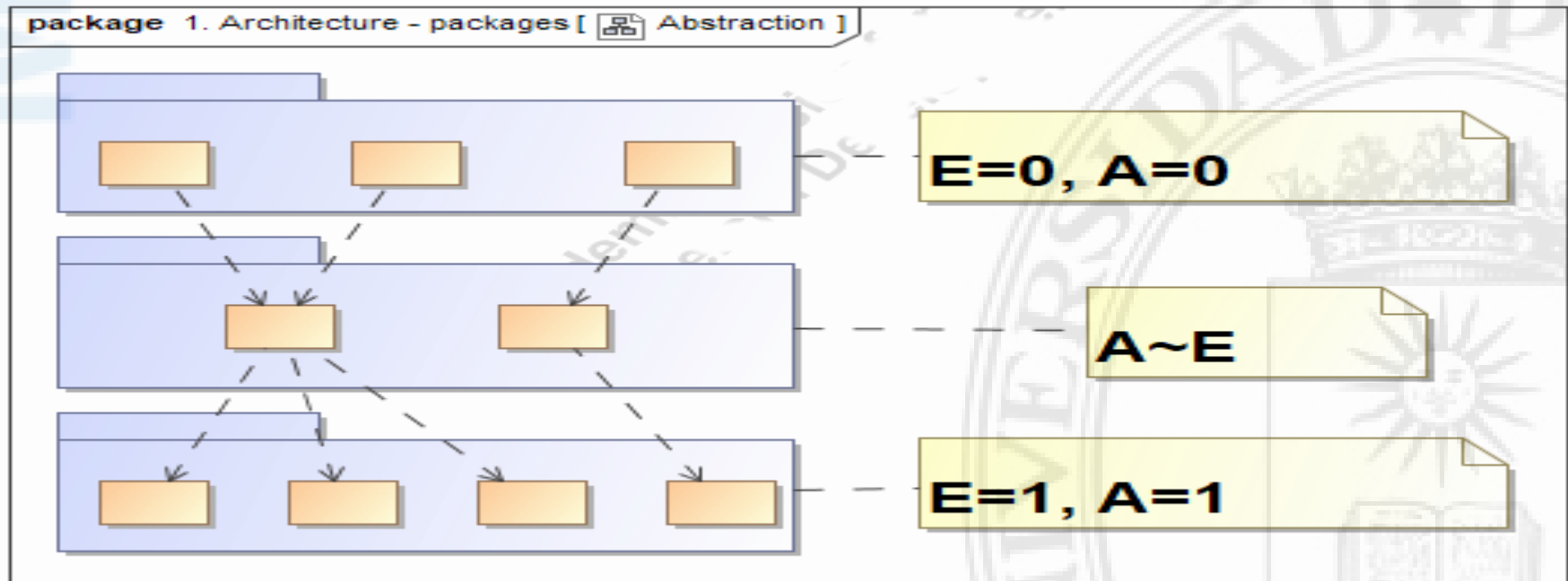
### ■ Principio de Dependencias Estables

- 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
  - $Aa$  - 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 - Estabilidad



# Principios: Robert Martin

- Principio de Abstracciones Estables
  - $Abstracción = \frac{Ca}{Ct}$ 
    - $Ca = N^{\circ}$  de clases abstractas
    - $Ct = N^{\circ}$  total de clases
- Abstracción debe ser  $\geq$  que la anterior, en cada capa
- Estabilidad  $\approx$  Abstracción



## Modelo de Aplicación Web

- Modelo-Vista-Controlador (**MVC**) es un estilo de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos.
- Smalltalk-80 (1979), GUI, Web, Frameworks...
  - MVC, MVP, MVP-PV, MVP-SC, MVP-PM, MVVM, **MV\***
- Web
  - Vista procesada en servidor
    - PERL, PHP, JSP, .NET..., *JEE, Struts, JSF, Spring, Symfony...*
  - Vista procesada en cliente (*JS*)
    - *Angular, React...*
- Estructurada por Capas.
  - Capa de Presentación.
  - Capa de Negocio.
  - Capa de Datos.

# Modelo de Aplicación Web

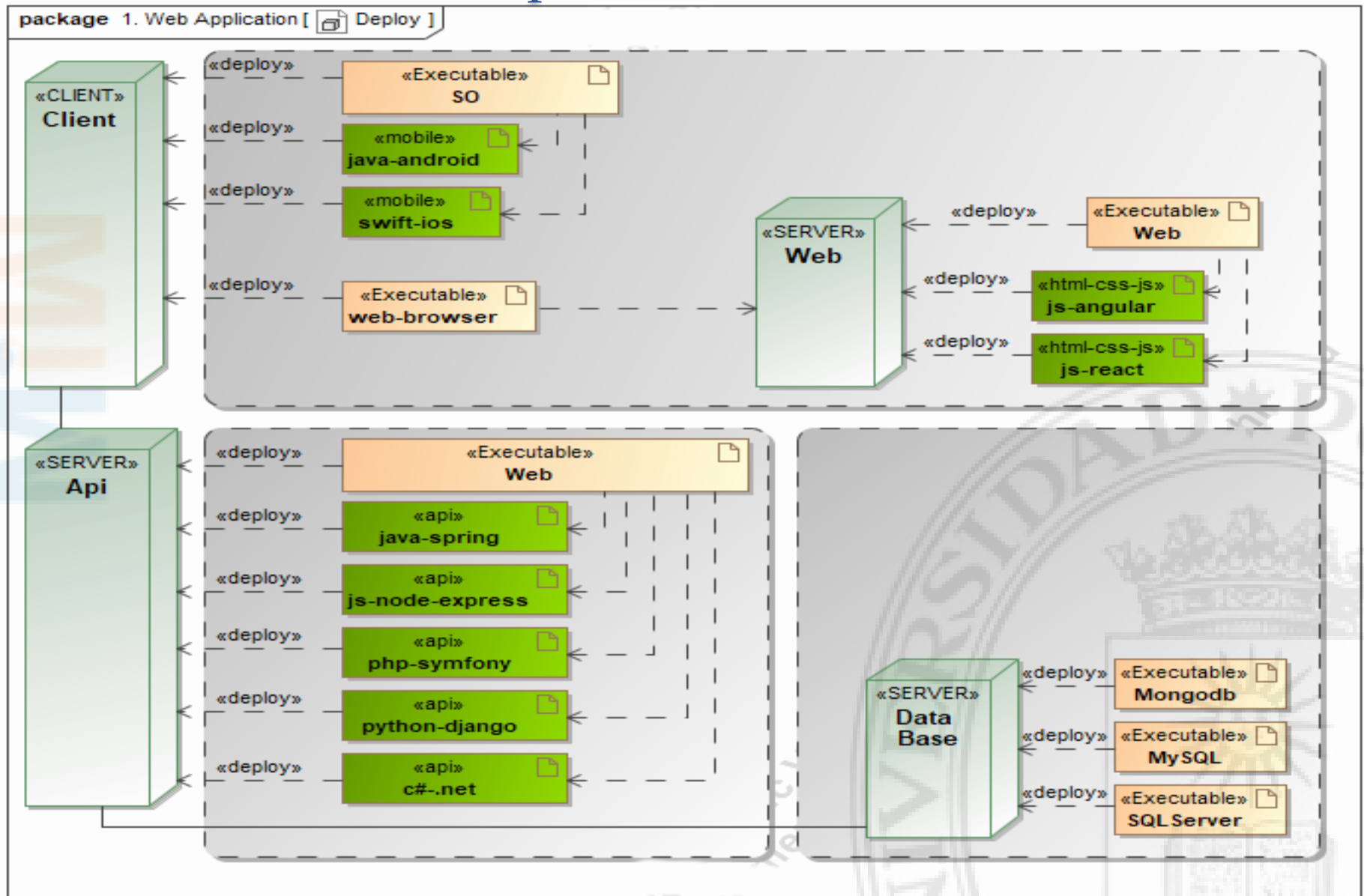
- Modelo distribuido multitarea. Esta dividido en componentes software (artefactos) dependiendo de su función e instalado en diferentes máquinas.
- Capa lógica-*Layer* vs Capa física-*Tier* (*deployment*).
- Estructurada por Capas. Existen varias organizaciones, dependiendo de los autores o de las tecnologías
  - **Presentation Layer**
    - Tendencia actual a ejecutarse en cliente
      - Framework JS
      - Mobile
  - **Business Layer**
    - SOAP (*Simple Object Access Protocol*). Es un protocolo para el intercambio de información estructurada (*HTTP*)
      - Servicios financieros, puertas de pago, servicios de telecomunicaciones.
    - REST (*Representational State Transfer*) (*HTTP*)
      - Redes sociales, chats, servicios de móviles...
    - GraphQL (*Query language*) (*Facebook, open-source 2015*)
      - Redes sociales, chats, servicios de móviles...
  - **Data Layer**
    - SQL
    - NoSQL



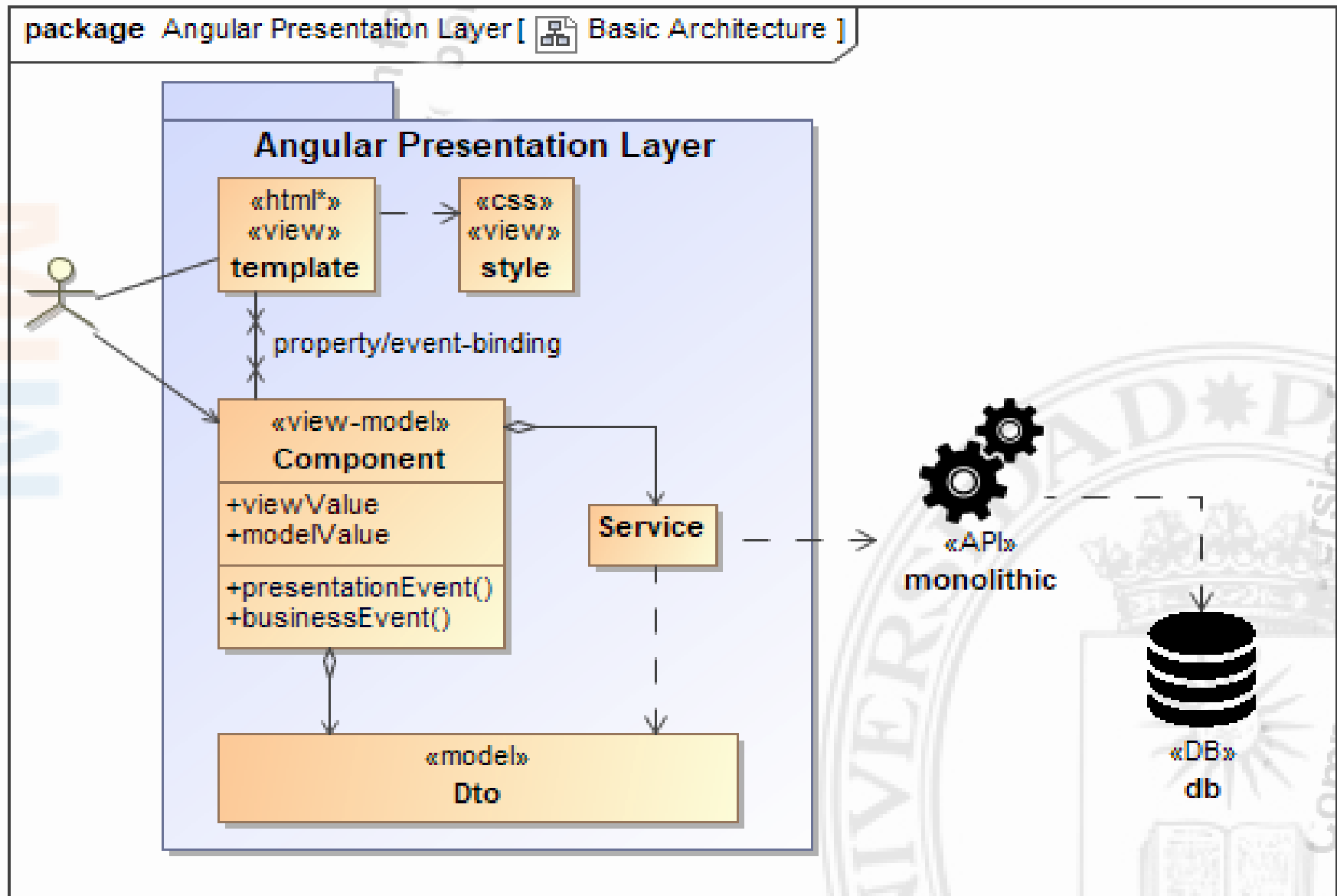
# API

- API (*Application Programming Interfaces*)
- OpenAPI (*API REST*)
  - Copyright © 2016 The Linux Foundation®.  
<https://www.openapis.org/>
  - Swagger. Son herramientas *open-source* para *OpenAPI*.  
<https://swagger.io/>
- GraphQL
  - Es un lenguaje para especificar *API's* y realizar peticiones
- Tendencias...
  - Asynchronous API
  - *OpenId*. Es un estándar de identificación digital descentralizado, con el que un usuario puede identificarse en una página web a través de una URL. <https://openid.net/>
  - *API Rest* vs *GraphQL*

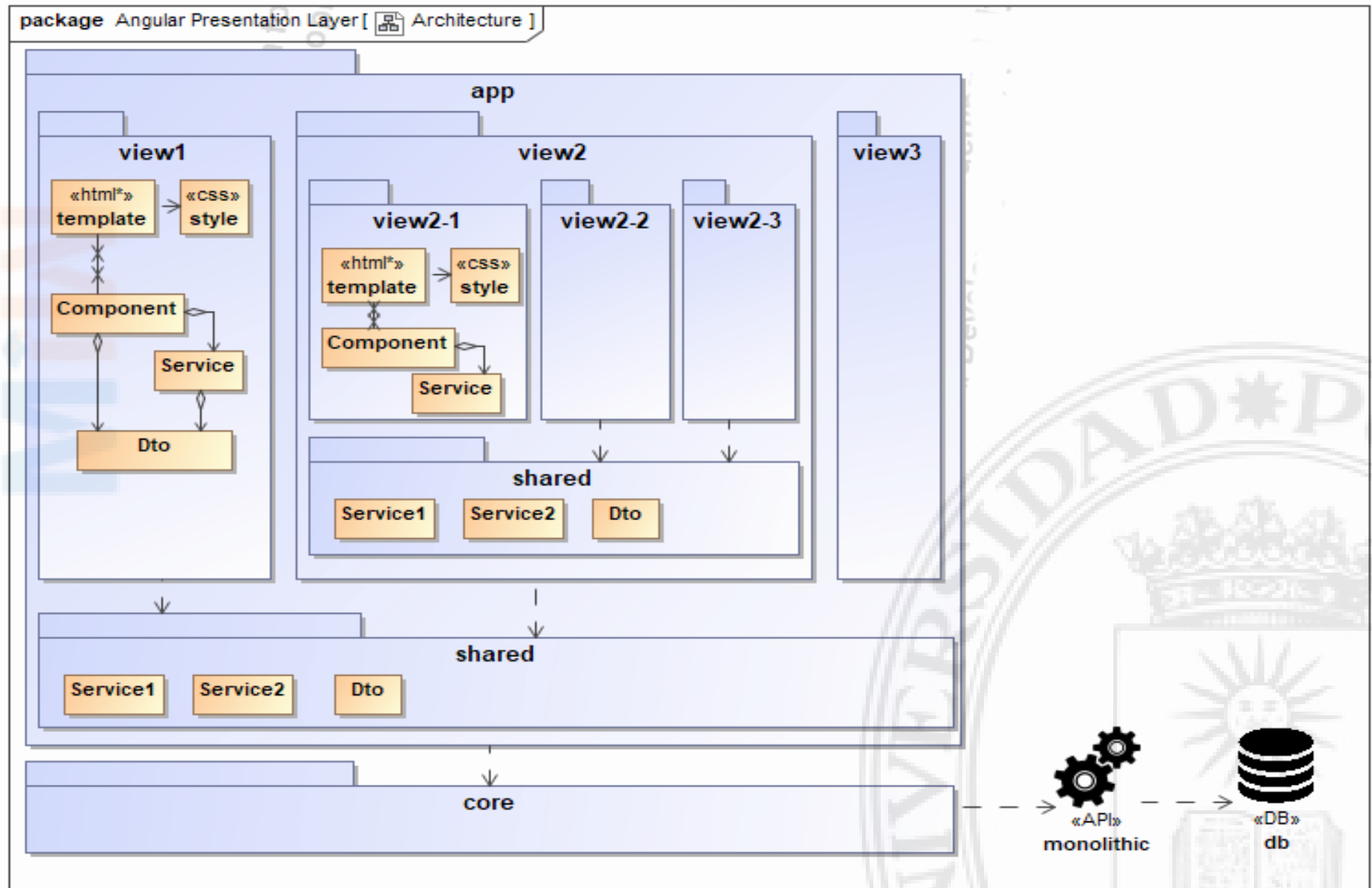
# Aplicación Web



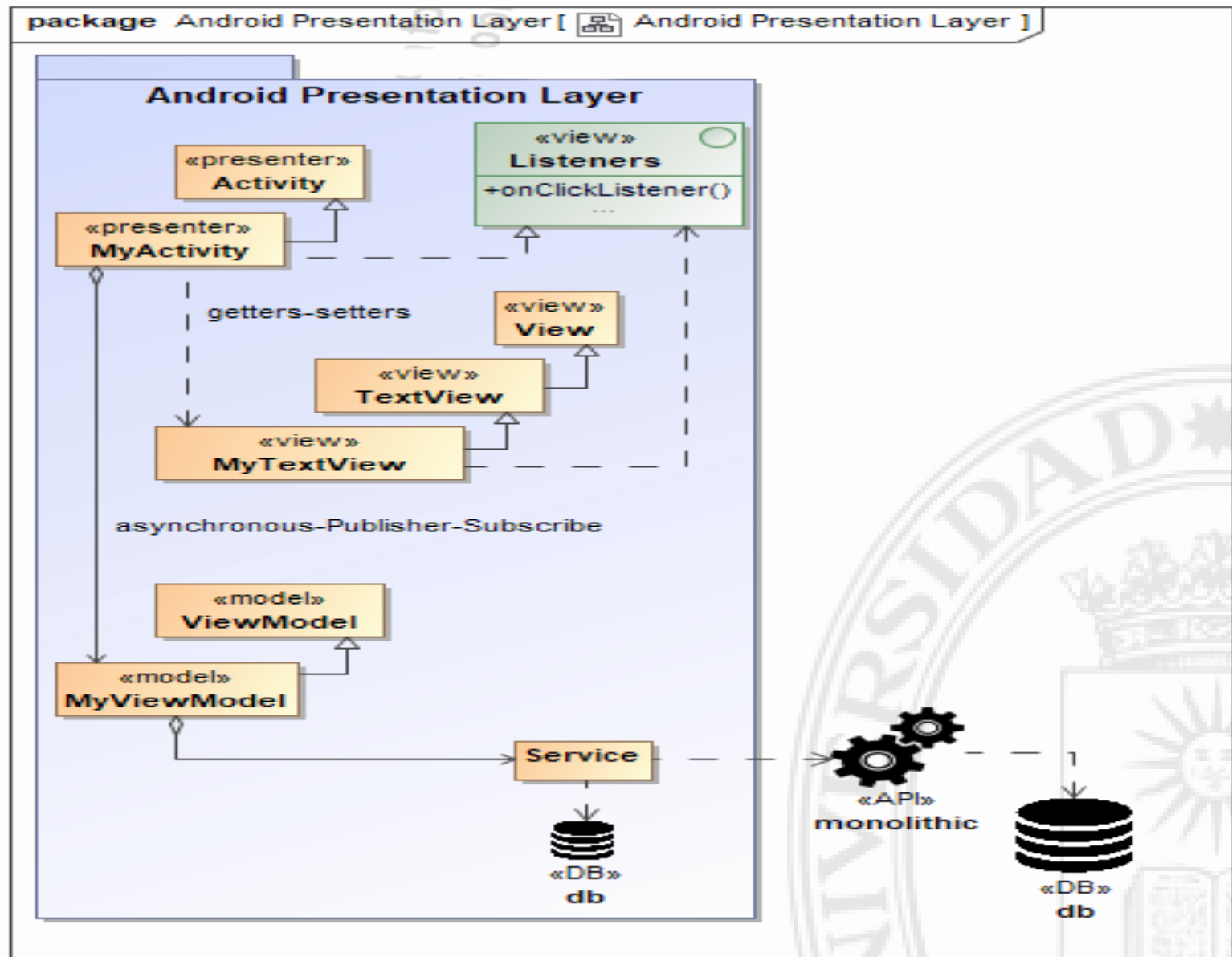
# Angular: arquitectura



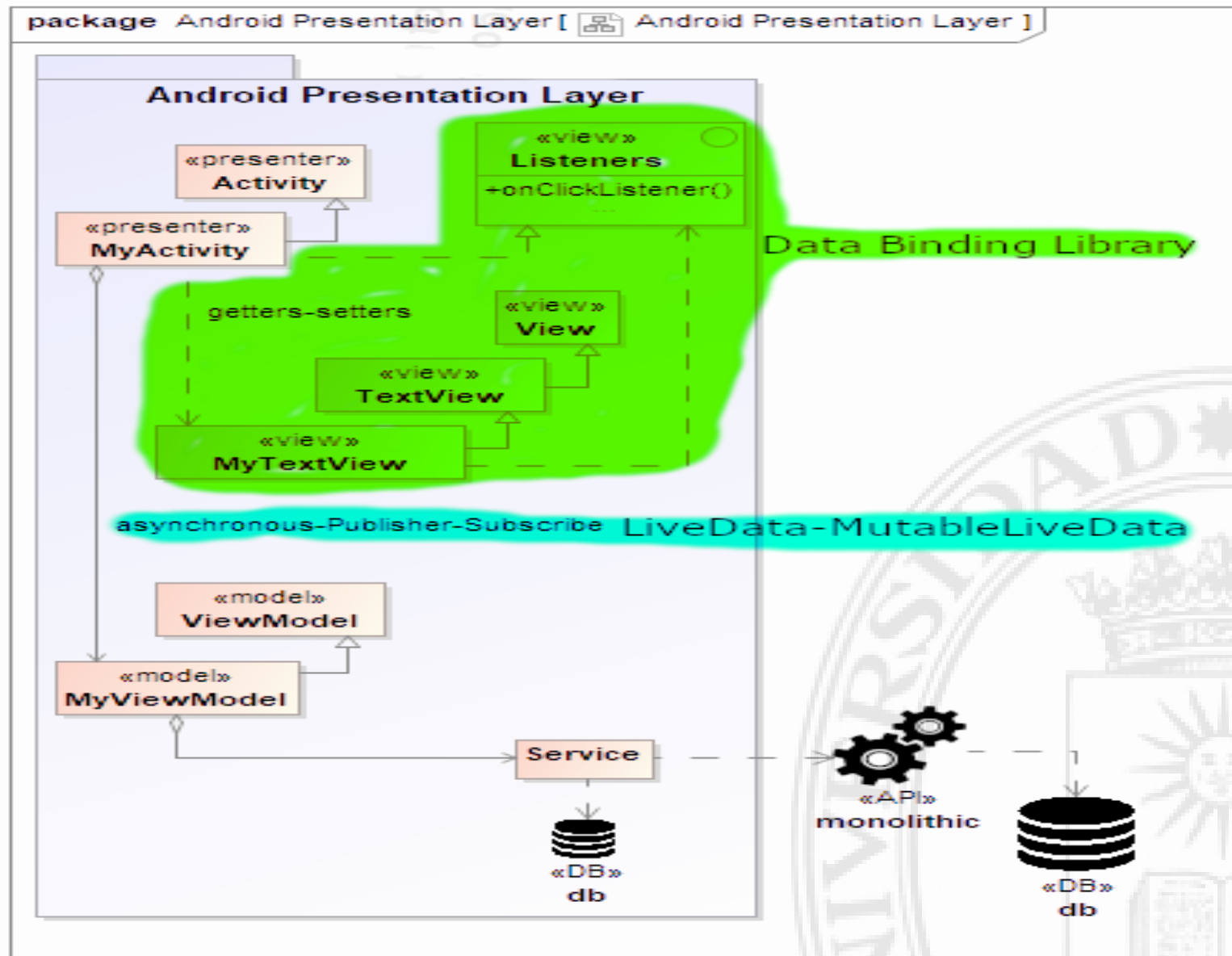
# Angular: arquitectura



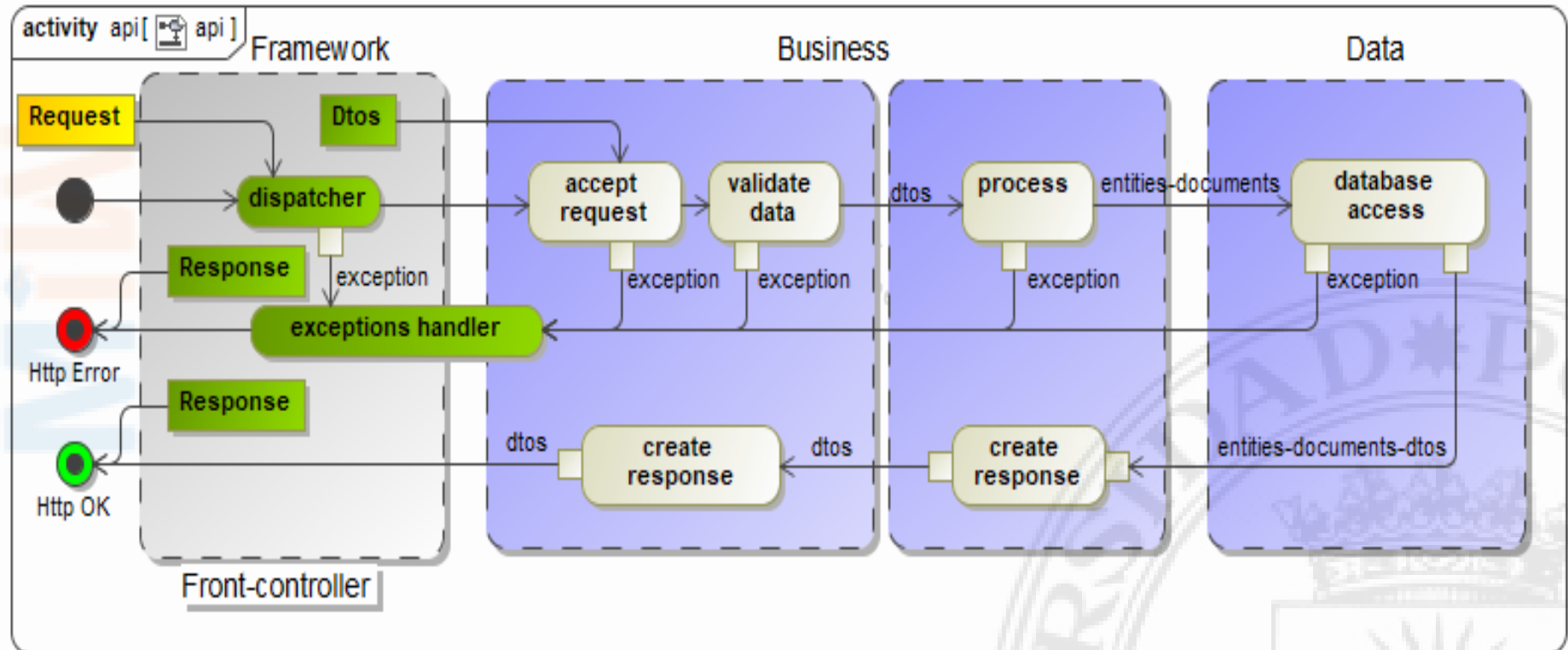
# Android: arquitectura



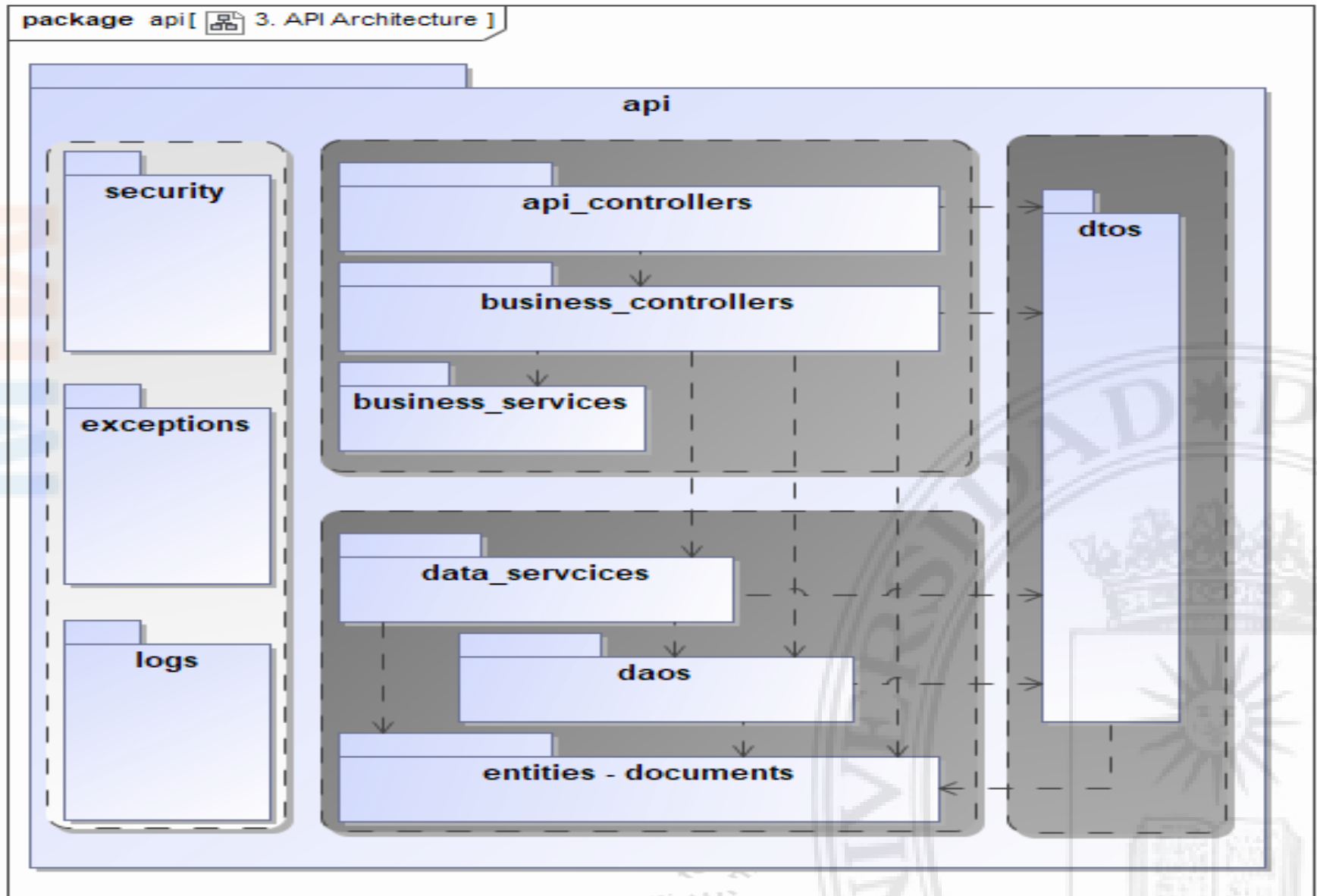
# Android: arquitectura



# API Rest. Flujo de datos

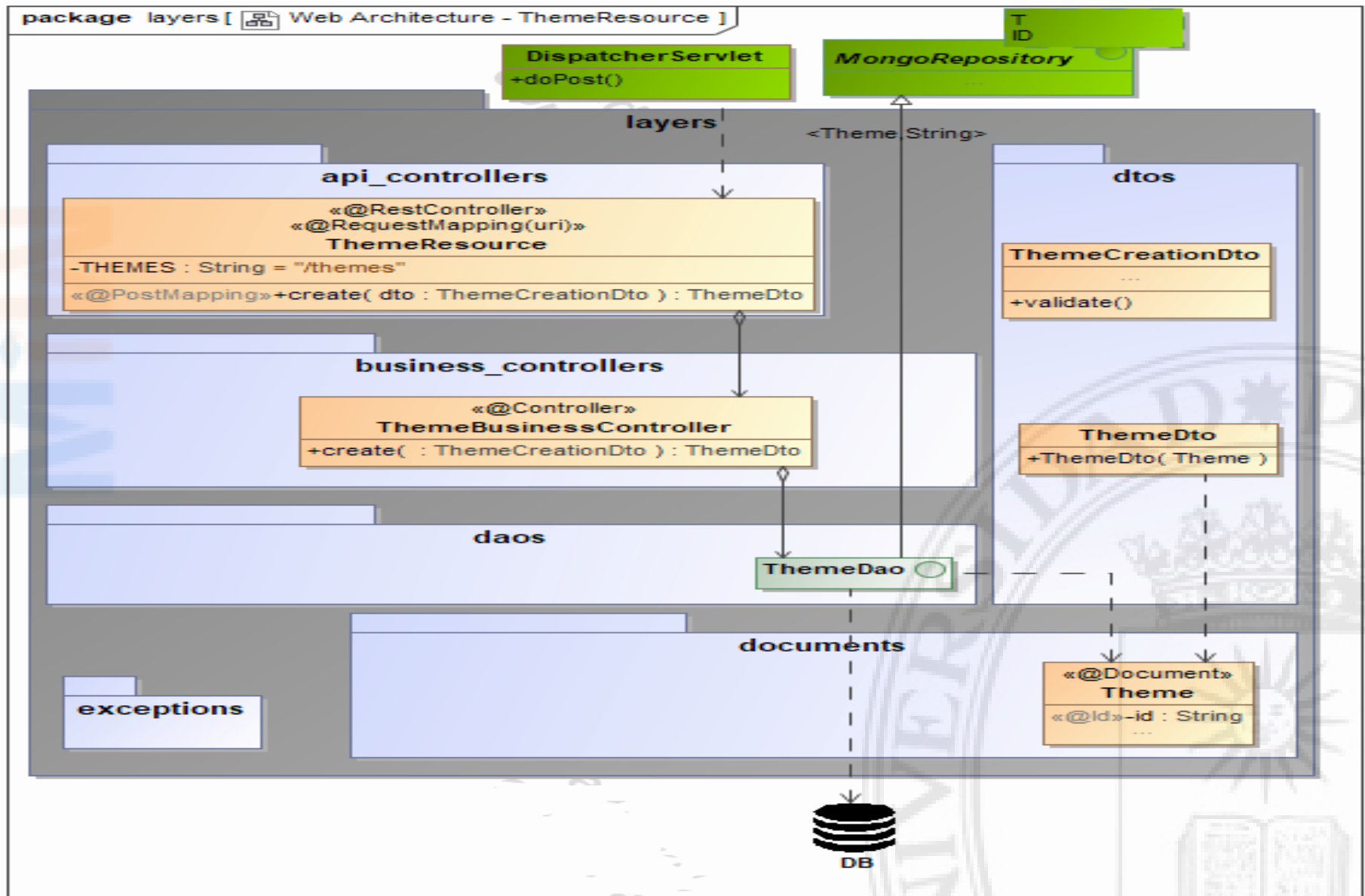


# Arquitectura Web. Layers






# Arquitectura Web. Layers

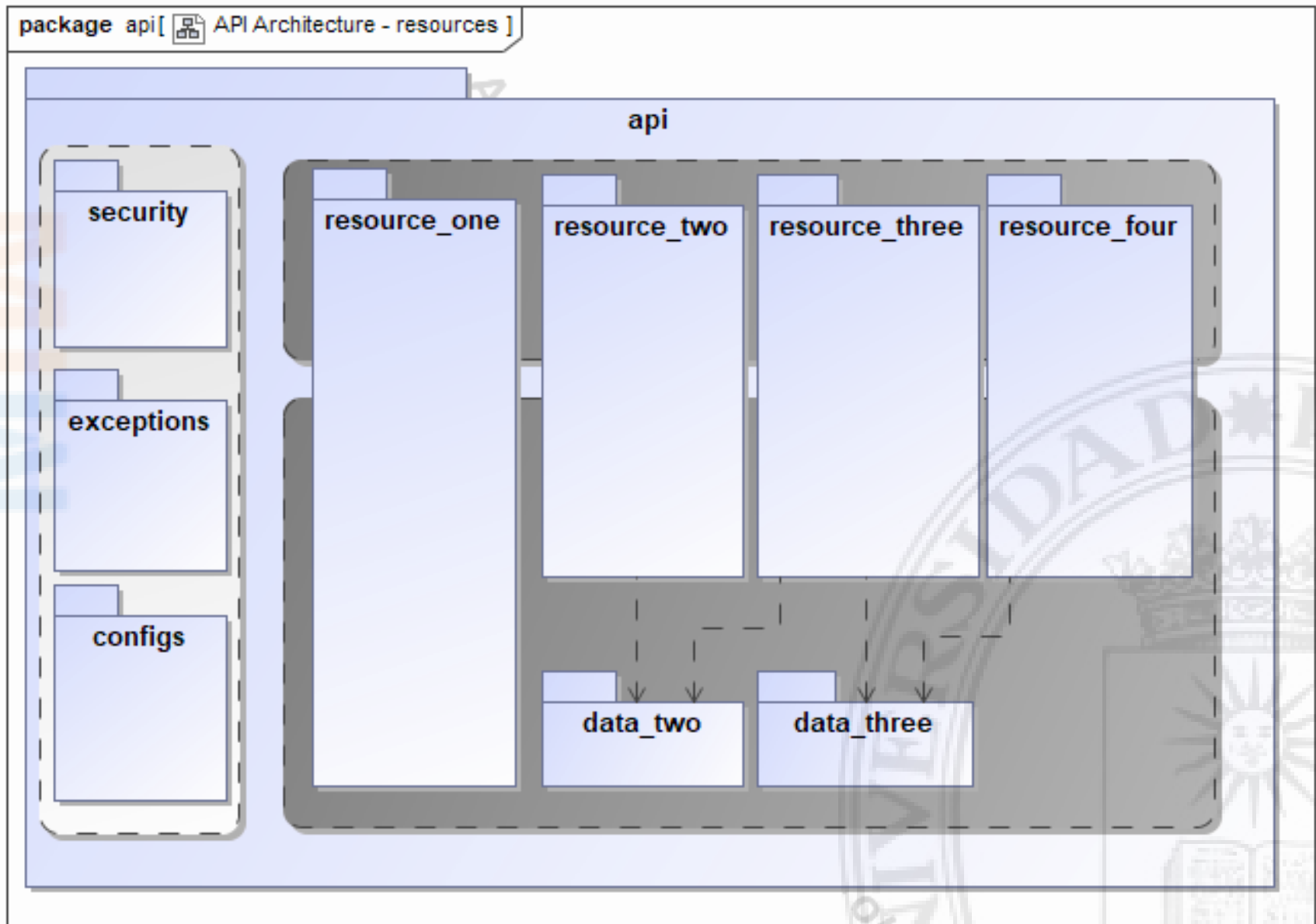


# Arquitectura Web. Layers

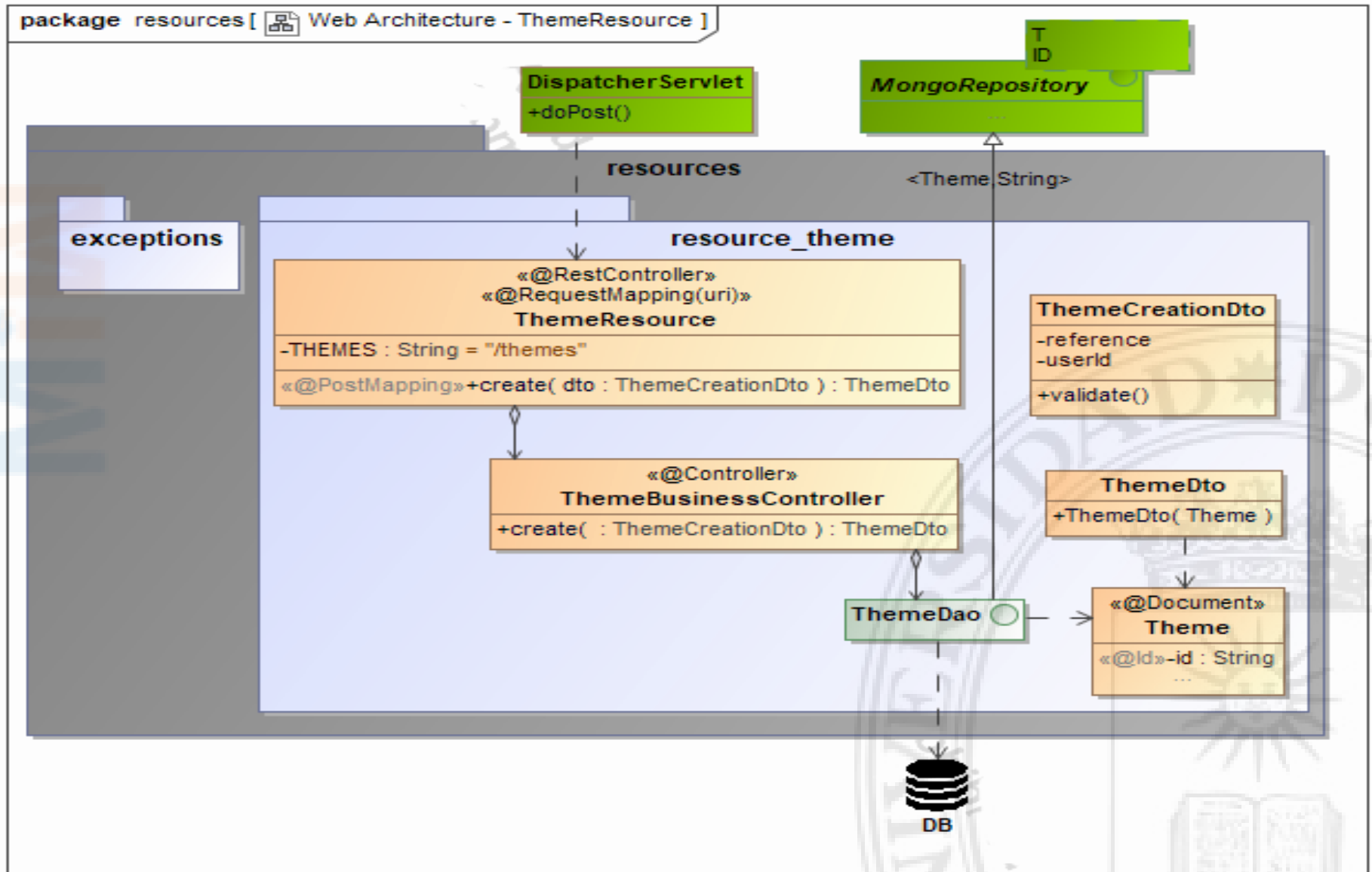
- Proyecto Themes: Architecture

-  <https://github.com/miw-upm/apaw-ep-themes>
  - Commit: 29f353b2 on 16/08/2019 at 22:34 (release-1.1)

# Arquitectura Web. Resources




# Arquitectura Web. Resources




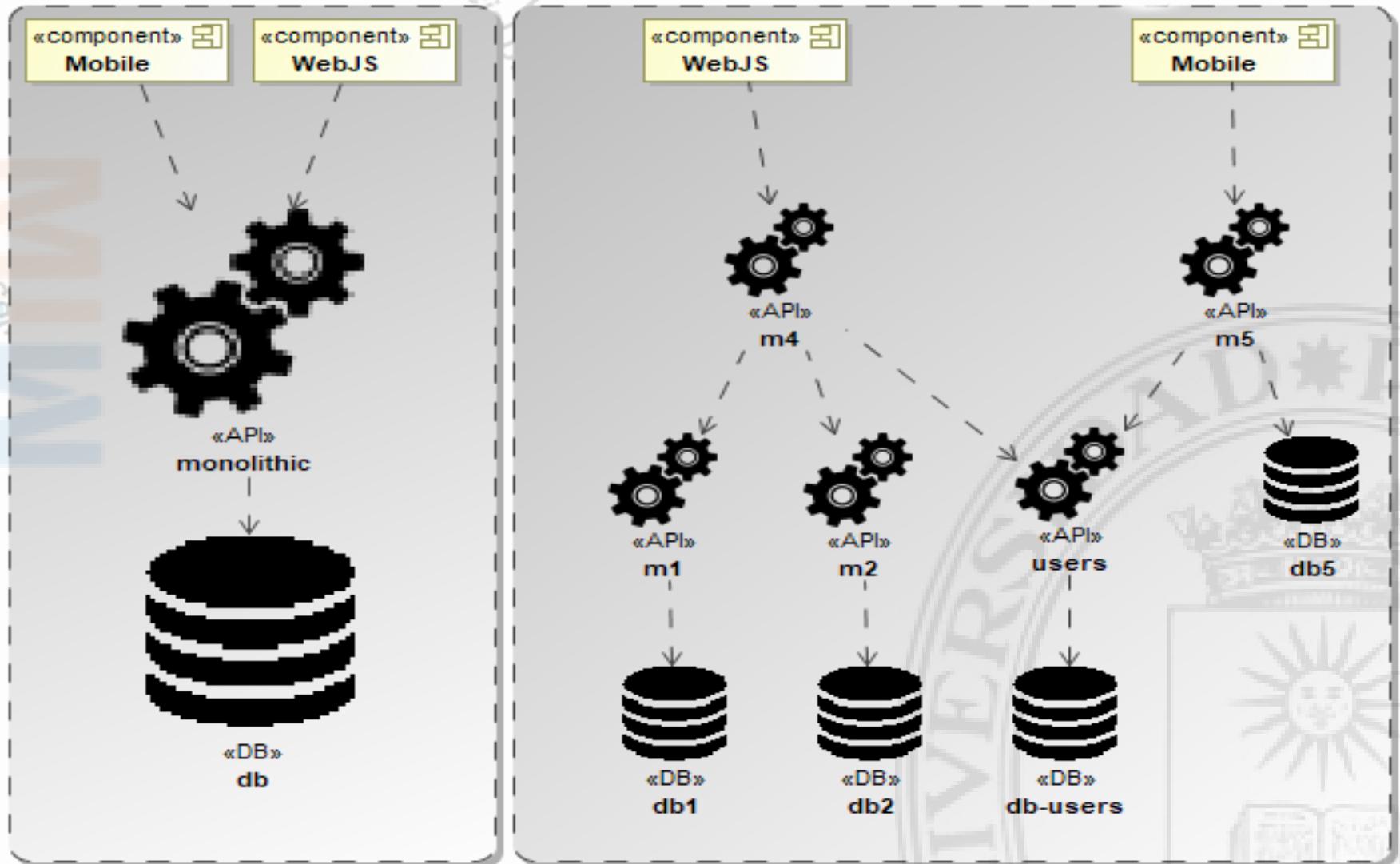
## Arquitectura Web. Resources

- Proyecto Themes: Architecture

-  <https://github.com/miw-upm/apaw-ep-themes>
  - Commit: 46dbd78c on 25/09/2019 at 15:46 (release-2.2)

# Monolítica VS Micro Servicios

package 4. Microservices Architecture [  Monolithic-Microservices ]



## Aplicación Monolítica

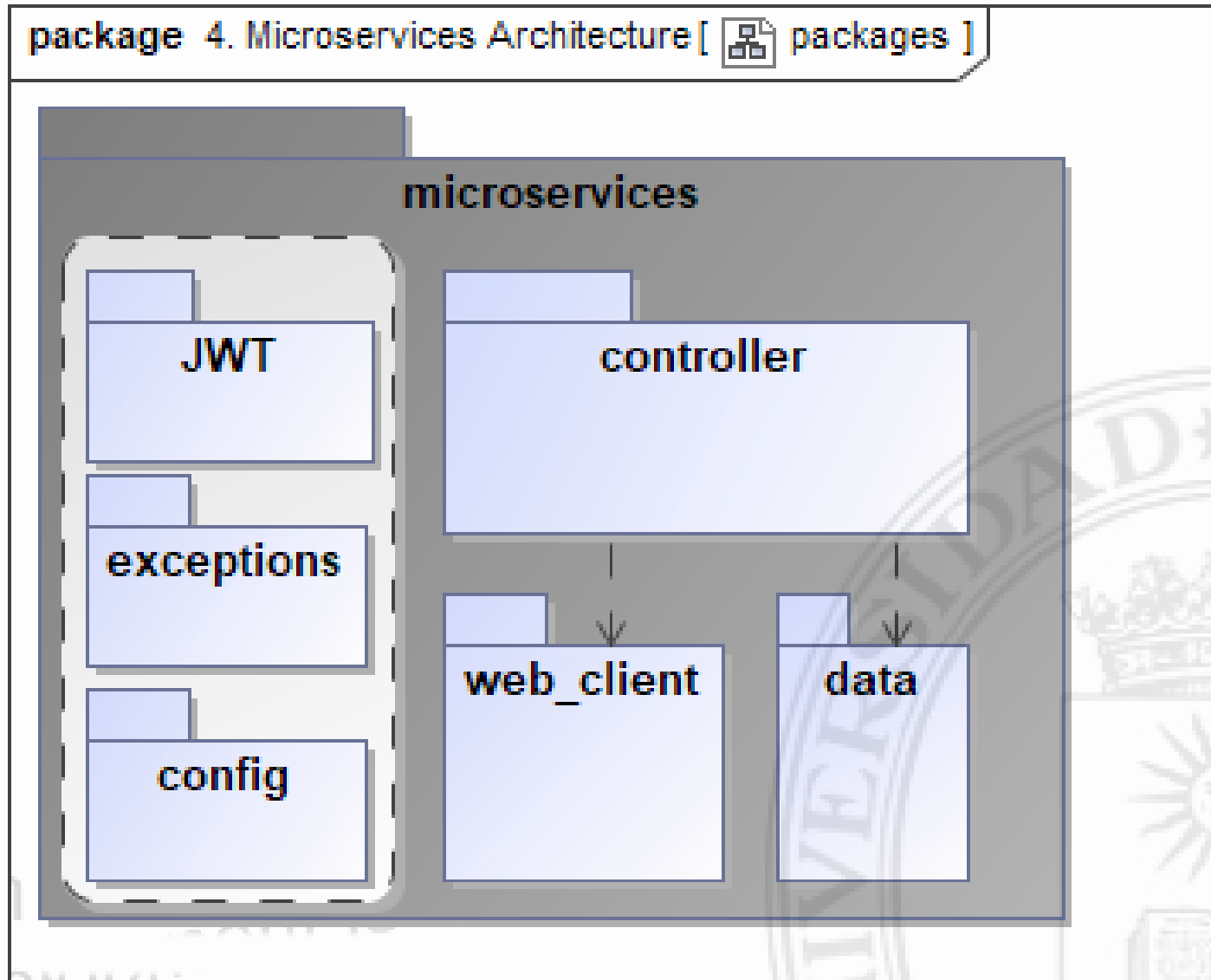
- Capa de negocio: *Stateless API Rest*
- Aplicación Monolítica
  - Un solo API Rest: fácil despliegue
  - Fácil gestión de versiones
  - Complejidad aumenta en el tiempo: pérdida de producción
  - Grandes equipos de desarrollo: especialización y complejidad en la interacción entre los equipos
  - Replicación costosa con difícil balanceo de carga
  - Grandes BD: migraciones complejas
  - Difícil reusabilidad de partes del proyecto
  - Una sola tecnología y difícil migración tecnológica

# Arquitectura Web. Micro Servicios

- Capa de negocio: *Stateless API Rest*
- Ventajas
  - Fuertemente modular
    - Permite tener equipos mas pequeños, multidisciplinarios
    - Filosofía de productos y no proyectos
    - Descentralización de BD y elimina la integración de BD
  - Despliegues independientes
    - Permite la evolución rápida y con menor riesgo
    - Replica de 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 compleja su despliegue
  - Consistencia
    - Es más complicado mantener la consistencia del sistema
  - Complejidad operacional
    - Se necesita acceder a varios servicios para realizar una operación
- <https://martinfowler.com/articles/microservices.html>
- <https://microservices.io/index.html>



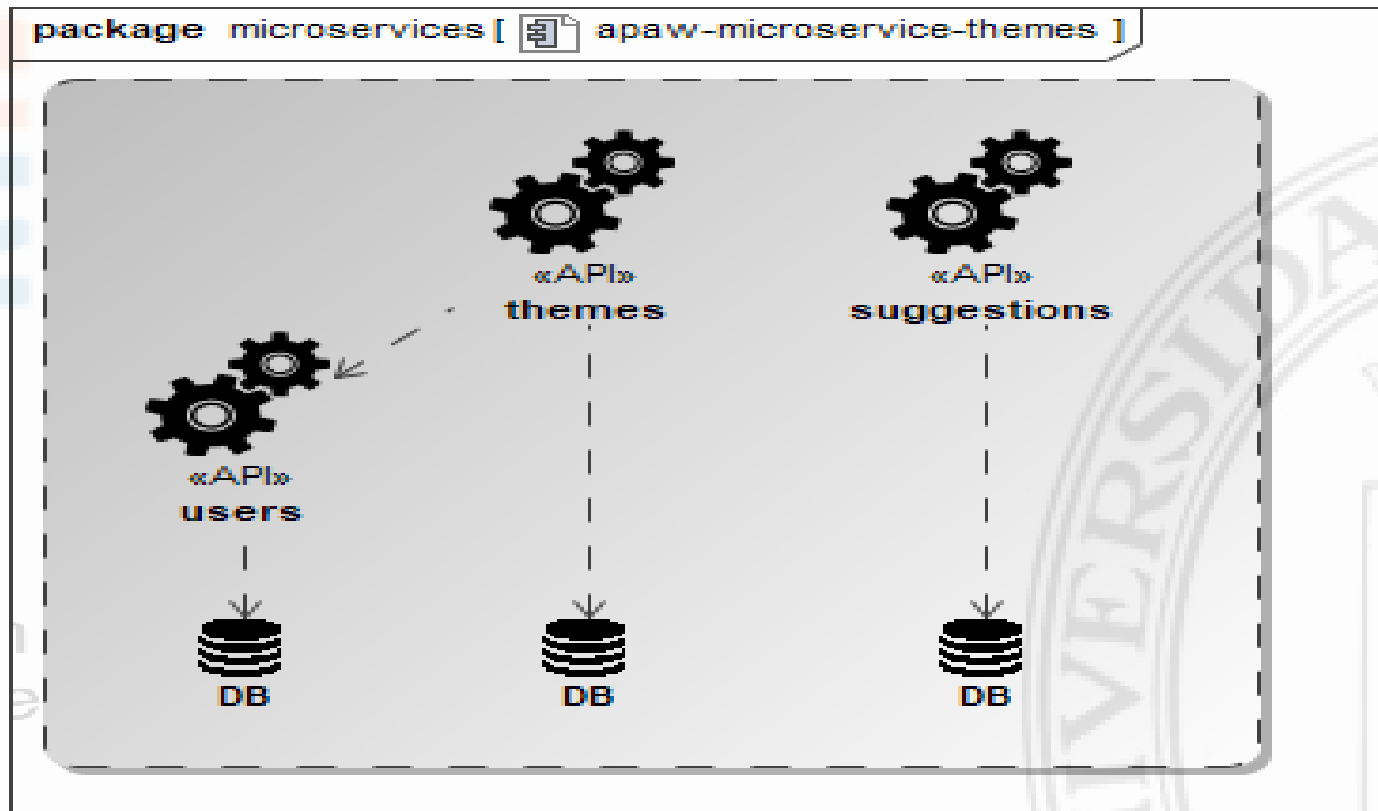
# Arquitectura Web. Micro Servicios



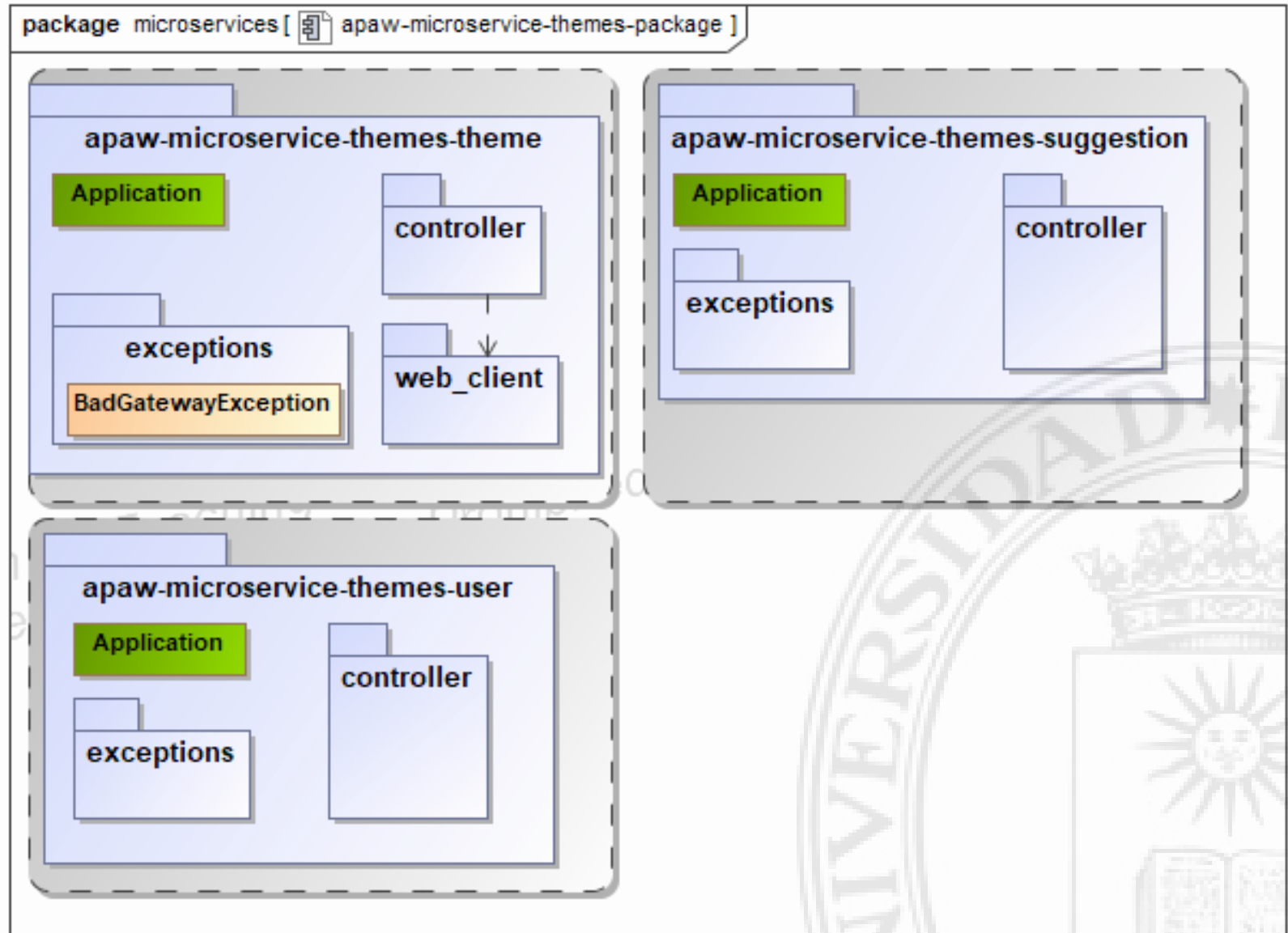
# Arquitectura Web. Micro Servicios

## ■ Aplicación Themes

- <https://github.com/miw-upm/apaw-microservice-themes-suggestion>
- <https://github.com/miw-upm/apaw-microservice-themes-user>
- <https://github.com/miw-upm/apaw-microservice-themes-theme>



# Arquitectura Web. Micro Servicios




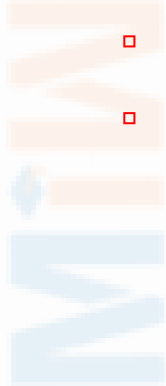
# Arquitectura Web. Micro Servicios

ApiClient.java X

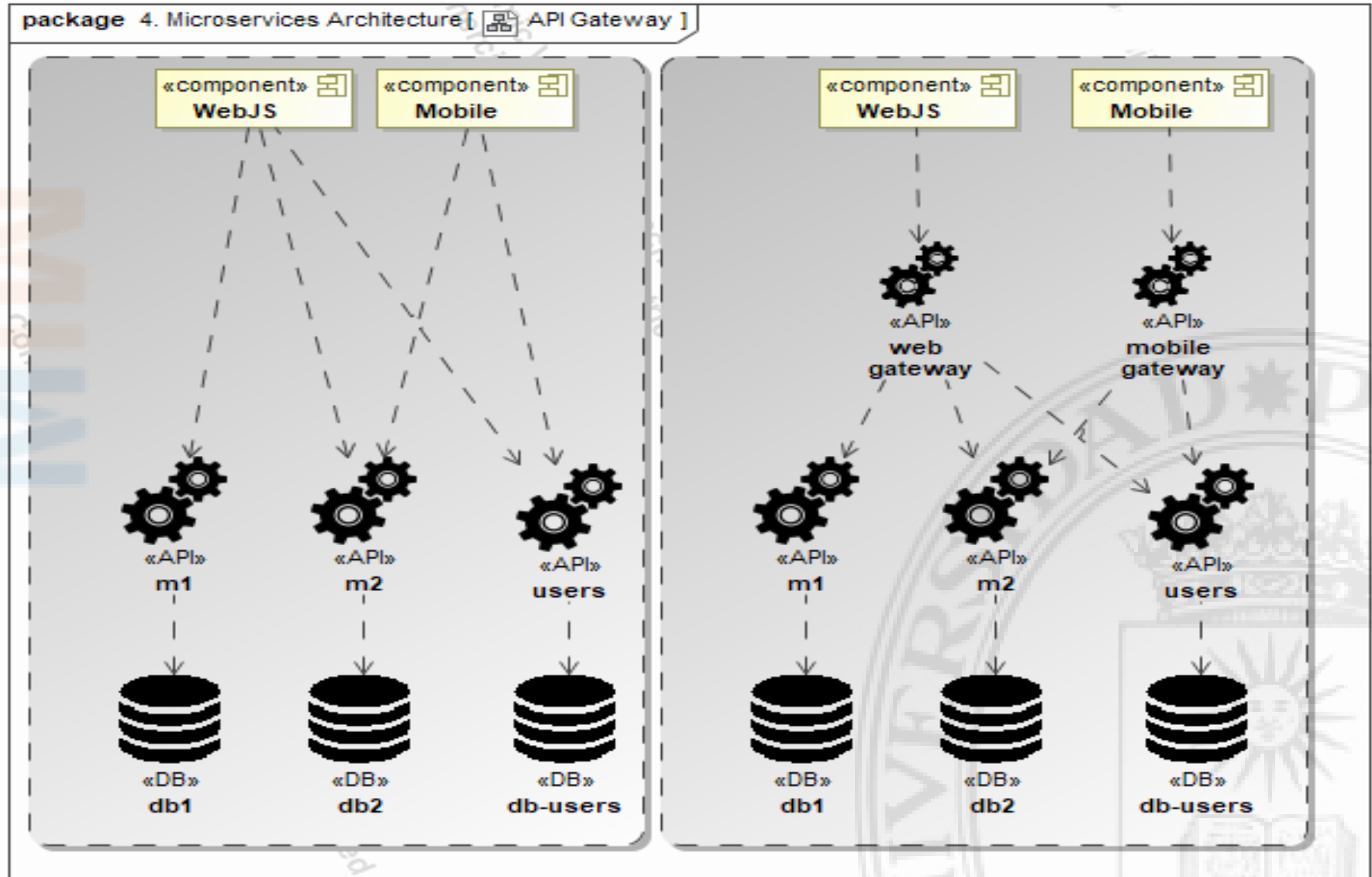
```
21 public void validateUserIdAssured(String userId) {
22     WebClientBuilder.build() WebClient
23         .get() RequestHeadersUriSpec<capture of ?>
24         .uri( s: "https://apaw-microservice-themes-user.herokuapp.com/users/" + userId + "/nick") capture of ?
25         .exchange() Mono<ClientResponse>
26         .onErrorResume(exception ->
27             Mono.error(new BadGatewayException("Unexpected error: Users API. " + exception.getMessage())))
28         .flatMap(response -> {
29             if (HttpStatus.NOT_FOUND.equals(response.statusCode())) {
30                 return Mono.error(new NotFoundException("User id: " + userId));
31             } else if (response.statusCode().isError()) {
32                 return Mono.error(new BadGatewayException("Unexpected error: Users API. "));
33             } else {
34                 return response.bodyToMono(Object.class);
35             }
36         }) Mono<Object>
37         .block();
38 }
```

# Arquitectura Web. Micro servicios

-  Proyecto Themes: Architecture
  - <https://github.com/miw-upm/apaw-microservice-themes-theme>
  - <https://github.com/miw-upm/apaw-microservice-themes-user>
  - <https://github.com/miw-upm/apaw-microservice-themes-suggestion>



## Patrones de Micro Servicios: Gateway (*Facade*)



## Patrones de Micro Servicios

- Pattern Access Token: JWT (JSON Web Token)
- Pattern: Externalized configuration
  - Se externaliza del servicio la configuración. Al arrancar, el servicio lee la configuración de una fuente externa
- Spring Cloud (<https://spring.io/projects/spring-cloud>)
  - Spring Cloud proporciona herramientas para que los desarrolladores construyan rápidamente algunos de los patrones comunes en sistemas distribuidos:
    - Configuration management
    - Service discovery
    - Circuit breakers
    - Intelligent routing
    - Micro-proxy
    - Control bus
    - One-time tokens
    - Global locks
    - Leadership election
    - Distributed sessions
    - Cluster state

# Introducción a GraphQL

- <https://graphql.org/>
- GraphQL es una lenguaje para especificar un API.

```
schema {  
  query: Query  
  mutation: Mutation  
}  
type Query {  
  hello: String!  
}  
type Mutation {  
  createPerson(name: String!, age: Int=18): String!  
}
```

```
extend type Mutation {  
  createUser(nick:String!,email:String,address:AddressInput):User  
  updateUser(nick:String,email:String,address:AddressInput):User  
  deleteUser(nick:String): Int  
}
```

```
type Dto {  
  id: ID,  
  attribute1: Boolean,  
  attribute2: [Float]!,  
  attribute3: [String]!  
}
```

```
input DtoInput {  
  stars: Int!  
  commentary: String  
}
```

```
enum Priority {  
  HIGH,  
  MEDIUM,  
  LOW  
}
```



# Introducción a GraphQL

```
introduction.graphqls x
schema {
  query: Query
  mutation: Mutation
}
type Query {
  suggestions(negative: String): [Suggestion]
  users: [User],
  user(nick: String): User,
}
type Mutation {
  createSuggestion(suggestion: SuggestionInput): Suggestion
  createTheme(themeCreation: ThemeCreation): Theme
  createUser(nick: String!, email: String, address: AddressInput): User
  updateUser(nick: String, email: String, address: AddressInput): User
  deleteUser(nick: String): Int
}
type Suggestion {
  id: String,
  negative: Boolean,
  description: String
}
input SuggestionInput {
  negative: Boolean,
  description: String
}
input ThemeCreation {
  reference: String,
  userId: String
}
type Theme {
  id: String,
  reference: String,
  date: String,
  user: User,
  votes: [Vote]
}
```

# Introducción a GraphQL

- GraphQL es un lenguaje para realizar *queries* o *mutations*
- Para realizar una petición: <http://localhost:8080/graphql?query>
  - ?query=query{users{id nick address{city}}}
  - ?query=query{users{id}}
  - ?query=mutation...

```
1 {  
2   users {  
3     id  
4   }  
5 }  
6
```

```
1 {  
2   "data": {  
3     "users": [  
4       {  
5         "id": "5d890b90f3896a3158db5c7a"  
6       },  
7       {  
8         "id": "5d890b90f3896a3158db5c7b"  
9       },  
10      {  
11        "id": "5d890b90f3896a3158db5c7c"  
12      }  
13    ]  
14  }  
15 }
```

```
{  
  users {  
    id  
    nick  
    address {  
      city  
    }  
  }  
}
```

```
mutation {  
  createUser(nick: "us1", email: "email1",  
    address: {country: "myCountry", city: "myCity"}) {  
    id  
    email  
  }  
}
```

```
1 {  
2   "data": {  
3     "createUser": {  
4       "id": "5d8910b6f3896a3158db5c7e",  
5       "email": "email1"  
6     }  
7   }  
8 }
```

# 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 reactiva:** *spring-boot-starter-webflux*
- **Tests:** *spring-boot-starter-test*
- **Cliente http:** *reactor-test*
- **OpenApi:** *org.springdoc:springdoc-openapi-ui*

## ■ Configuración

- *application.properties*
- *ApiLogs*



# Introducción a Spring

## ■ Arranque

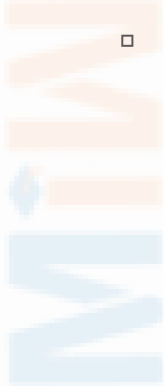
- *Application*

## ■ 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*

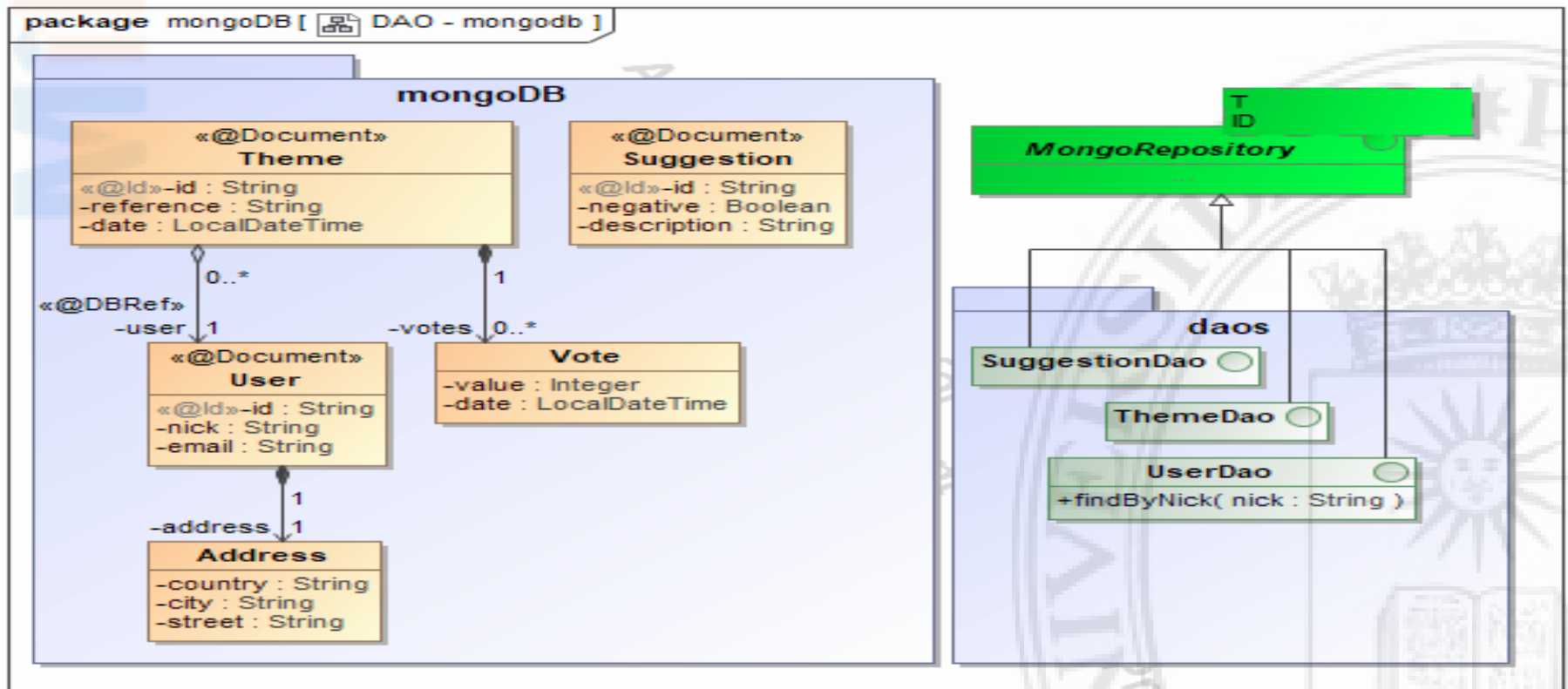
## ■ Inyección

- *@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 WebClient **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);