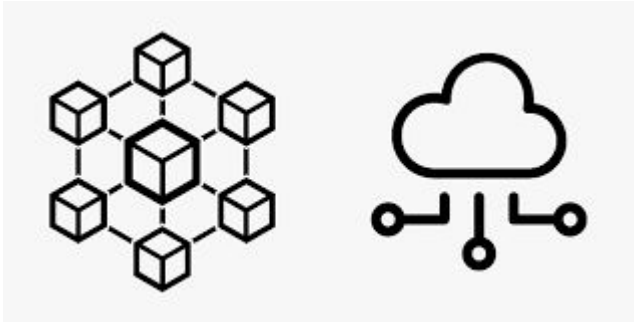
Java
Microservicios

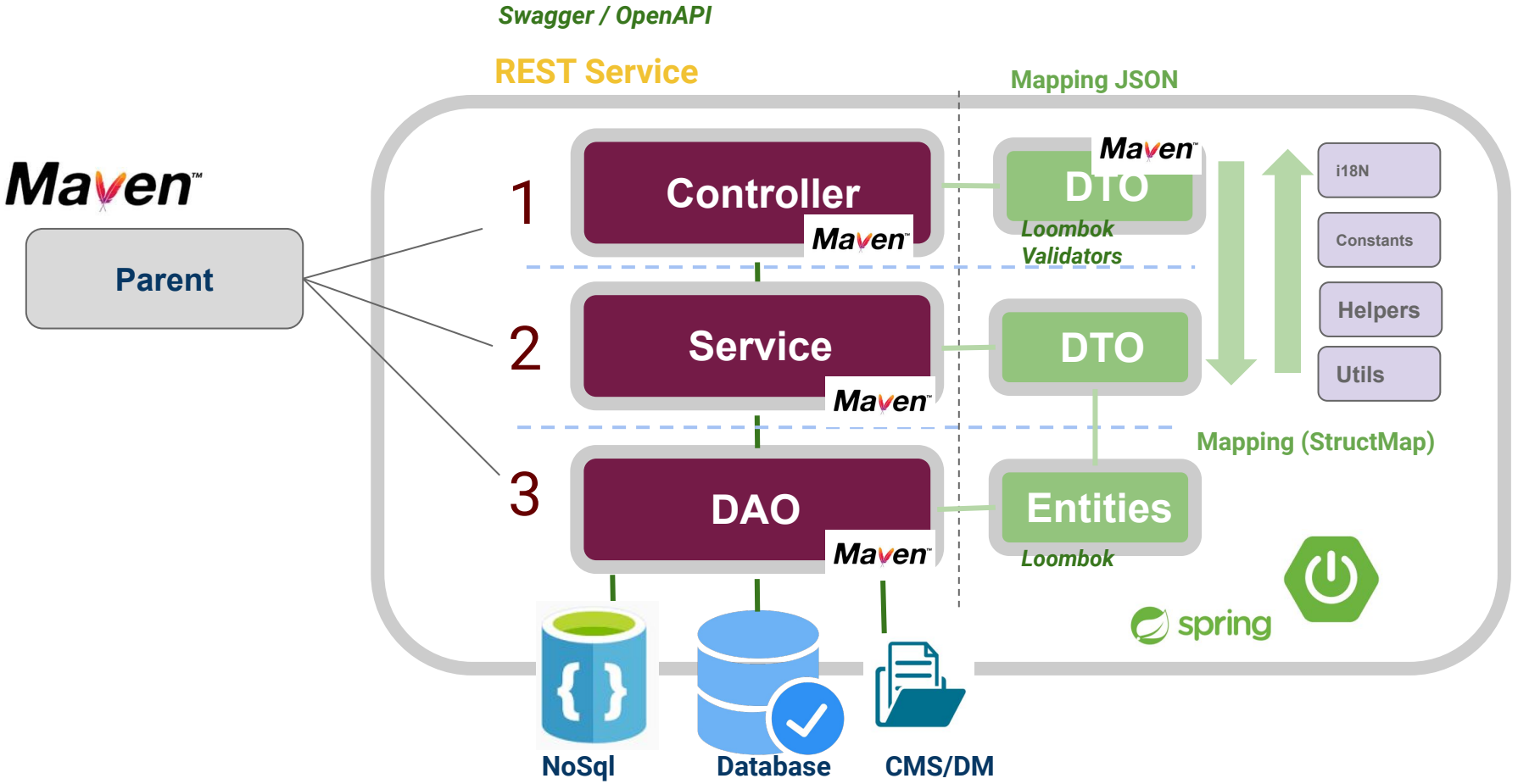
Ing. Rafael Benedettelli

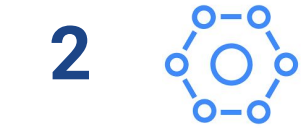
1. Arquitectura orientada a microservicios
2. Integración y despliegue continuo
3. Java (Situación actual) - de 8 a 14
4. Externalización de parámetros
5. Documentación REST con Swagger
6. Navegación REST con HATEOAS

Arquitectura orientada a Microservicios



Desarrollo

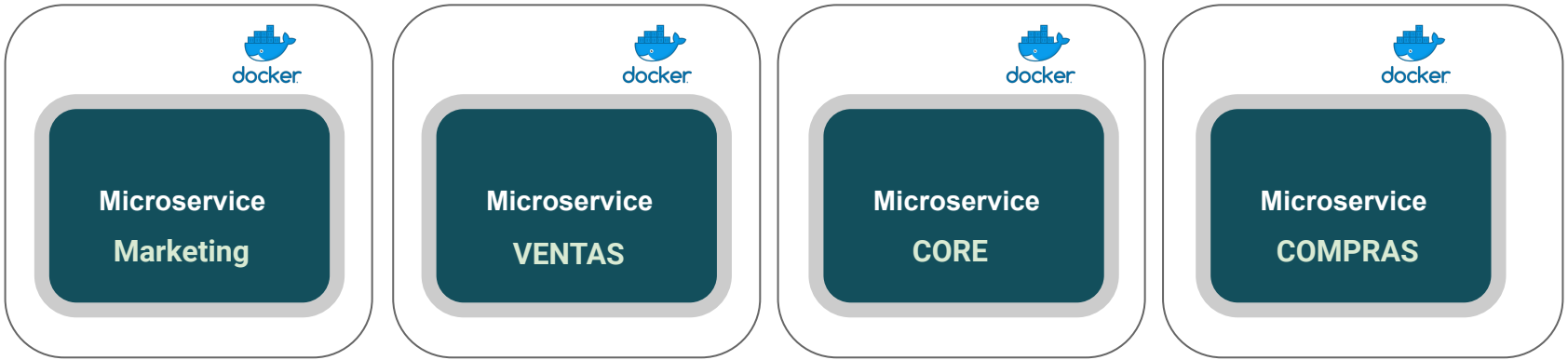

Swagger / OpenAPI
REST Service
Mapping JSON
Maven
Parent
1
Controller
Maven
DTO
Maven
Loombok
Validators
i18N
Constants
Helpers
Utils
2
Service
Maven
DTO
Mapping (StructMap)
3
DAO
Maven
Entities
Loombok
spring
NoSql
Database
CMS/DM



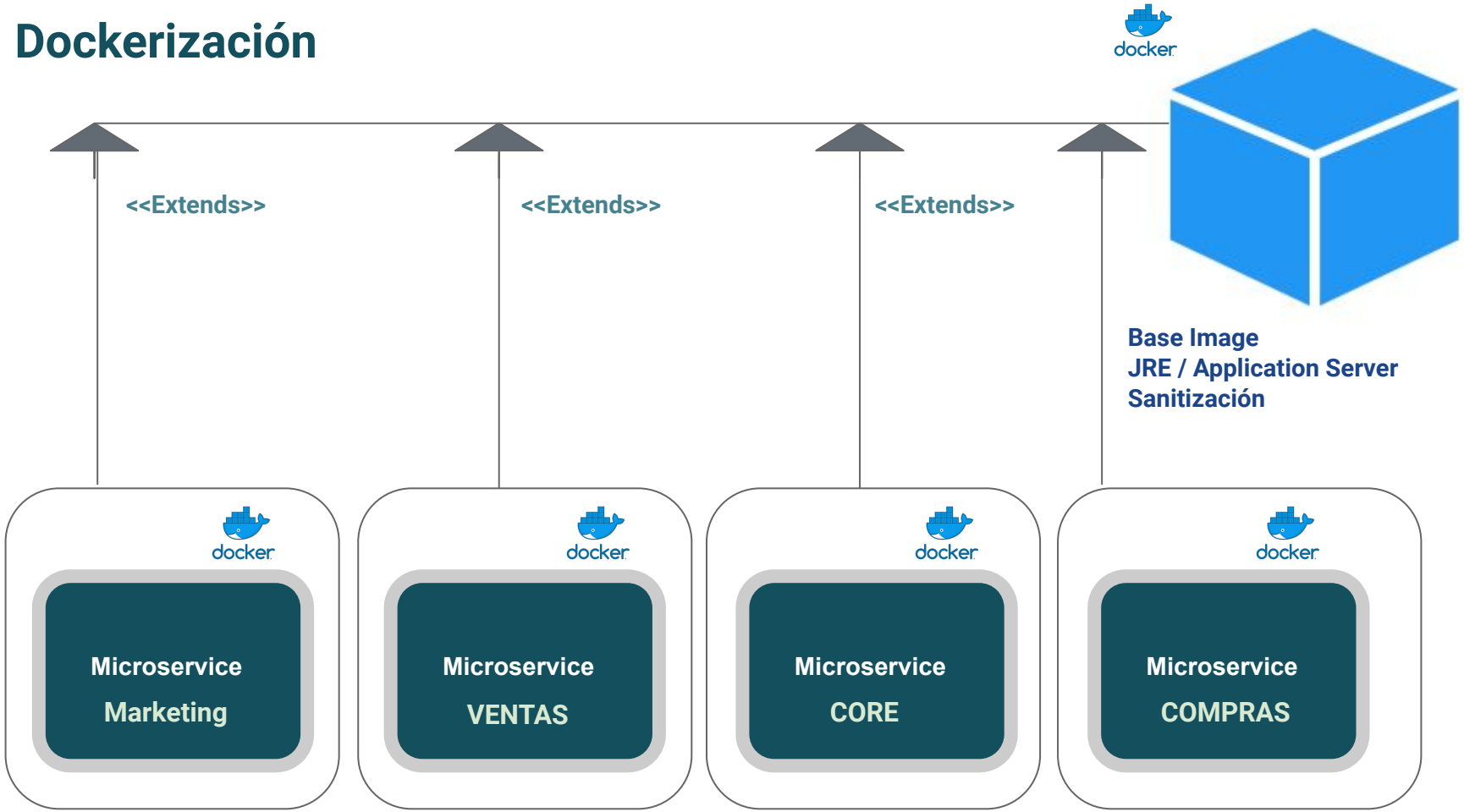
CI/CD

GitLab ♡ docker

Dockerización

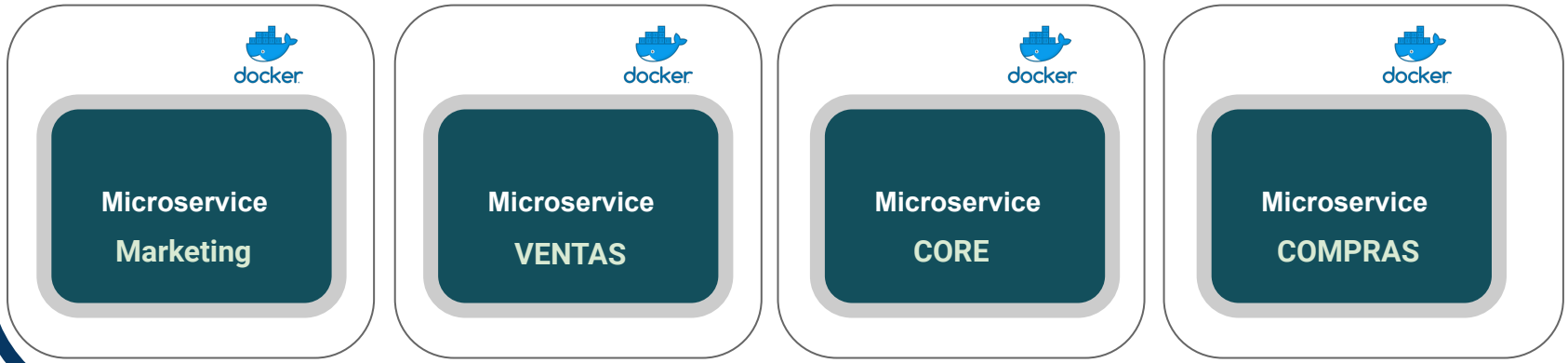


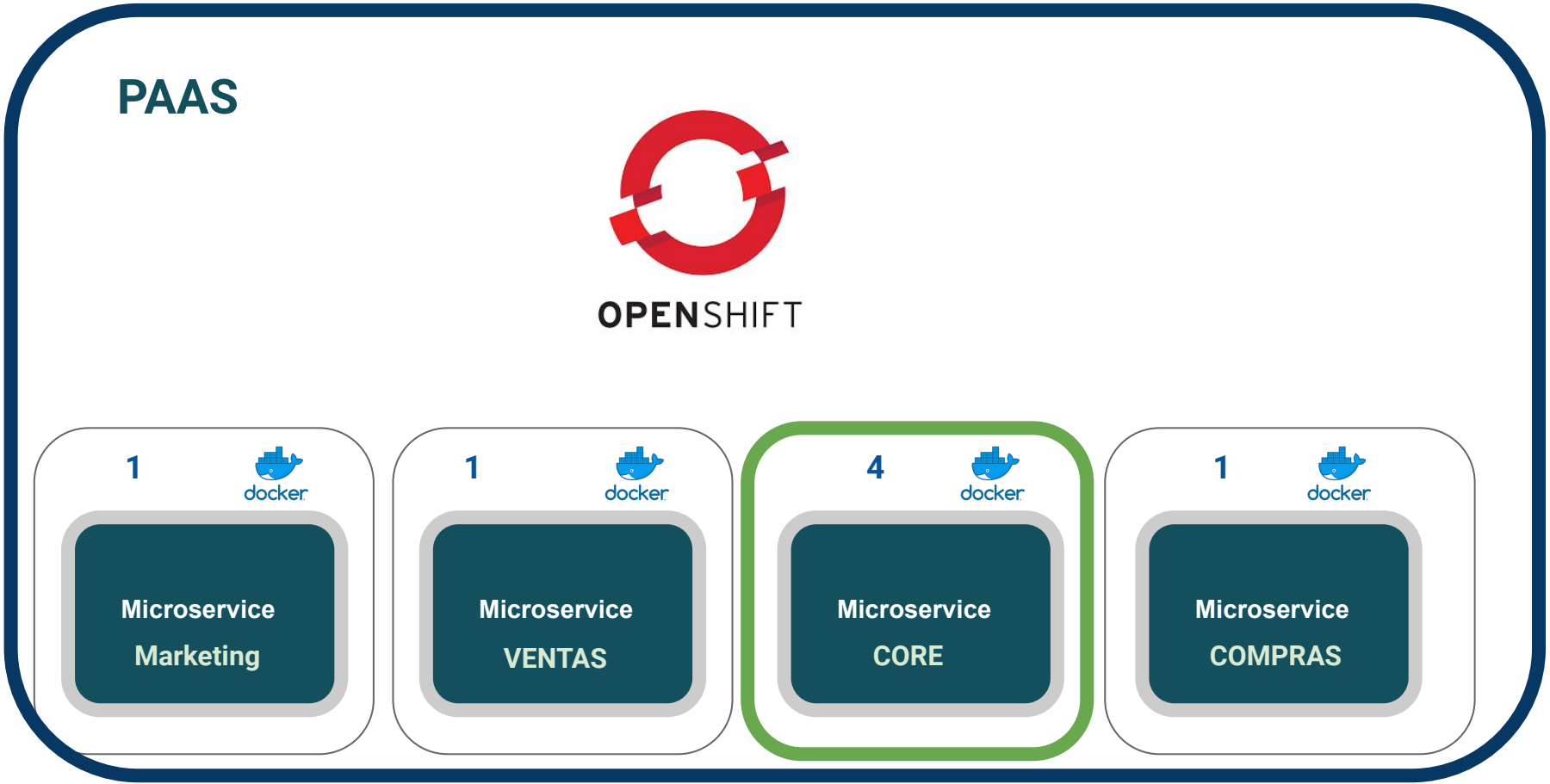
Dockerización



PAAS


OPENSHIFT


docker
Microservice
Marketing
docker
Microservice
VENTAS
docker
Microservice
CORE
docker
Microservice
COMPRAS


PAAS
OPENSHIFT
1
docker
Microservice
Marketing
1
docker
Microservice
VENTAS
4
docker
Microservice
CORE
1
docker
Microservice
COMPRAS

PostgreSQL vs MySql

Mejor rendimiento ante consultas complejas (joins y subconsultas)
Orientado a desarrolladores PL / JAVA /C
Licencia totalmente libre
Instalación mas compleja
Proyectos Medios-Grandes

Alto rendimiento en consultas simples
GUI mas amigable
Puesta a punto simple
Licencia Abierta pero bajo el ala de Oracle
Proyectos chicos / web

Roadmap de versiones

Java de 8 a 15

Java Politicas de Versionado

Dudas clase anterior

Dudas clase anterior
PATCH (partial update)

```
@PatchMapping(value = "/users/{id}")
public UserDTO updateAge(Map<String,Object> attributes, @PathVariable("id") String id) {

        UserDTO userDTO = new UserDTO();

        //Logica

        return userDTO;
}
```

Dudas clase anterior
QUERY FILTER

```
@GetMapping("/users")
public List<UserDTO> listAllUsers( @RequestParam(required = false) String name,
                                   @RequestParam(required = false) String lastName,
                                   @RequestParam(required = false) Integer age) {

        List<UserDTO> list = List.of(new UserDTO(1, "Rafael"), new UserDTO(2, "Miguel"), new UserDTO(3, "Alvaro"));

        list = list.stream().filter(u -> u.getName().contains(name)).collect(Collectors.toList());

        return list;

}
```

Dudas clase anterior
QUERY FILTER mecanismo query

```
https://api/rest/customers?q={"status": "GOLD"}
```

```
https://api/rest/people?q={"name": "Joe", "age": 17}
```

```
https://api/rest/people?q={"$or": [{"name": "Jane"}, {"name": "Donald"}]}
```

```
https://api/rest/people?q={"salary": {"$gt": 10000}}
```

application.properties


```
server.port=8081
server.servlet.context-path=/escuelait/api/v1/microservices
```

Unificar rutas por controller


```
@RequestMapping("/users")
```

CODIGOS DE RESPUESTAS HTTP

Rest User

		Código Errores HTTP
getById	GET	200 - OK
listAll	GET	200 - OK
create	POST	201 - CREATED
update	PUT	200 - OK
delete	DELETE	200 - OK

Rest User (Not Found)

getById	→	GET	404 - OK
listAll	→	GET	404 - OK
create	→	POST	404 - OK
update	→	PUT	404 - OK
delete	→	DELETE	404 - OK

Obtener URI:

```
URI location = ServletUriComponentsBuilder.
                                        fromCurrentRequest()
                                        .path("/{id}")
                                        .buildAndExpand(userDTO.getId())
                                        .toUri();
```

Códigos de error HTTP ResponseEntity

200 - OK
201 - Created
400 - Bad Request
401 - Unauthorized
404 - Not Found
500 - Internal error server

Not Found
Recurso no encontrado

Anidación de recursos

users/{id}/accounts

Rest Buenas Prácticas

Rest Buenas Practicas

REST es orientado a resources

- Usar sustantivos (no verbos)
- Usar plural

POST **/usuarios** 

/crearUsuarios 

Rest Buenas Practicas

REST concatenación de recursos de manera apropiada

GET	**/usuarios/**	(todos los usuarios)
GET	**/usuarios/{id}**	(el usuario con {id}
GET	**/usuarios/{id}/accounts/**	Todas la cuentas del usuario con id
GET	**/usuarios/{id}/accounts/{idAcc}**	La cuenta con id del usuario id

Rest Buenas Practicas

Entregar codigos HTTP adecuados

No mas de tres niveles de resources

equipos/{1}/jugadores/{1}/estadisticas/

JSON claros y no mas de 3 niveles

```
{
nombre: “Real Madrid”
jugadores: [
  {nombre: “Benzema”, estadisticas: [
                .....
        ]}
]
}
```

La cuenta con id del usuario id

Rest Buenas Practicas

Evitar!!!

/equipos/1/jugadores/1/partidos/2/goles/1/multimedia/1

/users/1/accounts/1/orders/1/products/1/

Rest Buenas Practicas

No entregar JSON “Enormes”

- Paginar
- Sortear
- Filtrar
- Crear un rest query con filters

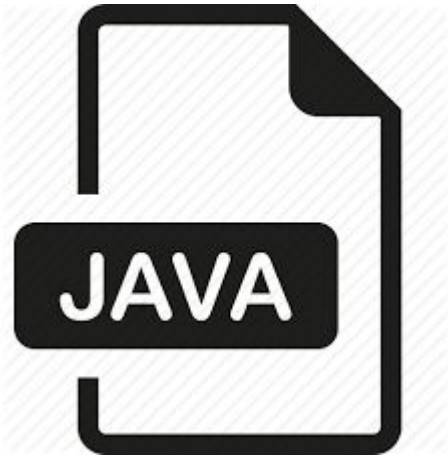
Rest Buenas Practicas

- Documentar la API
  - Open API - Swagger
- Generar links de navegación
  - HATEOAS
- Asegurar las apis definiendo roles

Rest Buenas Practicas

- Exportar clientes de demostración
  - Ej: POSTMAN
- Entregar errores definidos y verbosos
- Validación de campos
- Versionado apropiado de la API

Externalización



application.properties

Externalización

- @Value
- @ConfigurationProperties Prefix
- Listas de valores
- Variables de entornos
- Valor constante
- Cambio de archivo de propiedades:
  @PropertySource(value = "classpath:mensajes.properties")

Práctica

Práctica

1. Generar un proyecto Spring Boot desde cero
    a. Incorporando lombok y web
2. Crear un modelo Equipo -> Jugadores
    a. Lombok: Generar constructores/getter/setter/toString/Equals
    b. Para jugadores generar un constructor exclusivo para los campos Numero/Nombre
3. Crear un controlador CRUD Rest para la entidad equipo
4. Retornar los codigos de error apropiados
5. Generar un proyecto POSTMAN con todas las llamadas
    a. Generar variables de entorno para base url
    b. Exportar el proyecto
6. Agregar un servicio teams/{id}/players que devuelva un array con todos sus jugadores
7. Crear un CRUD controller para Players
8. Externalizar los siguientes datos:
    a. Nombre del juego
    b. Edicion
    c. Año
    d. Liga