

API Slides Companion

Anatomia API REST

curl -s

```
https://api.github.com/repos/didattica-forever/Chirpy (address API)  
https://api.github.com/repos/didattica-forever/Chirpy (address API)  
    <filtro_username>/<filtro_reponame> (filtrri)  
  
/repos/<filtro_username>/<filtro_reponame> API  
/repos/{username}/{reponame} API
```

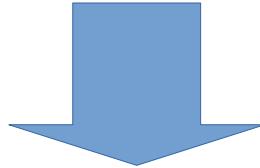
result

JSON

```
object {82}  
  id : 1054599009  
  node_id : "R_kgDOPtvnYQ"  
  name : Chirpy  
  full_name : "didattica-forever/Chirpy"  
  private : false  
  ↴ owner {19}  
    login : "didattica-forever"  
    id : 47675590  
    node_id : "MDQ6VXNlcjQ3Njc1NTkw"  
    avatar_url : "https://avatars.githubusercontent.com/u/47675590?v=4"  
    gravatar_id : ""
```

SOAP

```
curl -X POST https://www.example.com/weatherService \
  -H "Content-Type: text/xml" \
  -d '<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <GetCityWeather xmlns="http://weather.example.com/">
      <CityName>Rome</CityName>
    </GetCityWeather>
  </soap:Body>
</soap:Envelope>'
```

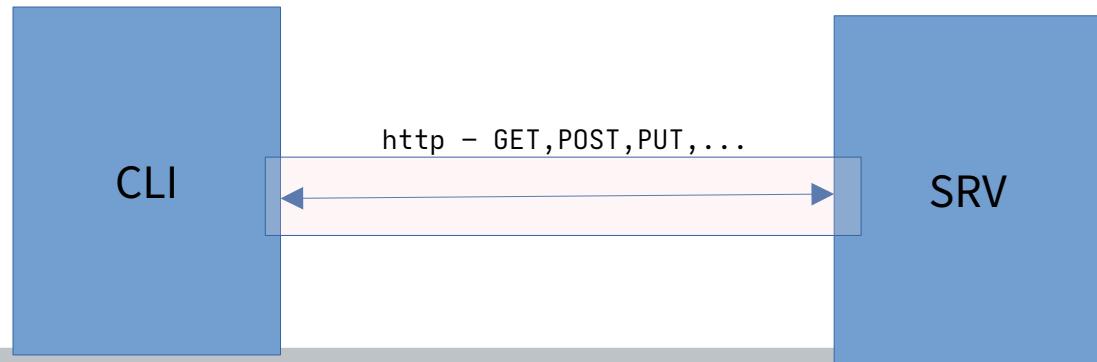


```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <GetCityWeatherResponse>
      <GetCityWeatherResult>
        <Temperature>18</Temperature>
        <Condition>Sunny</Condition>
      </GetCityWeatherResult>
    </GetCityWeatherResponse>
  </soap:Body>
</soap:Envelope>
```

REST

REST trasferimento stato (Es: da server a client)

- 1) POST /customers – crea nuovo cliente (payload necessario)
- 2) GET /customers/{id} – profilo cliente
- 3) elenco di tutti i clienti? ==> GET /customers



CRUD & HTTP

CRUD	SQL	HTTP	DAO/REPO
Create	insert	POST	create
Read	select	GET	find findAll findBy+sort
Update	update	PUT (100%) PATCH	update merge
Delete	delete	DELETE	delete remove

Idempotenza

PUT /devices/12 { "mode": "AUTO" } - IDEM

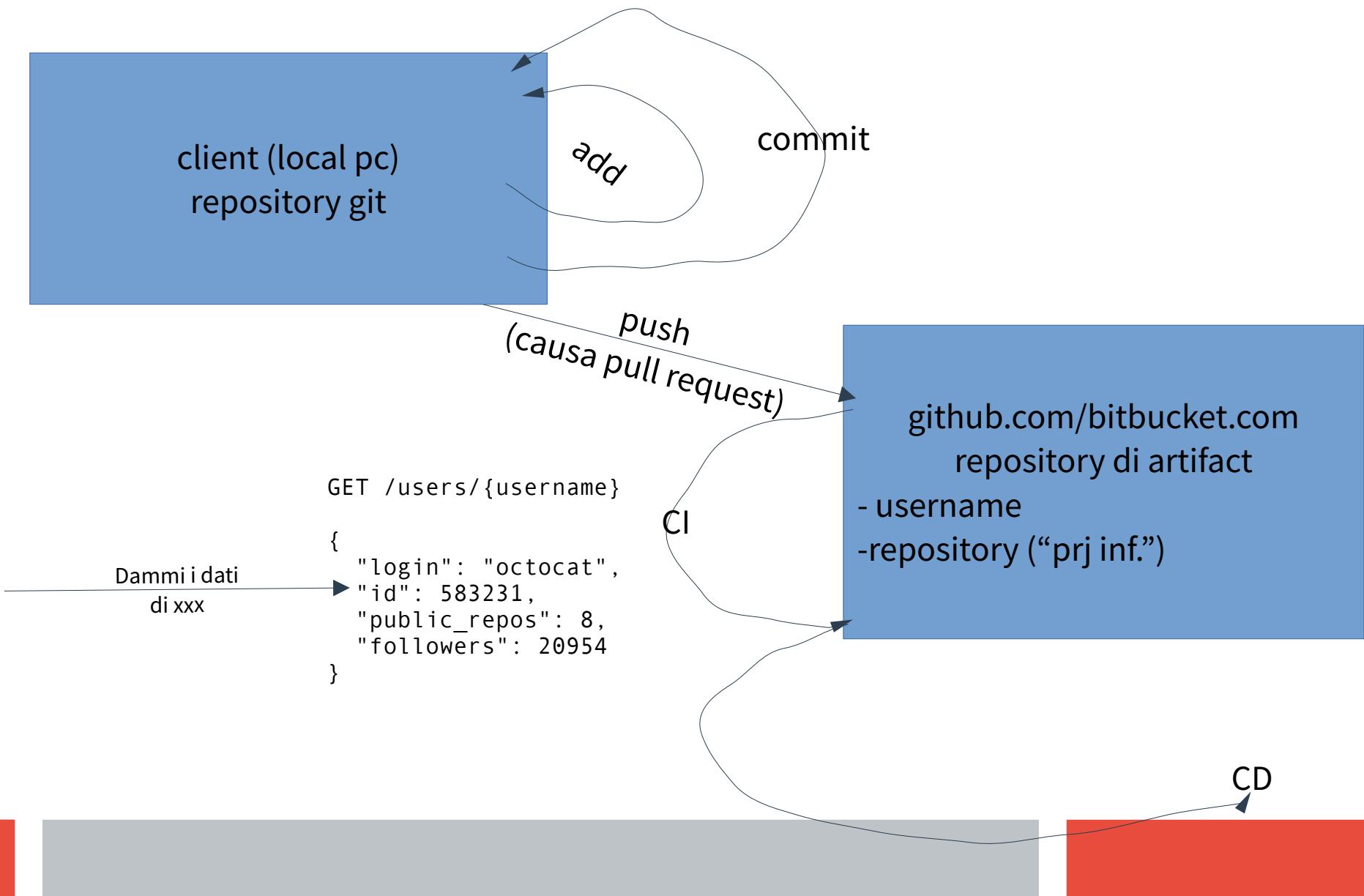
POST /logs { "event": "login" } - NO

DELETE /sessions/99 - IDEM

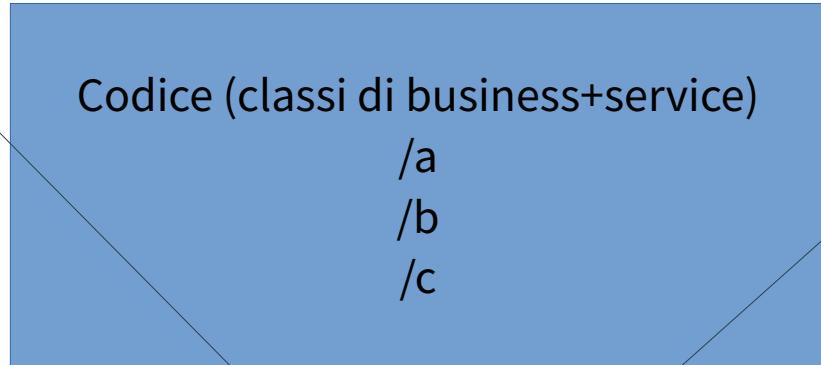
UPDATE accounts SET score = score + 1 - NO

UPDATE accounts SET saldo = credit + debit - NO

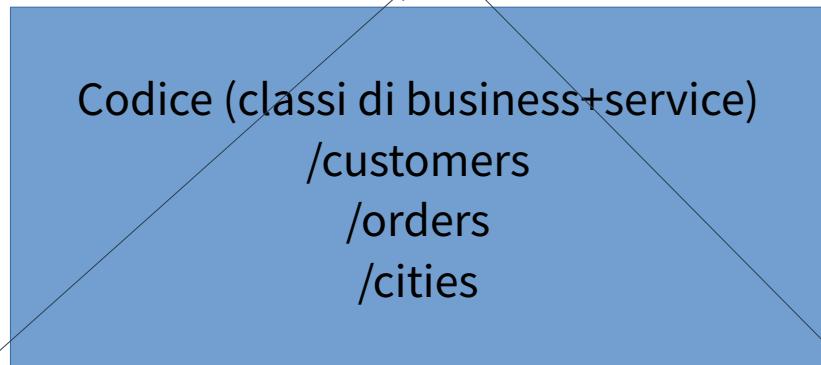
Git & Github



realizzazione API REST



rename

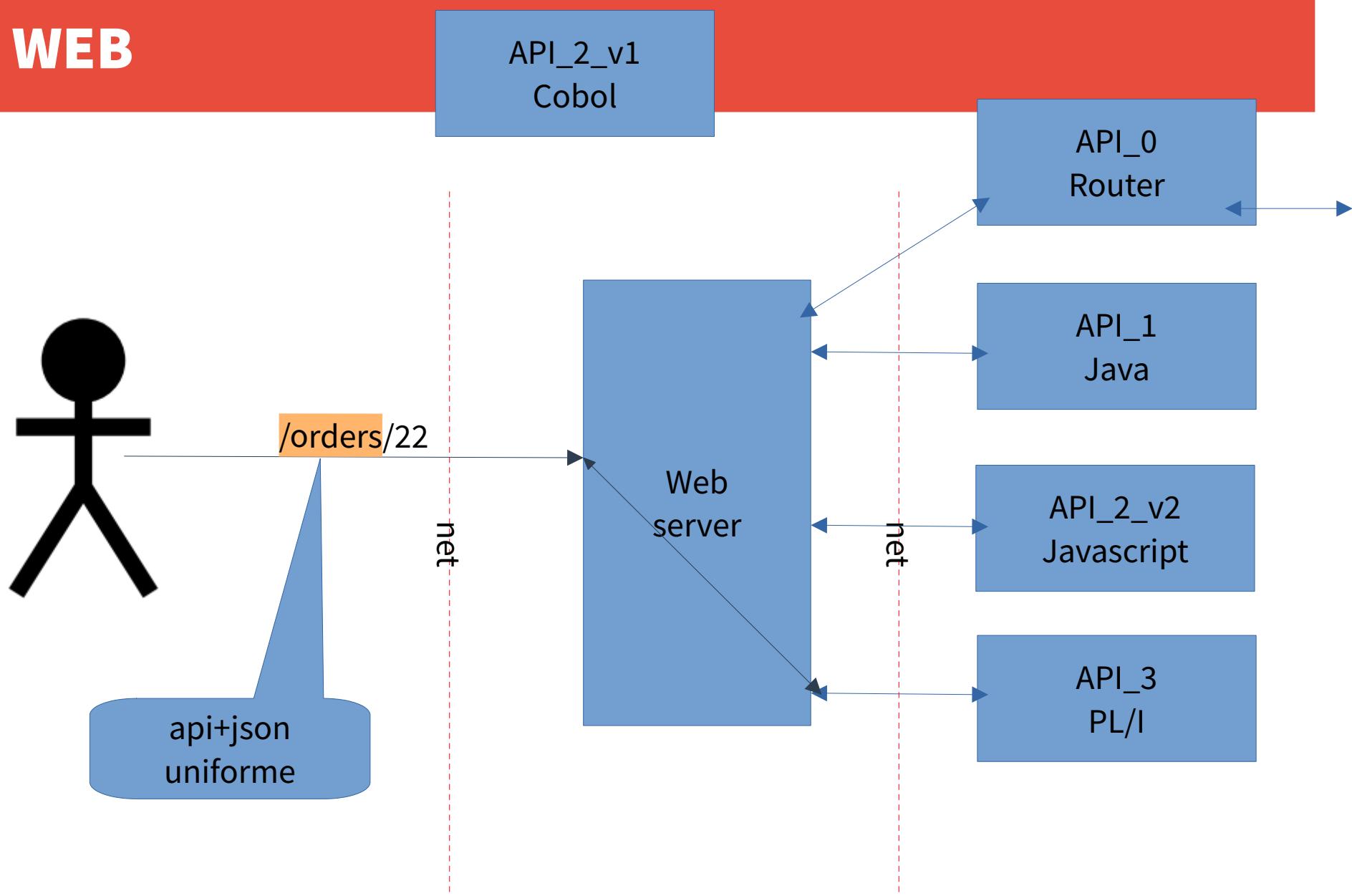


/customers
/orders
/cities

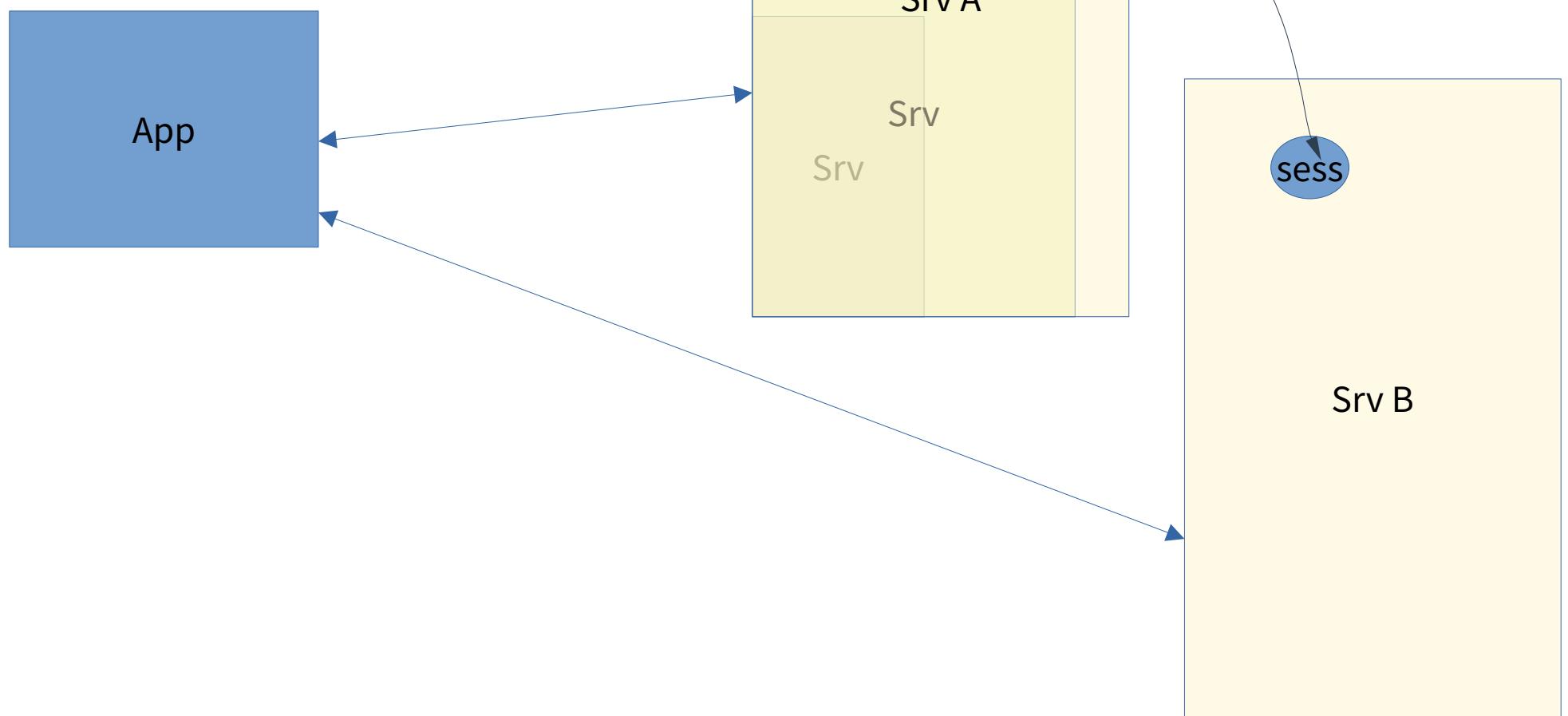
Linguaggio
del
business



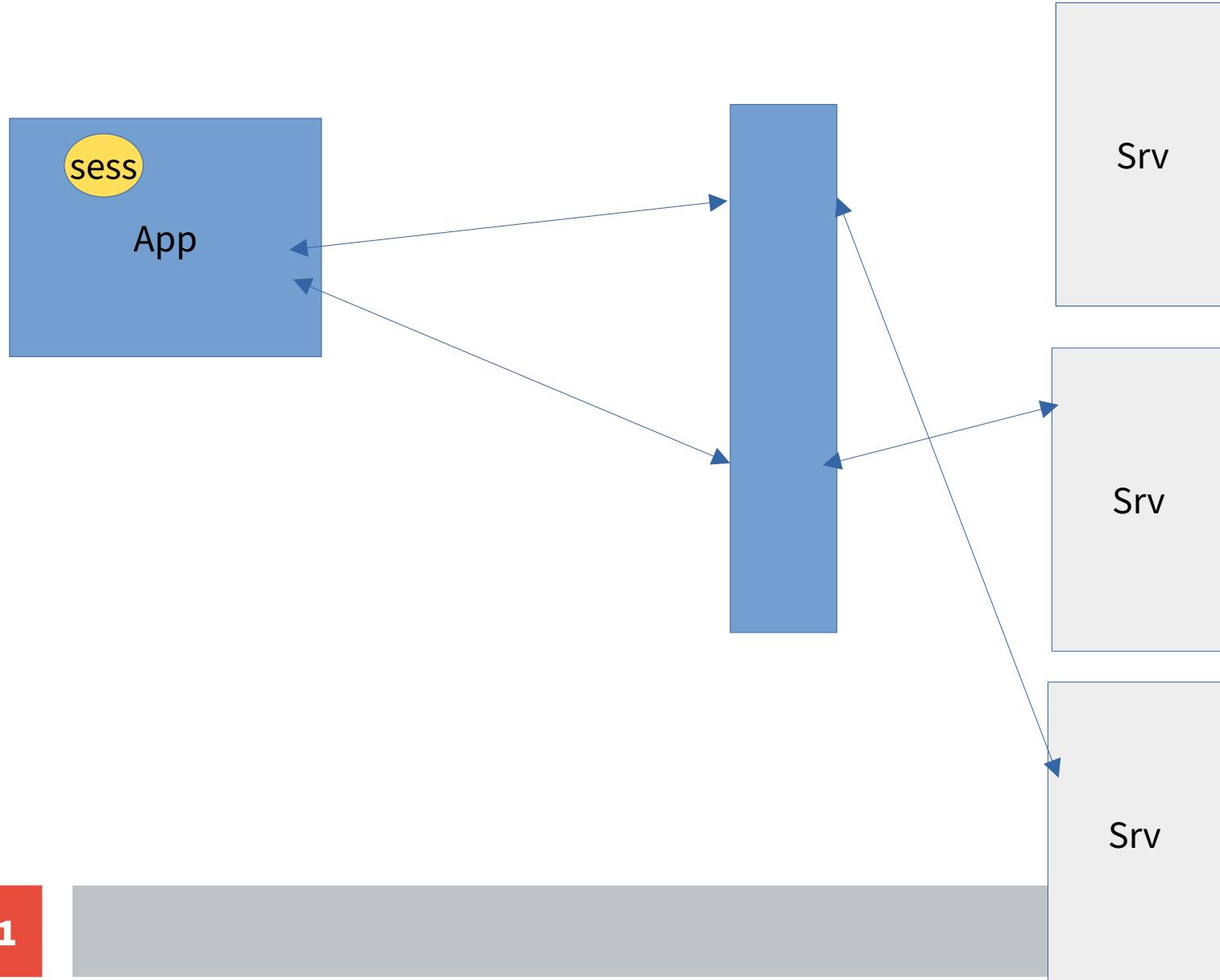
API WEB



Scalabilità verticale



Scalabilità orizzontale



Cache

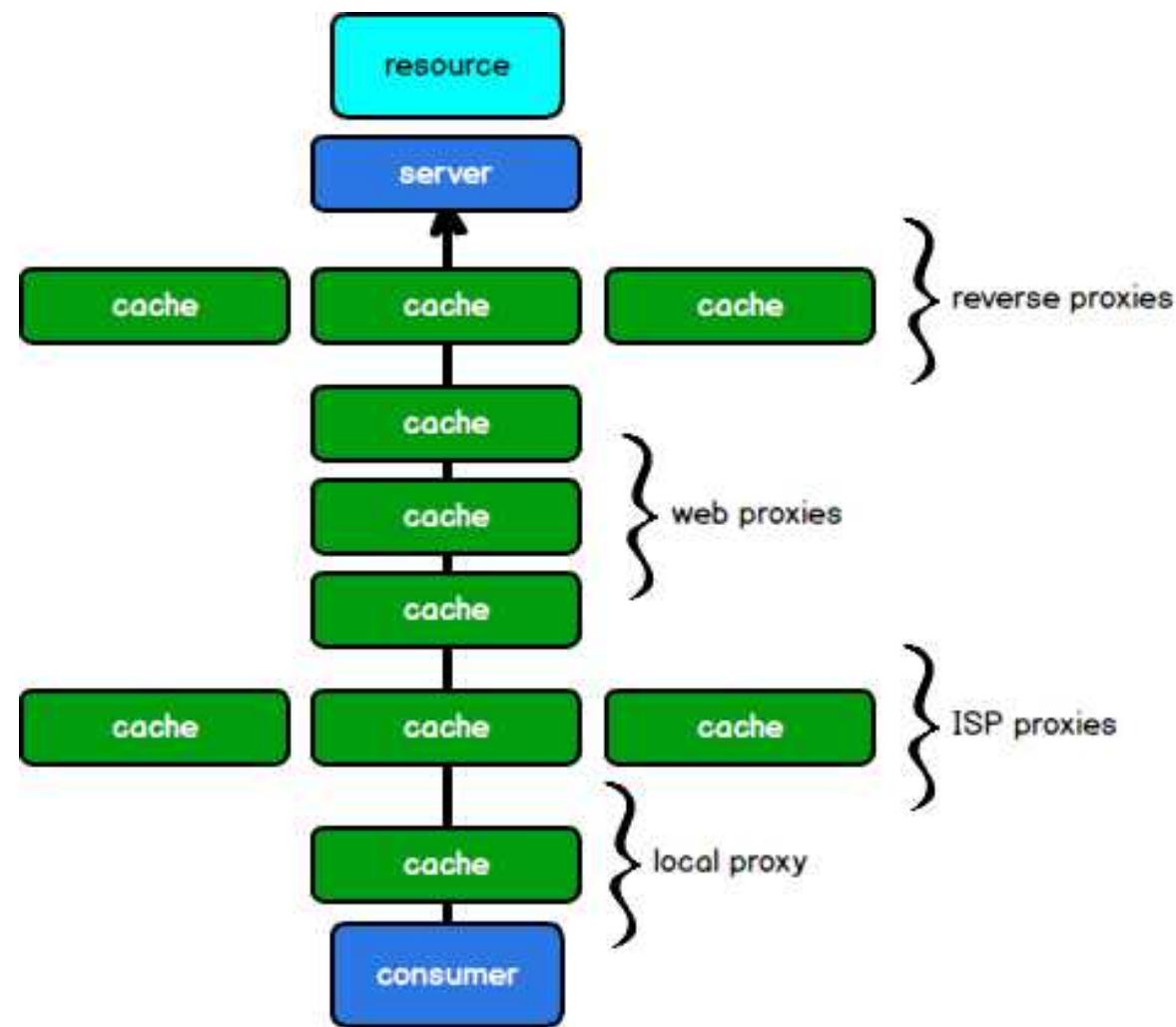
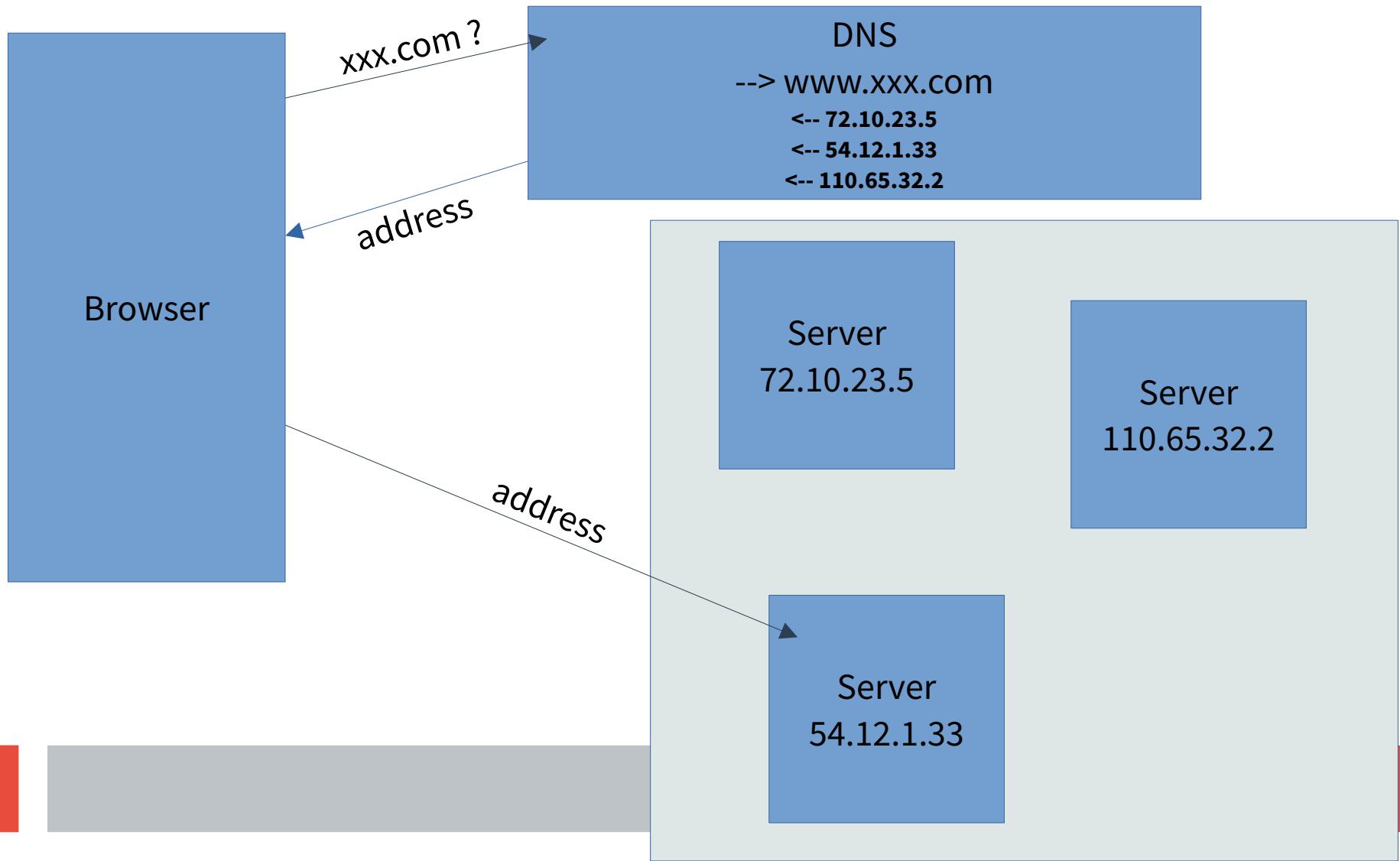


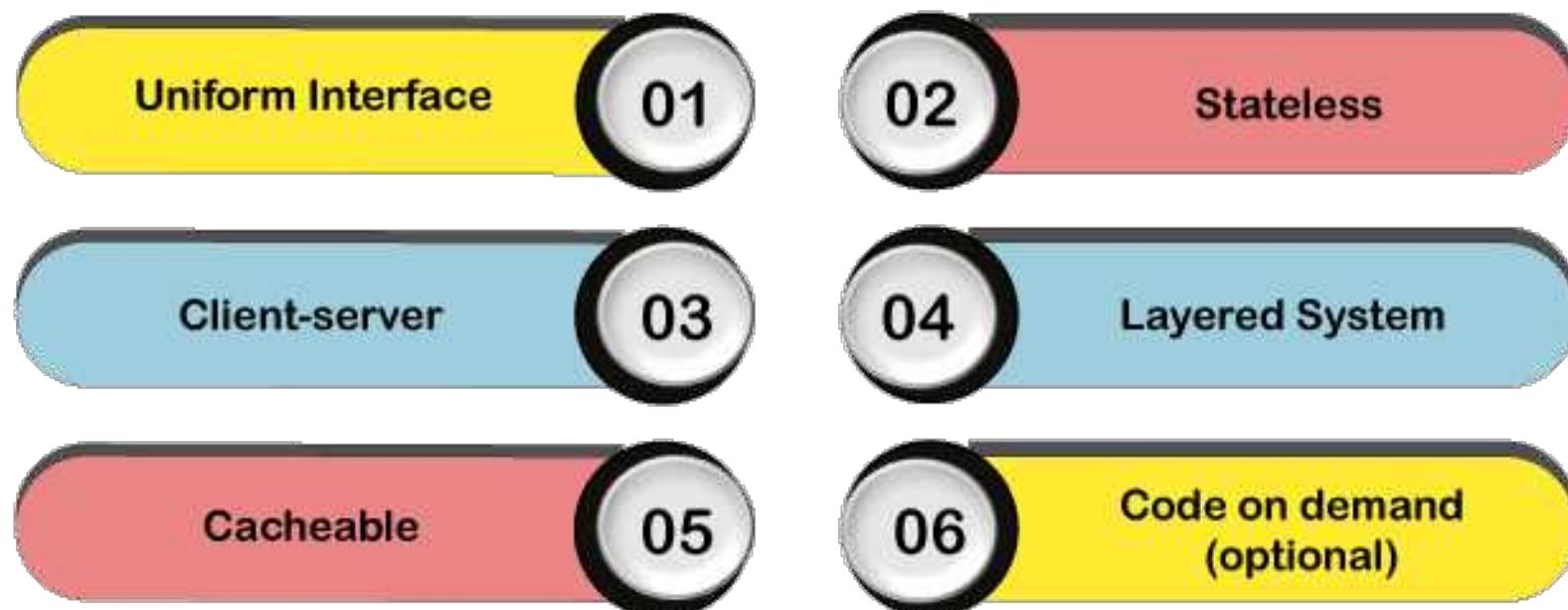
Figure 1. Web caches. REST In Practice, 2010.

DNS



Constraint Stile Architetturale REST

CONSTRAINTS OF REST ARCHITECTURE



BitBucket

Lista Commit di un repository

GET /rest/api/1.0/projects/{projectKey}/repos/{repoSlug}/commits

Lista commenti Pull Request filtrando quelli con action=="COMMENTED"

**GET /rest/api/latest/projects/{project}/repos/{repo}/pull-requests/{prId}/
activities**

Elencare permessi repository (cloud)

GET /repositories/{workspace}/{repo_slug}/permissions-config/users

BitBucket

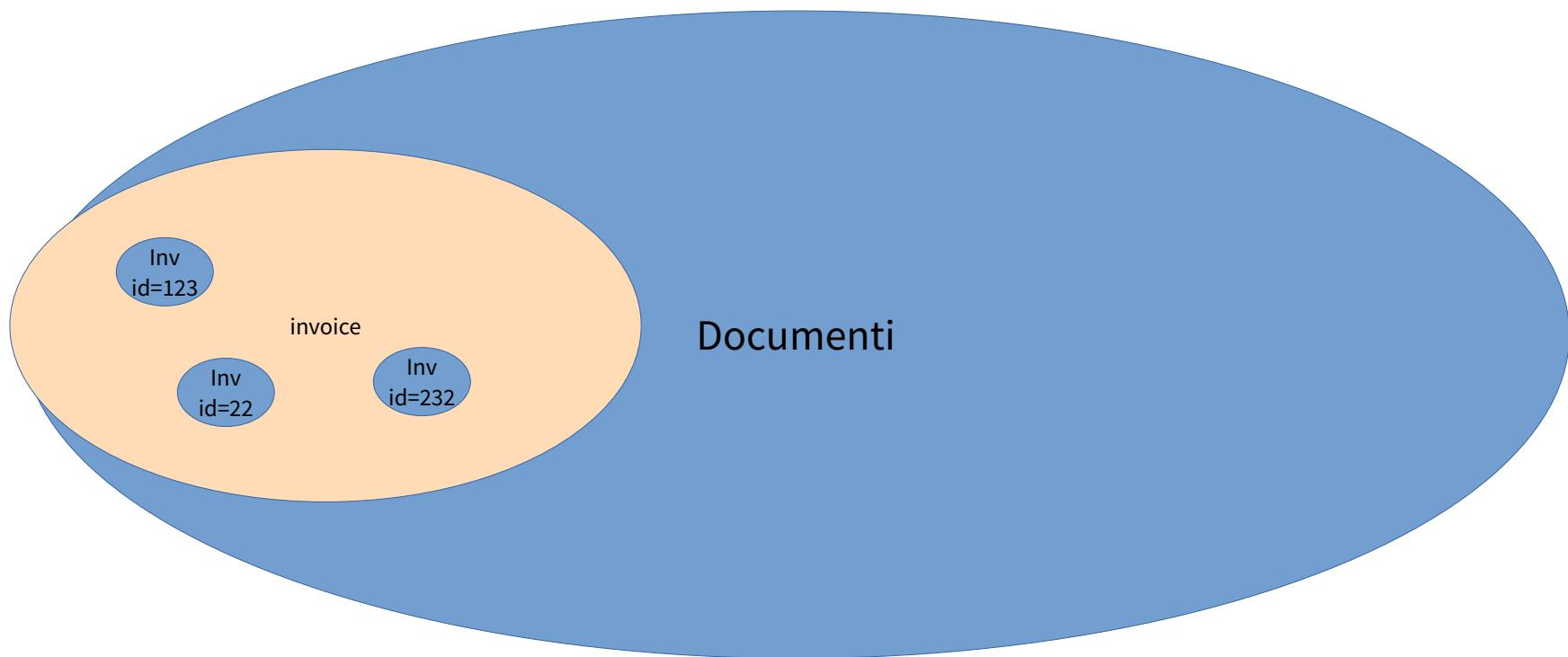
Lista Commit di un repository (ritorna un JSON)

```
curl -u USER:PASS \
```

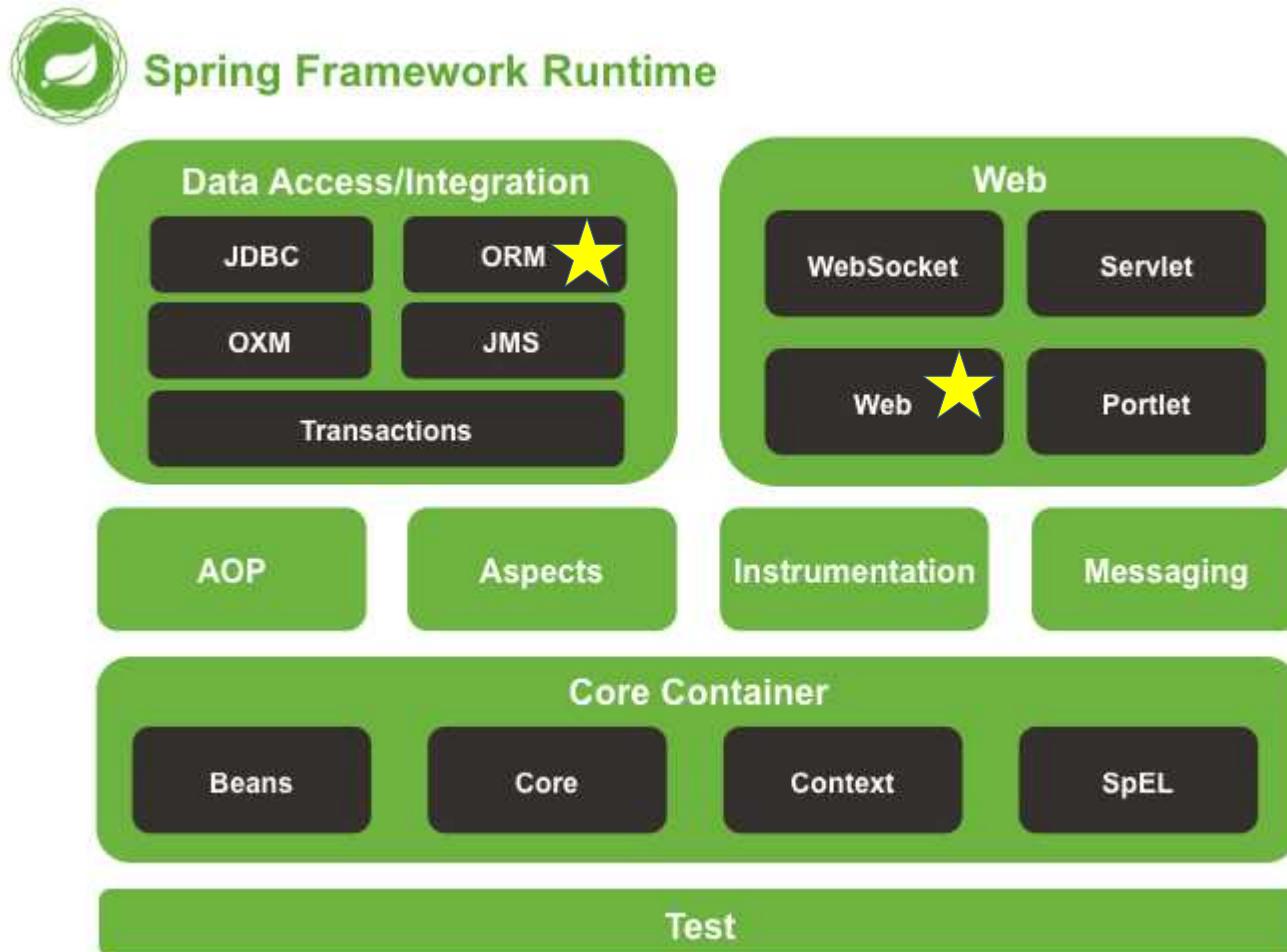
```
"https://your-bitbucket-server/rest/api/1.0/projects/PROJ/repos/my-repo/commits?until=master"
```

```
{
  "size": 2,
  "limit": 25,
  "isLastPage": true,
  "values": [
    {
      "id": "01f9c86...",
      "displayId": "01f9c86",
      "author": { "name": "Alice", "emailAddress": "alice@example.com" },
      "message": "Fix bug XYZ",
      "parents": [ { "id": "abcdef123" } ]
    },
    { /* altro commit */ }
  ],
  "start": 0,
  "nextPageStart": 2
}
```

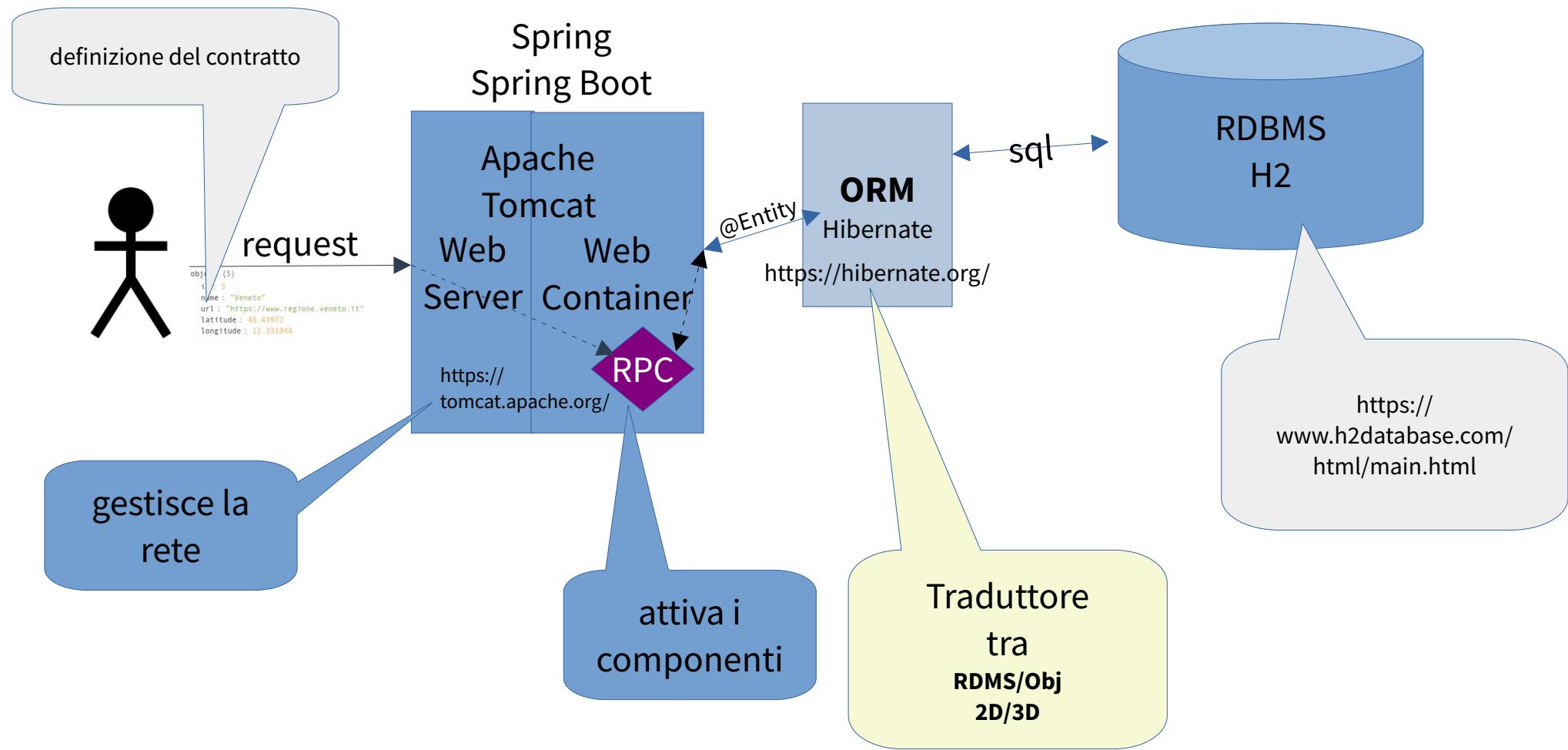
Classes o tipi & valori



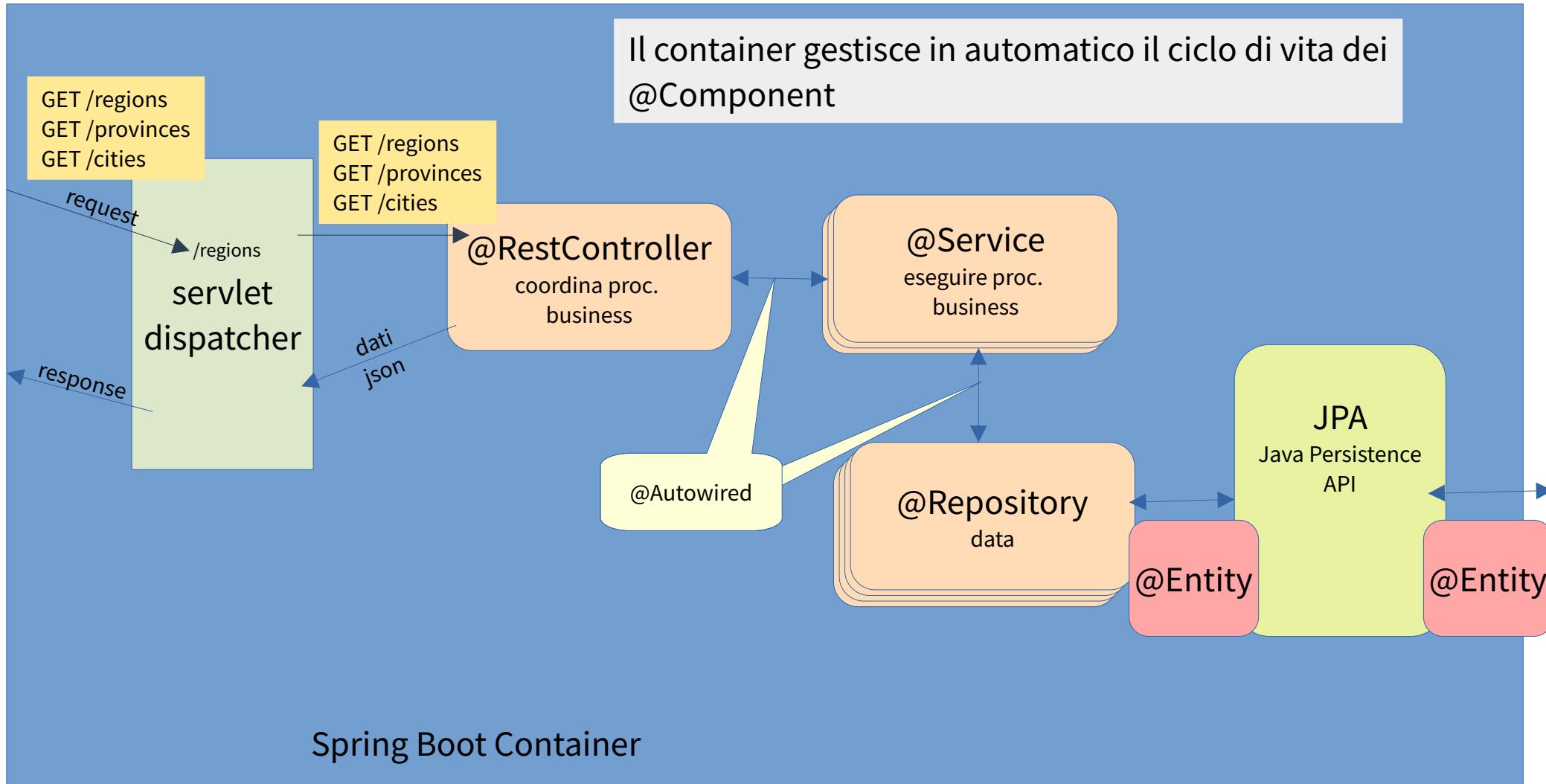
Server REST Backend Java+Spring(boot)



Spring Boot Application



Application in SpringBoot



RDBMS VS Objects

```
@Column(name = "codice_città_metropolitana") // visione lato rdbms
```

```
private String codiceCittàMetropolitana; // visione lato OOP
```

SELECT * FROM province;						
ID	ID_REGIONE	CODICE_CITTA_METROPOLITANA	NOME	SIGLA_AUTOMOBILISTICA	LATITUDINE	LONGITUDINE
1	1	201	Torino	TO	45.063299	7.669289
2	1	null	Vercelli	VC	45.320220	8.418508
3	1	null	Novara	NO	45.548513	8.515079
4	1	null	Cuneo	CN	44.597031	7.611422
5	1	null	Asti	AT	44.900765	8.206432
6	1	null	Alessandria	AI	44.847220	8.704629

SELECT * FROM REGIONI;				
ID	NAME	URL	LATITUDINE	LONGITUDINE
1	Piemonte	https://www.regione.piemonte.it	45.066666	7.7
2	Valle d'Aosta/Vallée d'Aoste	https://www.regione.vda.it	45.73722	7.320556
3	Lombardia	https://www.regione.lombardia.it	45.46416	9.190336
4	Trentino-Alto Adige/Südtirol	https://www.regione.taa.it	46.066666	11.116667
5	Veneto	https://www.regione.veneto.it	45.43972	12.331944
6	Eredi Veneto/Gidra	https://www.regione.veneto.it	45.826447	12.804127

2D

Regione

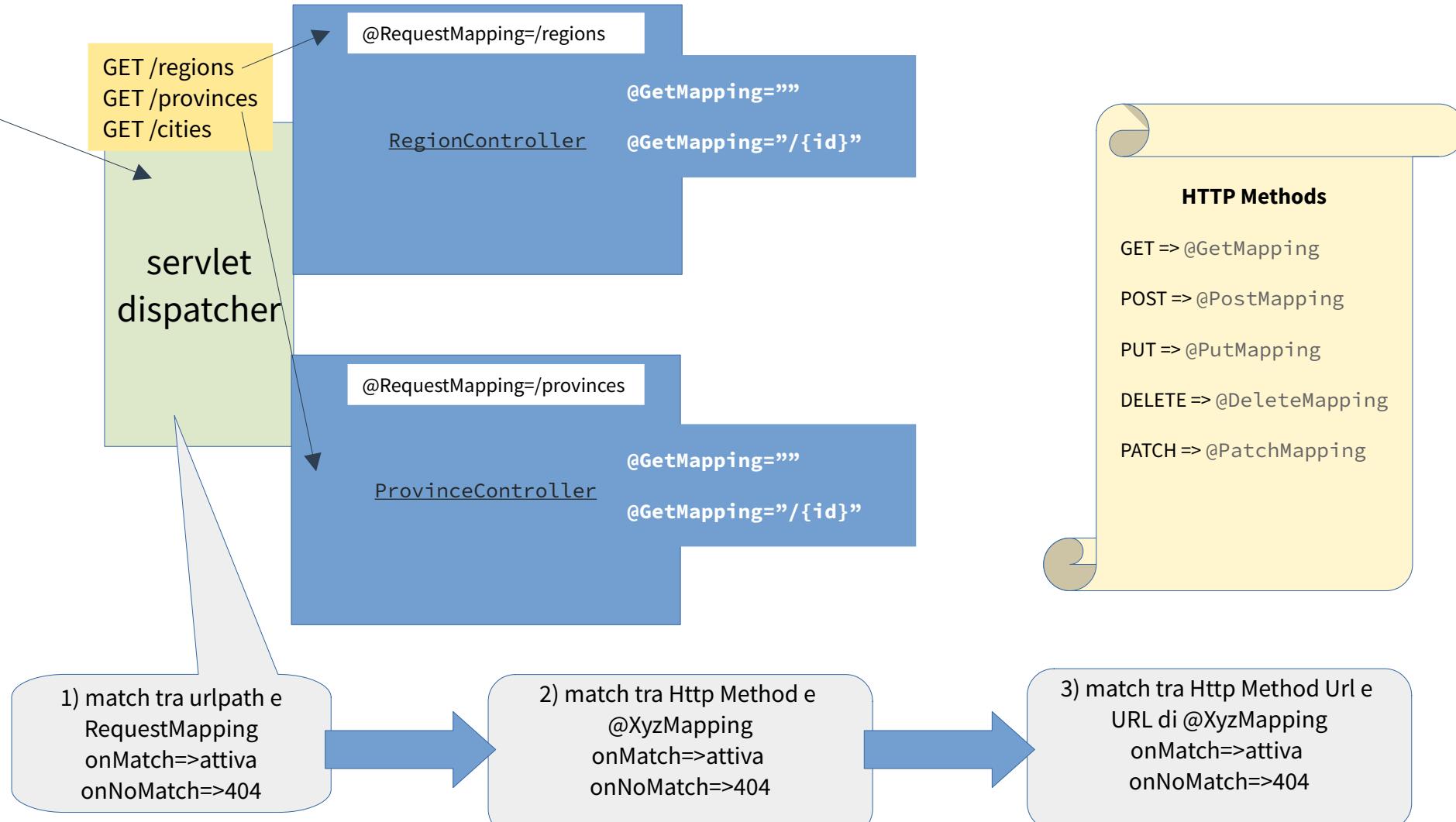
Provincia

Provincia

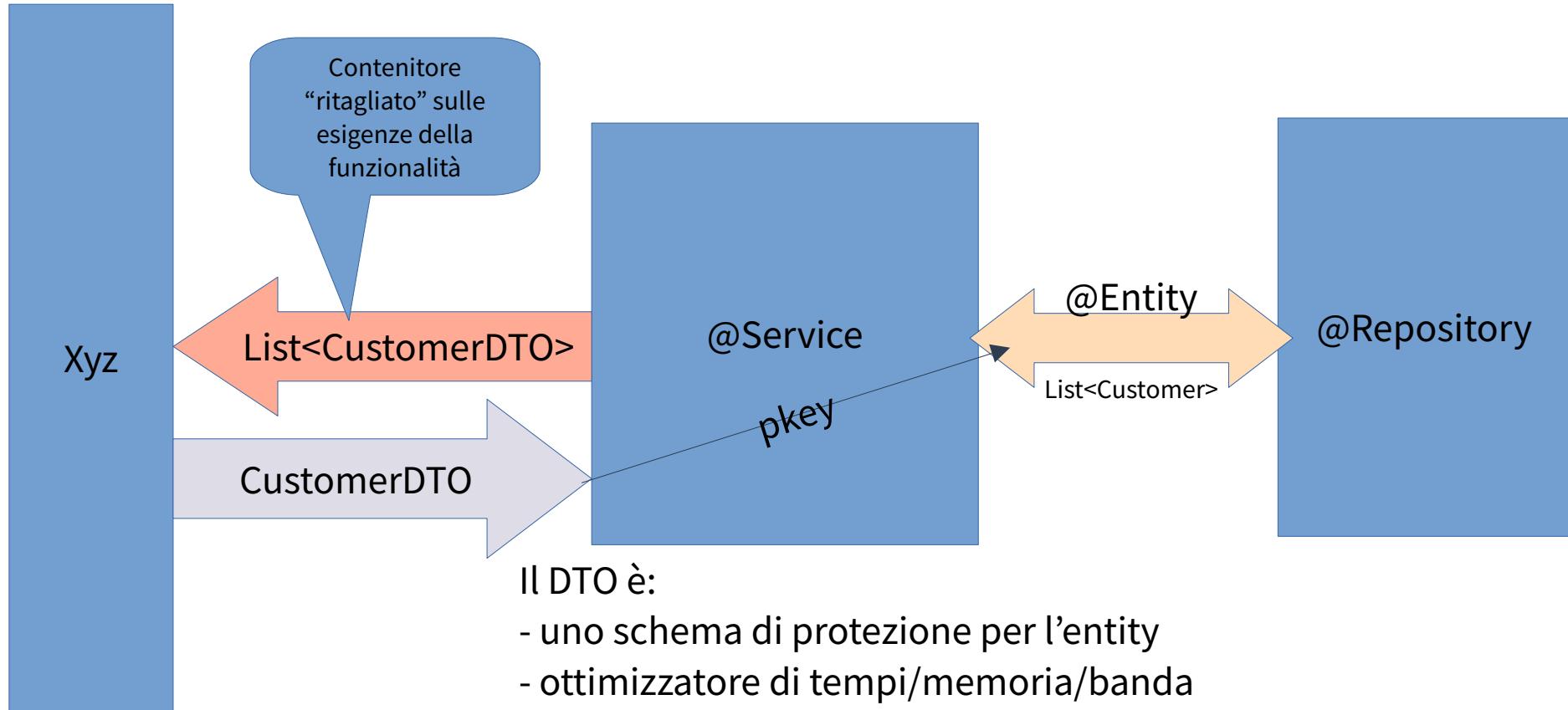
Regione

Al boot dell'applicazioni

<https://start.spring.io/>



DTO pattern (Data Transfer Object)



Il DTO è:

- uno schema di protezione per l'entity
- ottimizzatore di tempi/memoria/banda
- foundation per il JSON di scambio

Il DTO è associato ad un mapper che converte da e verso l'entity

News in V2

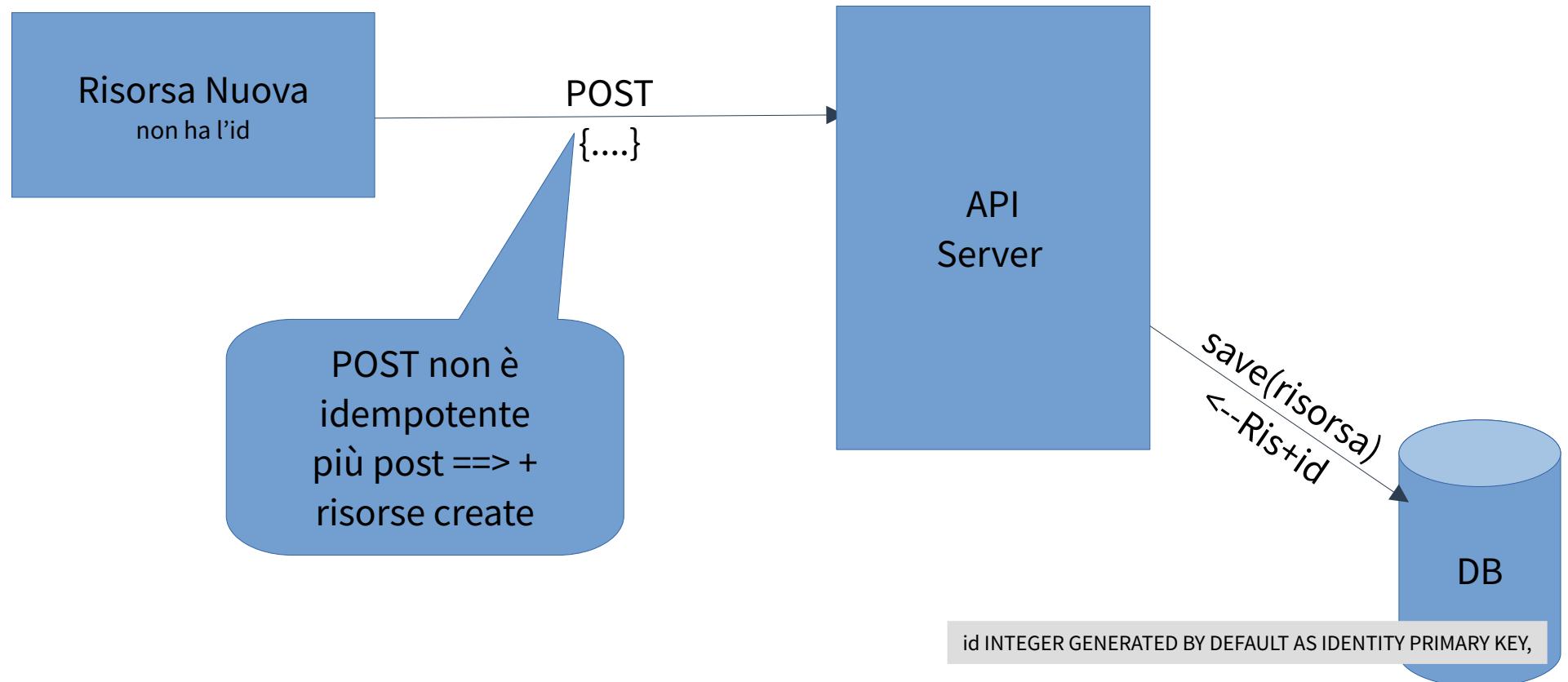
- new ApiError
- repo method **findByNomeContainingIgnoreCase(String nome)** di RegionRepository, ProvinceRepository, CityRepository (query added)
- RegionController, ProvinceController, CityController per aggiunta metodo di filtraggio e “query string”

```
@GetMapping  
public List<CityDTO> getAllCities(  
    @RequestParam(name = "regionId", required = false) Integer regionId,  
    @RequestParam(name = "provinceId", required = false) Integer provinceId,  
    @RequestParam(name = "name", required = false) String name) {  
    return cityService.getCities(regionId, provinceId, name);  
}
```

API: cosa serve? Lista della spesa.

- progettare la semantica (significato delle API), tenendo presente il linguaggio “funzionale” (DDD *ubiquitous language*), non il linguaggio del DB.
- Progettare il contratto: definire la struttura del JSON+DTO (o dell’XML) da usare nello “scambio dei dati”, ovvero l’interfaccia fornita dai servizi. [il DTO viene serializzato in JSON e deserializzato da JSON]
- Progettare il contratto relativo al messaging: definire la struttura del JSON (o dell’XML) da usare nella segnalazione degli errori (Possibilmente risolto dagli standard aziendali).

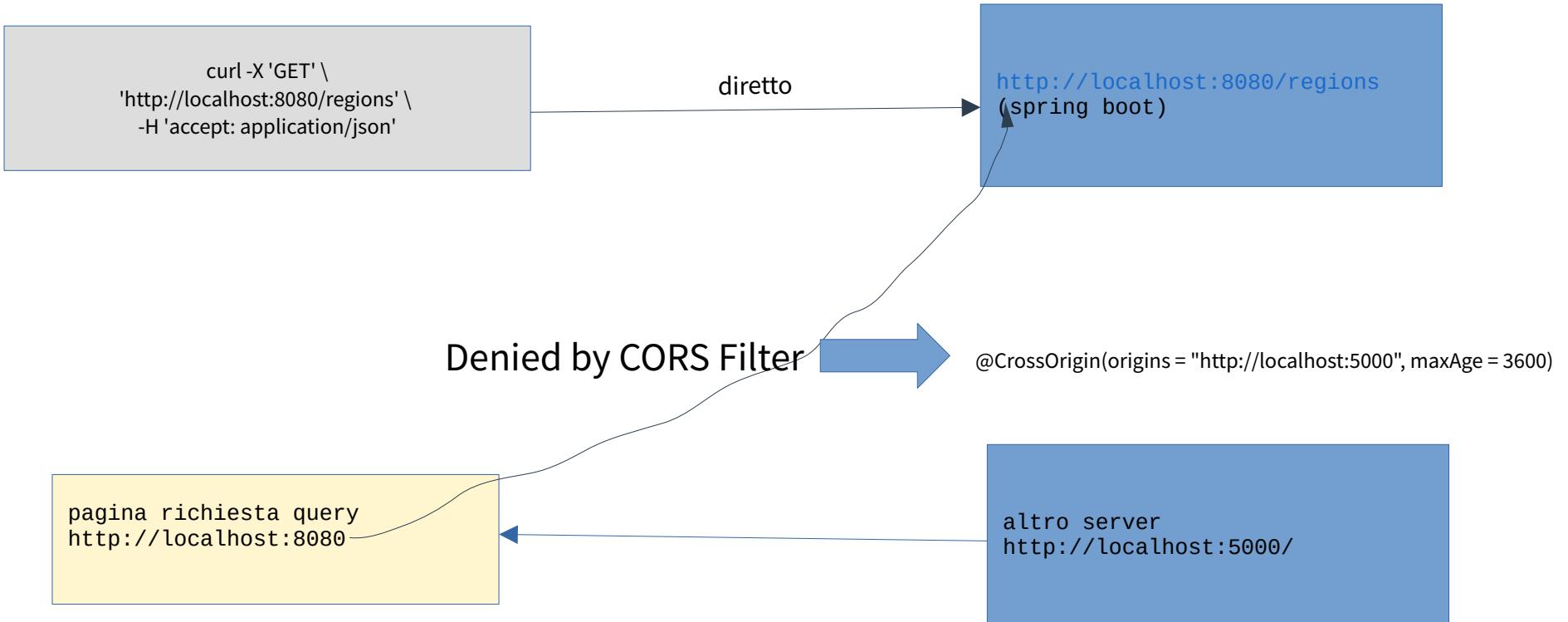
POST e Creazione risorse



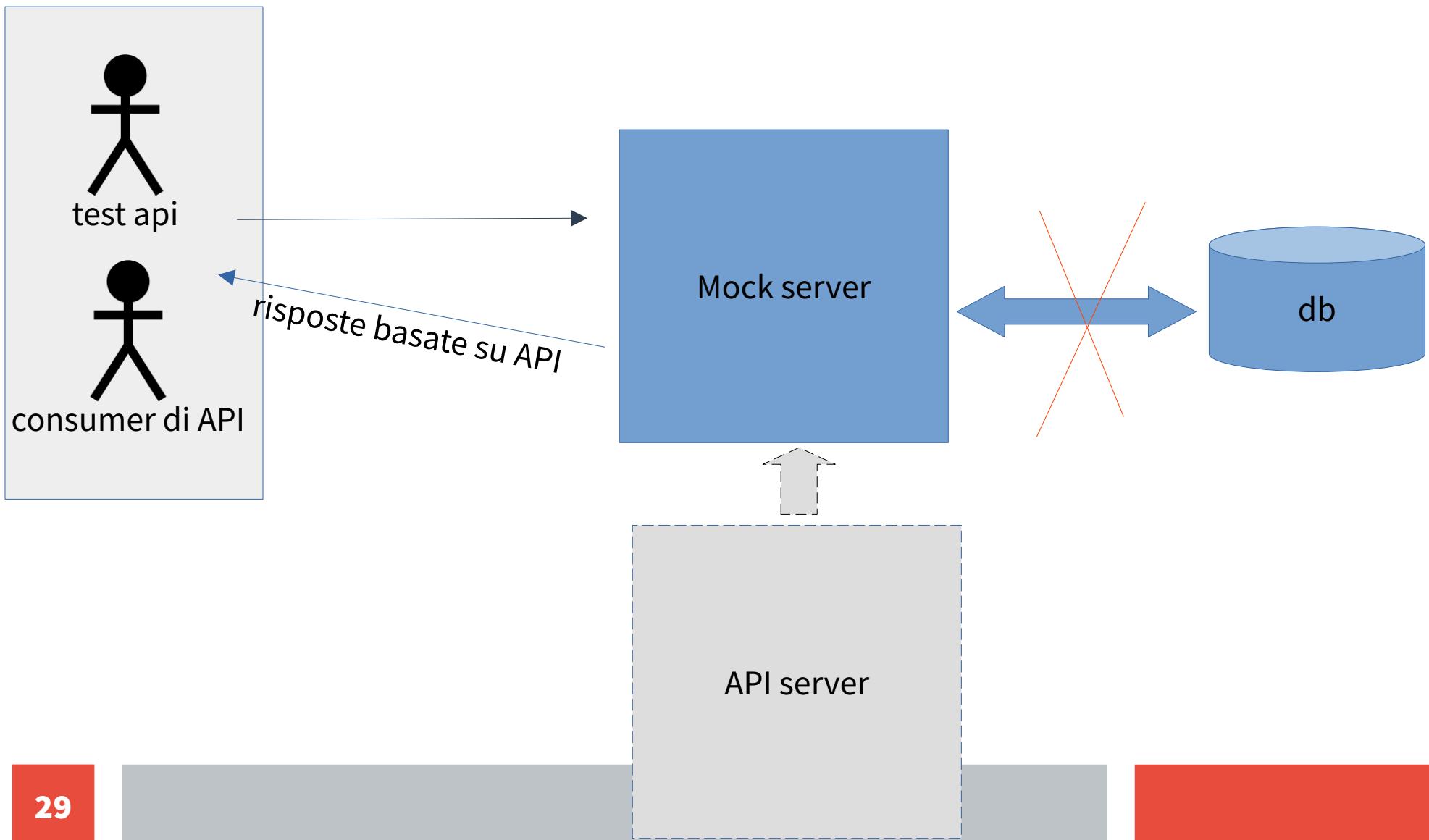
Navigazione con limit

- a) prima volta ==> /products?cursor=abc123&limit=50
- b) altre volte /products?cursor=abc123&limit=50&productId=xyz
dove productId è l'ultimo id dei prodotti ricevuti:
Il server calcola
 - in avanti
`select * from products where productId>xyz limit 50`
 - all'indietro
`select * from products where productId<=xyz limit 50`

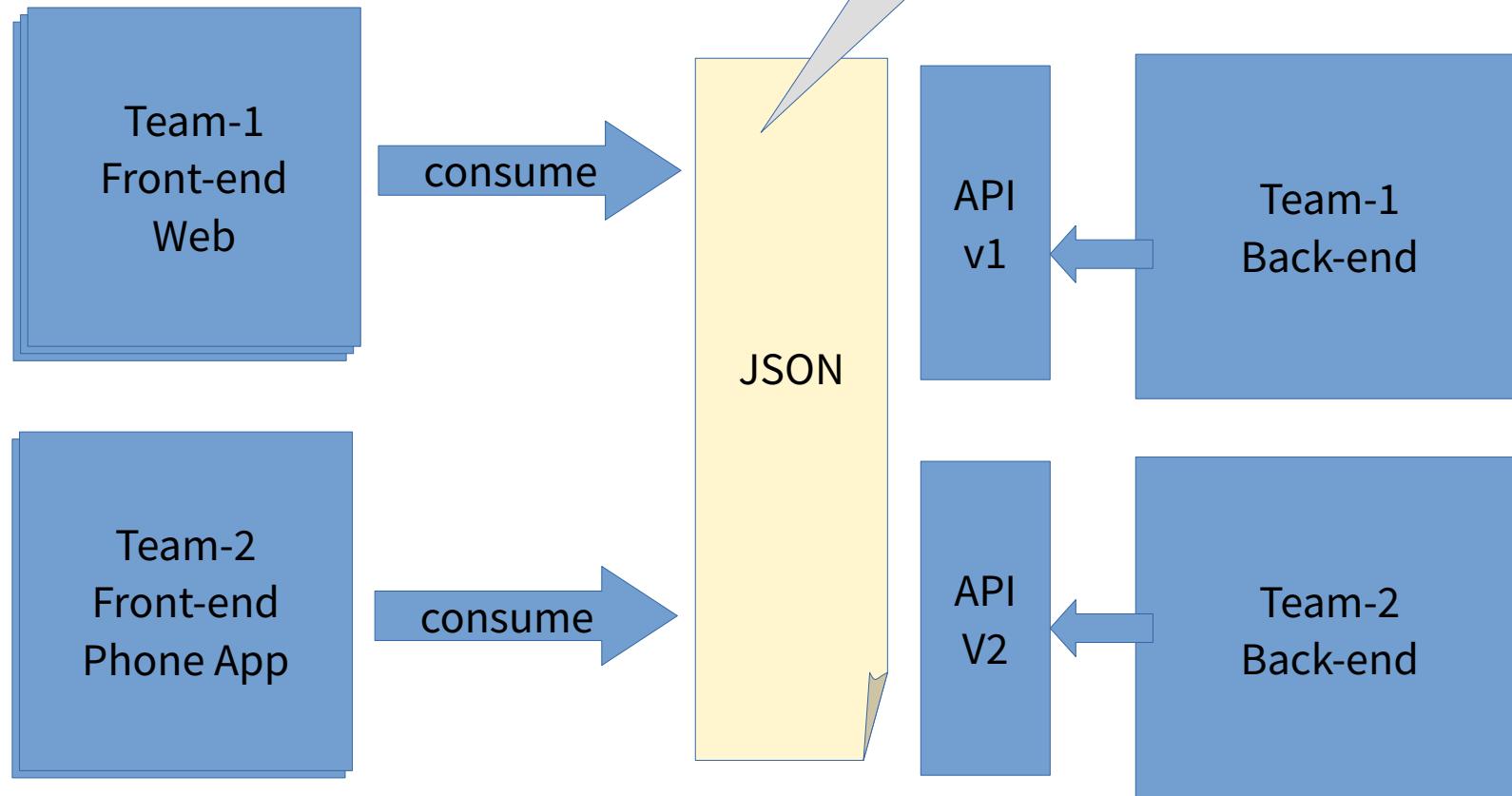
CORS



Mock Server



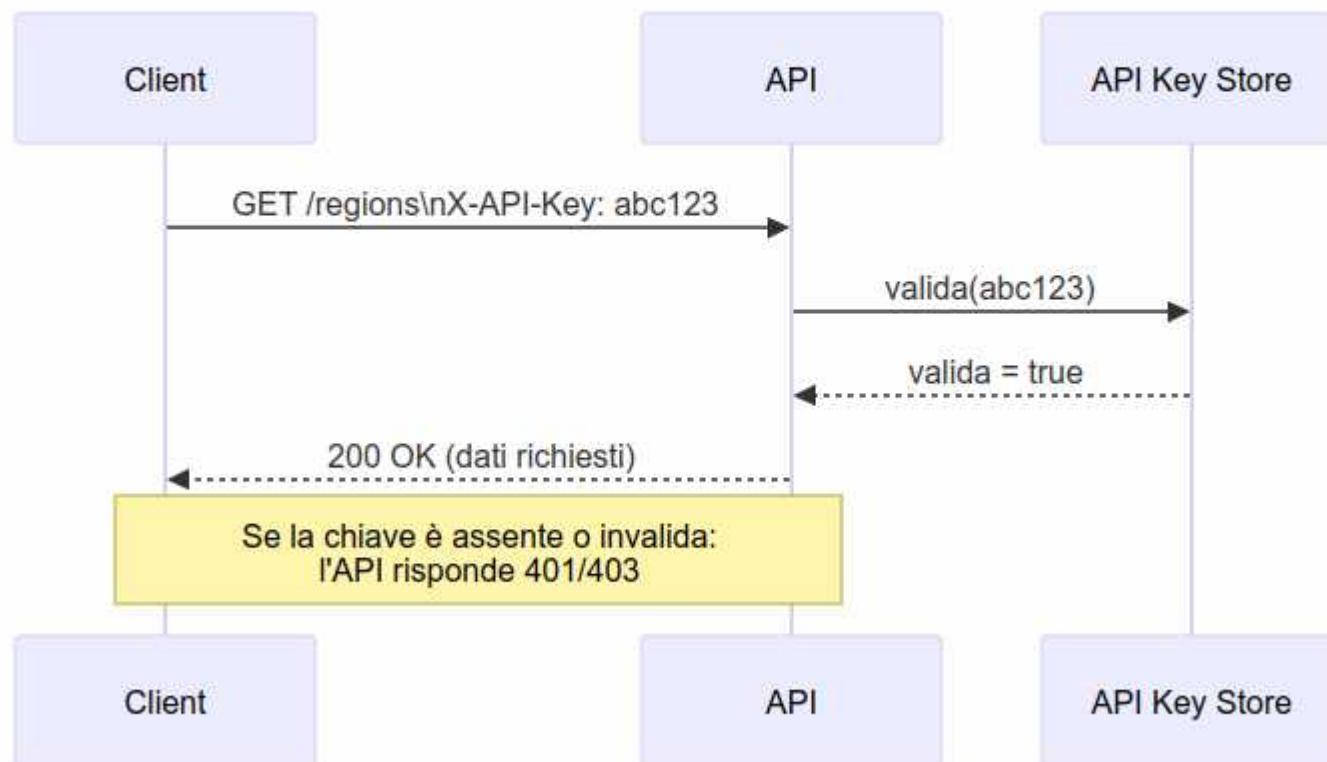
API & Frontend



AUTENTICAZIONE api-key



1. API Key — Sequence



API-KEY HTTP headers spec

```
curl -v -H "X-API-KEY: demo-key-123" "http://localhost:8080/provinces"
```

```
* Host localhost:8080 was resolved.  
* IPv6: ::1  
* IPv4: 127.0.0.1  
* Trying [::1]:8080...  
* Connected to localhost (::1) port 8080  
* using HTTP/1.x  
> GET /provinces HTTP/1.1  
> Host: localhost:8080  
> User-Agent: curl/8.15.0  
> Accept: */*  
> X-API-KEY: demo-key-123  
>  
* Request completely sent off
```

API KEY inviata
dalla request

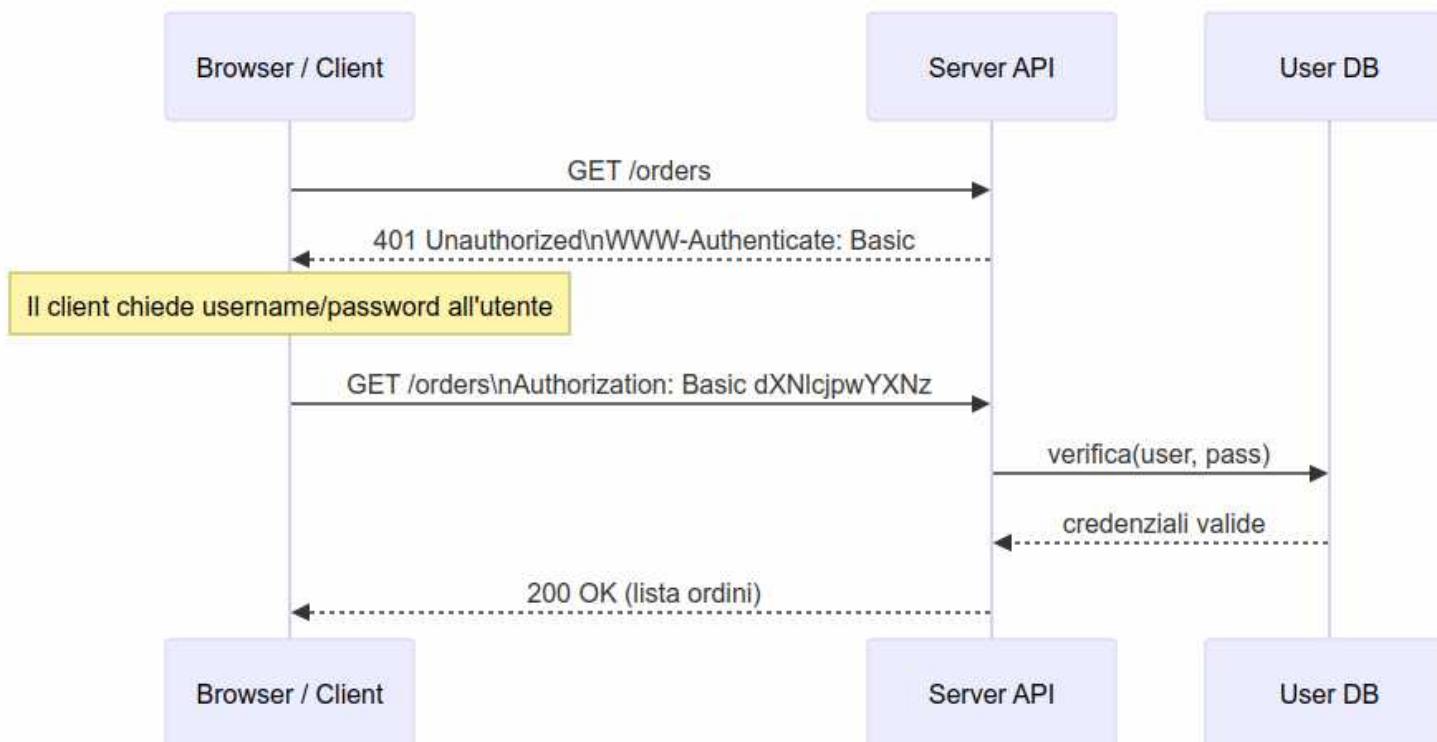
```
< HTTP/1.1 200  
< X-Content-Type-Options: nosniff  
< X-XSS-Protection: 0  
< Cache-Control: no-cache, no-store, max-age=0, must-revalidate  
< Pragma: no-cache  
< Expires: 0  
< X-Frame-Options: DENY  
< Content-Type: application/json  
< Transfer-Encoding: chunked  
< Date: Tue, 09 Dec 2025 14:42:00 GMT  
<  
[{"id":1,"name":"Torino","regionId":1,
```

Body
json delle province

Autenticazione basic

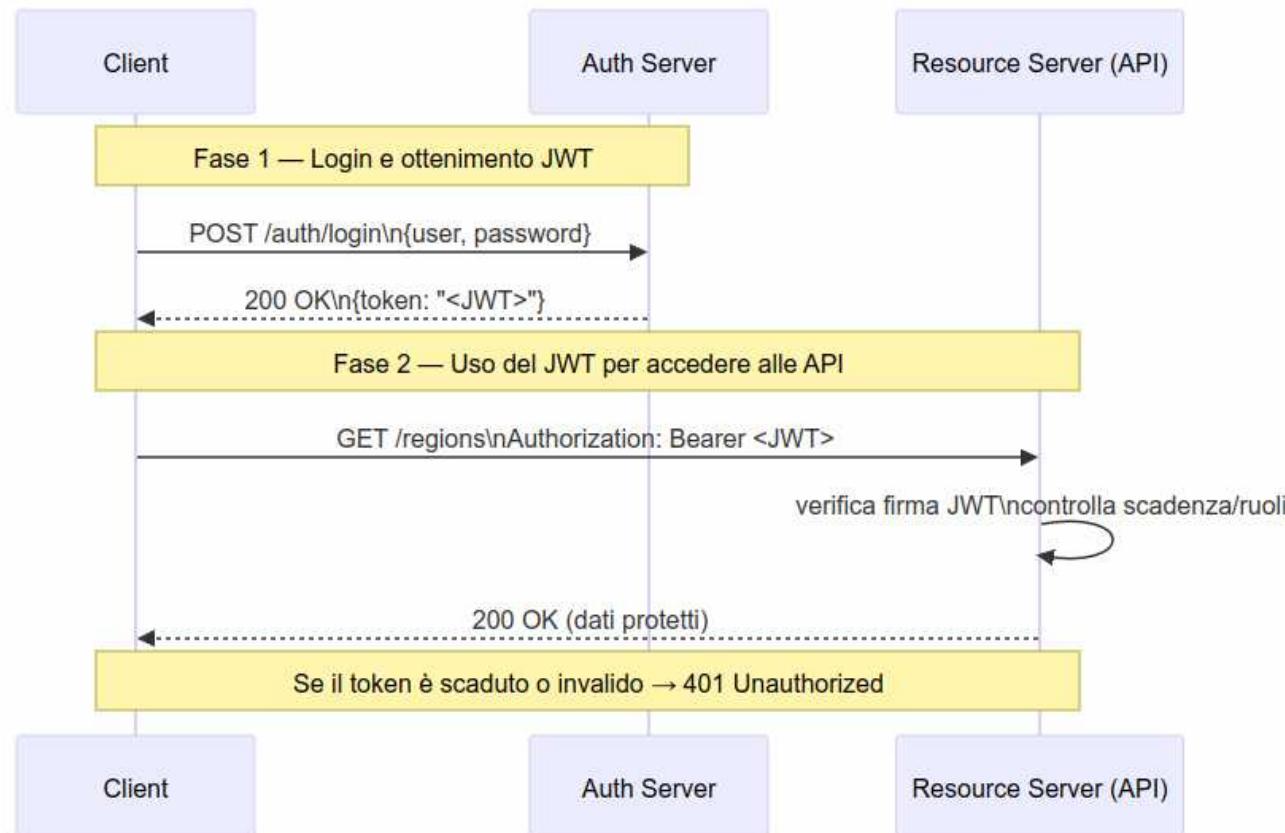


2. Basic Auth — Sequence

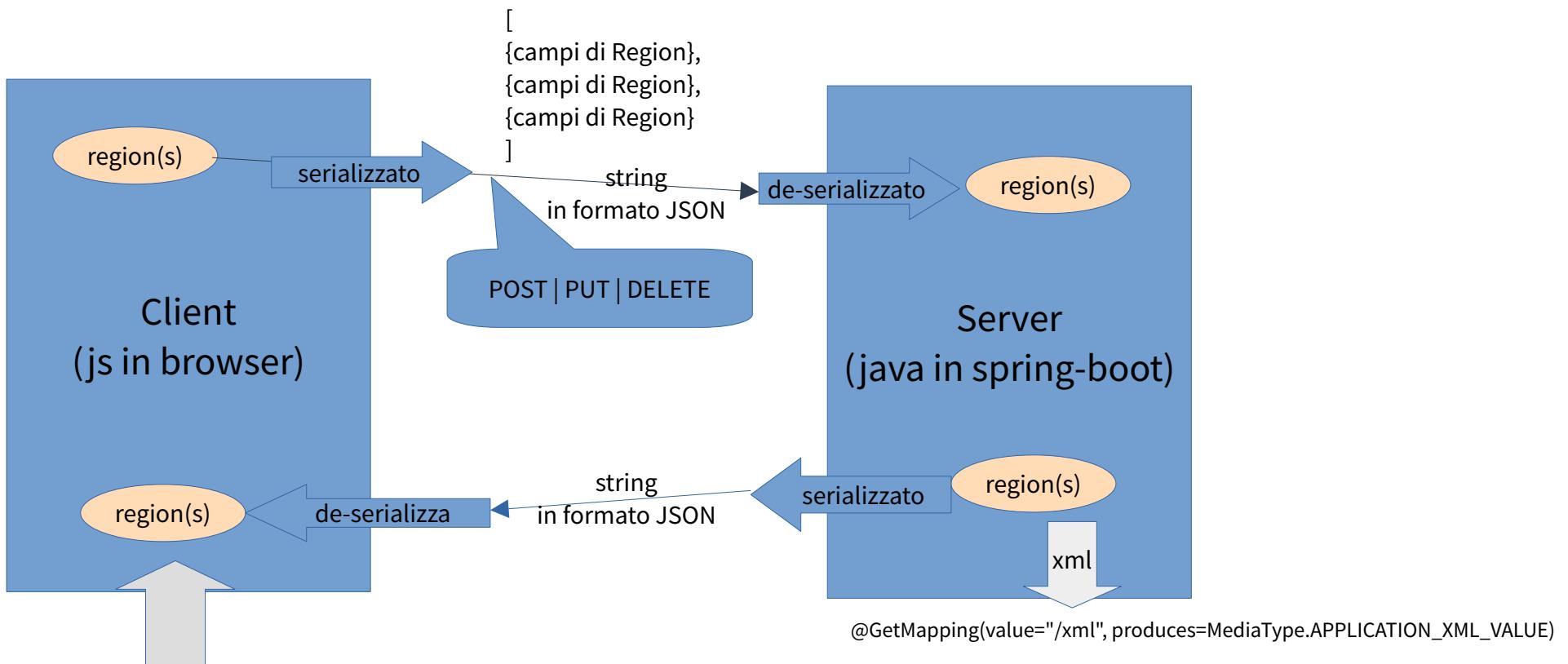


Jwt (Json Web Token) autentication

3. JWT — Sequence (login + uso token)



Serializzazione/Deserializzazione



Cosa serve per provare

- ambiente java

- java jdk (java.oracle.com)
- IDE es: eclipse da www.eclipse.org
- curl (da <https://curl.se/windows/>)
- browser

- base per lo sviluppo

- scheletro di progetto pronto start.spring.io (es: restapi.zip) da importare nell'IDE
- modificare application.properties
- struttura database schema.sql (struttura) data.sql (dati) da posizionare in <progetto>/src/main/resources

Carica schema.sql e data.sql automaticamente

```
spring.sql.init.mode=always
spring.sql.init.schemaLocationsclasspath:schema.sql
spring.sql.init.dataLocationsclasspath:data.sql,.....
```

start.spring.io

The screenshot shows the Spring Initializr web application interface. On the left, there's a sidebar with a menu icon (three horizontal lines), a refresh icon, and a theme switcher icon (sun and moon).

Project section:

- Build Tools: Gradle - Groovy (radio button), Gradle - Kotlin (radio button), Maven (radio button, selected).
- Language: Java (radio button, selected), Kotlin (radio button), Groovy (radio button).

Spring Boot section:

- Version: 4.0.1 (SNAPSHOT) (radio button), 4.0.0 (radio button), 3.5.9 (SNAPSHOT) (radio button), 3.5.8 (radio button, selected), 3.4.13 (SNAPSHOT) (radio button), 3.4.12 (radio button).

Project Metadata section:

- Group: com.example
- Artifact: restapi
- Name: restapi
- Description: Progetto base per la realizzazione API con Java
- Package name: com.example.restapi
- Packaging: Jar (radio button), War (radio button, selected).
- Configuration: Properties (radio button, selected), YAML (radio button).
- Java: 25 (radio button), 21 (radio button, selected), 17 (radio button).

Dependencies section:

ADD DEPENDENCIES... CTRL + B

- GraphQL DGS Code Generation** [DEVELOPER TOOLS]: Generate data types and type-safe APIs for querying GraphQL APIs by parsing schema files. (Remove)
- Spring Boot DevTools** [DEVELOPER TOOLS]: Provides fast application restarts, LiveReload, and configurations for enhanced development experience. (Remove)
- Lombok** [DEVELOPER TOOLS]: Java annotation library which helps to reduce boilerplate code. (Remove)
- Spring Web** [WEB]: Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container. (Remove)
- Spring for GraphQL** [WEB]: Build GraphQL applications with Spring for GraphQL and GraphQL Java. (Remove)
- Spring HATEOAS** [WEB]: Eases the creation of RESTful APIs that follow the HATEOAS principle when working with Spring / Spring MVC. (Remove)
- Spring Security** [SECURITY]: Highly customizable authentication and access-control framework for Spring applications. (Remove)
- Spring Data JPA** [SQL]: Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate. (Remove)
- H2 Database** [SQL]: Provides a fast in-memory database that supports JDBC API and R2DBC access, with a small (2mb) footprint. Supports embedded and server modes as well as a browser based console application. (Remove)

Roadmap

- 1 – struttura db
- 2 – mappare le @Entity
- 3 – @Entity --> cosa mostrare --> dto
- 4 – @Repository (accesso ai dati)
- 5 – @Service (logica di business)
- 6(0) – disegno delle API significato di business+path+parametri
- 7 - @Controller
- 8 – test

@Entity è il legame stretto tra la ns applicazione e l'ORM (Object Relational Mapper). Non usiamo più un RDBMS, ma un prodotto che da un lato fornisce oggetti e dall'altro si unisce al RDMBS.



Roadmap

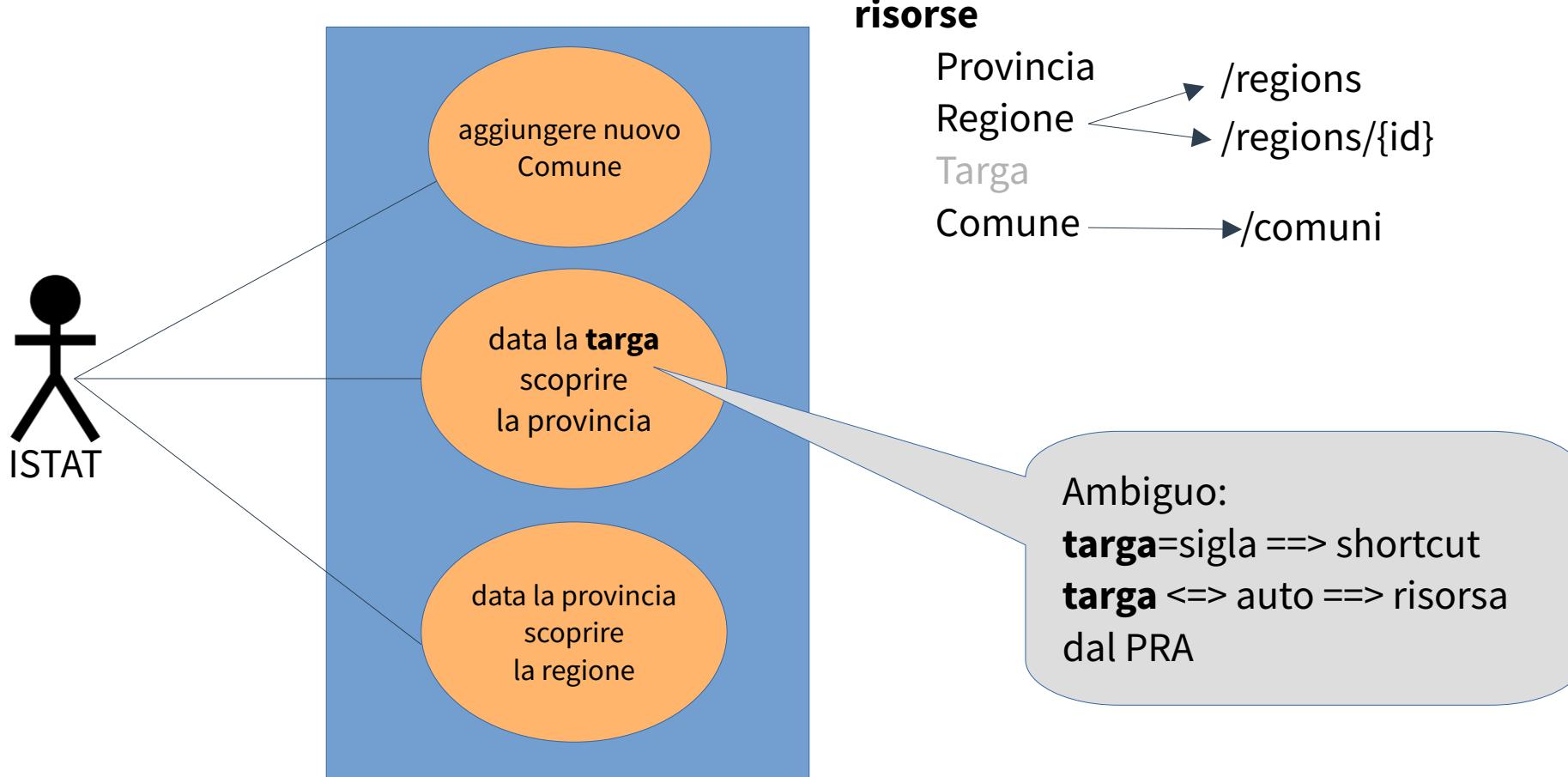
Progettare una piccola API design-first, usando quello che abbiamo visto:

- risorse
- URI
- metodi HTTP
- parametri
- pagination
- esempi
- OpenAPI (livello base)

Statement aggiuntivo del problema:
i **clienti** filtrano per

Cliente implica:
essere una risorsa registrata
livello di controllo accessi (security)

Risorse esempio (italia=>regioni,province,comuni)



E-commerce API

UML Use Case Diagram

Domande guida:

A cosa serve questa realizzazione?

Quali parametri servono davvero al consumer?

Ci sono casi d'uso reali?

Risorse

Prodotto

Ordine

OrdineDettaglio

Categoria



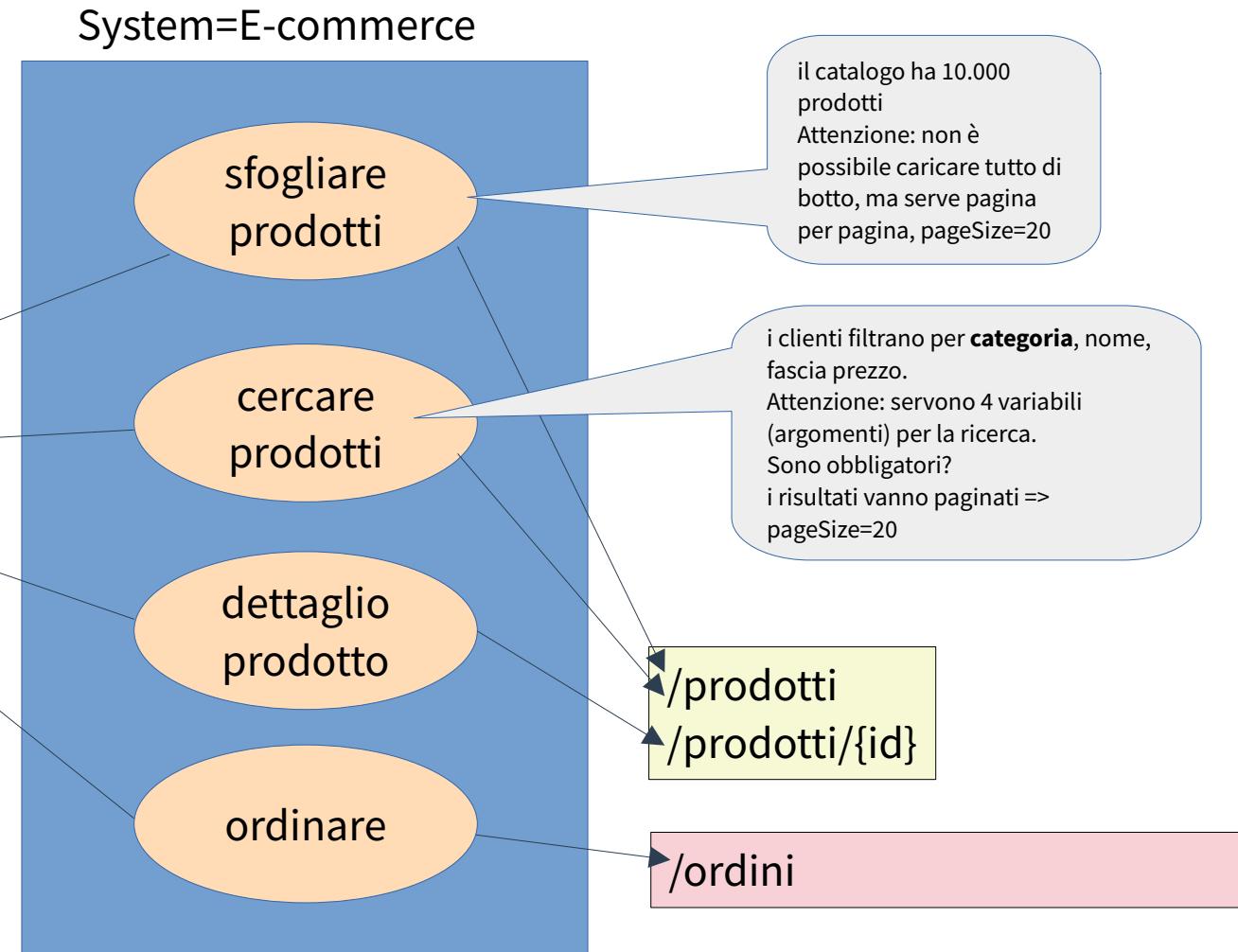
risorse?

URI?

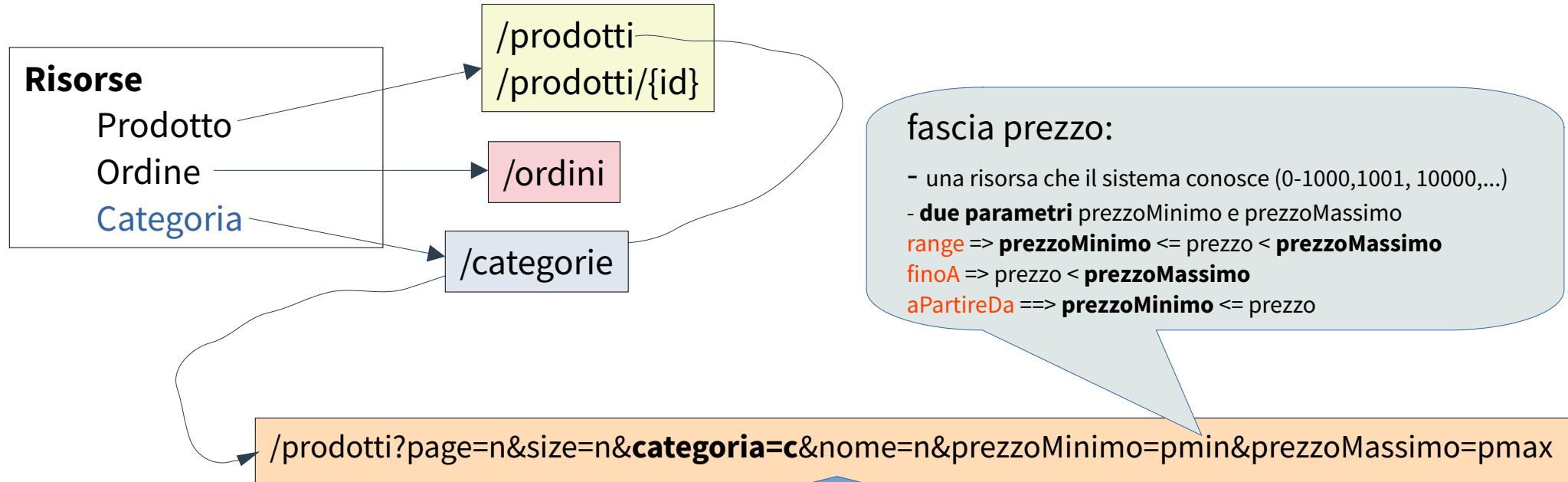
metodi HTTP?

parametri?

pagination?

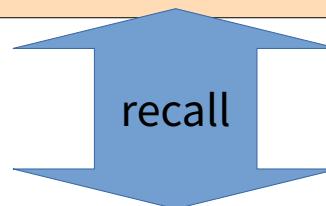


Risorse



fascia prezzo:

- una risorsa che il sistema conosce (0-1000,1001, 10000,...)
- **due parametri** prezzoMinimo e prezzoMassimo
 - range => **prezzoMinimo** <= prezzo < **prezzoMassimo**
 - finoA => prezzo < **prezzoMassimo**
 - aPartireDa ==> **prezzoMinimo** <= prezzo



```
getAllCities(  
    @RequestParam(name = "regionId", required = false) Integer regionId,  
    @RequestParam(name = "provinceId", required = false) Integer provinceId,  
    @RequestParam(name = "name", required = false) String name,  
    @RequestParam(name = "page", required = false) Integer pageParam,  
    @RequestParam(name = "size", required = false) Integer sizeParam)
```

URI

GET /prodotti?page=n&size=n&categoria=c&nome=n&prezzoMinimo=pmin&prezzoMassimo=pmax
GET /prodotti/{id}

GET /ordini?page=n&size=n&clientId=cid

GET /ordini/{id}

POST /ordini

DELETE /ordini/{id}

PUT /ordini/{id}

PATCH /ordini/{id}

GET /categorie

GET /categorie{id}

Manca il sorting, per i prodotti potrebbe essere

sort=price,asc

Mancano default per paging

size = 20

max size = 100

TODO-LIST

per le GET che ritornano liste
occorre definire il criterio di sort

GET lettura

POST crea nuova risorsa

PUT modifica tutta la risorsa

PATCH modifica parzialmente la risorsa

DELETE elimina risorsa

JSON delle risorse Prodotto e Categoria

Prodotto

```
{  
  "idProdotto" : "number",  
  "nome" : "string",  
  "categoria" : "number",  
  "prezzo" : "number",  
  "stock" : "number"  
}
```

Categoria

```
{  
  "idCategoria" : "number",  
  "nome" : "string"  
}
```

ProdottoResponse

```
{  
  "idProdotto" : "number",  
  "nome" : "string",  
  "categoria" : "number"  
  "prezzo" : "number",  
  "stock" : "number"  
}
```

CategoriaResponse

```
{  
  "idCategoria" : "number",  
  "nome" : "string"  
}
```

Per pattern DTO

CONTRATTO

JSON delle risorse Ordine (segue swagger editor)

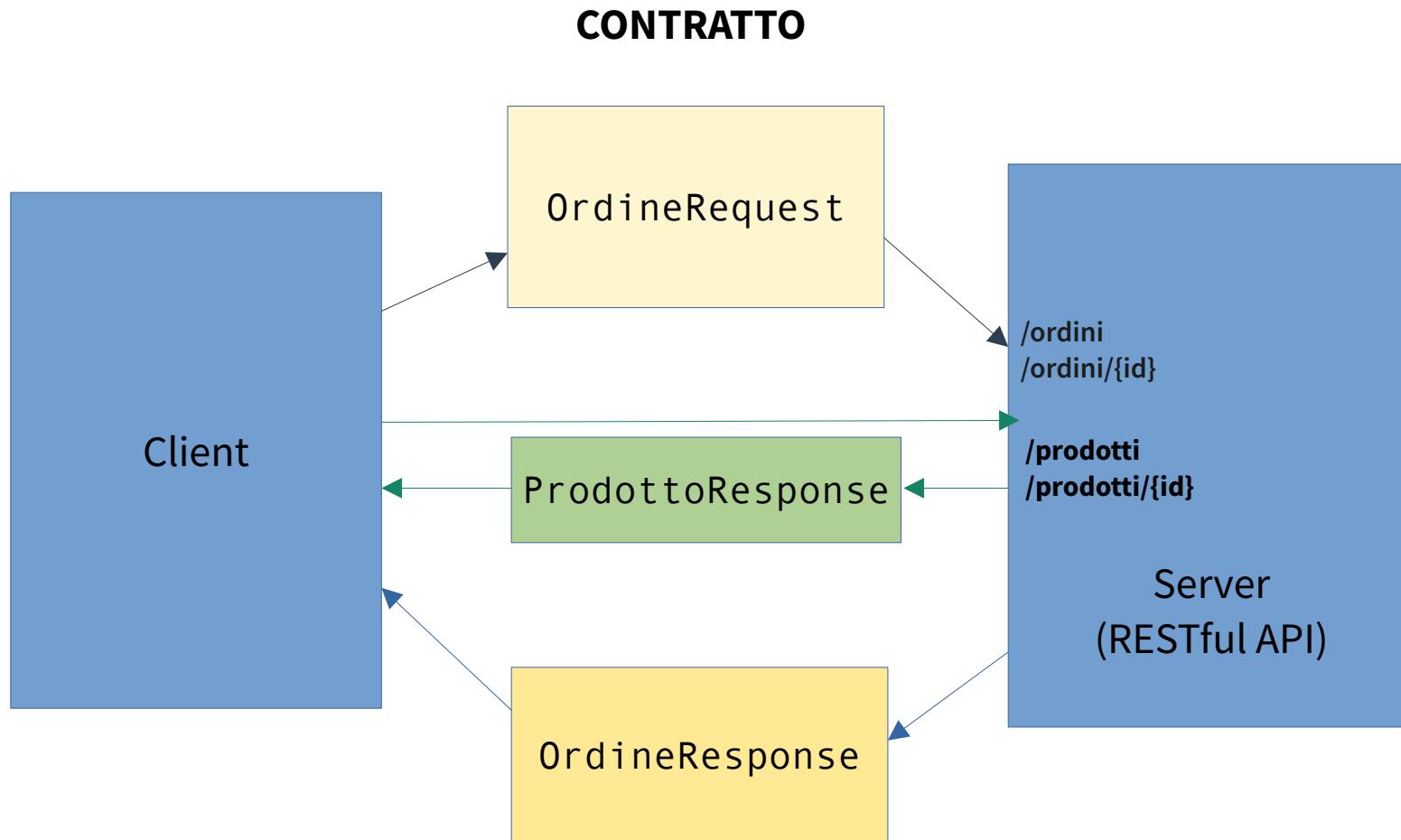
```
Ordine
{
  "idOrdine" : "number",
  "data" : "string",
  "valore" : "number",
  "customer" : "?",
  "linee" : [
    {
      "idProdotto" : "number",
      "quantita" : "number",
      "prezzo_unitario" : "number"
    }
  ]
}
```

```
OrdineRequest
{
  "data" : "string",
  "valore" : "number",
  "customer" : "?",
  "linee" : [
    {
      "idProdotto" : "number",
      "quantita" : "number",
      "prezzo_unitario" : "number"
    }
  ]
}
```

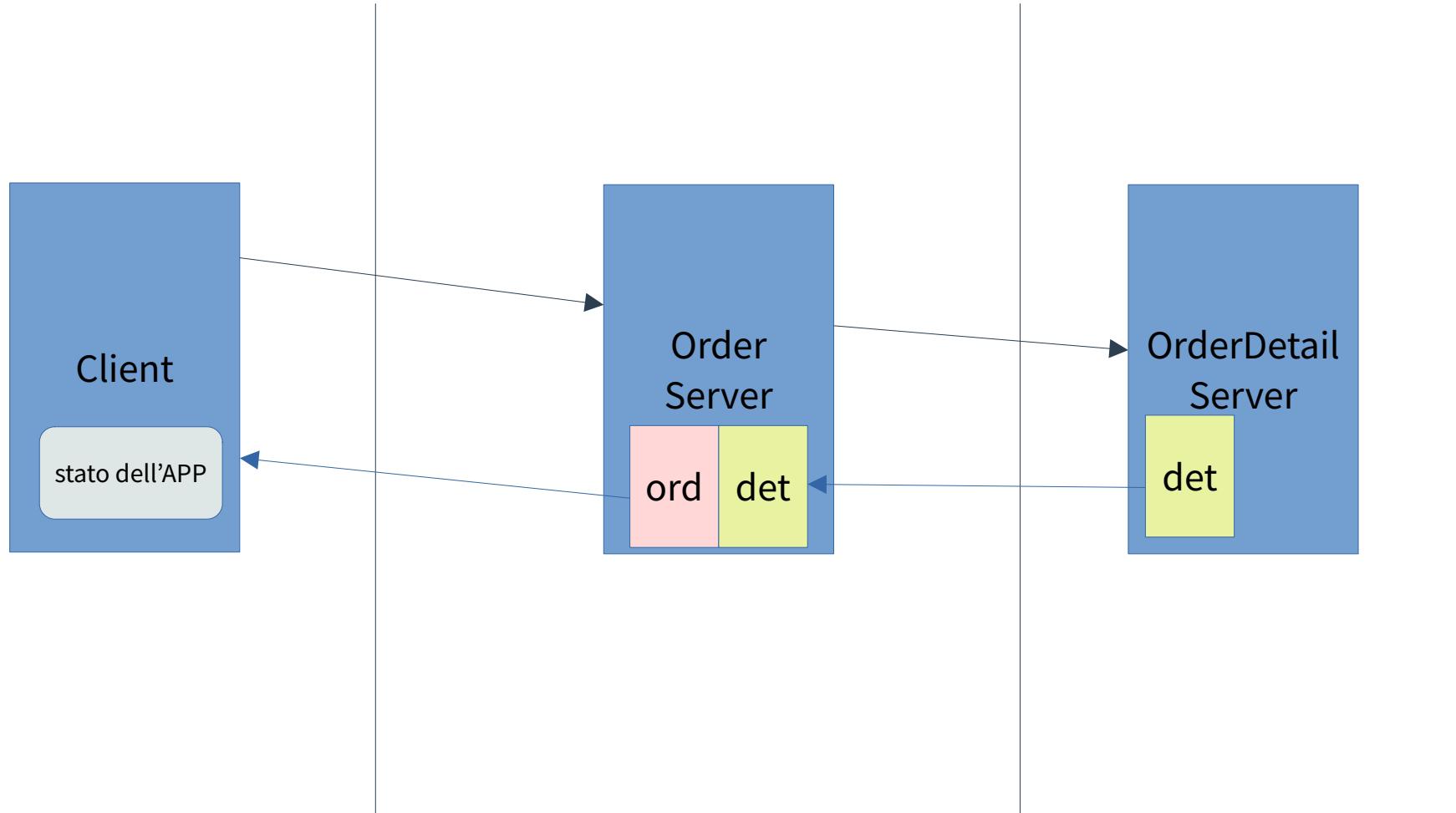
```
OrdineResponse
{
  "idOrdine" : "number",
  "data" : "string",
  "valore" : "number",
  "customer" : "?",
  "linee" : [
    {
      "idProdotto" : "number",
      "quantita" : "number",
      "prezzo_unitario" : "number"
    }
  ]
}
```

CONTRATTO

uso del contratto



layered system



POST e idempotenza

POST non è idempotente by default

se mi serve una POST idempotente devo poter distinguere la risorsa da creare in qualche modo e controllare che non sia stata già creata. Implicare una GET con criteri

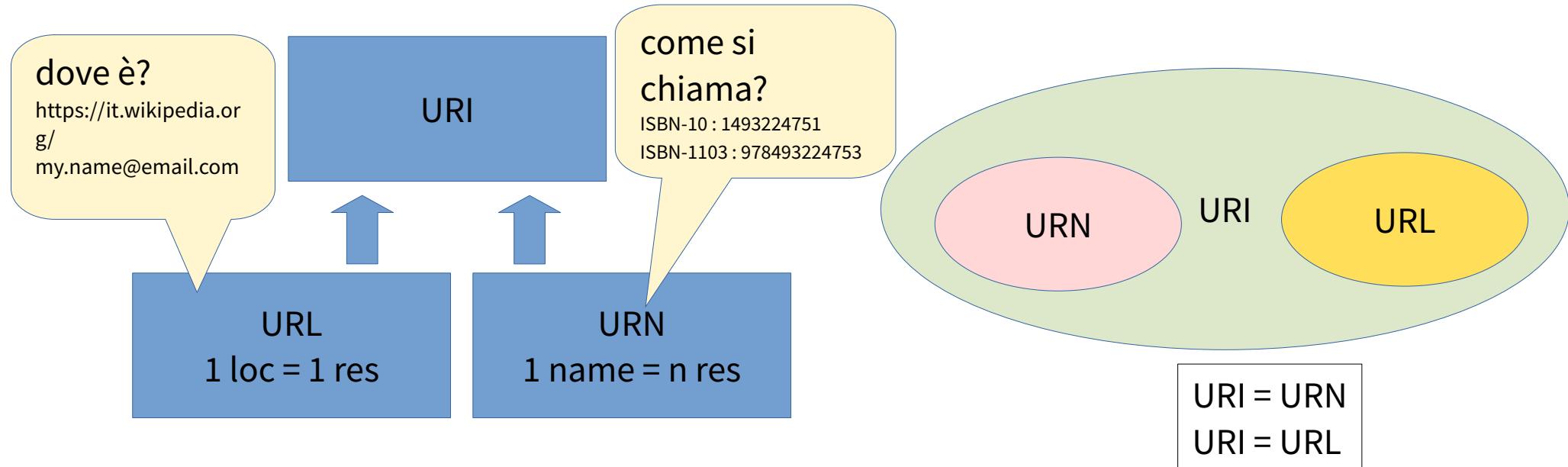
esempio di POST e idempotenza logica

URI <==> URL e URN

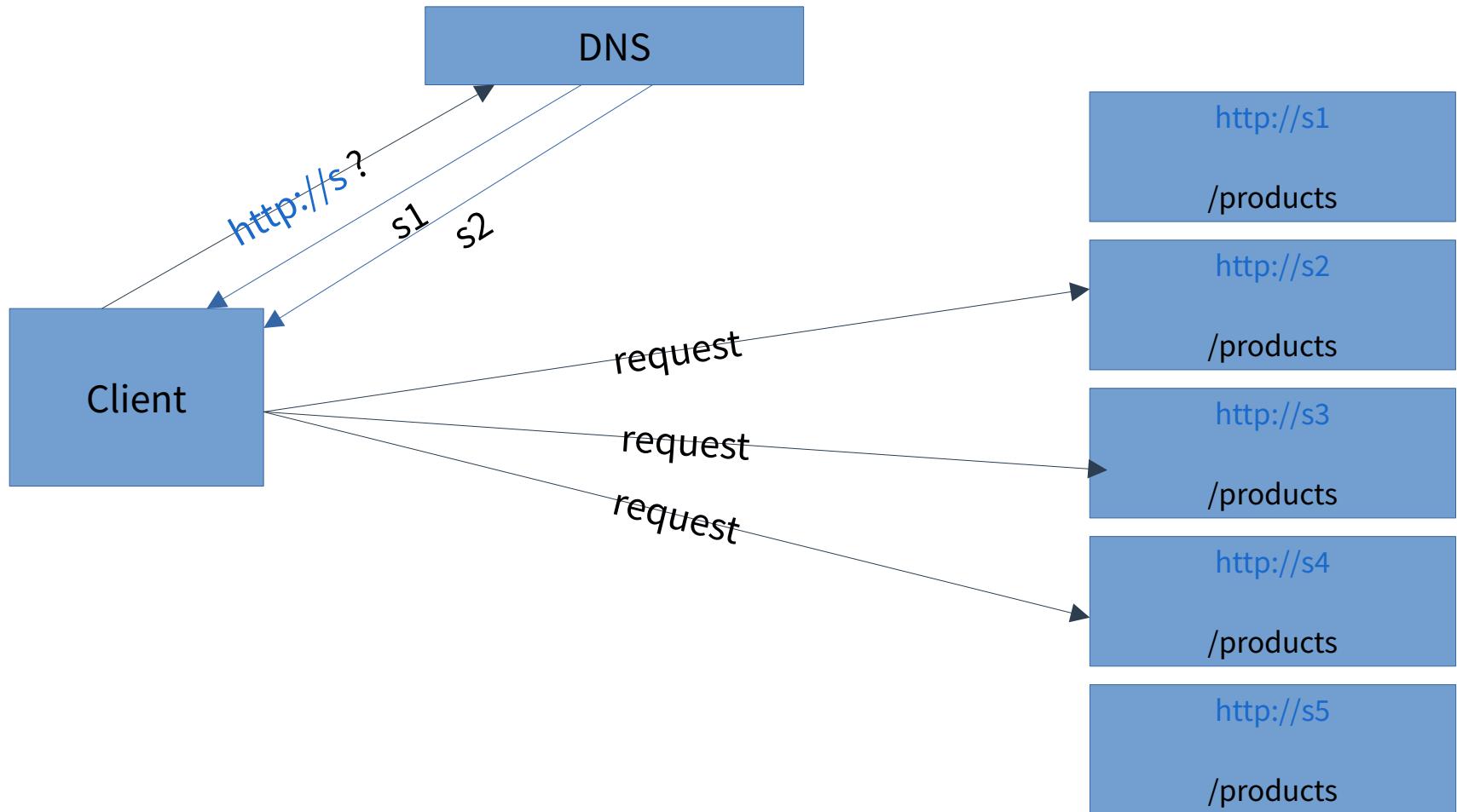
URI = Uniform Resource Identifier

URL = Uniform Resource Locator

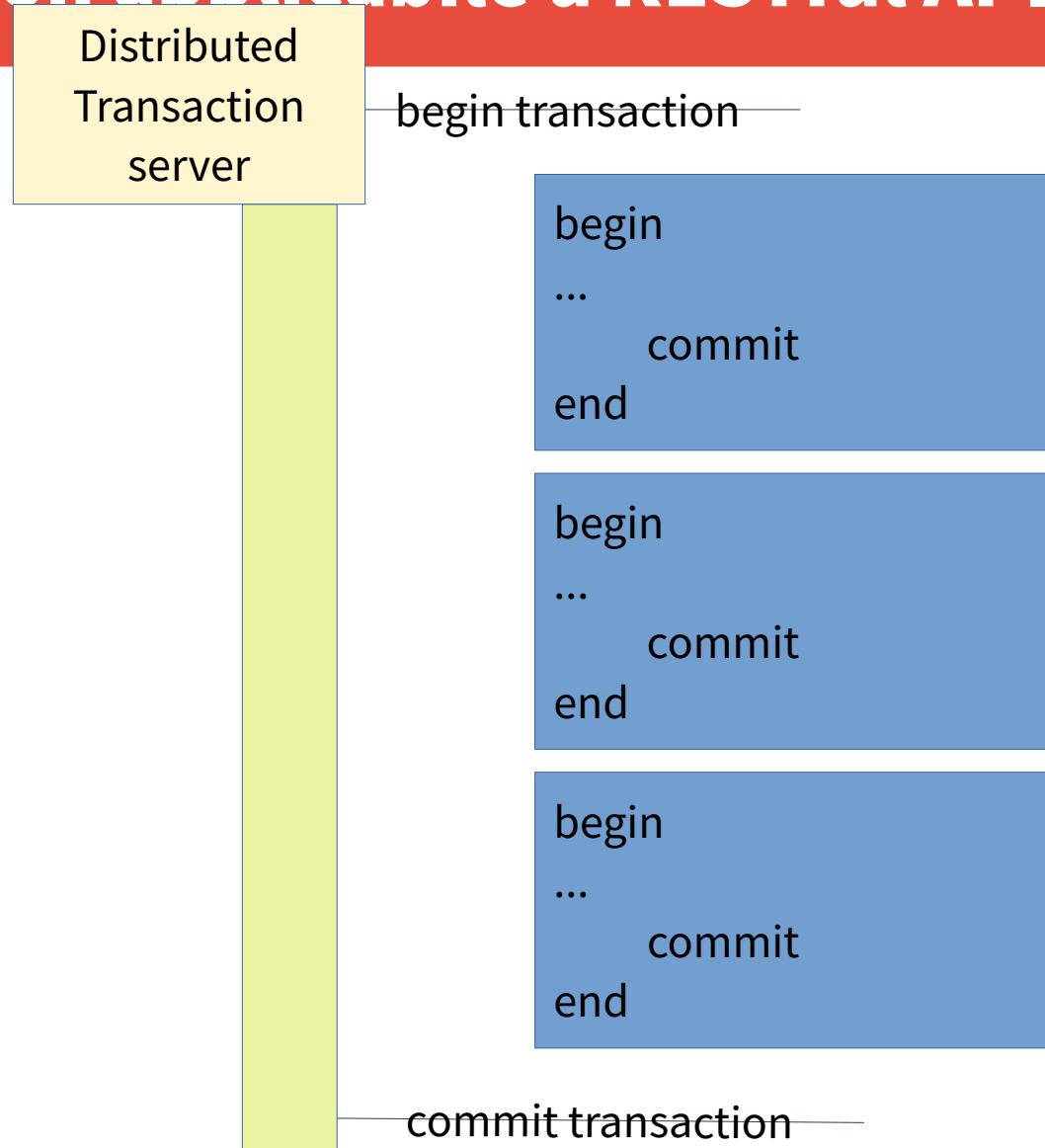
URN = Uniform Resource Name



API-URL-Server

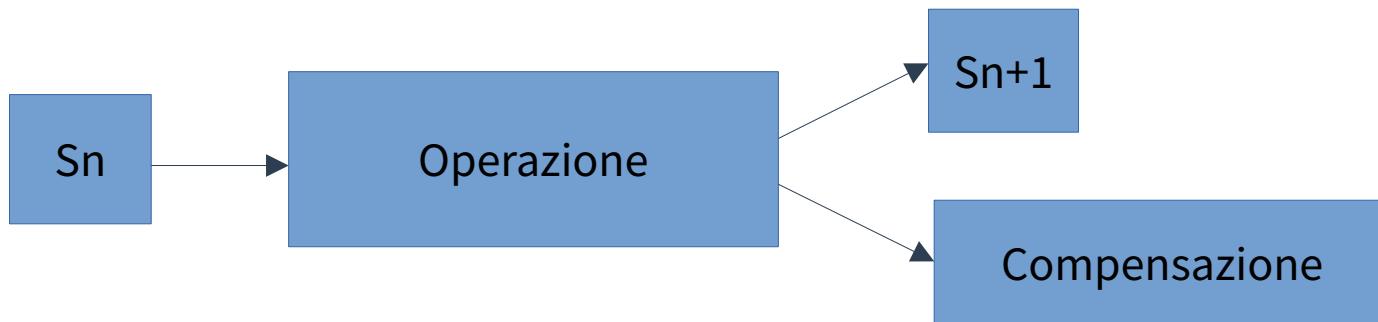


2PC non applicabile a RESTful API

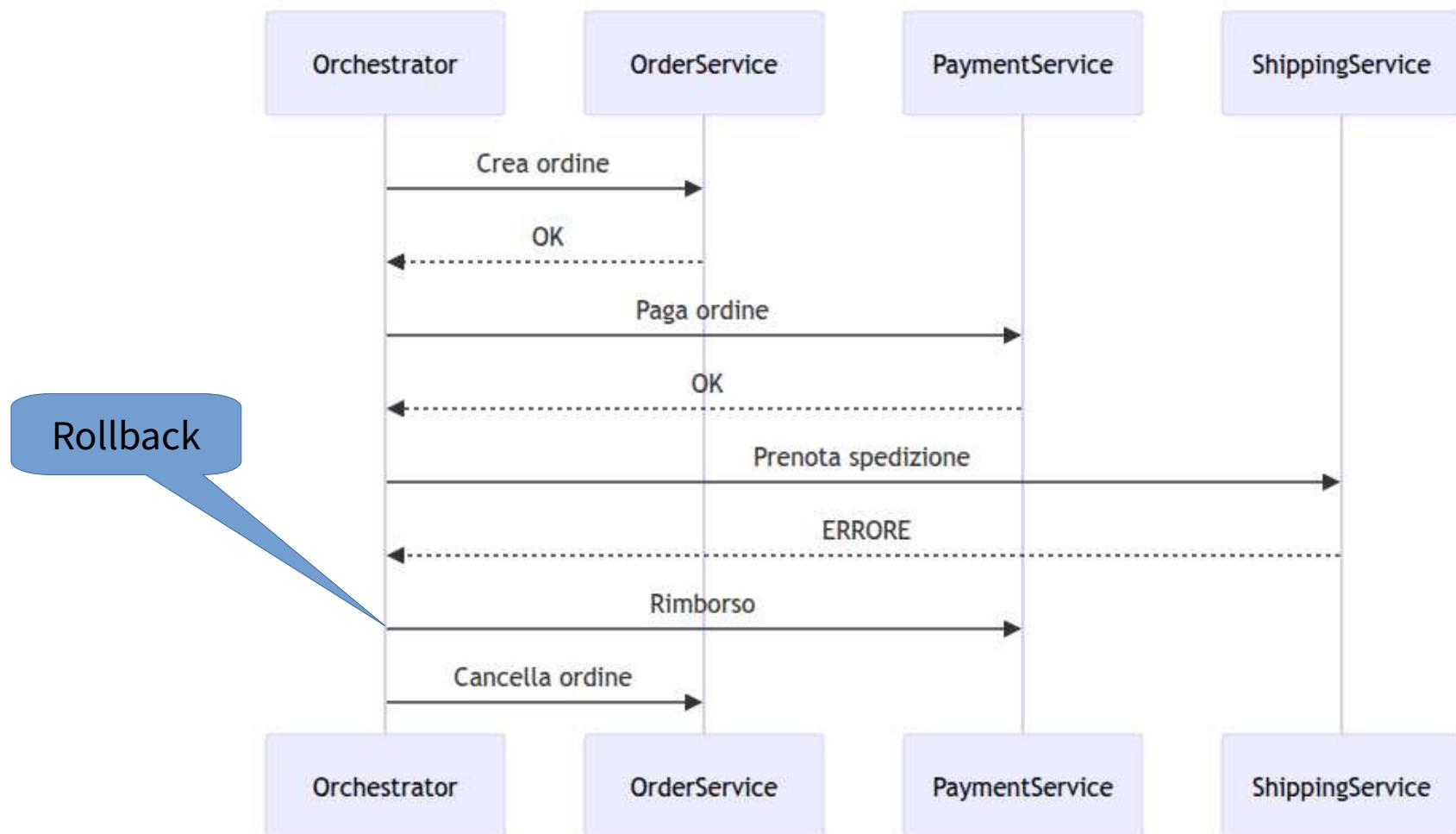


Saga Pattern

- 1) spezzare il percorso in tappe ben definite (local transaction?)
- 2) definire stato di arrivo, eseguo operazione
- 3.1) se tutto ok la macchina a stati evolve `stato_attuale -> stato_passo_completato`
- 3.2) se qualcosa va storto devo eseguire una compensazione (equivale alla rollback)
- 4) passo allo step successivo e ritorno a 2

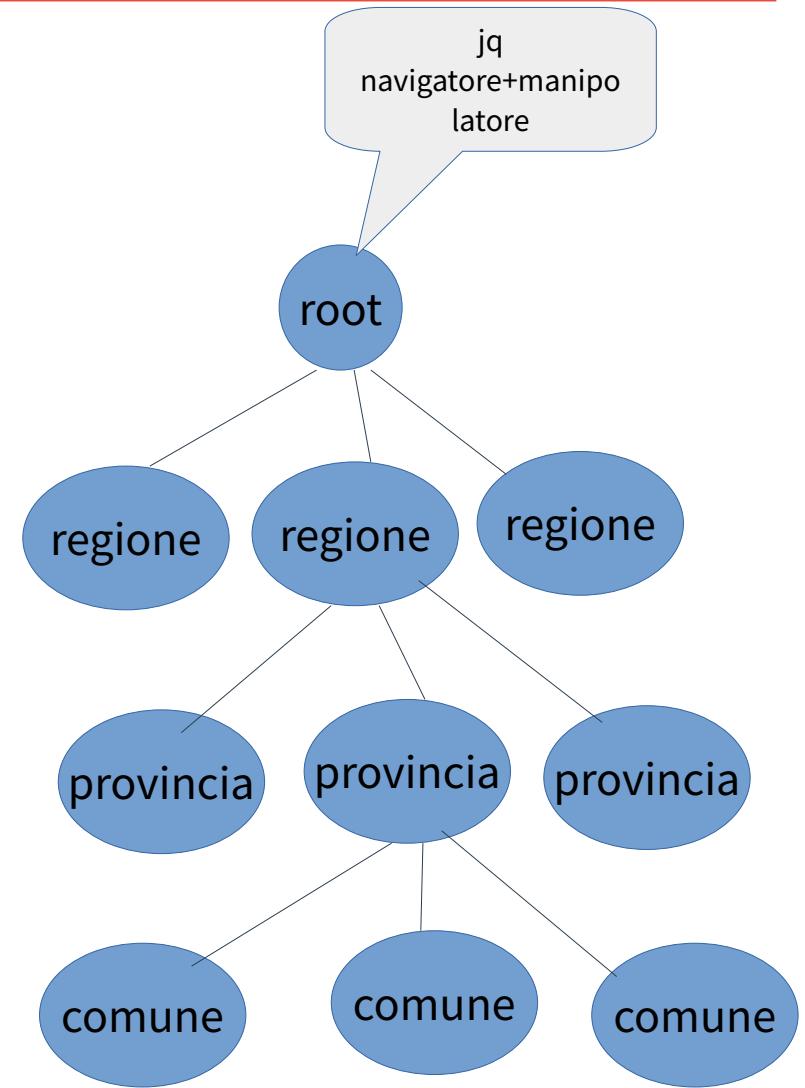


Compensazioni



italy_fixed.json & jq (<https://jqlang.org/>)

```
{  
    "regioni": [  
        {  
            "_key": "nome_regione",  
            "id": 123,  
            "nome": "Nome Regione",  
            "latitudine": 42.0,  
            "longitudine": 13.0,  
            "province": [  
                {  
                    "_key": "nome_provincia",  
                    "id": 456,  
                    "nome": "Nome Provincia",  
                    "sigla_automobilistica": "XX",  
                    "latitudine": 42.0,  
                    "longitudine": 13.0,  
                    "comuni": [  
                        {  
                            "_key": "nome_comune",  
                            "id": 789,  
                            "nome": "Nome Comune",  
                            "codice_catastale": "A123",  
                            "latitudine": 42.0,  
                            "longitudine": 13.0  
                        }  
                    ]  
                }  
            ]  
        }  
    ]  
}
```



Estrarre i comuni da una specifica provincia

```
jq '.regioni[].province[] | select(.nome == "Milano") | .comuni[].nome' italy_fixed.json
```

1. ` `.regioni[]` - itera su tutte le regioni dell'array principale
2. ` `.province[]` - Per ogni regione, itera su tutte le sue province
3. ` select(.nome == "Chieti")` - filtra selezionando solo la provincia con nome "Chieti"
4. ` `.comuni[]` - Si accede all'array dei comuni della provincia selezionata
5. ` `.nome` - Si estrae il campo "nome" di ogni comune
6. Il risultato è una lista di nomi di comuni, uno per riga

Pausa Caffè

rientro 16:00