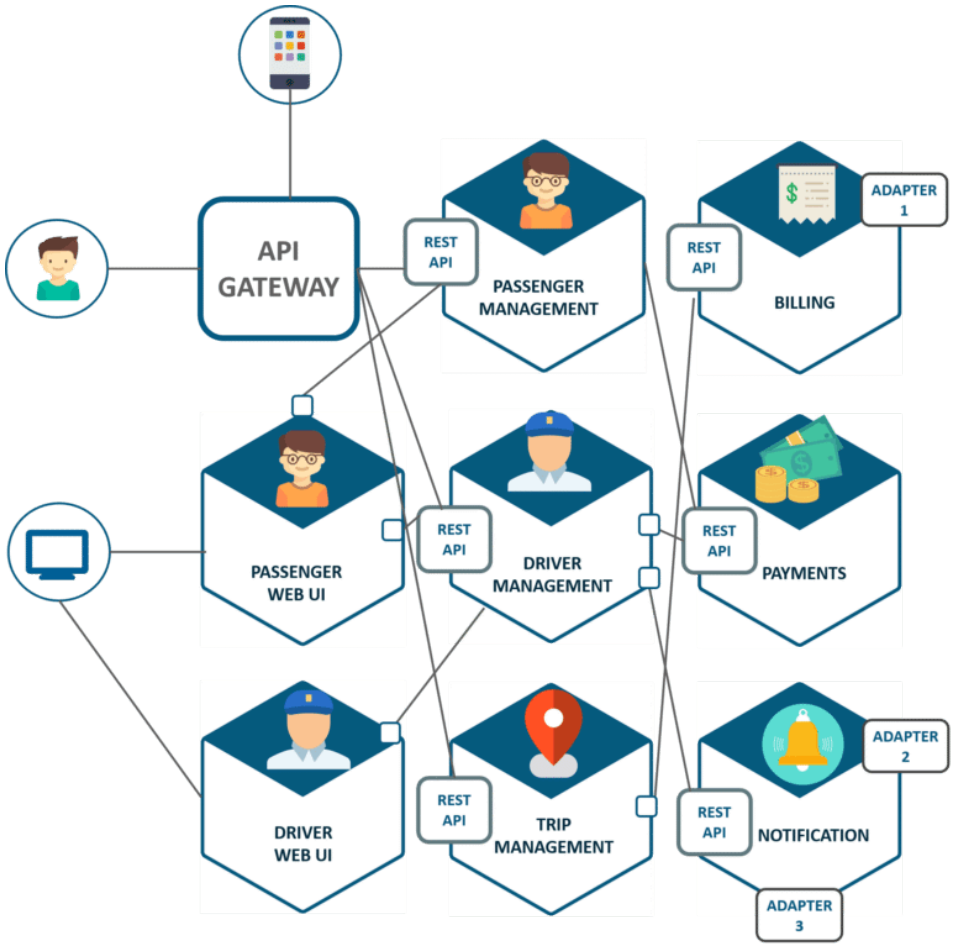
ALOM

 INTEROPERABILITY

UBER


API
GATEWAY
REST
API
PASSENGER
MANAGEMENT
REST
API
BILLING
ADAPTER
1
PASSENGER
WEB UI
REST
API
DRIVER
MANAGEMENT
REST
API
PAYMENTS
DRIVER
WEB UI
REST
API
TRIP
MANAGEMENT
REST
API
NOTIFICATION
ADAPTER
2
ADAPTER
3

PROBLÉMATIQUE

- Comment communiquer avec les autres micro-services?
- Comment communiquer avec les partenaires?

COMMENT FAIRE COMMUNIQUER DES PROCESSUS ?

Sur une même machine : IPC - Inter Process Communication

- mémoire partagée
- message queue
- sémaphores

Sur des machines séparées : Réseau

- sockets

EN JAVA ☕ - RMI (REMOTE METHOD INVOCATION)

Communication entre 2 JVM

`java.rmi.*`

Définition d'une interface qui `extends` `java.rmi.Remote`.

Paramètres sérialisés en binaire, interface `java.io.Serializable` à implémenter.

CONTRAINTES ⛓

Toutes les applications ne sont pas écrites dans le même langage (Java, .Net, NodeJS, PHP, Ruby, Python...)

Les partenaires n'ont pas forcément les mêmes environnements (réseaux, firewall)

SOLUTION 🎉

Définition d'une norme de communication, basée sur des standards.

WEB SERVICES & WEB-SOCKETS 🌍

- Protocole HTTP(S)
  - Facile à implémenter (texte)
  - Passe les firewalls (port 80/443)
  - Sécurisation avec SSL/TLS 🔒
- Formats de données
  - SOAP : XML
  - REST : JSON
- Contrat de service
  - SOAP : WSDL
  - REST : Swagger...

WEB SERVICES REST

WEB SERVICES REST

REpresentational

State

Transfert

WEB SERVICES REST

PRINCIPES ARCHITECTURAUX

- Architecture découplée client/serveur
- Sans état (pas de session)
- Accès à des ressources:
  - Identifiées de manière unique
  - Manipulées via des représentations (JSON, XML, HTML...)
  - Compatible avec une mise en cache
  - Données Hypermédia

WEB SERVICES REST

Utilisation des codes HTTP (Http Status Dogs)

❌ don’t

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 15

{"error":"Content Not Found"}
```

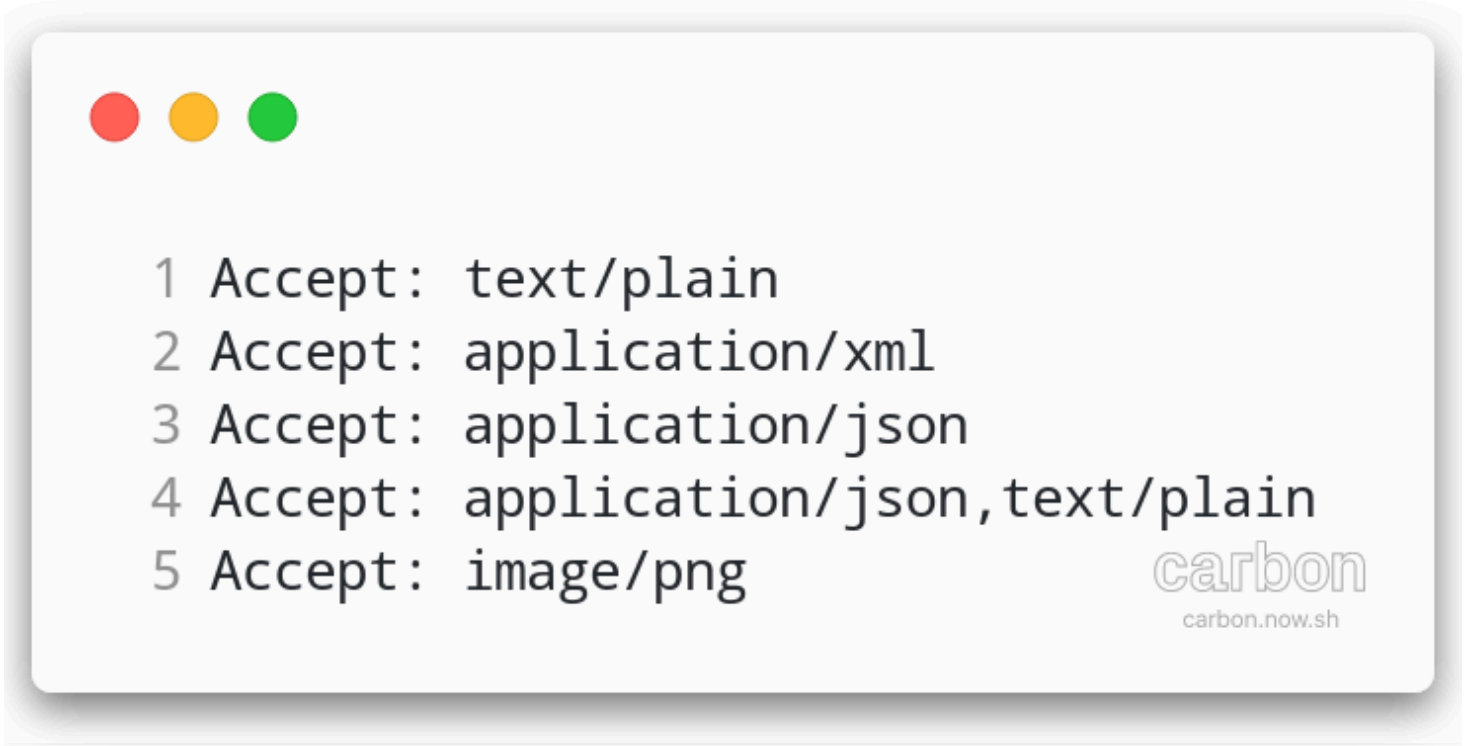
✅ do

```
HTTP/1.1 404 NOT FOUND
Content-Type: application/json
Content-Length: 15

{"error":"Pokemon with id {152} does not exists"}
```

NÉGOCIATION DE CONTENU

Principes au coeur du web

Le client indique au serveur ses attentes via des headers HTTP



NÉGOCIATION DE CONTENU

- Format des données : header `Accept`
- Traduction : header `Accept-Language`
- Retour avec les headers `Content-Type` et `Content-Language`

La RFC 4229 liste les headers possibles.

```
1 GET /pokemons-types/25
2 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
3 Accept-Language: en-US,en;q=0.5
4
5 HTTP/1.1 200
6 Content-Type: application/json;charset=UTF-8
7 {...}
```

WEB SERVICES REST

HATEOAS : HYPERMEDIA AS THE ENGINE OF APPLICATION STATE

Le message contient les informations permettant de manipuler l'application

WEB SERVICES REST : EXEMPLE JSON

```
GET /account/12345 HTTP/1.1
Host: somebank.org
Accept: application/json
...

HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: ...

{
    "account_number" : "12345"
    "balance" : 100.00,
    "currency" : "usd",
    "links" : [
        { "rel": "deposit", "href" : "http://somebank.org/account/12345/deposit" },
        { "rel": "withdraw", "href" : "http://somebank.org/account/12345/withdraw" },
        { "rel": "transfert", "href" : "http://somebank.org/account/12345/transfert" },
        { "rel": "close", "href" : "http://somebank.org/account/12345/close" }
    }
}
```

HATEOAS

Excellent talk de Julien Topçu

CONTRAT DE SERVICE REST

OPENAPI (EX SWAGGER)

OpenAPI est la spec, *Swagger* une implémentation

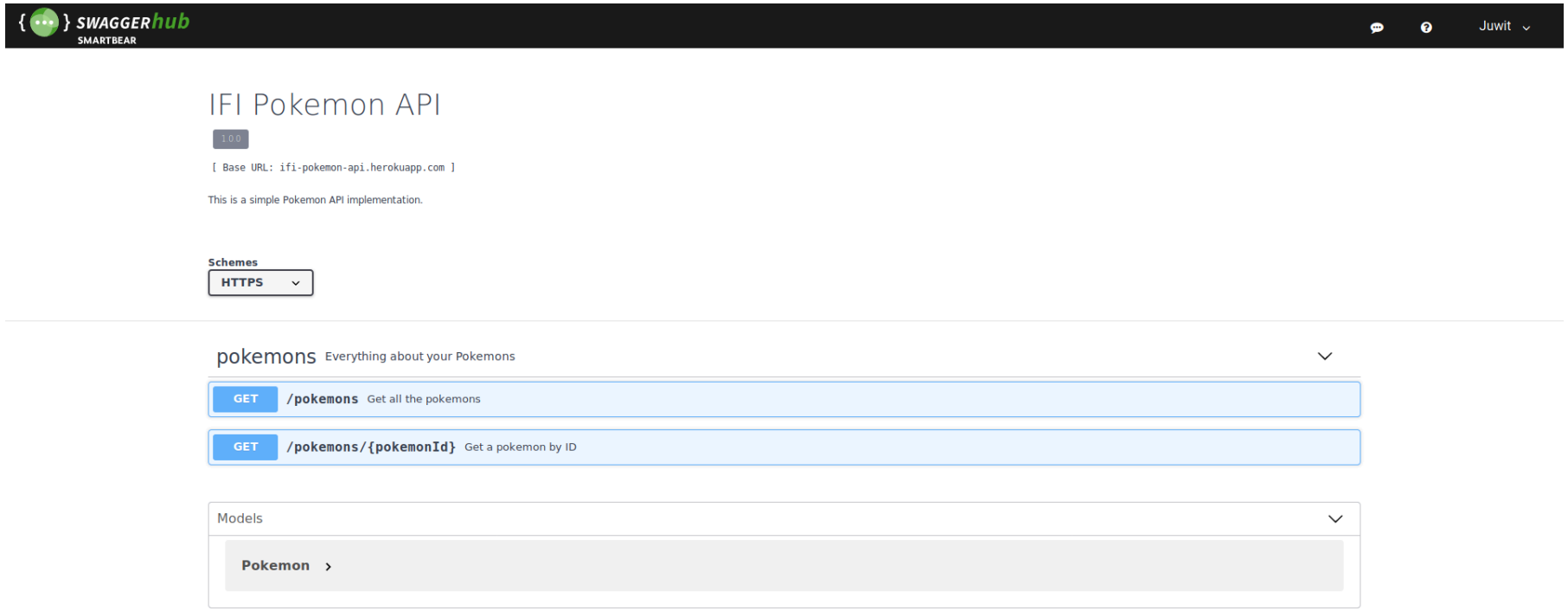
Description des API au format JSON ou YAML : *OpenApi*

Rendu Web + "Try It" : *Swagger*

Pokemon API

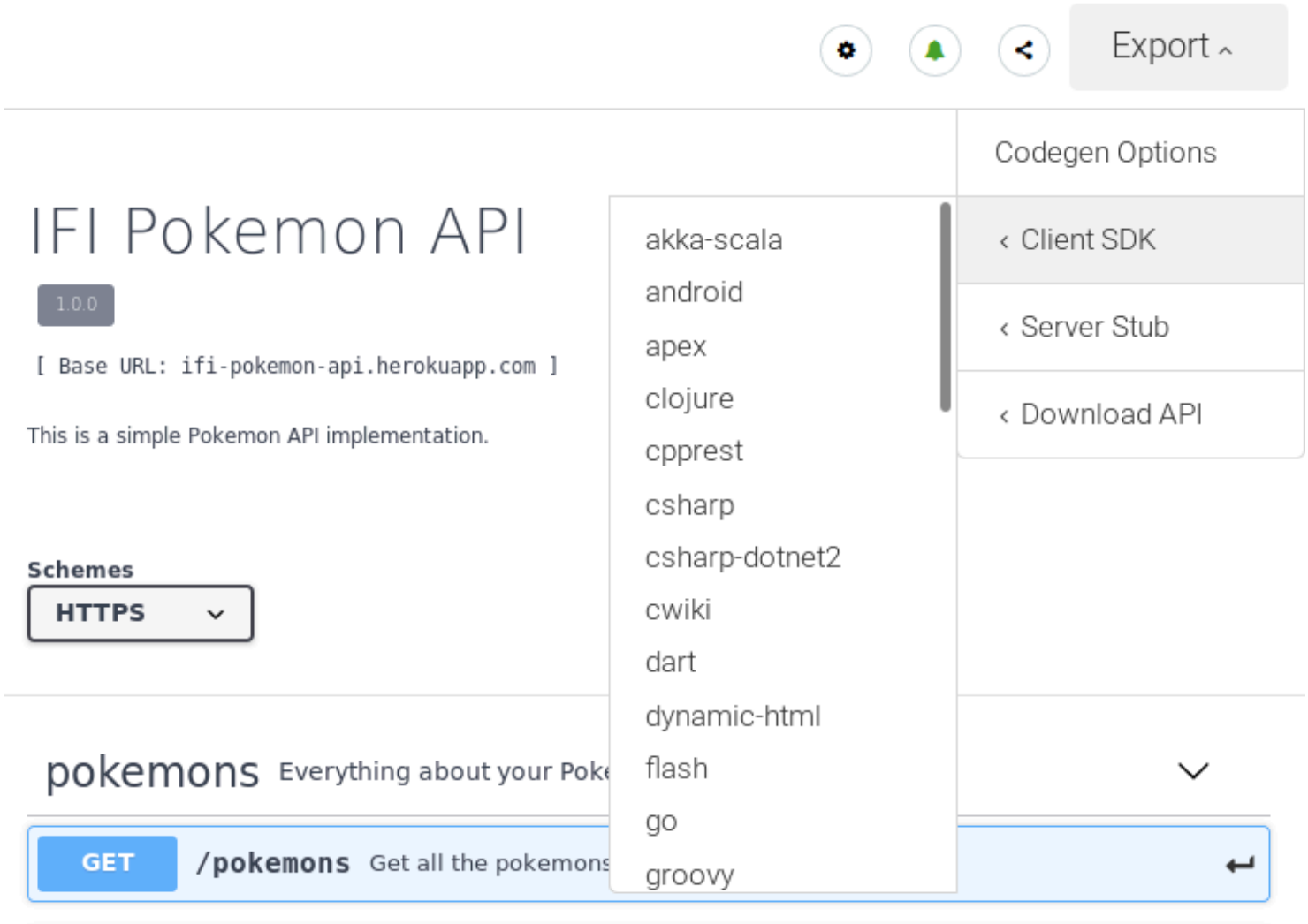
SWAGGER

Affichage de la documentation sous forme de page web



SWAGGER

Génération de squelettes clients/serveur



SWAGGER ET *OPENAPI* EN SPRING BOOT

Pas d'implémentation de la part de Spring

1 projet Open Source

springdoc-openapi (doc)

Exposition d'un service REST Spring

```java
/**
 * A convenience annotation that is itself annotated with
 * {@link Controller @Controller} and {@link ResponseBody @ResponseBody}.
 * <p>
 * Types that carry this annotation are treated as controllers where
 * {@link RequestMapping @RequestMapping} methods assume
 * {@link ResponseBody @ResponseBody} semantics by default.
  */
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Controller
@ResponseBody
public @interface RestController {

    /**
     * The value may indicate a suggestion for a logical component name,
     * to be turned into a Spring bean in case of an autodetected component.
     * @return the suggested component name, if any (or empty String otherwise)
     */
    @AliasFor(annotation = Controller.class)
    String value() default "";

}

```

EN SPRING

/api/pokemon-types/{id}

/api/pokemon-types?orderBy=name

/api/pokemon-types?type=poison

- `@RequestMapping` : écouter une URI
- `@PathVariable` : récupérer les variables d'URI entre '{}'
- `@RequestParam` : récupérer les paramètres de requête (query-strings '?a=b&c=d')
- `@RequestBody` : récupérer le corps de la requête

@REQUESTMAPPING

`/api/pokemon-types`

```java
@RestController
@RequestMapping("/api")
public class PokemonController {

    @GetMapping("/pokemon-types")
    public Iterable<Pokemon> getAllPokemons() {
        ...
    }
}
```

@PATHVARIABLE

`/api/pokemon-types/{id}`

```java
@RestController
@RequestMapping("/api")
public class PokemonController {

    @GetMapping("/pokemon-types/{id}")
    public Pokemon getPokemon(@PathVariable String id) {
        ...
    }
}
```

@REQUESTPARAM

`/api/pokemon-types?orderBy=name`

```java
@RestController
@RequestMapping("/api")
public class PokemonController {

    @GetMapping("/pokemon-types")
    public Iterable<Pokemon> getAllPokemons(@RequestParam String orderBy) {
        ...
    }
}
```

@REQUESTPARAM

`/api/pokemon-types?type=poison`

```java
@RestController
@RequestMapping("/api")
public class PokemonController {

    @GetMapping(path = "/pokemon-types", params = {"orderBy"})
    public Iterable<Pokemon> getAllPokemons(@RequestParam String orderBy) {
        ...
    }

    @GetMapping(path = "/pokemon-types", params = {"type"})
    public Iterable<Pokemon> getAllPokemons(@RequestParam String type) {
        ...
    }
}
```

@REQUESTBODY

`POST /api/trainers`

```java
@RestController
@RequestMapping("/api")
public class TrainerController {

    @PostMapping("/trainers")
    public Trainer createTrainer(@RequestBody Trainer trainer) {
        ...
    }
}
```

CONSOMMATION REST EN SPRING

`RestTemplate` (maintenance)

```
@Service
class PokemonTypeServiceImpl implements PokemonTypeService{

    private RestTemplate restTemplate;

    PokemonTypeServiceImpl(RestTemplate restTemplate) {
        this.restTemplate = restTemplate;
    }

    @Override
    public List<PokemonType> listPokemonsTypes() {
        var pokemonTypes = restTemplate
            .getForObject(pokemonServiceUrl+"/pokemon-types",
        return Arrays.asList(pokemonTypes);
    }
```

SPRING RESTTEMPLATE

Classe utilitaire pour effectuer des appels REST

- Exécute les requêtes HTTP : GET/POST/PUT/PATCH/OPTIONS/DELETE/HEAD
- Utilise `jackson-databind` pour convertir les objets Java en JSON !

HTTP INTERFACE

On définit une interface avec des méthodes annotées `@HttpExchange`.

Utilisation des annotations `@RequestParam`, `@PathVariable`, `@RequestBody`, `@RequestHeader` pour les paramètres.

On déclare en type de retour le type attendu, ou `ResponseEntity<T>`.

Spring génère un proxy dynamique qui implémente l'interface et exécute les appels HTTP (comme les Spring data repository).

HTTP INTERFACE

On déclare une interface

```
@HttpExchange("/pokemon-types")
public interface PokemonTypeApiRepository {

    @GetExchange
    List<PokemonType> getAllPokemons();

    @GetExchange("/{id}")
    PokemonType getPokemonFromId(@PathVariable int id);

}
```

On configure un client pour cette interface

```
// configuration à mettre dans une classe annotée @Configurati
@Bean
PokemonTypeApiRepository configurePokemonTypeApiRepository(@Va
    var restClient = RestClient.builder().baseUrl(pokemonTypeS
    var adapter = RestClientAdapter.create(restClient);
    var factory = HttpServiceProxyFactory.builderFor(adapter).
    return factory.createClient(PokemonTypeApiRepository.class
}
```

On reçoit le bean en injection de dépendance comme d'habitude.

WEBCLIENT

```xml
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-webflux</artifactId>
</dependency>
```

Client pour la programmation réactive & synchrone.

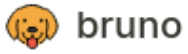
```java
// reactive
Flux<PokemonType> pokemonsFlux = client.get().uri("/pokemon-ty
    .retrieve()
    .bodyToFlux(PokemonType.class);

// synchrone
List<PokemonType> pokemonsList = pokemonsFlux
    .collectList()
    .block();
```

OUTILLAGE

💪 `$> curl`

ou

POSTMAN/INSOMNIA/BRUNO



LES AUTRES MOYENS DE COMMUNICATION RÉSEAU

- Reactive Streams - Support avec Spring WebFlux
- GraphQL - Support avec Spring GraphQL
- gRPC - pas d'implémentation officielle par Spring

TP

Interoperability