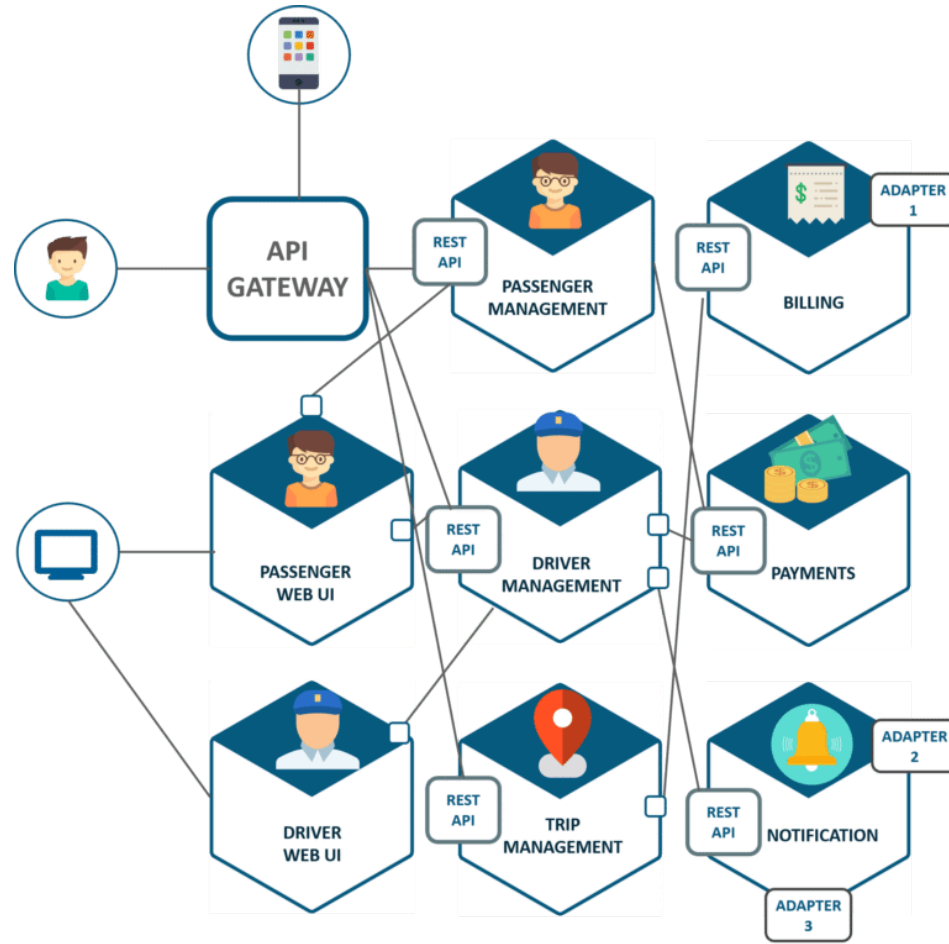


# ALOM



## INTEROPERABILITY



# 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



RESTful API

DELETE POST PUT GET



# 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



```
1 Accept: text/plain
2 Accept: application/xml
3 Accept: application/json
4 Accept: application/json,text/plain
5 Accept: image/png
```

carbon  
carbon.now.sh



# 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" }
  ]
}
```

carbon  
carbon.now.sh



# 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

The screenshot shows the SwaggerHub web interface for the IFI Pokemon API. At the top, there is a black header with the SwaggerHub logo (a green circle with three dots) and the text "SWAGGERhub SMARTBEAR". On the right side of the header, there are icons for chat, help, and a user profile labeled "Juwit".

Below the header, the main content area displays the API title "IFI Pokemon API" in a large, light blue font. Underneath the title, there is a small blue box containing the version number "1.0.0". Below this, the base URL is shown in brackets: "[ Base URL: ifi-pokemon-api.herokuapp.com ]". A short description follows: "This is a simple Pokemon API implementation."

Below the description, there is a "Schemes" section with a dropdown menu currently set to "HTTPS".

The main section of the interface is titled "pokemons" with the subtitle "Everything about your Pokemons". It contains two API endpoints listed in light blue boxes:

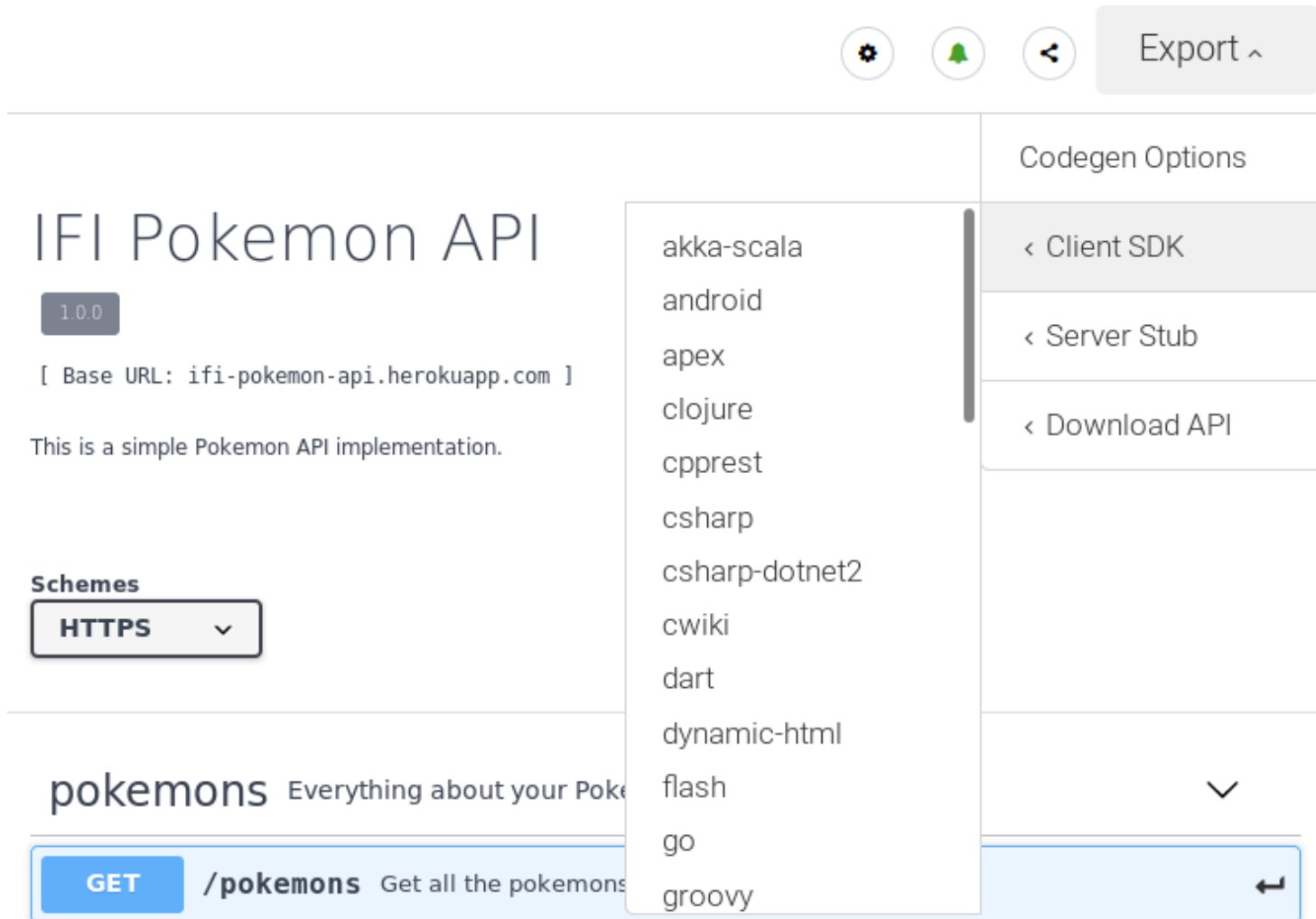
- GET /pokemons** Get all the pokemons
- GET /pokemons/{pokemonId}** Get a pokemon by ID

Below the endpoints, there is a "Models" section with a dropdown menu. Under the "Models" section, there is a single model listed: "Pokemon" with a right-pointing arrow next to it.

At the bottom right of the page, there is a promotional banner that says "Ready to Design a New API?" next to a green icon with curly braces. To the right of this banner is the "MAGE Lille" logo, which features the word "MAGE" in orange and "Lille" in white on a blue background.

# SWAGGER

## Génération de squelettes clients/serveur



The image shows the Swagger UI for the IFI Pokemon API. At the top right, there are icons for settings, notifications, and a share button, followed by an "Export" button with a dropdown arrow. The main content area on the left displays the API title "IFI Pokemon API", version "1.0.0", and the base URL "[ Base URL: ifi-pokemon-api.herokuapp.com ]". Below this, it states "This is a simple Pokemon API implementation." and a "Schemes" dropdown menu is set to "HTTPS". On the right, the "Codegen Options" panel is open, showing a list of languages: akka-scala, android, apex, clojure, cpprest, csharp, csharp-dotnet2, cwiki, dart, dynamic-html, flash, go, and groovy. The "Client SDK" option is selected and highlighted. At the bottom, a search bar contains the text "pokemons" and a "GET" button is visible next to the endpoint "/pokemons".

IFI Pokemon API

1.0.0

[ Base URL: ifi-pokemon-api.herokuapp.com ]

This is a simple Pokemon API implementation.

Schemes

HTTPS

Codegen Options

- < Client SDK
- < Server Stub
- < Download API

akka-scala

android

apex

clojure

cpprest

csharp

csharp-dotnet2

cwiki

dart

dynamic-html

flash

go

groovy

GET /pokemons

Get all the pokemons

# ***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

```
1 /**
2  * A convenience annotation that is itself annotated with
3  * {@link Controller @Controller} and {@link.ResponseBody @ResponseBody}.
4  * <p>
5  * Types that carry this annotation are treated as controllers where
6  * {@link.RequestMapping @RequestMapping} methods assume
7  * {@link.ResponseBody @ResponseBody} semantics by default.
8  */
9 @Target(ElementType.TYPE)
10 @Retention(RetentionPolicy.RUNTIME)
11 @Documented
12 @Controller
13 @ResponseBody
14 public @interface RestController {
15
16     /**
17      * The value may indicate a suggestion for a logical component name,
18      * to be turned into a Spring bean in case of an autodetected component.
19      * @return the suggested component name, if any (or empty String otherwise)
20      */
21     @AliasFor(annotation = Controller.class)
22     String value() default "";
23
24 }
25
```

# 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



```
1 @RestController
2 @RequestMapping("/api")
3 public class PokemonController {
4
5     @GetMapping("/pokemon-types")
6     public Iterable<Pokemon> getAllPokemons() {
7         ...
8     }
9 }
```

# @PATHVARIABLE

/api/pokemon-types/{id}



```
1 @RestController
2 @RequestMapping("/api")
3 public class PokemonController {
4
5     @GetMapping("/pokemon-types/{id}")
6     public Pokemon getPokemon(@PathVariable String id) {
7         ...
8     }
9 }
```

carbon  
carbon.now.sh



# @RequestParam

/api/pokemon-types?orderBy=name



```
1 @RestController
2 @RequestMapping("/api")
3 public class PokemonController {
4
5     @GetMapping("/pokemon-types")
6     public Iterable<Pokemon> getAllPokemons(@RequestParam String orderBy) {
7         ...
8     }
9 }
```

carbon  
carbon.now.sh



# @RequestParam

/api/pokemon-types?type=poison

```
1 @RestController
2 @RequestMapping("/api")
3 public class PokemonController {
4
5     @GetMapping(path = "/pokemon-types", params = {"orderBy"})
6     public Iterable<Pokemon> getAllPokemons(@RequestParam String orderBy) {
7         ...
8     }
9
10    @GetMapping(path = "/pokemon-types", params = {"type"})
11    public Iterable<Pokemon> getAllPokemons(@RequestParam String type) {
12        ...
13    }
14 }
```

# @REQUESTBODY

POST /api/trainers



```
1 @RestController
2 @RequestMapping("/api")
3 public class TrainerController {
4
5     @PostMapping("/trainers")
6     public Trainer createTrainer(@RequestBody Trainer trainer) {
7         ...
8     }
9 }
```

carbon  
carbon.now.sh

# 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 @Configuration
@Bean
PokemonTypeApiRepository configurePokemonTypeApiRepository(@Value("${pokemon.type.url}") String url) {
    var restClient = RestClient.builder().baseUrl(url).build();
    var adapter = RestClientAdapter.create(restClient);
    var factory = HttpServiceProxyFactory.builderFor(adapter).build();
    return factory.createClient(PokemonTypeApiRepository.class);
}
```

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



# WEBCLIENT

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

Client pour la programmation réactive & synchrone.

```
// reactive  
Flux<PokemonType> pokemonsFlux = client.get().uri("/pokemon-type")  
    .retrieve()  
    .bodyToFlux(PokemonType.class);  
  
// synchrone  
List<PokemonType> pokemonsList = pokemonsFlux  
    .collectList()  
    .block();
```

# OUTILLAGE



```
$> curl
```

ou

## POSTMAN/INSOMNIA/BRUNO



A small yellow dog head emoji. **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