GROUPE ISI

Les webservices

MIAGE2

Nov. 2021

M. LY

PARTIE 1

Plan du cours (1/2)

- Théorie des Services web
  - Pourquoi les web services ?
  - Qu'est-ce qu'un service web ?
  - Origines
  - Usages
  - Rappel de quelques notions de base
    - Serveur / client
    - Protocole
    - HTTP
  - Types de services web
    - REST
    - SOAP

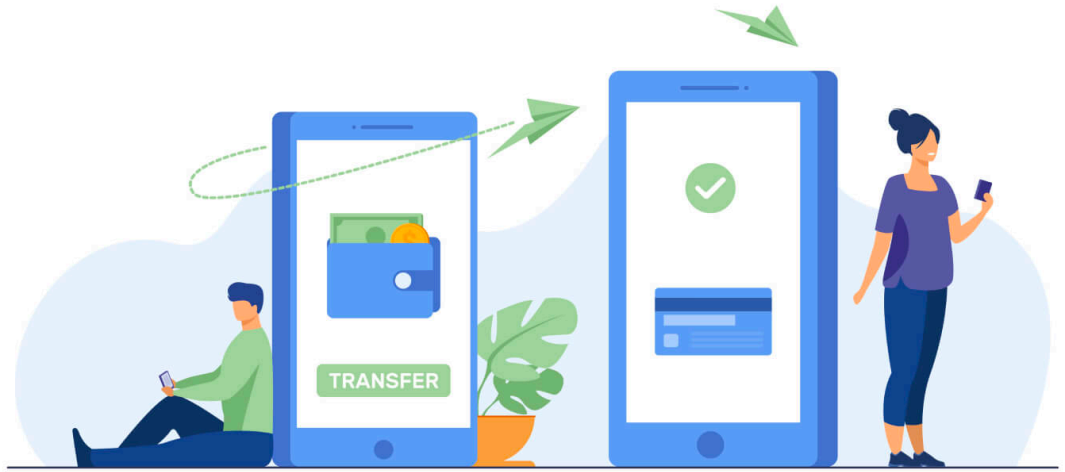
Plan du cours (2/2)

- Création de Services web avec Springboot
  - Configurations de base
  - Services Web REST
  - Services Web SOAP
  - Travaux pratiques
- GraphQL
- Java API for XML Web Services (JAX-WS)
- Java API for Restful Web Services (JAX-RS)

Théorie des services web

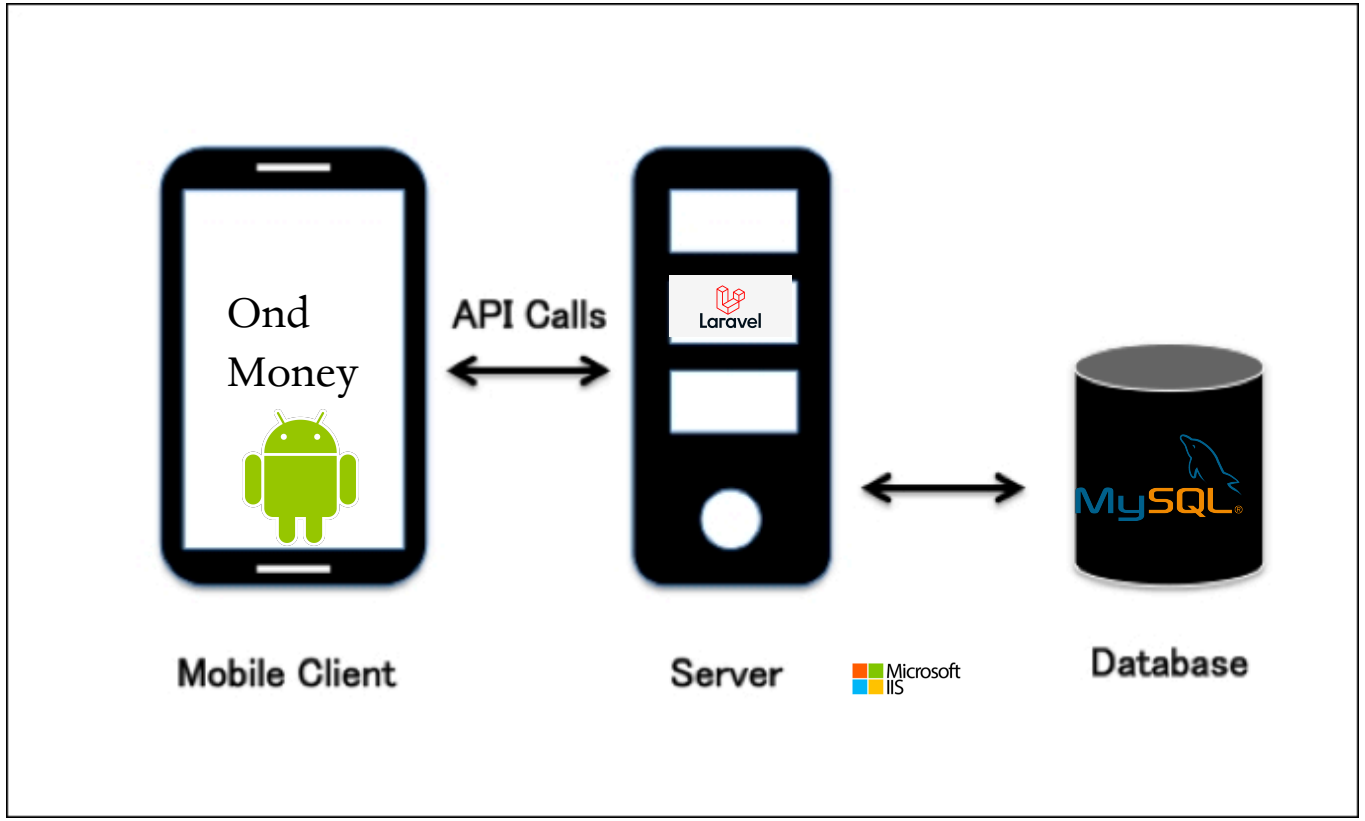
Pourquoi les web services ?

- **Etude de cas**:
  - Cahier de charge **V1** : Réaliser une application mobile ***Ond Money*** qui permet de faire du transfert d'argent entre utilisateurs.
    - Analyse
    - Conception
    - Développement
    - Déploiement

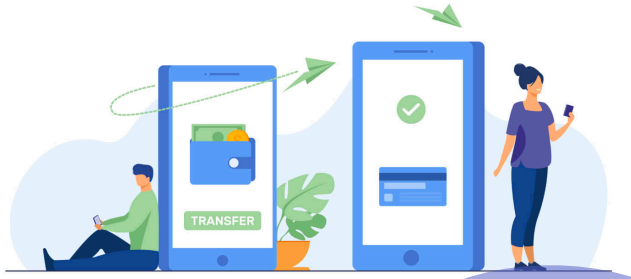


Pourquoi les web services ?

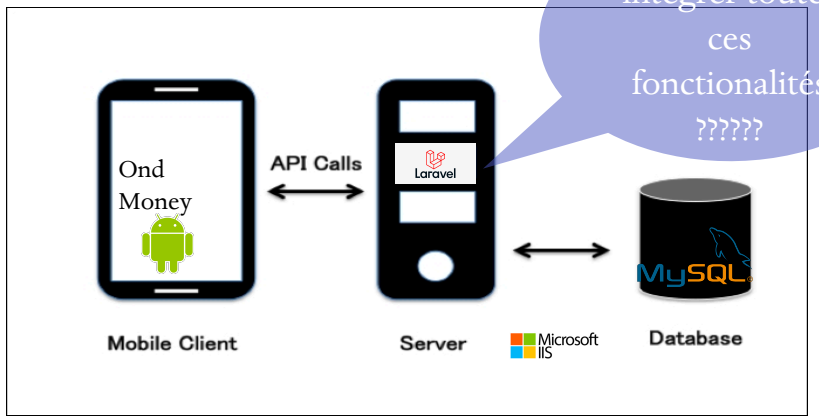
- Etude de cas
  - Architecture
    - Mobile App
    - Backend
    - Database

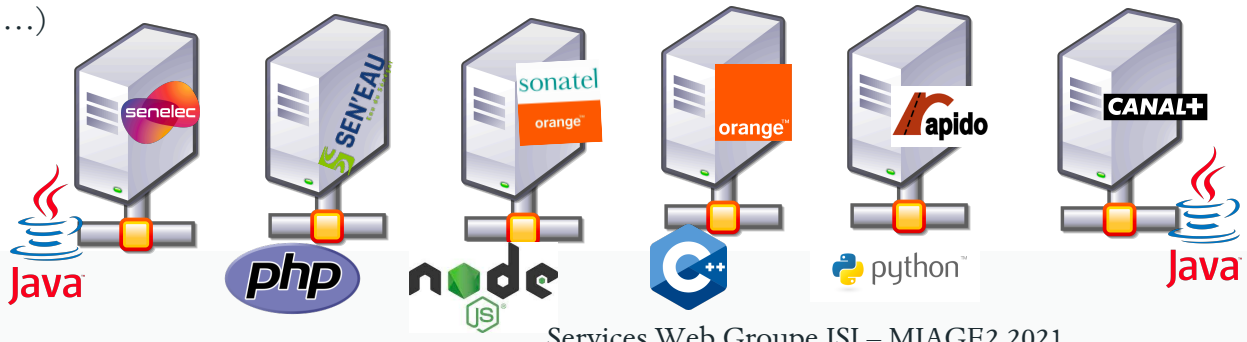


Pourquoi les web services ?



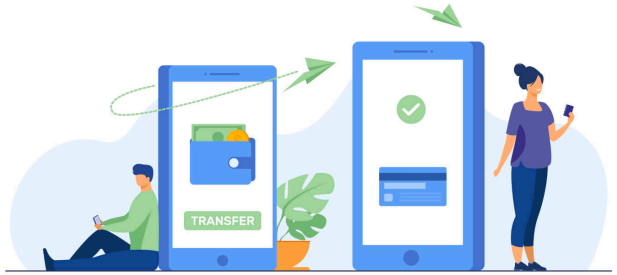
- Etude de cas
  - Plus tard
    - Succes de l'application
    - Plusieurs millions d'utilisateurs
    - Clients de plus en plus nombreux
    - Clients de plus en plus exigents
      - De nouvelles fonctionnalités réclamées
        - Paiement de factures (Senelec, Sen'eau, Sonatel …)
        - Achat de crédit
        - Paiement de services (Rapido, Canal+ …)



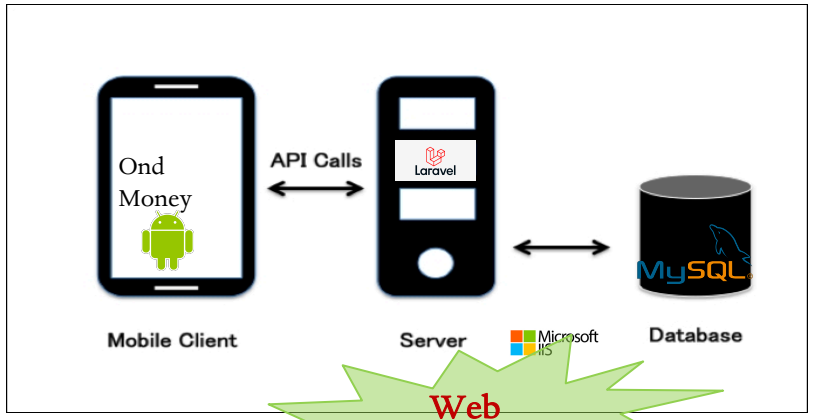


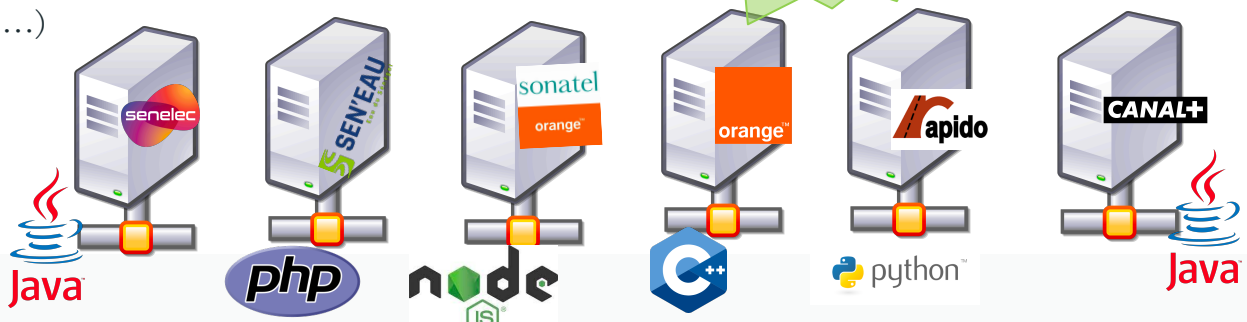
Red Hat

Pourquoi les web services ?



- Etude de cas
  - Plus tard
    - Succes de l'application
    - Plusieurs millions d'utilisateurs
    - Clients de plus en plus nombreux
    - Clients de plus en plus exigents
      - De nouvelles fonctionnalités réclamées
        - Paiement de factures (Senelec, Sen'eau, Sonatel …)
        - Achat de crédit
        - Paiement de services (Rapido, Canal+ …)





Red Hat

Pourquoi les web services ?

- Cycle de développement d'un projet informatique
- Qualité / Coût / Délai
- Performance
- Maintenabilité
- Evolutivité
- Scalabilité

Qu'est ce qu'un service web ?

- Composants web
- Basés sur internet HTTP
- Qui effectuent des tâches précises
- Selon un format bien déterminé

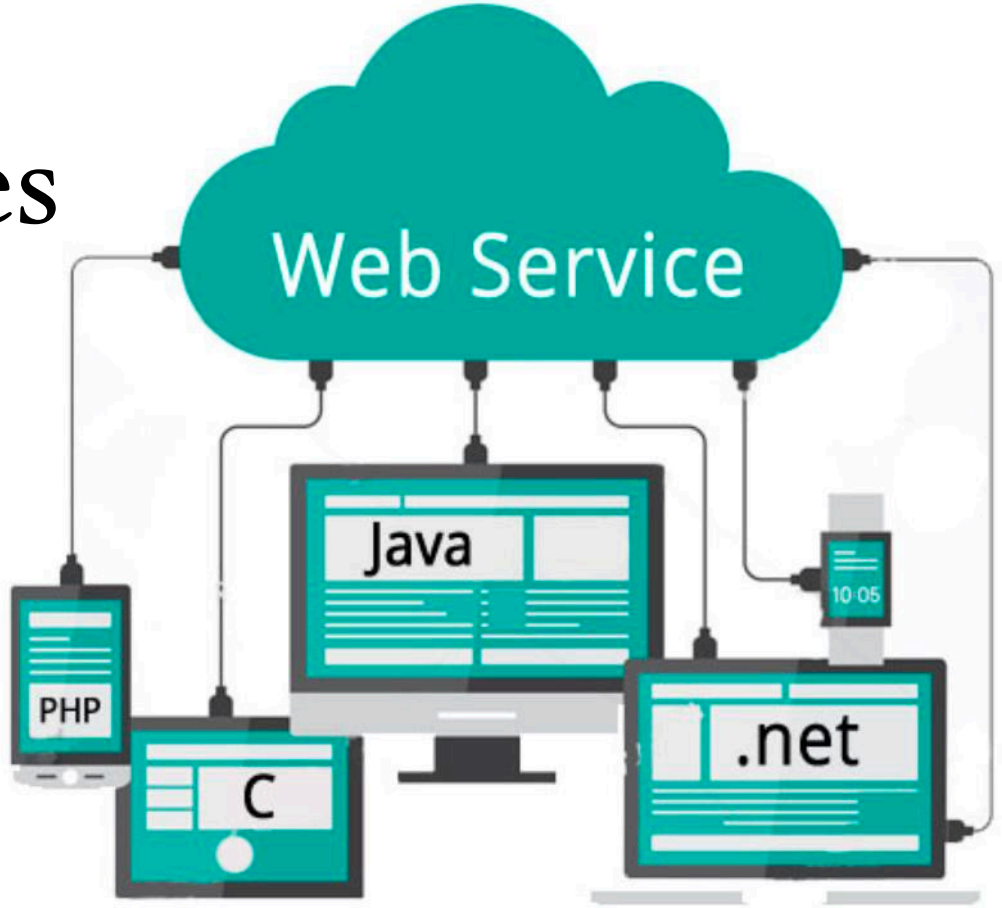
- Moyen de communications entre applications via le réseau
  - Indépendamment de la plateforme
  - Indépendamment du langage

Origine des webservices

- Besoin d'échanges de données automatisés entre organisations
- Mettre fin à l'isolement des systèmes informatiques
- En 1975, l'EDI (Echanges de Données Informatisées) est créé.
  - Format standard d'échanges d'un certain type de données, d'ordinateur à ordinateur
  - Gain de temps et d'argent
  - Incomplète
  - Mise en œuvre complexe et couteuse
- Création de technologies dans les années 80
  - CORBA
  - COM

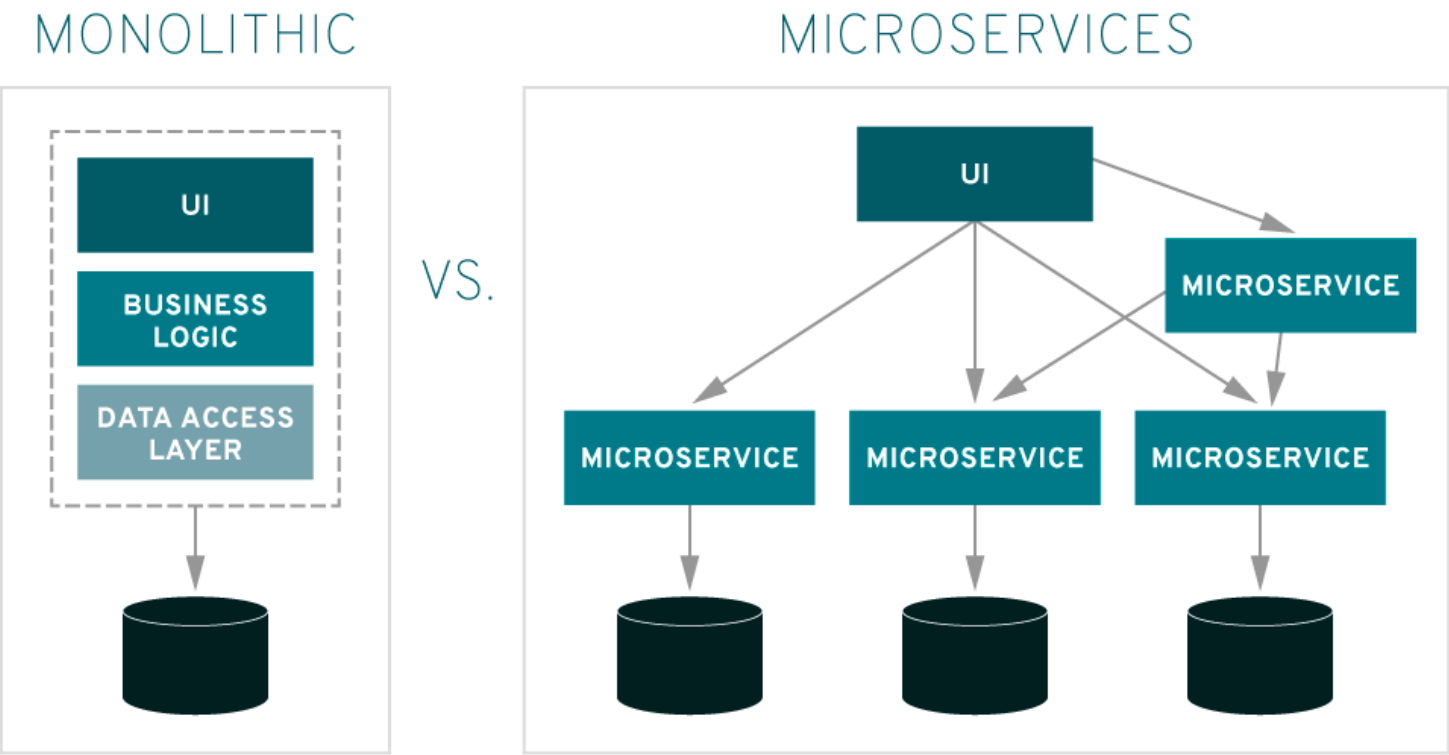
Usages des web services

- Moteur de recherche
- Boutique en ligne
- Météo
- Conversion de devises
- Etc…



Usages des web services

- Architectures microservices



Usage des web services

- TP: mise en place d'une architecture microservice
  - JHipster

    Générateur d'application Java avec le framework Springboot comme backend et Angular / React ou Vue.js comme front
  - Docker

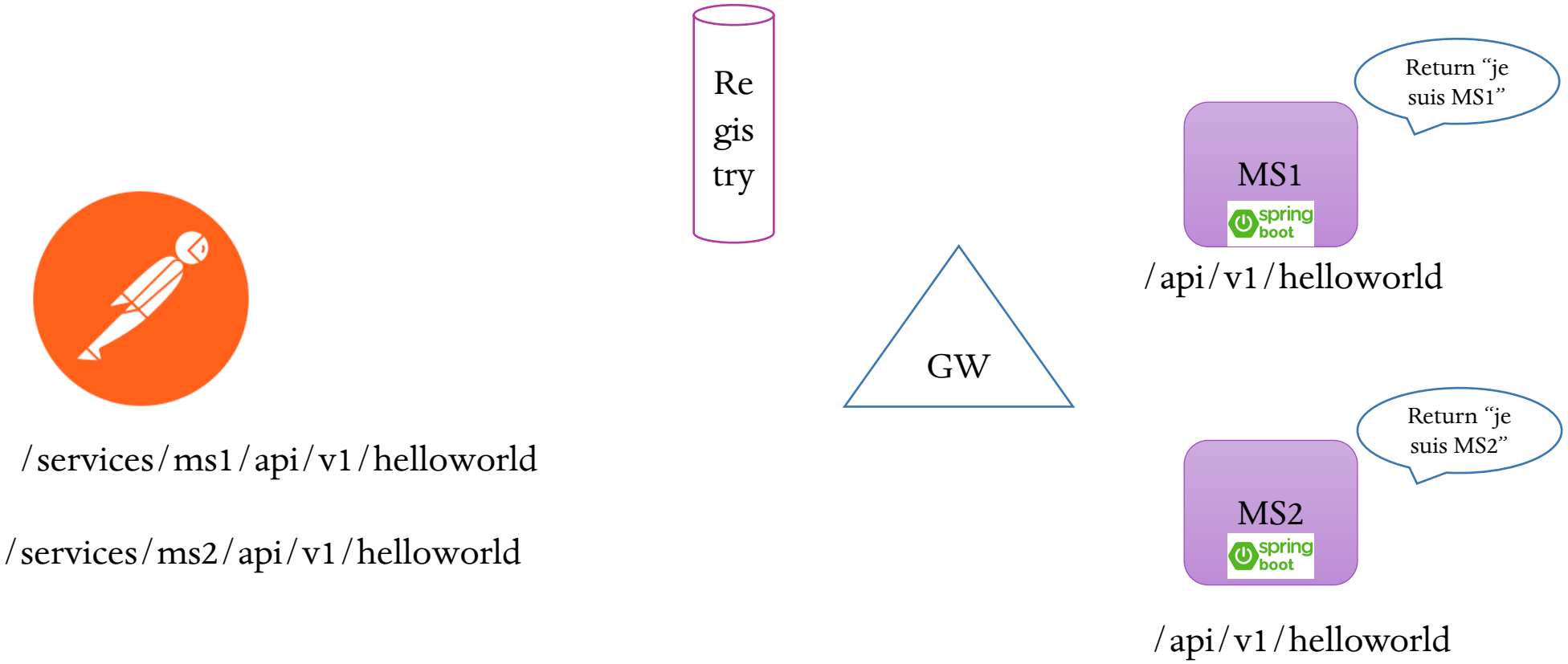
    Technologie de containerisation qui permet la création et l'utilisation de container Linux.
  - Registry

    Service, dans une architecture microservice permettant de faire du "discovery" ou decouverte de services
  - Gateway

    Point d'entrée unique pour les APIs et microservices backend.
  - Microservice

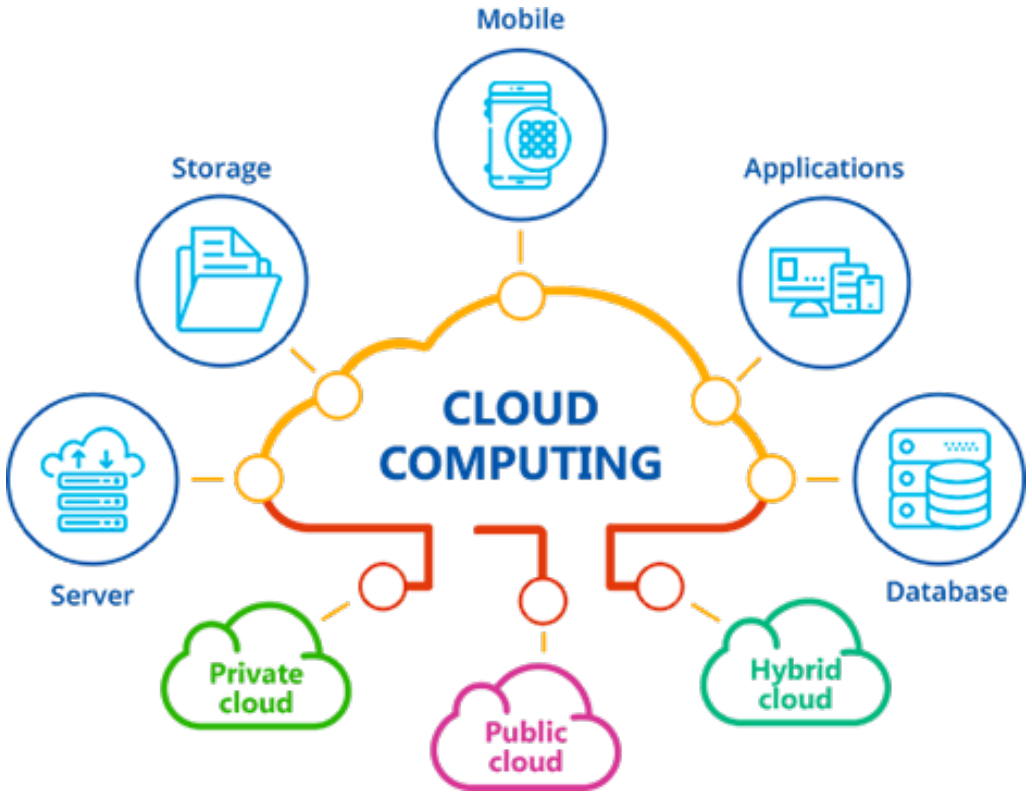
    Service autonome et indépendant permettant de faire un traitement

Architecture microservices



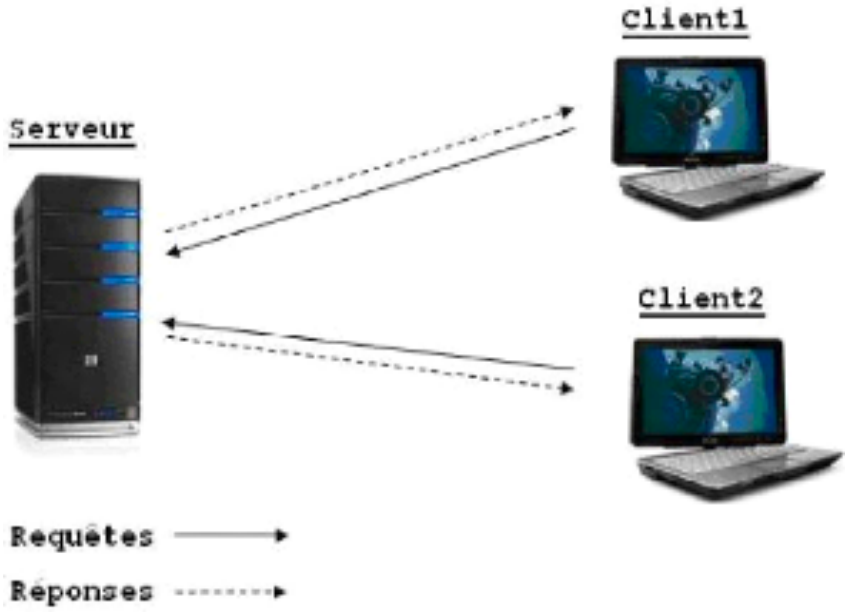
Usage des web services

- Le Cloud computing



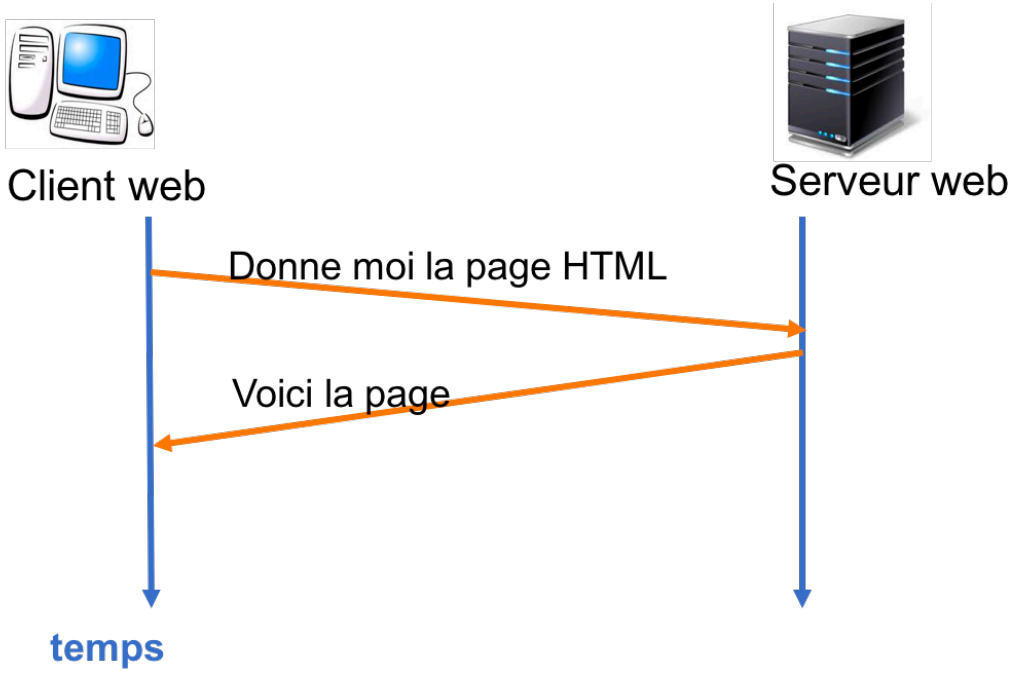
Rappel de quelques notions de base

- Client / Serveur
  - Mode de communication entre 2 ou plusieurs entités. Celle qui envoie les demandes/requêtes représente le client. Celle qui traite les demandes le serveur.



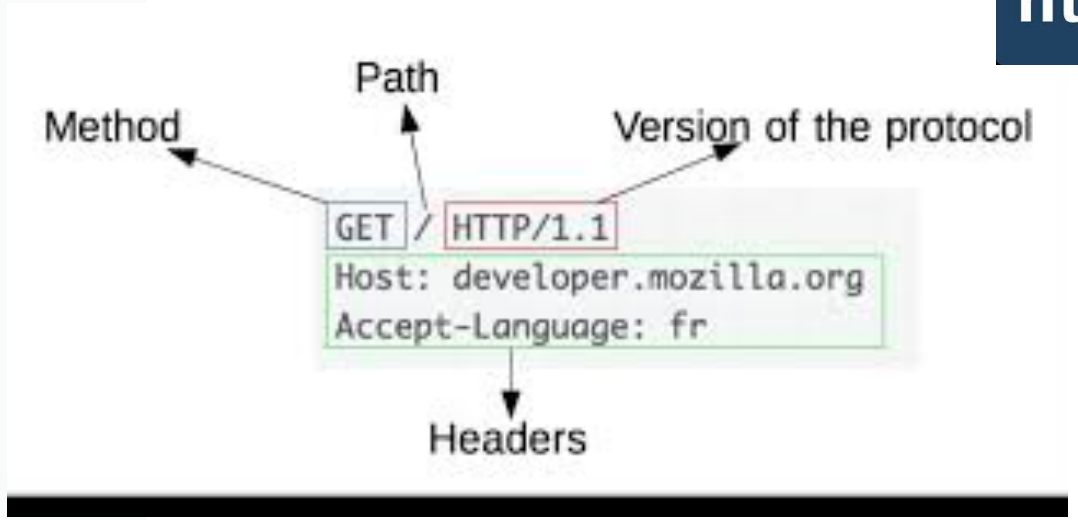
Rappel de quelques notions de base

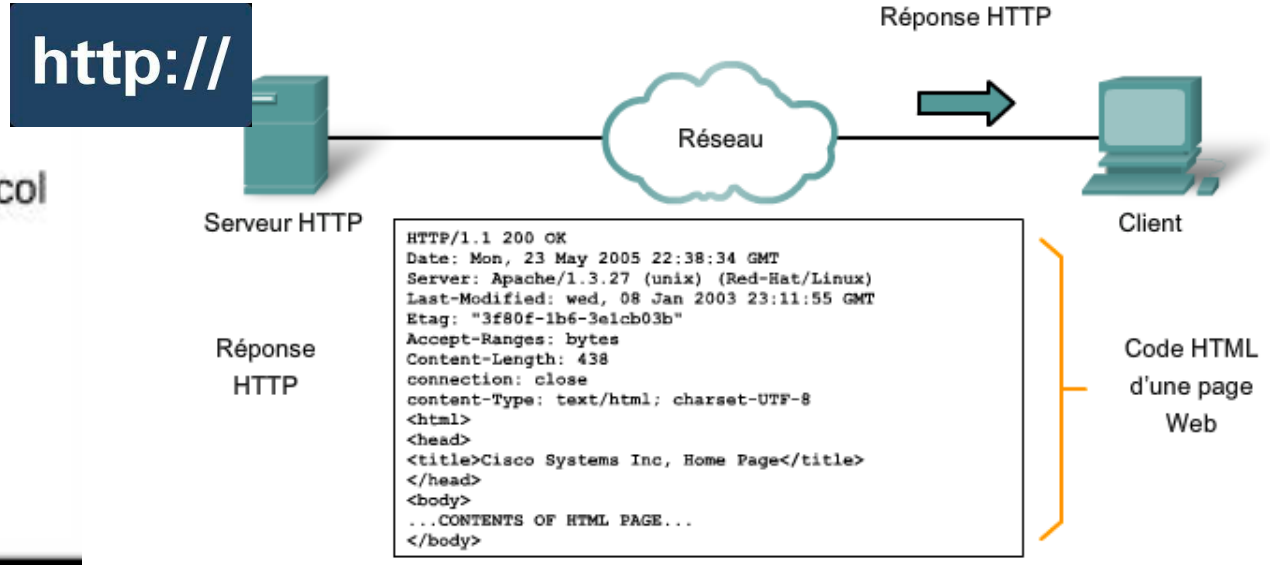
- Protocole
  - Ensemble de règles qui régissent les échanges de données entre 2 ou plusieurs entités.



Rappel de quelques notions de base

- HTTP
  - Protocole de communication client-serveur développé pour le world wide web.

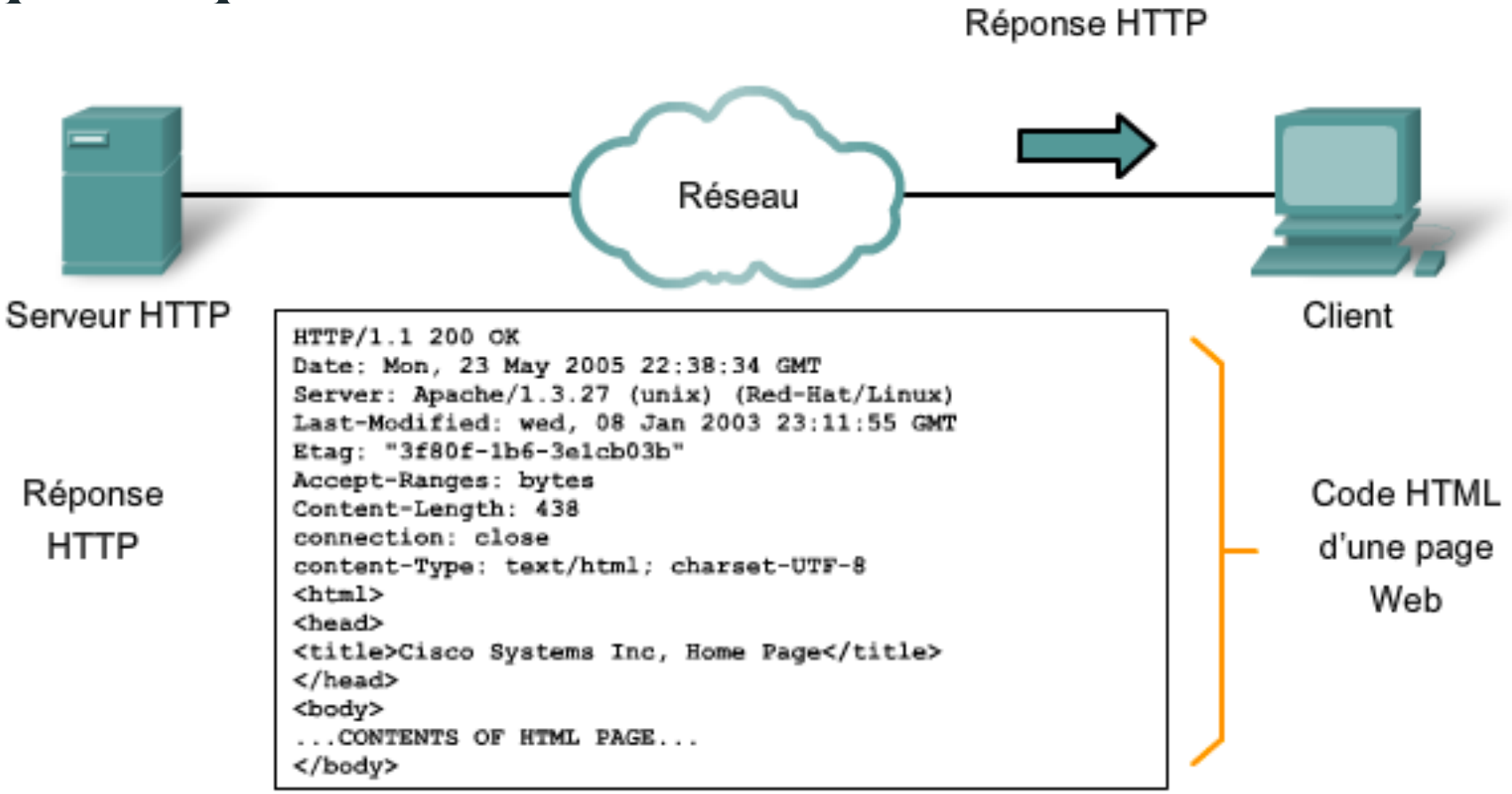




En réponse à la requête, le serveur HTTP retourne le code d'une page Web.

Rappel de quelques notions de base

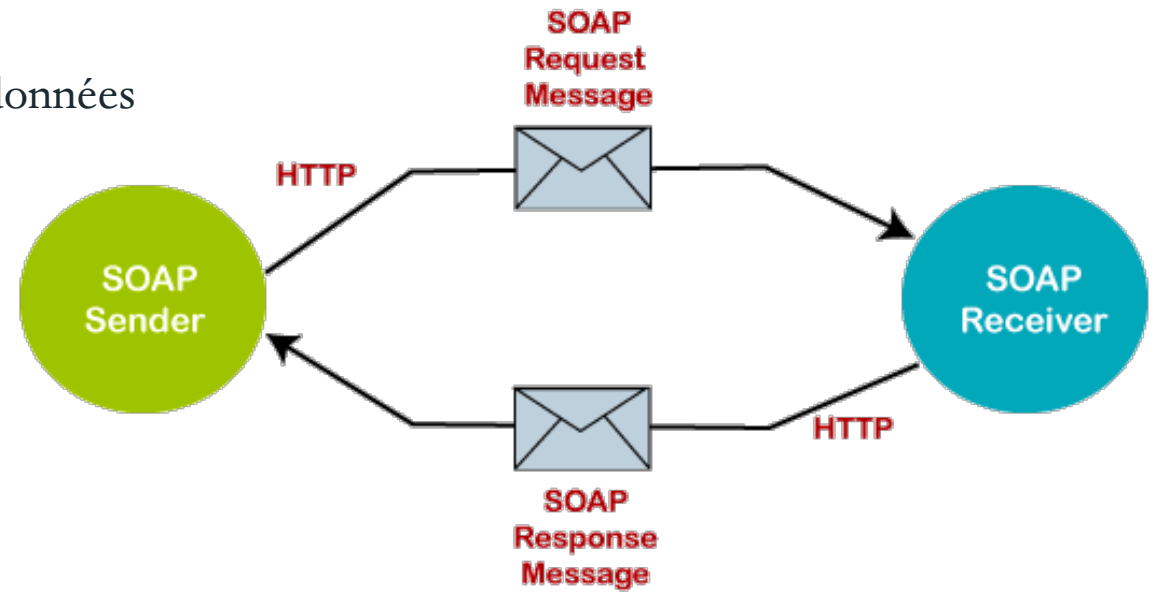
http://



En réponse à la requête, le serveur HTTP retourne le code d'une page Web.

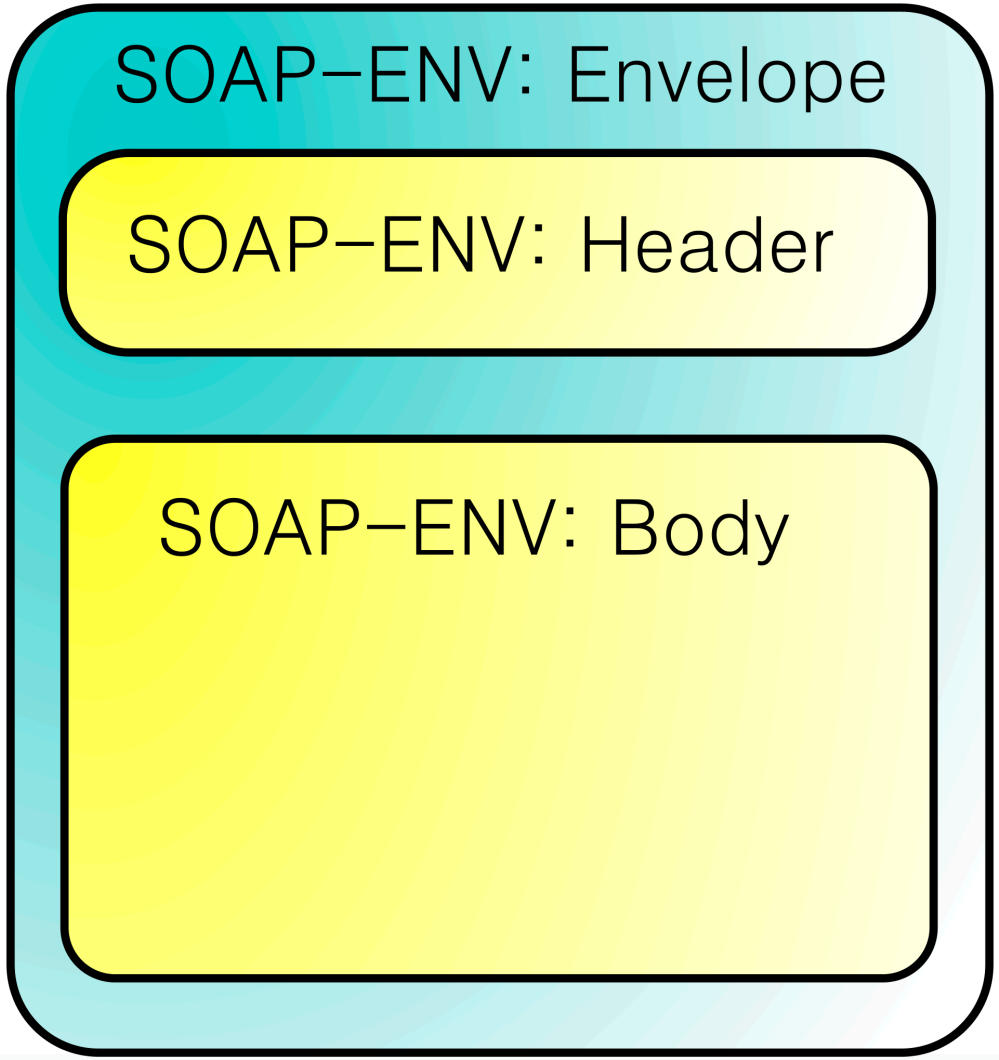
Types de services web

- **SOAP**
  - Anciennement « Simple Object Access Protocol ». Cependant cet acronyme ne convient plus depuis la version 1.2
  - Protocole permettant l'échange de données structurées en XML.

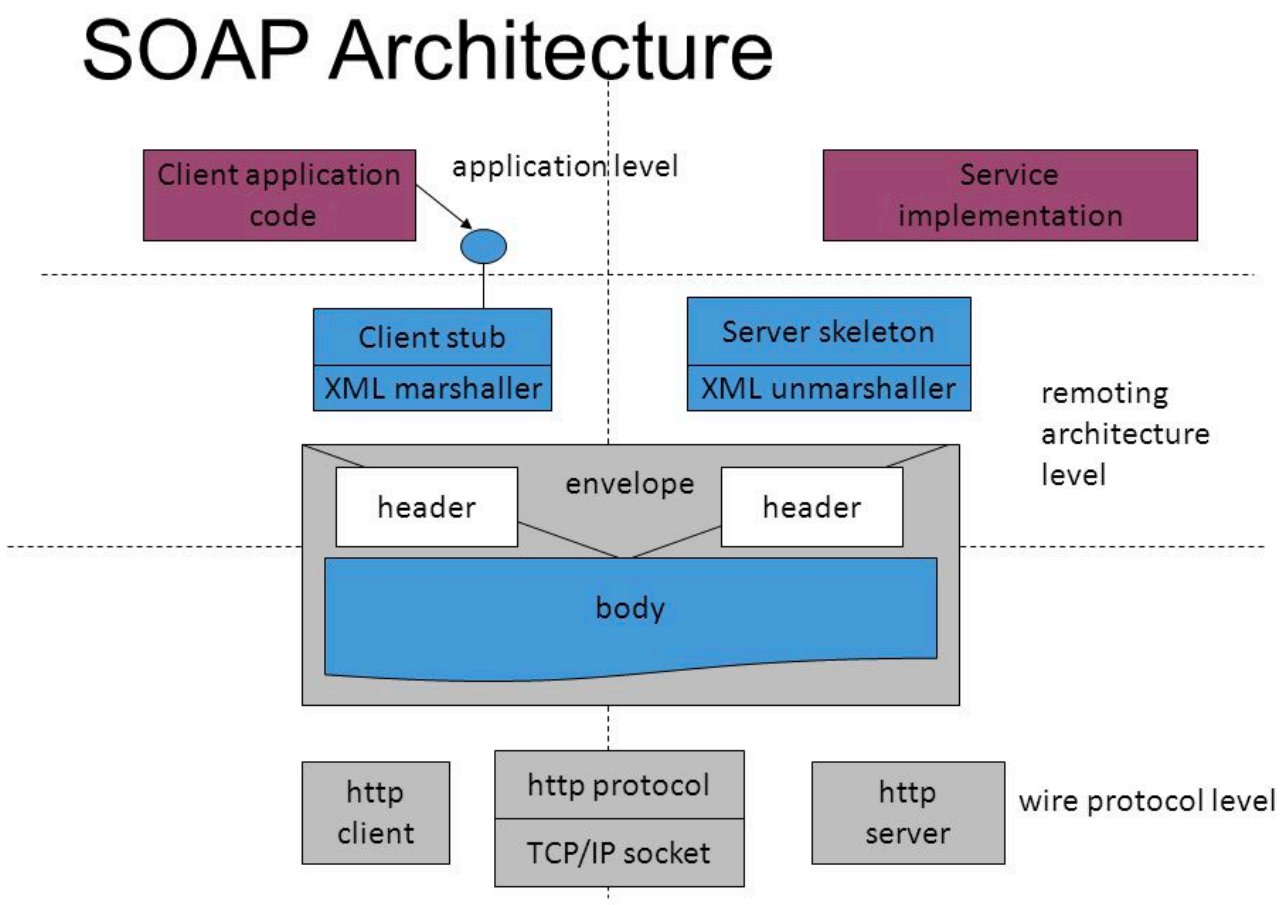


Types de services web

- SOAP

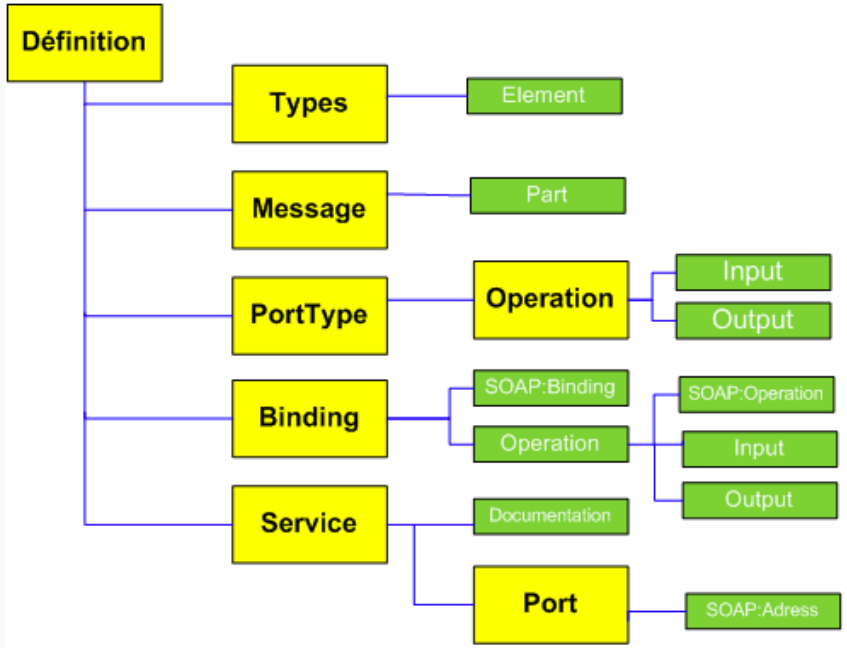


Types de services web

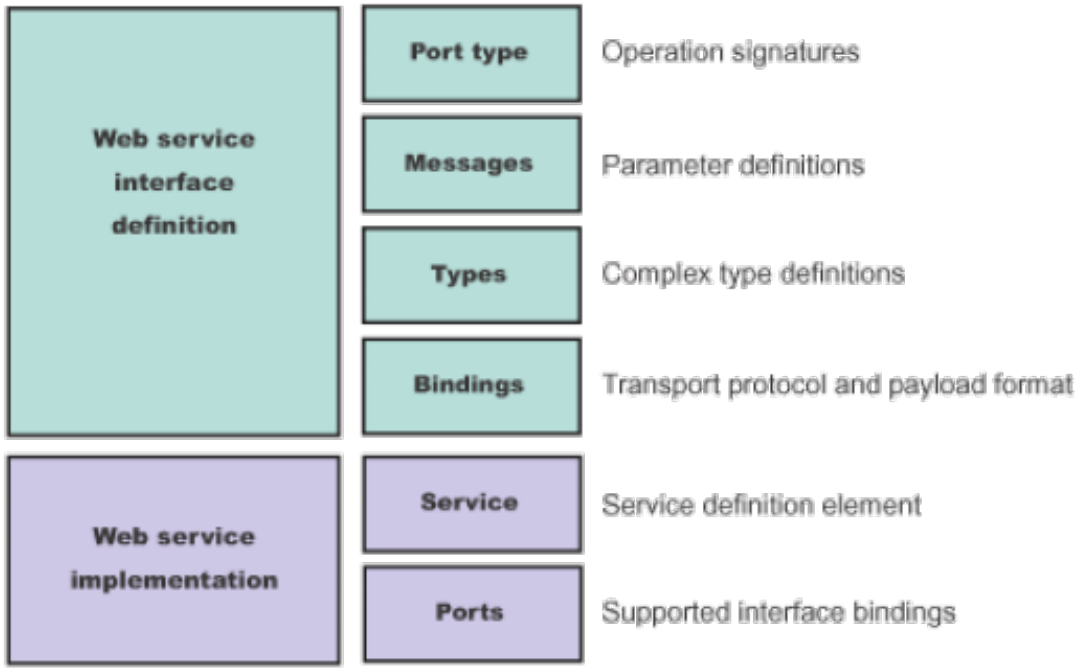


Types de services web

Le WSDL

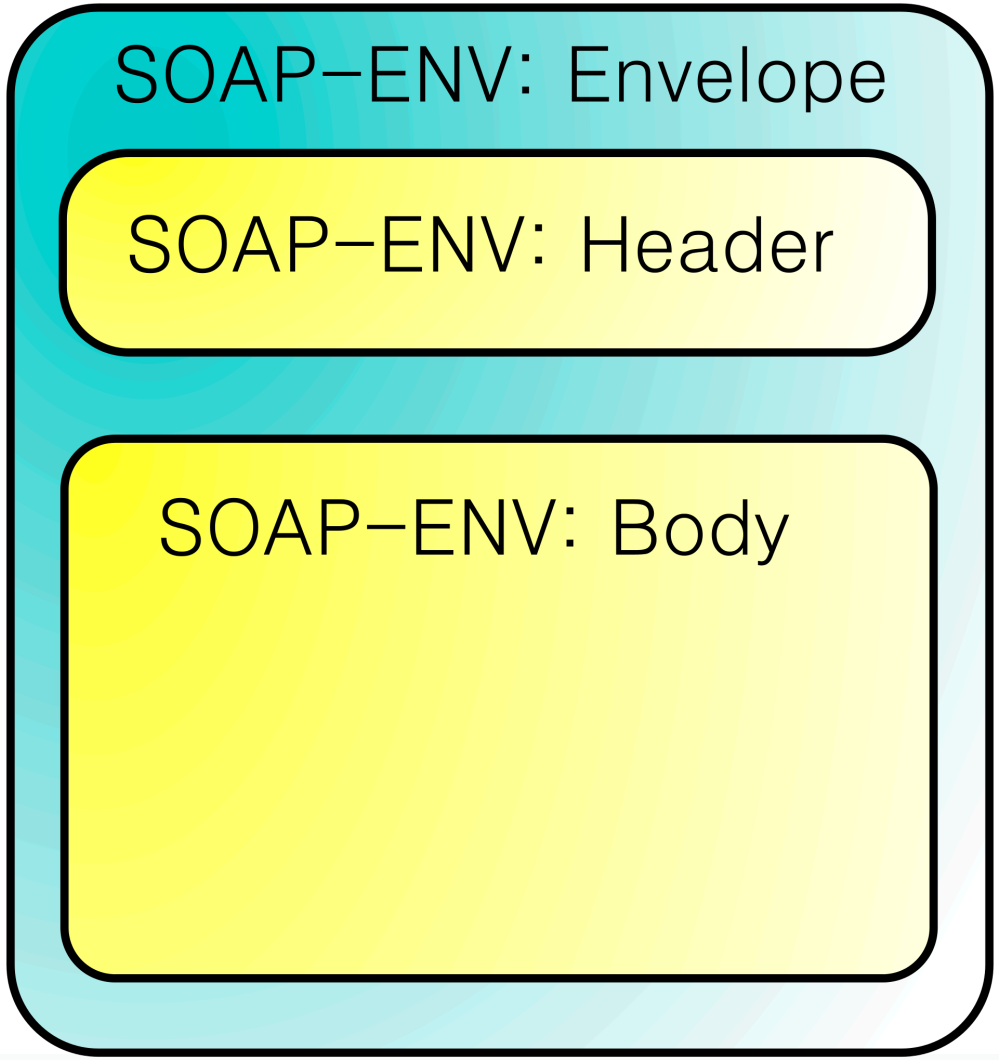


Structure d'un document WSDL



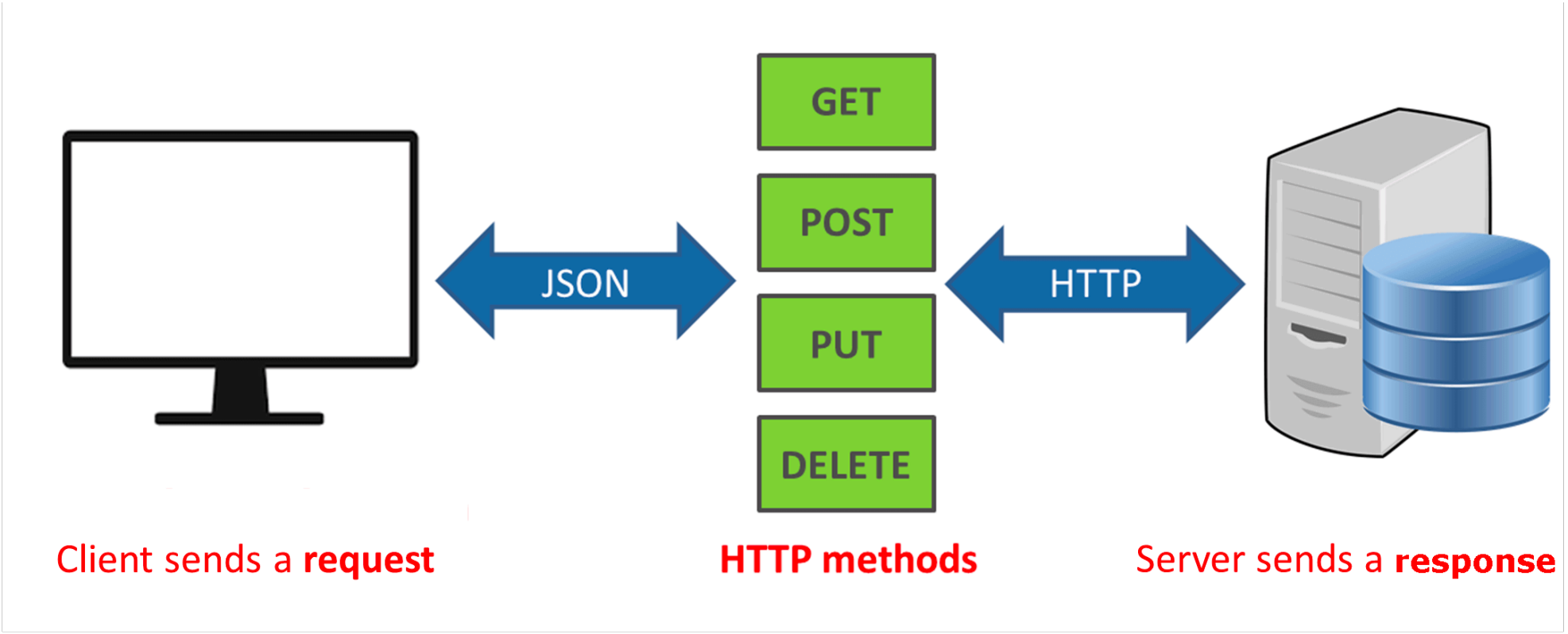
Types de services web

- **SOAP**
  - Structure d'un message SOAP
    - **Enveloppe**: contient le message et ses différents sous-bloc
    - **Header**: bloc optionnel contenant les informations d'entête du message. Il vient toujours avant le bloc body et se trouve dans Enveloppe.
    - **Body**: contient le corps du message. Il est unique dans le message et doit être dans le bloc Enveloppe
    - **Fault**: il est le seul bloc dans Body et sert à reporter les erreur lors du traitement.



Types de services web

- RESTful

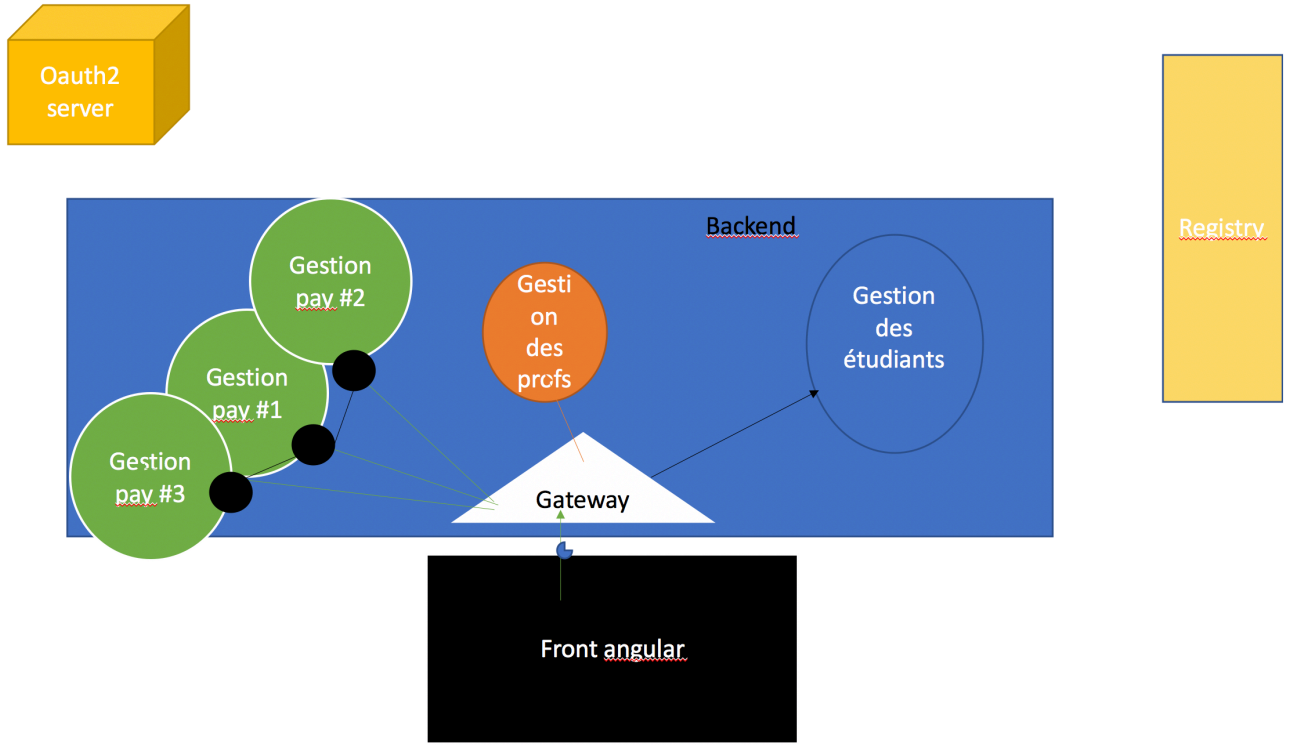


Types de services web

- **RESTful**
  - Basé sur l'architecture REST (Representational State Transfer)
  - Les informations cotés serveur sont considérées comme des ressources accessible de façon uniforme à travers des URIs (Uniform Resource Identifier)
  - Les objets référencées par une URI peuvent avoir différentes représentation ou format (JSON, XML, TXT …)
  - Requêtes basées sur le protocole HTTP comme protocole d'échange
  - Les services sont STATELESS
  - Les services Restful sont légers et représentent une alternative au SOAP
  - Utilise les méthodes(verbes) du protocole HTTP pour designer les opérations à effectuer

Création de web services avec Springboot

- Configuration de base
  - Travaux pratiques:
    - JHipster
    - Docker
    - Keycloak (JWT, OAuth2)
  - Achitecture Microservices
    - Présentation
    - Microservices
    - Gateway
    - Registry



Création de web services RESTFul avec Springboot

- Ajout dépendance SPRING MVC contenu dans le starter spring-boot-starter-mvc
  - <**dependency**>
    <**groupId**>org.springframework.boot</**groupId**>
    <**artifactId**>spring-boot-starter-web</**artifactId**>
    </**dependency**>
- Création d'un controller web simple :

  /src/main/java/sn/isi/mg2l/web/rest/EtudiantResource.java

Création de web services avec Springboot

```
package sn.isi.mg2l.web.rest;
import sn.isi.mg2l.service.EtudiantService;
import
sn.isi.mg2l.web.rest.errors.BadRequestAlertException;
import sn.isi.mg2l.web.rest.util.HeaderUtil;
import sn.isi.mg2l.service.dto.EtudiantDTO;
…
/**
 * REST controller for managing Etudiant.
 */
@RestController
@RequestMapping("/api")
public class EtudiantResource {


   @GetMapping("/etudiants")
   public List<EtudiantDTO> getAllEtudiants() {
      log.debug("REST request to get all Etudiants");
      return etudiantService.findAll();
   }

}
```

- @RestController: annotation pour définir la classe EtudiantResource comme un composant de Spring MVC capable de traiter des requêtes.
- @RequestMapping("/api") définit la route (/api) que gère ce controller.
- @GetMapping("/etudiants") : effectue un mapping de la route /api/etudiants à la méthode getAllEtudiants(). En d'autres termes les requêtes http avec la méthode GET dont l'URI pointe vers /api/etudiant sont traités par la méthode getAllEtudians()

Création de web services RESTFul avec Springboot

- Spring MVC offre à coté des annotations vues précédemment:
  - **@RequestParam** : Mapping d'un paramètre de la requête à une variable
    - Exemple

```
@GetMapping("/api/etudiants")
 @ResponseBody
 public String getFoos(@RequestParam String id)
{ return "ID: " + id; }
```

La requête: http://localhost:8080/api/etudiant?id=135

Création de web services RESTFul avec Springboot

- **@PathVariable**: pour recupérer une variable contenue dans le chemin de la ressource
  - Exemple:

```
@DeleteMapping("/etudiants/{id}")
public ResponseEntity<Void> deleteEtudiant(@PathVariable Long id) {

    etudiantService.delete(id);
    return ResponseEntity.ok().build();
}
```

Création de web services avec Springboot

- **@RequestBody**

  Réalise un mapping du contenu du HTTPRequest body à un Objet ou un DTO pour une déserialization automatique.

  Exemple:

```
@PutMapping("/etudiants")
public ResponseEntity<EtudiantDTO> updateEtudiant(@RequestBody EtudiantDTO etudiantDTO) throws
URISyntaxException {

    return ResponseEntity.ok().body(result);
}
```

Création de web services RESTFul avec Springboot

- **@ResponseBody**

Indique au controller que l'objet retournée par la méthode est automatiquement sérializée en JSON et passé au HTTPResponse (réponse de la requête).

NB: cette annotation devient non-nécessaire lorsque le controller est annoté @RestController car dans ce cas Spring le fait automatiquement.

Création de web services RESTful avec Springboot

Exemple: A supposer la classe ResponseEtudiant avec l'implémentation suivante

```
public class ResponseEtudiant { private String message; // getters & setters }
```

Au niveau du controller:

```
@Controller
@RequestMapping("/api") public class EtudiantController {

    @PostMapping("/etudiant")
    @ResponseBody
    public ResponseEtudiant postResponseController( @RequestBody Etudiant etudiant) {
            return new ResponseEtudiant("Bonjour !!!");
            }
    }
```

La réponse à la requete sera:

```
{"message":"Bonjour !!!"}
```

Création de web services RESTful avec NodeJS

Package.json

package.json > ...

```
1   {
2     "name": "m2gl_app",
3     "version": "0.0.1",
      ▷ Debug
4     "scripts": {
5       "start": "node app.js"
6     },
7     "dependencies": {
8       "express": "^4.17.1"
9     }
10  }
11
```

app.js

app.js > ...

```
1   var express = require('express');
2
3   var app = express();
4
5   var etudiant = {id: 1, nom:'toto', prenom: 'tintin'}
6
7   app.get('/etudiant', function (req, res) {
8     res.send(etudiant);
9   });
10
11  app.listen(3000, function () {
12    console.log('Started at port 3000');
13  });
14
```

Création de web services SOAP avec Springboot

- **Web service SOAP**
  - Ajout dépendances
    - Spring-boot-starter-web-services

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

    - WSDL4J

```
<dependency>
    <groupId>wsdl4j</groupId>
    <artifactId>wsdl4j</artifactId>
</dependency>
```

Création de web services SOAP avec Springboot

- Définition **Schéma XSD** *(/src/main/resources/schema.xsd)*

```xml
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://ws.groupeisi.com">

    <xs:element name="getInfoRequest">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="input" type="xs:string" minOccurs="0"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:element name="getInfoResponse">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="output" type="xs:string" />
            </xs:sequence>
        </xs:complexType>
    </xs:element>

</xs:schema>
```

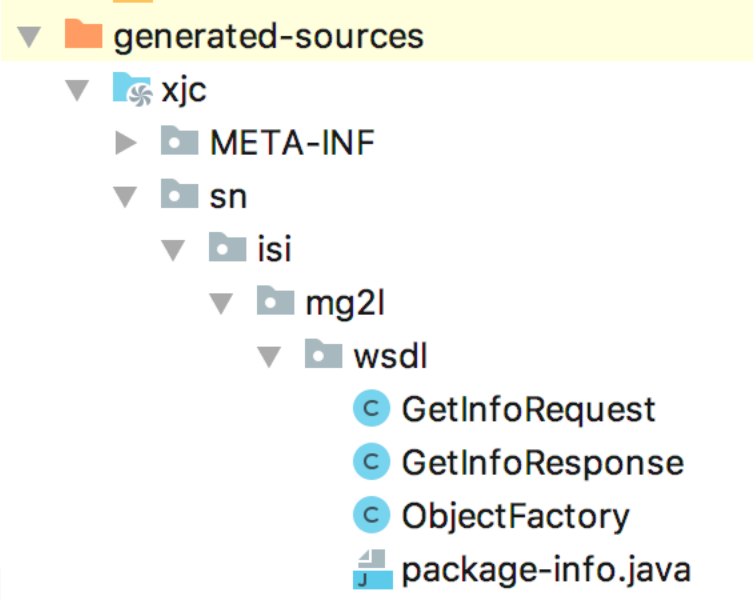
Création de web services SOAP avec Springboot

- Génération des classe à partir du schéma XSD
- Ajout et configuration plugin **maven-jaxb2-plugin**

```
<plugin>
    <groupId>org.jvnet.jaxb2.maven2</groupId>
    <artifactId>maven-jaxb2-plugin</artifactId>
    <version>0.12.2</version>
    <executions>
        <execution>
            <goals>
                <goal>generate</goal>
            </goals>
        </execution>
    </executions>
    <configuration>
        <generatePackage>sn.isi.mg2l.wsdl</generatePackage>
        <schemaDirectory>${project.basedir}/src/main/resources/</schemaDirectory>
    </configuration>
</plugin>
```

Création de web services SOAP avec Springboot

- Génération des classe à partir du schéma XSD
  - Génération à travers Maven
  - **# mvn generate-sources**



Création de web services avec Springboot

- Configuration *(/src/main/java/sn/isi/mg2l/config/WebServiceConfig.java)*

```
@Configuration
public class WebServiceConfig extends WsConfigurerAdapter {

    public static final String HTTP_WS_M2GL = "http://ws.groupeisi.com";

    @Bean
    public ServletRegistrationBean messageDispatcherServlet(ApplicationContext applicationContext) {
        MessageDispatcherServlet servlet = new MessageDispatcherServlet();
        servlet.setApplicationContext(applicationContext);
        servlet.setTransformWsdlLocations(true);
        return new ServletRegistrationBean(servlet, "/ws/*");
    }

}
```

Création de web services avec Springboot

```
@Configuration
public class WebServiceConfig extends WsConfigurerAdapter {
    //messageDispatcherServlet …
    @Bean
    @Qualifier("infoSchema")
    public XsdSchema infoSchema() {
        return new SimpleXsdSchema(new ClassPathResource("schema.xsd"));
    }

    @Bean(name = "infos")
    public DefaultWsdl11Definition infosWsdlDefinition(@Qualifier("infoSchema") XsdSchema infoSchema) {
        DefaultWsdl11Definition wsdl11Definition = new DefaultWsdl11Definition();
        wsdl11Definition.setPortTypeName("infoPort");
        wsdl11Definition.setLocationUri("/ws");
        wsdl11Definition.setTargetNamespace(HTTP_WS_M2GL);
        wsdl11Definition.setSchema(infoSchema);
        return wsdl11Definition;
    }}
```

Bean de configuration du WSDL

Le WSDL sera accessible via :
http://adresse:port/ws/infos.wsdl

Création de web services avec Springboot

- Création d'un endpoint (/src/main/java/sn/isi/mg2l/web/endpoints/M2GLEndpoint.java)

```
@Endpoint
public class M2GLEndpoint {

  @PayloadRoot(namespace = "http://ws.groupeisi.com",localPart = "getInfoRequest")
  public @ResponsePayload GetInfoResponse getInfoRequest (@RequestPayload GetInfoRequest request) {
    GetInfoResponse response= new GetInfoResponse();
    response.setOutput("Bonjour M2GL "+ request.getInput());
    return response;
  }

}
```

Création de web services avec Springboot

- **@Endpoint**

Inscrit la classe qui porte cette annotation auprès de Spring WS comme potentiel candidate à traiter des messages SOAP qui arrivent au serveur.

- **@PayloadRoot**

Spring WS se base sur cette annotation pour trouver la méthode capable de traiter une requête SOAP en se basant sur le namespace et le localpart du message.

- **@RequestPayload**

Indique que le message sera mappé au paramètre request de la méthode

- **@ResponsePayload**

A travers cette annotation, Spring WS réalise le mapping de la valeur retournée par la méthode au payload de la réponse

Introduction à GraphQL

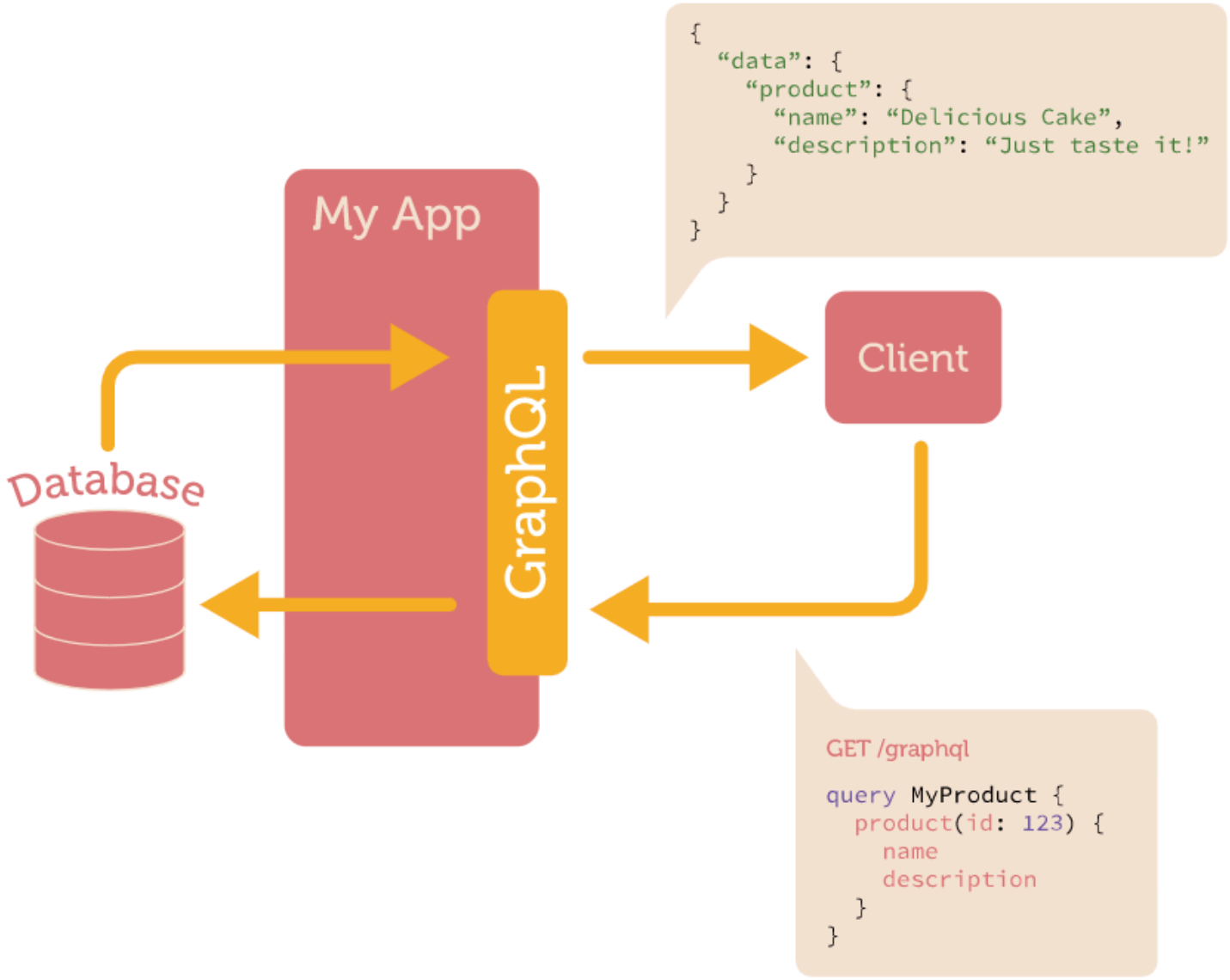
GraphQL

- Language de requête
- Environnement d'exécution
- Créé par Facebook en 2012
- S'appuie sur le modèle client/serveur
- Une alternative aux API REST

GraphQL

- Le client décrit les données qu'il souhaite recevoir
  - Champs
  - Ordre
- Les spécifications de la requêtes sont envoyées en HTTP via la méthode POST
- Le serveur traite et répond spécifiquement à cette requête en suivant la description du client

GraphQL

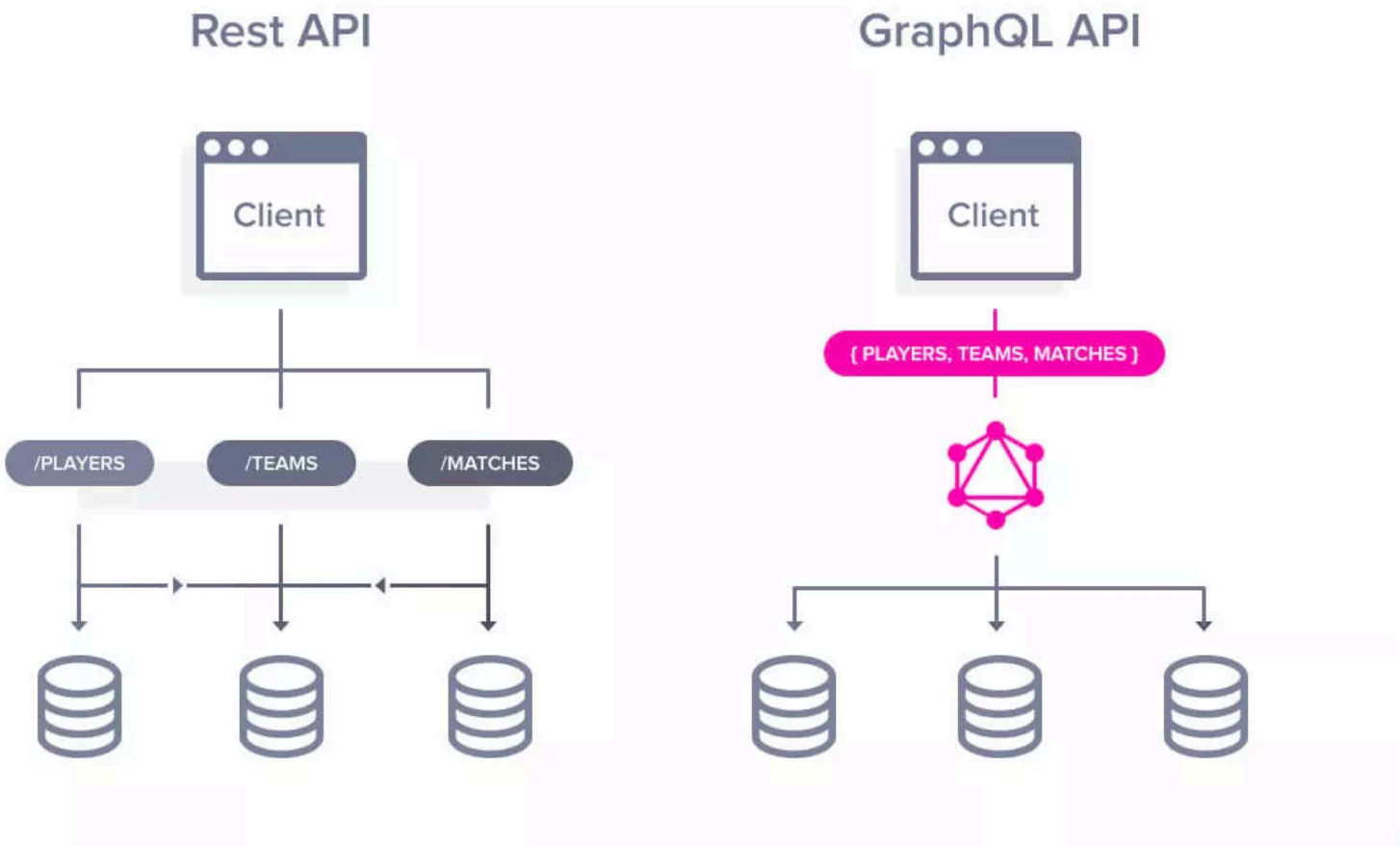

{
"data": {
"product": {
"name": "Delicious Cake",
"description": "Just taste it!"
}
}
}
My App
GraphQL
Client
Database
GET /graphql
query MyProduct {
product(id: 123) {
name
description
}
}

GraphQL : Exemple

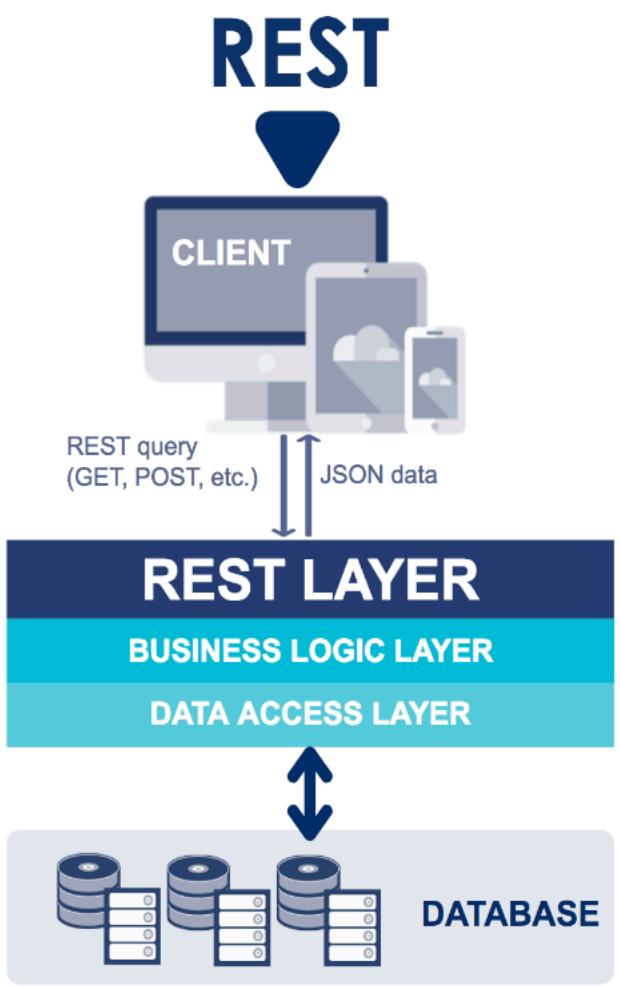
```
{
  orders {
    id
    productsList {
      product {
        id
        name
        weight
        price
      }
      quantity
    }
    totalAmount
  }
}
```

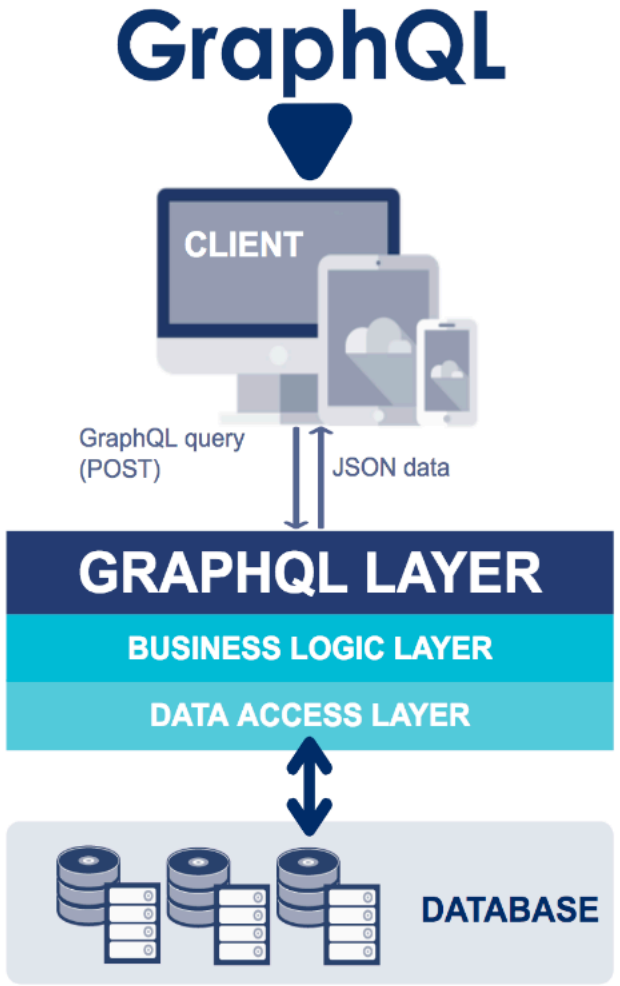
```
{
  "data": {
    "orders": [
      {
        "id": 0,
        "productsList": [
          {
            "product": {
              "id": 1,
              "name": "orange",
              "weight": 0.3,
              "price": 1.5
            },
            "quantity": 100
          }
        ],
        "totalAmount": 250
      }
    ]
  }
}
```

GraphQL Vs REST API

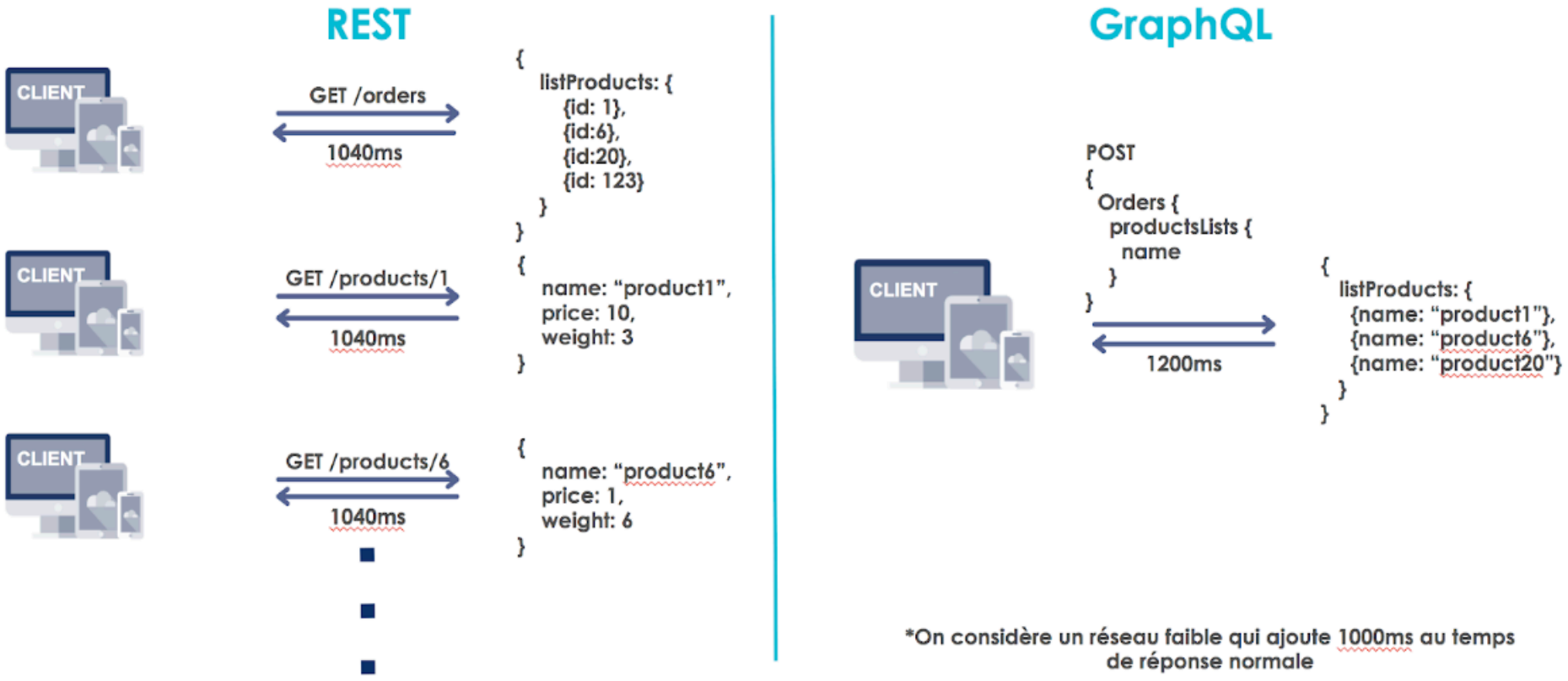


GraphQL Vs REST API





GraphQL vs REST API



GraphQL

- Query

```
query MyProduct {
  product(id: 123) {
    name
    description

    picture {
      width
      height
      url
    }
  }
}
```

```
{
  "data": {
    "product": {
      "name": "Delicious Cake",
      "description": "Just taste it!"

      "picture": {
        "width": 150,
        "height": 150,
        "url": "http://..."
      }
    }
  }
}
```

GraphQL

- Mutation
  - Create
  - Update
  - Delete

```
mutation {
  createPerson(name: "Bob", age: 36) {
    name
    age
  }
}
```

GraphQL

- Subscription
  - Connexion en “realtime” sur un serveur pour être notifié des événements imortants
  - En mode Stream

```
subscription {
  newPerson {
    name
    age
  }
}
```

```
{
  "newPerson": {
    "name": "Jane",
    "age": 23
  }
}
```

Java API for XML Web Services (JAX-WS)

- A la fois un standard et une impléméntation
- Composante de JEE
- Propose un modèle java d'implémentation de services web qui communiquent via des message XML de type SOAP
- Basé sur les POJO (Plain Old Java Object)
- Requiert uniquement des annotation Java

Java API for XML Web Services (JAX-WS)

- **JAX-WS** est bati autour d'un ensemble de **JSR** (*Java Specification Requests*).
- **JSR** est un système normalisé ayant pour but de faire **évoluer la plateforme java** :
  - **JSR 224** (*Java API for XML web service (JAX-WS) 2.0*):

    JAX-WS a été spécifié par cette JSR.
  - **JSR 181** (*Web Services Metadata for Java Platform*) : propose un ensemble d'annotations qui permettent de définir les services web.
- **JSR 109** et **JSR 921** (*Implementing Enterprise Web Services*) : décrit comment déployer, gérer et accéder aux services web par un serveur d'applications.

Java API for XML Web Services (JAX-WS)

- **JSR 222** : Java architecture for XML Binding (**API JAXB 2.0**)

  JAXB 2.0 propose une API et des outils pour automatiser le **mapping** d'un **document XML** et des **objets Java**. En effet, à partir d'une description du document XML (Schéma XML), des classes sont générées pour effectuer automatiquement l'analyse du document XML et le mapping de ce dernier dans des objets Java.

- **JSR 183** (*Web Services Message Security APIs*) : traite de la sécurité des messages SOAP.
- **JSR 250** : Common Annotation for the Java platform.

Java API for XML Web Services (JAX-WS)

- **JAX-WS** repose sur l'utilisation massive d'annotations pour
- la configuration d'un Service Web

  JSR utilisées (JSR 224, 222, 181 et 250)
- Les principales **annotations** utilisées sont les suivantes :
  - **@WebService** : POJO implémentant un service Web
  - **@WebMethod**: Paramétrer une opération
  - **@WebParam** : Paramétrer un message d'entrée
  - **@WebResult** : Paramétrer un message de sortie
  - **@WebFault** : Parmétrer un message fault

Java API for Restful Web Services (JAX-RS)

- JAX-RS est l'API conçue pour implémenter des API Web (aussi appelées Web Services RESTful).
- JAX-RS est une API de programmation pour implémenter rapidement des applications basées sur HTTP.
- JAX-RS 2.x est défini par la JSR 339
- Il existe plusieurs implémentations de cette spécification :
  - Jersey (l'implémentation de référence),
  - RestEasy (intégré dans le serveur Wildfly),
  - Apache CXF.

Java API for Restful Web Services (JAX-RS)

- L'URI représente une **ressource**.
- Une ressource offre une interface uniforme qui est définie dans HTTP par l'ensemble des méthodes : GET, HEAD, POST, PUT, DELETE, OPTIONS…
- Chaque requête vient avec une méthode qui indique le type d'opération que le serveur devra effectuer sur la ressource.

Java API for Restful Web Services (JAX-RS)

- **GET**

  Demande au serveur une représentation de la ressource cible.

- **HEAD**

  Comme un GET sauf que la réponse ne contient jamais de corps. Cette méthode est utile pour obtenir les informations des en-têtes HTTP et valider une requête sans envoyer ni recevoir de corps de message.

- **PUT**

  Crée ou met à jour l'état d'une ressource identifiée par l'URI.

- **DELETE**

  Détruit l'association de l'URI avec l'état de la ressource.

- **POST**

  Cette méthode est utilisable dans différentes situations:

  - Le client souhaite créer une ressource sur le serveur en laissant au serveur le choix de l'URI de la ressource.
  - Le client souhaite ajouter une ressource à une ressource représentant une collection.
  - Le client souhaite que le serveur effectue un traitement.
  - Le client souhaite modifier partiellement une ressource.

Java API for Restful Web Services (JAX-RS)

- **OPTIONS** Permet d'obtenir les options de communication (par exemple : les méthodes autorisées pour l'URI). Le serveur doit retourner ces informations dans les en-têtes de réponse. Ainsi l'en-tête de réponse Allow liste les méthodes HTTP autorisées pour cette URI.
- **TRACE** Permet de simuler un écho de la requête. Cette méthode n'est pas utilisée pour la réalisation d'API Web car elle est surtout utile pour tester la configuration du réseau et obtenir des informations des proxies.
- **CONNECT** Établit un tunnel à travers un proxy. Cette méthode n'est pas utilisée pour la réalisation d'API Web.

Java API for Restful Web Services (JAX-RS)

Exemple

```java
package sn.isi.mg2l.web.endpoints;

import javax.ws.rs.DELETE;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.PUT;
import javax.ws.rs.Path;

@Path("/etudiant")
public class UserResource {

  @GET
  public User retourner() {
    //....
  }

  @PUT
  public User creerOuMettreAjour() {
    //....
  }

  @DELETE
  public void supprimer() {
    //....
  }

  @POST
  @Path("/inscription")
  public void inscrire() {
    //....
  }
```