# Web Service

REST

## INTRODUCTION AUX SERVICES WEB RESTFUL

REST ?

- Acronyme de **RE**presentational **S**tate **T**ransfert
- REST n'est pas un protocole ou un format, contrairement à SOAP, HTTP ou RPC, c'est un style d'architecture inspiré de l'architecture du web fortement basé sur le protocole HTTP
- Il peut utiliser d'autres protocoles

## REQUÊTES REST

### RESSOURCES

Identifiée par une URI ( http://univ.domain/formations/ )

➢ Méthodes permettant de manipuler les ressources (identifiants)

Méthodes HTTP : GET, POST, PUT, DELETE

➢ Représentation: vue sur l'état de la ressource

Format d'échanges entre le client et le serveur (JSON, XML, text/plain,…)



## REQUÊTES REST

### MÉTHODES

- Une ressource peut subir quatre opérations de bases CRUD correspondant aux quatre principaux types de requêtes HTTP (GET, PUT, POST, DELETE)
- REST s'appuie sur le protocole HTTP pour effectuer ces opérations sur les objets

  - CREATE ➔ POST : crée une nouvelle ressource sur le système
  - RETRIEVE ➔ GET : renvoie une représentation de la ressource tel qu'elle est sur le système
  - UPDATE ➔ PUT : met à jour de la ressource sur le système
  - DELETE ➔ DELETE : supprime la ressource identifiée par l'URI sur le serveur

## REQUÊTES REST

### ❑ REPRÉSENTATIONS

- Une représentation désigne les données échangées entre le client et le serveur pour une ressource:
  - HTTP GET ➔ Le serveur renvoie au client l'état de la ressource
  - PUT, POST ➔ Le client envoie l'état d'une ressource au serveur
- Peut être sous différents formats :
  - JSON
  - XML
  - XHTML
  - CSV
  - Text/plain
  - .....



## AVANTAGES REST

- Simplicité
  → Liens structurés et de façon universelle
- Stateless
  → Consommation optimale de mémoire
  → Mise au point plus simple, moins de cas à traiter
- URI uniques
  → Mise en cache possible donc meilleure montée en charge
- Proche de la philosophie d'HTTP
  → Moins de dépendances à une implémentation particulière
- Evolutivité

## REST vs SOAP

Les API REST sont plus légères ➔ adaptées aux applications IoT, développement d'applications mobiles et les services légers.
Les services web SOAP intègrent des spécifications de sécurité et de conformité des transactions ➔ par conséquent plus lourds.

| SOAP | REST |
|---|---|
| Un protocole de message basé le XML | Une architecture |
| Utilise exclusivement XML (WSDL) | propose différents formats JSON, XML, CSV, etc |
| Utilise des services en appelant des méthodes RPC | Utilise des services en appelant des URLs (PUT, GET, POST, DELETE) |
| Requière plus de bande passante et de ressources | Requière moins de bande passante et de ressources |
| Lent | Rapide |



## REST vs SOAP

- SOAP fonctionne efficacement dans un environnement distribué, alors que les API REST fonctionnent efficacement dans le cadre d'une communication directe/point à point.
- SOAP offre des protocoles de sécurité intégrés tels que les protocoles de sécurité WS. Cela incluait WS-Security, qui fournit le chiffrement, les signatures numériques et la sécurité au niveau des messages pour améliorer la sécurité des services Web basés sur SOAP
- Les API SOAP offrent des options intégrées pour la gestion des erreurs.
- REST est plus simple et plus flexible en termes d'utilisation.
- L'API REST ne nécessite pas d'outils coûteux ni de langages de programmation avancés pour connecter les applications.
- L'API REST est facile à apprendre et sa courbe d'apprentissage est plus courte.
- REST est plus efficace car il n'utilise pas le format XML pour la transmission des messages.

## JAKARTA RESTFUL WS

- Acronyme de Jakarta RESTful Web Services
- Version courante 3.1.9 (publiée le 17 octobre 2024)
- Depuis la première version, il fait partie intégrante de la spécification Jakarta EE
- Décrit la mise en œuvre des services REST web coté client
- Après le transfert de Java EE vers la fondation Eclipse, le nom Jakarta EE a remplacé Java EE → Le standard JAX-RS a été renommé en Jakarta RESTful Web Services.
- Implémentations:
  - JERSEY : implémentation de référence fournie par Oracle
  - RESTEasy : fournie par Red Hat/JBOSS
  - Apache CXF
  - ….
- **Package principal :** jakarta.ws.rs.*



## ANNOTATIONS JAKARTA REST

- @Path : définit l'uri de la ressource
- @GET, @POST, @PUT, @DELETE
  - Permettent de mapper une méthode à un type de requête HTTP
  - Ne sont utilisables que sur des méthodes
  - Le nom de la méthode n'a pas d'importance, JAX détermine la méthode à exécuter en fonction de la requête

## ANNOTATIONS JAKARTA-REST

- Paramètres des requêtes
  - @PathParam : Permet au consommateur de service de transmettre l'entrée dans l'URI de service.
  - @QueryParam : valeurs des paramètres de la requête
  - @FormParam : Valeurs des paramètres de formulaire
  - @HeaderParam: Valeurs dans l'en tète de la requête
  - @CookieParam : Valeurs des cookies
  - @Context : Informations liés au contexte de la ressource



## ANNOTATIONS JAKARTA-REST

- Exemple

```
@Path("/calcul")
public class Calcul {
public static int euro=40;
@GET
@Path("/val")
@Produces("text/plain") //http://localhost:8080/RestAtelier1/calcul/val
public int getValeur() {
return 10;
}
@GET
@Produces("text/plain")
@Path("/prd/{param1}/{param2}")
    //http://localhost:8080/RestAtelier1/calcul/prd/5/8
public int produit(@PathParam("param1")int a,@PathParam("param2")int b) {
return a*b;
}
@GET
@Produces("text/plain")
@Path("/sum")
    //http://localhost:8080/RestAtelier1/calcul/sum?param1=3&param2=6
public int somme(@QueryParam("param1")int a,@QueryParam("param2")int b) {
return a+b;
}
```

## ANNOTATIONS SPRINGMVC

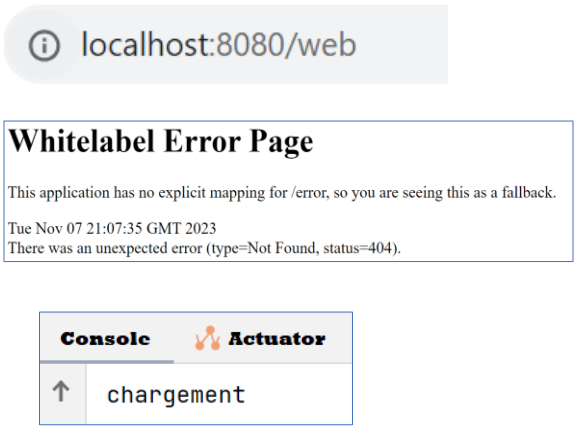
- Spring MVC et Jakarta REST servent tous deux à créer des API REST en Java, mais ils appartiennent à deux mondes différents
- Non standard, mais très populaire dans l'industrie
- Orienté productivité et simplicité de développement
- `org.springframework.web.bind.annotation.*`



## ANNOTATIONS SPRINGMVC

```
@Controller
public class SimpleController {
    @GetMapping("/web")
    public String loadPage(){
        System.out.println("chargement");

        return "test";
    }
```

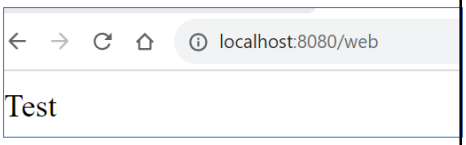


## ANNOTATIONS SPRINGMVC

```
@Controller
public class SimpleController {

    @ResponseBody
    @GetMapping("/web")
    public String loadPage(){
        System.out.println("chargement");

        return "Test";
    }
```






## ANNOTATIONS SPRINGMVC

- SpringMVC n'est pas une implémentation du standard JAKARTA REST mais une alternative offerte par spring assurant toutes les fonctionnalités REST
- @RestController : déclare un contrôleur Rest
- Les opérations Rest sont assurés par :
  - @GetMapping( ou @RequestMapping())
  - @PostMapping ( ou @RequestMapping(method=RequestMethod.POST))
  - @PutMapping ( ou @RequestMapping(method=RequestMethod.PUT))
  - @DeleteMapping ( ou @RequestMapping(method=RequestMethod.DELETE))

## ANNOTATIONS SPRINGMVC

- **@RequestBody**

Mappe le corps du HttpRequest à une entité permettant ainsi la désérialisation automatique du corps HttpRequest vers un objet Java

- **@ResponseBody**

Indique à un contrôleur que l'objet renvoyé est sérialisé automatiquement dans un JSON et renvoyé dans l'objet HttpResponse

- Paramètres des requêtes
  - @PathVariable (/api/client/{id})
  - @RequestParam (/api/client?name=emsi)



## @RESTCONTROLLER

| @Controller | @RestController |
|---|---|
| @Controller est utilisé pour marquer les classes comme Spring MVC Controller. | L'annotation @RestController est un contrôleur spécial utilisé dans les services Web RESTful, et c'est la combinaison de l'annotation @Controller et @ResponseBody. |
| Il s'agit d'une version spécialisée de l'annotation @Component. | Il s'agit d'une version spécialisée de l'annotation @Controller. |
| Dans @Controller, nous pouvons renvoyer une vue dans Spring Web MVC. | Dans @RestController, nous ne pouvons pas renvoyer de vue. |
| L'annotation @Controller indique que la classe est un « contrôleur » comme un contrôleur Web. | L'annotation @RestController indique que la classe est un contrôleur où les méthodes @RequestMapping assument la sémantique @ResponseBody par défaut. |
| Dans @Controller, nous devons utiliser @ResponseBody sur chaque méthode de gestionnaire. | Dans @RestController, nous n'avons pas besoin d'utiliser @ResponseBody sur chaque méthode de gestionnaire. |
| Il a été ajouté à la version Spring 2.5. | Il a été ajouté à la version Spring 4.0. |

## ANNOTATIONS

- @Controller vs @RestController

```
@Controller
public class SimpleController {
    @GetMapping("/web")
    @ResponseBody
    public String loadPage(){
        System.out.println("chargement");
        return "test";
  }
```

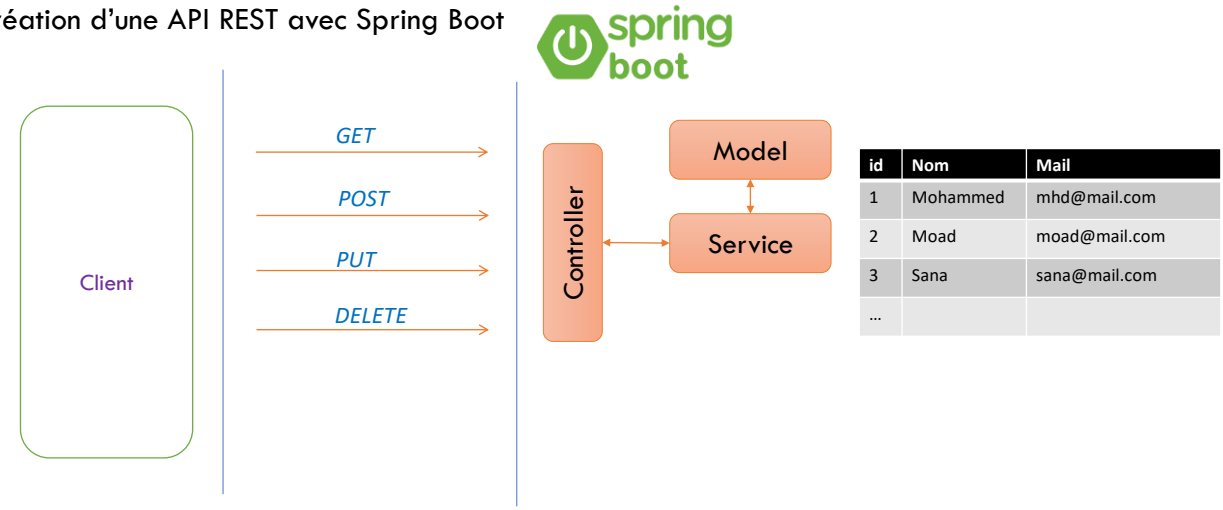
```
@RestController
public class SimpleController {
    @RequestMapping(method = RequestMethod.GET, value="/web")
    public String loadPage(){
        System.out.println("chargement");
        return "test";
    }
}
```



## APPLICATION- MISE EN ŒUVRE DES WEB SERVICES AVEC SPRING REST CONTROLLER

- @RestController
- @GetMapping
- @PostMapping
- @RequestParam
- @DeleteMapping

## Création d'une API REST avec Spring Boot

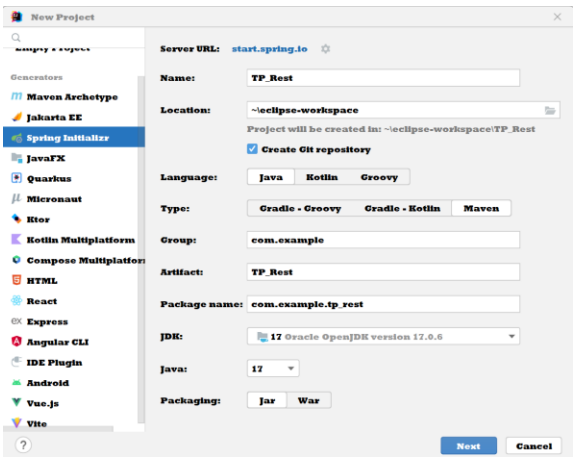


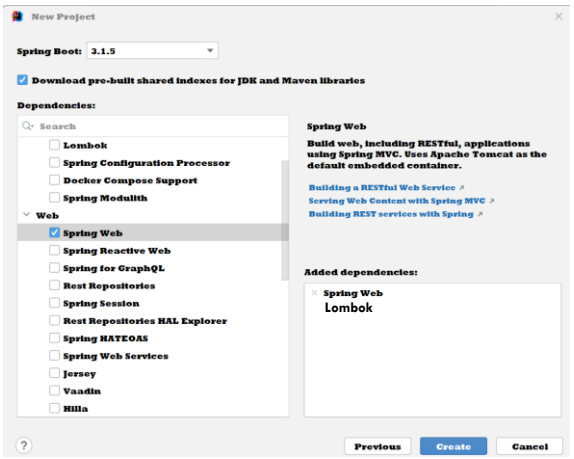

| id | Nom | Mail |
|---|---|---|
| 1 | Mohammed | mhd@mail.com |
| 2 | Moad | moad@mail.com |
| 3 | Sana | sana@mail.com |
| … | | |





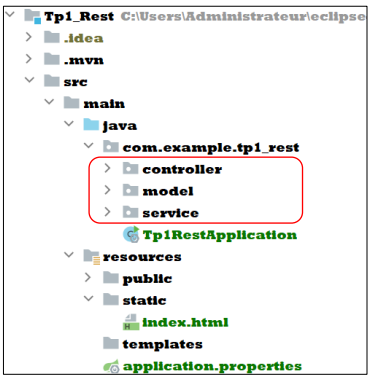
- Créer un nouveau Projet



## APPLICATION- MISE EN ŒUVRE DES WEB SERVICES AVEC SPRING REST CONTROLLER

- Ajouter les packages « model », « service » et « controller », dans le package com.examle.tp1_Rest






## APPLICATION- MISE EN ŒUVRE DES WEB SERVICES AVEC SPRING REST CONTROLLER

- Ajouter la classe « User » au modèle (model)

```
@Data
@AllArgumentContructor
@NoArgumentConstructor
public class User {
        int id;
        String nom;
        String mail;
}
```

## APPLICATION- MISE EN ŒUVRE DES WEB SERVICES AVEC SPRING REST CONTROLLER

- Ajouter la classe « UserService » au metier (package service)

```
@Service
public class UserService {
    private List<User> userList;

    public UserService() {
        userList = new ArrayList<>();
        User user1=new User(1,"Mohammed","mhd@mail.com");
        User user2=new User(2,"Moad","moad@mail.com");
        User user3=new User(3,"Sana","sana@mail.com");
        User user4=new User(4,"Ahlam","ahlam@mail.com");
        userList.addAll(Arrays.asList(user1,user2,user3,user4));
    }
public User getUser(Integer id){
        for(User usr:userList)
        if(id==usr.getId()) {
         return usr;
        }
    return null;
}
```

```
public void suppUser(Integer id){
        for(User usr:userList)
            if(id==usr.getId()) {
                userList.remove(usr);
            }
    }
    public List<User> getUsers(){

        return userList;
    }
}
```



## APPLICATION- MISE EN ŒUVRE DES WEB SERVICES AVEC SPRING REST CONTROLLER

- Ajouter la classe « UserService » au metier (suite)

```
public String updateUser(int id,User user){
    int i=0;
    for(User usr:userList) {
        if (usr.getId() == id) {
            userList.set(i, user);
            return String.format("l'utilisateur %s est bien modifié !", id);
        } else
            i++;
    }
    return String.format("lutilisateur %s n'existe pas !",id);
}

public String addUser(User user){
    userList.add(user);
    return String.format("lutilisateur %s est bien ajouté !",user.getId());
}
```

SUITE

## Ajouter la classe « UserController » au controleur (package controller)

```java
@RestController
public class UserController
{
    private UserService userService;
    @Autowired
    public UserController(UserService userService) {
        this.userService = userService;
    }
    @GetMapping("/user/{p}")
    public User getUser(@PathVariable("p") Integer id){
        User user=userService.getUser(id);
        if(user!=null){
            return user;
        }
        return null;
    }
    @GetMapping("/users")
    public List<User> getUsers(){
        return userService.getUsers();
    }

}
```

```java
@PostMapping("/add")
public String addUtilisateur(@RequestBody User user){
    return userService.addUser(user);
}
```

```java
@DeleteMapping("/supp")
    public void SupprimerUser(@RequestParam("p")
Integer id){
        userService.suppUser(id);
    }
```

```java
@PutMapping("/update/{p}")
public String UpdateUser(@PathVariable("p") int id,@RequestBody User user){
    return userService.updateUser(id,user);
}
```



## APPLICATION- MISE EN ŒUVRE DES WEB SERVICES AVEC SPRING REST CONTROLLER

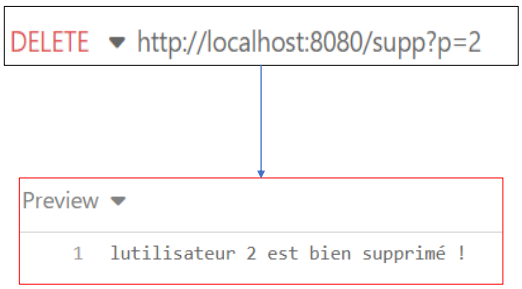
- Démarrer l'application
- Tester la liste des services : Récupération

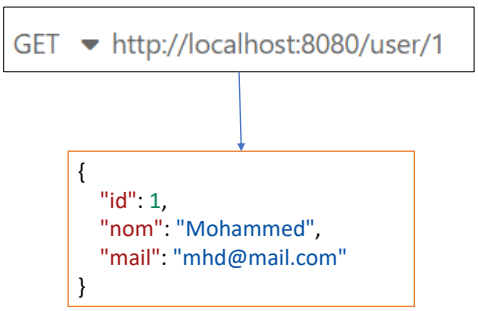
GET http://localhost:8080/users

```json
[
  {
    "id": 1,
    "nom": "Mohammed",
    "mail": "mhd@mail.com"
  },
  {
    "id": 2,
    "nom": "Moad",
    "mail": "moad@mail.com"
  },
  {
    "id": 3,
    "nom": "Sana",
    "mail": "sana@mail.com"
  },
  {
    "id": 4,
    "nom": "Ahlam",
    "mail": "ahlam@mail.com"
  }
]
```

## APPLICATION- MISE EN ŒUVRE DES WEB SERVICES AVEC SPRING REST CONTROLLER

- Démarrer l'application
- Récupération et suppression

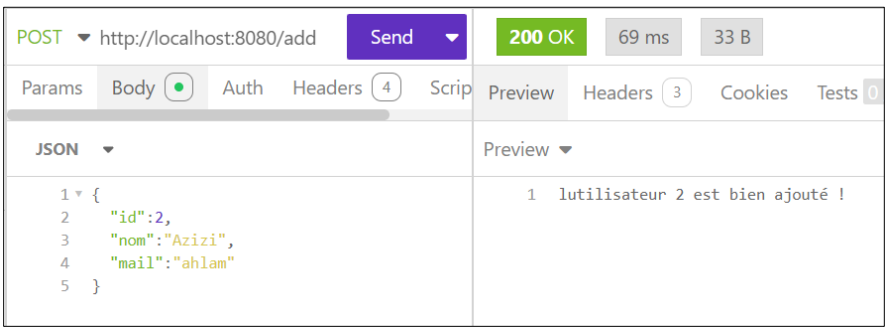




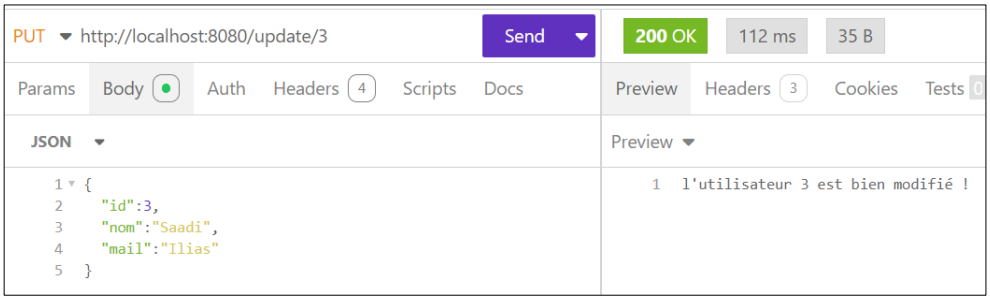

## APPLICATION- MISE EN ŒUVRE DES WEB SERVICES AVEC SPRING REST CONTROLLER

- Démarrer l'application
- Ajout



## APPLICATION- MISE EN ŒUVRE DES WEB SERVICES AVEC SPRING REST CONTROLLER

- Démarrer l'application
- Mise à jour






## REST VS RESTFULL

- L'API REST est un style architectural pour une API qui utilise des requêtes HTTP pour accéder aux données, les utiliser et les échanger en toute sécurité sur Internet. L'API REST est un moyen pour deux systèmes informatiques de communiquer

- L'API RESTful est une interface qui permet à deux systèmes différents d'échanger des informations sur Internet avec une sécurité renforcée. Les API RESTful offrent une méthode simple et évolutive pour créer des API applicables à divers langages et plates-formes de programmation.

## REST VS RESTFULL

- REST est un style d'architecture logicielle qui exploite essentiellement la technologie et les protocoles existants du Web.
- RESTful est généralement utilisé pour faire référence aux services Web implémentant l'architecture REST.
- Considérer REST comme une « classe » architecturale tandis que RESTful est « l'instance » bien connue de cette classe.
- REST (REpresentation State Transfer) est une architecture à l'aide de laquelle les WebServices sont créés.
- RESTful est un moyen d'écrire des services en utilisant les architectures REST. Les services RESTful exposent les ressources pour identifier les cibles pour interagir avec les clients.