

WEB SERVICE

Définition

Un web service (WS) est un programme qui permet une communication et un échange de données entre applications et systèmes hétérogènes (même si ces dernières sont construites dans des applications différentes) dans un environnement distribué.

Parmi les web services les plus connus, on peut citer : **SOAP**, **REST** ou **HTTP**. Elles sont utilisées généralement sur des **infrastructure cloud**, en **cloud public**, **privé** ou en **cloud hybride**.

Fonctionnement d'un web service

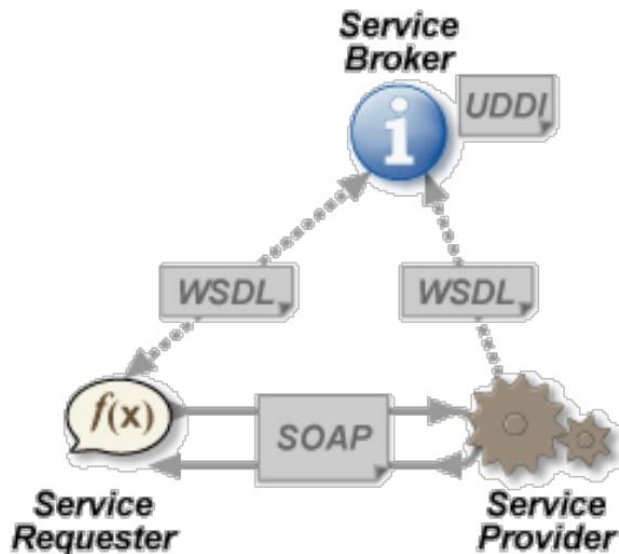
1. Le client effectue une requête dans un des langage : **XML**, **JSON**, **HTTP**.
2. Cette requête est transmise à un serveur distant via les protocoles **SOAP**, **REST** ou **HTTP**.
3. La réponse est ensuite délivrée sous le même format que sa demande.

Ce processus est décrit grâce à un modèle en couche :

Invocation : elle décrit la structure des messages échangés par le client et le serveur (standards **XML-RPC** ou **SOAP**).

Découverte : c'est la phase de recherche et de localisation des données demandées (protocole **UDDI**).

Description : stipule les paramètres des fonctions et les types de données des services web utilisés (protocole standard **WDSL** qui repose sur la notation **XML**).



Les avantages du Web services

1. L'ordinateur du client peut recevoir des informations d'un serveur distant sans pour autant devoir stocker toutes les données.
2. Un serveur distant peut être interrogé par un grand nombre de personnes de manière simultanée.
3. Les informations peuvent être cryptées. C'est le cas en utilisant la technologie SSL présente dans le protocole HTTPS.

Les API

Une API (Application Programming Interface) peut être vue comme un « pont » ou bien une sorte de « passerelle » entre deux applications. Une API permet d'accéder à un service comme des données ou des fonctionnalités fournies par un système tiers. On dit alors que **le système tiers expose une API**.

API vs Web services

L'unique différence entre ces deux concepts est que le web service a pour mission de **faciliter la communication entre deux machines**. À la différence de l'API qui elle va servir d'interface utilisée pour la communication entre les deux applications.

Une API n'a pas nécessairement besoin de réseau pour fonctionner alors qu'un web service a toujours besoin d'un réseau. Par conséquent toutes les API ne sont pas des web services, mais tous les web services sont des API.

L'API est une architecture légère alors que le web service ne dispose pas d'une architecture légère car il a besoin de protocoles pour fonctionner.

RESTfull

On appelle RESTfull une API compatible REST (RESTfull est développé avec le web service REST). Elle permet d'effectuer des requêtes HTTP pour obtenir (GET), placer (PUT), publier (POST) et supprimer (DELETE) des données. Elle est aujourd'hui utilisée par de nombreux sites comme **Google, Amazon, Twitter et LinkedIn**.

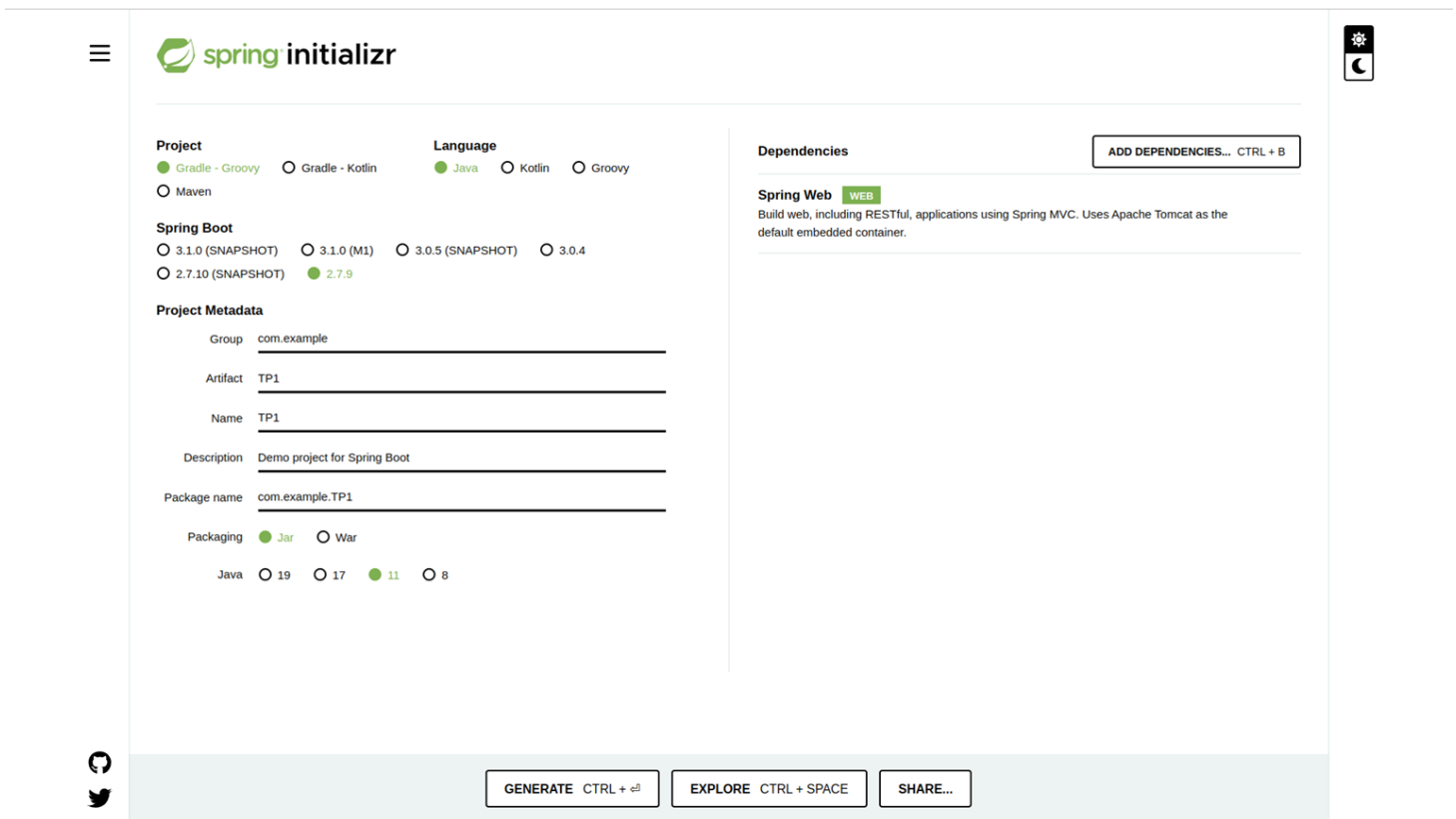
Exemple avec RESTfull :

1. Un client envoie une requête **GET** pour récupérer un livre spécifique dans une librairie.
2. Une demande **POST** ajoutera un nouveau livre à la librairie
3. **PUT** permet de mettre à jour le contenu d'un livre avec un ID
4. **DELETE** une requête supprime un enregistrement de livre de la librairie

TP : Gestion d'un service de location de voitures

Technologies: Java, java spring boot

1. Utiliser Spring Initializer pour créer un projet vide : <https://start.spring.io/>



The image shows the Spring Initializr web interface. It is a form for generating a Spring project. The interface is divided into several sections: Project, Language, Spring Boot, Project Metadata, Dependencies, and a footer with social media icons and action buttons.

Project

- ☒ Gradle - Groovy ☐ Gradle - Kotlin ☐ Maven

Language

- ☒ Java ☐ Kotlin ☐ Groovy

Spring Boot

- ☐ 3.1.0 (SNAPSHOT) ☐ 3.1.0 (M1) ☐ 3.0.5 (SNAPSHOT) ☐ 3.0.4
- ☐ 2.7.10 (SNAPSHOT) ☒ 2.7.9

Project Metadata

Group:

Artifact:

Name:

Description:

Package name:

Packaging

- ☒ Jar ☐ War

Java

- ☐ 19 ☐ 17 ☒ 11 ☐ 8

Dependencies

Spring Web ☒ WEB

Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Footer

2. Décompresser le projet

3. Ouvrir le projet sur Eclipse (import gradle project) ou sur IntelliJ (open project)

4. Créer dans src/main/java/package/... une classe Car

5. Créer un attribut **plateNumber** (numéro de la plaque), un attribut **brand** (marque de la voiture) de type String et un attribut **price** (prix location de la voiture) de type String dans la classe Car.

6. Créer deux constructeurs dans la classe Car:

→ Le constructeur par défaut

→ Le constructeur qui initialise les attributs

7. Initialiser les getter et les setter pour les attributs de la classe Car.

8. Surcharger la méthode **toString** pour afficher les attributs de la classe Car

9. Créer une classe CarRentalService et compléter avec le script ci-dessous

```
@RestController
public class CarRentalService {
    6 usages
    private ArrayList<Car> cars = new ArrayList<Car>();

    public CarRentalService() {
        cars.add(new Car( plateNumber: "11AA22", brand: "Ferrari", price: 1000));
        cars.add(new Car( plateNumber: "33BB44", brand: "Porsche", price: 2222));
    }
}
```

10. Récupérer tous les cars du service avec une requête **GET** : compléter le script ci-contre .

```
// Get list of cars
@GetMapping("/cars")
public ArrayList<Car> getListOfCars(){
    ...
}
```

Voir le résultat avec : <http://localhost:8080/cars> ou
`curl -H "Content-Type: application/json" -X GET http://localhost:8080/cars/`

11. Rechercher un car à partir de son numéro de plaque (plateNumber) : Compléter le script ci-contre .

```

@GetMapping("/cars/{plateNumber}")
public Car getCar(@PathVariable(value="plateNumber") String plateNumber){
    ...
}

```

Faire un test avec : `http://localhost:8080/cars/11AA22` ou `curl -H "Content-Type: application/json" -X GET http://localhost:8080/cars/11AA22`

12. Mise à jour d'un car avec la requête **PUT** : compléter le script ci-contre.

```

@PutMapping("/cars")
public void rent(@RequestBody Car car){
    System.out.println(car);
    // parcourir le tablea à la recherche d'une voiture de plaque plaque
    // si voiture trouvé
    //     si rented = true => louer
    //     sinon ramener
    // si voiture non trouvé
    // envoyer HttpStatus NOT-FOUND
}

```

Faire un test avec : `curl -H "Content-Type: application/json" -X PUT -d '{"plateNumber":"11AA22", "brand":"aa"}' http://localhost:8080/cars`

13. Ajouter un car avec la requête **POST** : compléter le script ci-contre.

```

// add a new car
@PostMapping("/cars")
public ArrayList<Car> addCar(@RequestBody Car car) throws Exception{
    System.out.println(car);
}

```

Faire un test avec : `curl -v -H "Content-Type: application/json" -X POST -d '{"plateNumber":"1234PL","brand":"ferari", "price":10000}' http://localhost:8080/cars`