EPSI l'École d'ingénierie informatique

Module : Services Web

Version 2

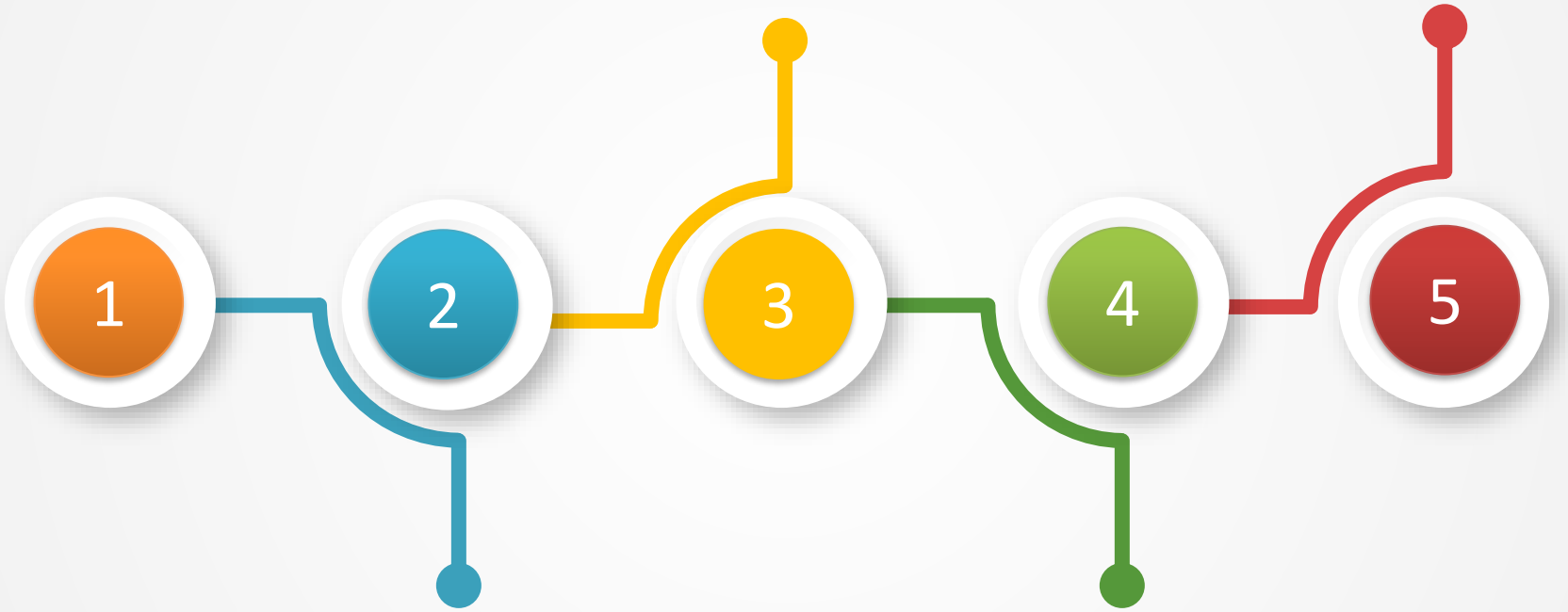
Formateur :

DANGLA Loïc
loic.dangla1@reseau-cd.net

Authentification
Différentes méthodes

XML-RPC
Concept général
Avantages

Spring boot
Mise en place



WSDL
Règles générales
Concept

UDDI
Concept général
Avantages

API

Authentification

Authentification des API

- L'authentification est un mécanisme essentiel pour sécuriser les API et contrôler l'accès aux ressources.
- Dans cette section, nous explorerons différentes méthodes d'authentification des API.

Il existe plusieurs méthodes pour sécuriser les API :

Méthode 1 : Authentification basée sur les clés d'API (API Key)

- L'authentification basée sur les clés d'API est une méthode simple mais efficace pour contrôler l'accès aux API.
- Chaque client d'API possède une clé unique qui est incluse dans chaque requête.
- Le serveur vérifie la validité de la clé et autorise ou refuse l'accès en fonction de cette vérification.

API

Authentification

Méthode 2 : Authentification basée sur les jetons d'accès (Access Tokens)

- L'authentification basée sur les jetons d'accès est couramment utilisée pour les API qui nécessitent une interaction continue avec l'utilisateur.
- L'utilisateur se connecte avec ses identifiants et reçoit un jeton d'accès.
- Le jeton d'accès est inclus dans les requêtes ultérieures pour prouver l'authentification.

Méthode 3 : Authentification basée sur Oauth

- OAuth est un protocole d'autorisation standard utilisé pour l'authentification sécurisée entre applications.
- Il permet à une application d'agir au nom de l'utilisateur sans partager les identifiants de connexion.
- OAuth utilise des jetons d'accès pour permettre aux applications d'accéder aux ressources protégées.

API

Authentification

Différence entre OAuth et OAuth 2.0 ?

Elles sont deux versions différentes du protocole d'autorisation OAuth. Voici les principales différences entre OAuth 1.0 et OAuth 2.0 :

- **Complexité** : OAuth 1.0 est considéré comme plus complexe à mettre en œuvre que OAuth 2.0. OAuth 2.0 a été conçu pour être plus simple, plus flexible et plus adapté aux besoins actuels des applications.

- **Flux d'autorisation** : OAuth 1.0 utilise le flux de signature de requête, qui nécessite la signature de chaque requête avec des clés secrètes. En revanche, OAuth 2.0 utilise des jetons d'accès et des rafraîchissements de jeton, ce qui simplifie le flux d'autorisation.

- **Authentification** : OAuth 1.0 ne spécifie pas de méthode d'authentification standard, laissant aux développeurs le soin de choisir la méthode appropriée. En revanche, OAuth 2.0 introduit des mécanismes d'authentification standard tels que l'authentification basée sur les clés d'API (API Key) et l'authentification basée sur les jetons d'accès (Access Tokens).

API

Authentification

```java
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Value("${security.oauth2.resource.jwt.key-value}")
    private String jwtSigningKey; // Clé pour signer et vérifier les JWT

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
                .authorizeRequests()
                .antMatchers("/api/public").permitAll() // Endpoint public accessible sans authentification
                .anyRequest().authenticated()
                .and()
                .oauth2ResourceServer()
                .jwt()
                .signatureAlgorithm("RS256") // Algorithme de signature utilisé pour les JWT
                .jwkSetUri(jwtSigningKey); // Clé publique pour vérifier les JWT
    }
}
```

API

Authentification

- **Évolution** : OAuth 1.0 a été publié en 2010 et a connu des problèmes de sécurité et de complexité. OAuth 2.0 a été introduit pour remédier à ces problèmes et est largement adopté comme la version préférée d'OAuth.

- **Compatibilité descendante** : En raison de différences majeures dans la conception et la mise en œuvre, OAuth 2.0 n'est pas rétrocompatible avec OAuth 1.0. Les serveurs et les clients doivent être spécifiquement conçus pour prendre en charge OAuth 2.0.

Il convient de noter que la version 1.0 de OAuth n'est plus recommandée pour de nouvelles implémentations, et OAuth 2.0 est la version recommandée et largement utilisée pour les scénarios d'autorisation dans les applications modernes. OAuth 2.0 offre une meilleure simplicité, flexibilité et adaptabilité aux besoins actuels des applications.

API

Authentification

Méthode 4 : Authentification basée sur les certificats :

- L'authentification basée sur les certificats utilise des certificats numériques pour établir l'identité d'une partie.
- Les clients et les serveurs échangent des certificats pour s'authentifier mutuellement.
- Cette méthode est couramment utilisée dans les environnements hautement sécurisés.

Meilleures pratiques de l'authentification des API :

- Utiliser HTTPS pour chiffrer les communications entre le client et le serveur.
- Ne jamais transmettre d'informations d'identification sensibles en clair.
- Utiliser des mécanismes de renouvellement automatique des jetons d'accès.
- Limiter les autorisations accordées aux utilisateurs ou aux applications en fonction de leurs besoins.

API

Authentification

Exemple d’un projet en Oauth 2 :

- Structure du projet

```
src
├── main
│   ├── java
│   │   ├── com
│   │   │   ├── example
│   │   │   │   ├── Application.java
│   │   │   │   ├── config
│   │   │   │   │   └── SecurityConfig.java
│   │   │   │   ├── model
│   │   │   │   │   └── User.java
│   │   │   │   ├── resource
│   │   │   │   │   ├── SecureUserResource.java
│   │   │   │   │   └── UserResource.java
│   │   │   │   └── exception
│   │   │   │       └── UnauthorizedException.java
│   ├── resources
│   │   └── META-INF
│   │       └── persistence.xml
│   └── webapp (pour les projets déployés en tant que WAR)
│       └── WEB-INF
│           └── web.xml (facultatif, uniquement pour les projets déployés en tant que WAR)
└── test
    └── java
        └── com
            └── example
                └── // Tests unitaires ou d'intégration (facultatif)
```

API

Authentification

- **Explications de la structure du projet :**

- **Application.java** : Classe principale contenant la méthode main pour démarrer l'application Spring Boot.

- **config** : Package contenant les configurations Spring, telles que la configuration de la sécurité (SecurityConfig).

- **model** : Package contenant les classes de modèle, comme User, pour représenter les entités stockées dans la base de données.

- **resource** : Package contenant les ressources JAX-RS, telles que UserResource et SecureUserResource, qui définissent les endpoints de l'API.

- **exception** : Package contenant les classes d'exceptions personnalisées, telles que UnauthorizedException, pour gérer les erreurs spécifiques de l'API.

- **resources/META-INF/persistence.xml** : Fichier de configuration pour JPA, définissant la source de données et d'autres propriétés de persistance.

API

Authentification

- **Application.java :**

```java
package com.example;

import javax.ws.rs.ApplicationPath;
import javax.ws.rs.core.Application;

@ApplicationPath("/api")
public class Application extends Application {
    // Pas besoin de codage supplémentaire ici, l'application est configurée automatiquement en utilisant JAX-RS.
}
```

API

Authentification

- **SecurityConfig.java :**

```java
package com.example.config;

import javax.ws.rs.container.ContainerRequestContext;
import javax.ws.rs.container.ContainerRequestFilter;
import javax.ws.rs.core.Response;
import java.io.IOException;

no usages
public class SecurityConfig implements ContainerRequestFilter {

    1 usage
    private static final String API_KEY = "YOUR_API_KEY"; // Remplacez par votre propre clé API

    @Override
    public void filter(ContainerRequestContext requestContext) throws IOException {
        String apiKey = requestContext.getHeaderString("api-key");
        if (!API_KEY.equals(apiKey)) {
            requestContext.abortWith(Response.status(Response.Status.UNAUTHORIZED).entity("Clé API invalide.").build());
        }
    }
}
```

API

Authentification

- **User.java :**

```java
package com.example.model;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import java.util.UUID;

@Entity
public class User {

    no usages
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    no usages
    private String username;
    no usages
    private String password;
    1 usage
    private String apiKey;

    // Constructeurs, getters et setters (omis pour la simplicité)

    public User() {
        this.apiKey = UUID.randomUUID().toString();
    }
}
```

API

Authentification

- **UserResource.java :**

```java
package com.example.resource;

import com.example.model.User;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import javax.ws.rs.*;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;
import java.util.List;
import java.util.UUID;

no usages
@Path("/users")
public class UserResource {

    no usages
    @POST
    @Consumes(MediaType.APPLICATION_JSON)
    public Response createUser(User user) {
        // Génération d'une clé API unique pour le nouvel utilisateur
        user.setApiKey(UUID.randomUUID().toString());

        // Enregistrement de l'utilisateur dans la base de données
        EntityManagerFactory emf = Persistence.createEntityManagerFactory("userPU");
        EntityManager em = emf.createEntityManager();
        em.getTransaction().begin();
        em.persist(user);
        em.getTransaction().commit();
        em.close();
        emf.close();

        return Response.status(Response.Status.CREATED).entity("Utilisateur enregistré avec succès. Clé API : " + user.getApiKey()).build();
    }
}
```

API

Authentification

- **SecureUserResource.java :**

```java
package com.example.resource;

import com.example.model.User;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.NoResultException;
import javax.persistence.Persistence;
import javax.ws.rs.GET;
import javax.ws.rs.HeaderParam;
import javax.ws.rs.Path;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;

no usages
@Path("/user")
public class SecureUserResource {

    no usages
    @GET
    @Path("/details")
    public Response getUserDetails(@HeaderParam("api-key") String apiKey) {
        // Vérification de la clé API dans la base de données
        EntityManagerFactory emf = Persistence.createEntityManagerFactory("userPU");
        EntityManager em = emf.createEntityManager();

        try {
            User user = em.createQuery("SELECT u FROM User u WHERE u.apiKey = :apiKey", User.class)
                    .setParameter("apiKey", apiKey)
                    .getSingleResult();

            return Response.status(Response.Status.OK).entity("Détails de l'utilisateur : " + user.getUsername()).build();
        } catch (NoResultException e) {
            return Response.status(Response.Status.UNAUTHORIZED).entity("Clé API invalide.").build();
        } finally {
            em.close();
            emf.close();
        }
    }
}
```

API

Authentification

- **UnauthorizedException.java :**

```java
package com.example.exception;

import javax.ws.rs.WebApplicationException;
import javax.ws.rs.core.Response;

no usages
public class UnauthorizedException extends WebApplicationException {

    no usages
    public UnauthorizedException(String message) {
        super(Response.status(Response.Status.UNAUTHORIZED).entity(message).build());
    }
}
```

API

Authentification

- **persistence.xml :**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence http://xmlns.jcp.org/xml/ns/persistence/persistence_2_2.xsd"
             version="2.2">
<persistence-unit name="userPU" transaction-type="RESOURCE_LOCAL">
<class>com.example.model.User</class>
```

API

Authentification

Récapitulatif :

- L'authentification des API est essentielle pour garantir la sécurité des ressources.
- Les méthodes d'authentification varient en fonction des besoins et des exigences de chaque API.
- Il est important de choisir la méthode d'authentification appropriée en fonction du contexte et des exigences de sécurité.

Technologies associées

Introduction

Qu'est-ce que le WSDL ?

- WSDL est un langage basé sur XML (eXtensible Markup Language) utilisé pour décrire les services Web.
- Il fournit une description formelle des fonctionnalités offertes par un service Web, ainsi que des informations sur la manière d'y accéder.

Objectif du WSDL

- Le WSDL permet de décrire l'interface d'un service Web, c'est-à-dire les opérations qu'il expose, les messages échangés et les protocoles utilisés.
- Il facilite la communication entre les fournisseurs de services Web et les clients en définissant un contrat clair et standardisé.

Technologies associées

Introduction

Éléments clés du WSDL

1. Types de données :
   - Le WSDL permet de définir les types de données utilisés dans les messages échangés entre le client et le service Web.
   - Les types peuvent être des types de base XML (comme string, integer, etc.) ou des types complexes définis par l'utilisateur.
2. Opérations :
   - Les opérations décrivent les actions ou les fonctionnalités spécifiques fournies par le service Web.
   - Chaque opération spécifie les messages d'entrée et de sortie, ainsi que les types de données associés.
3. Messages :
   - Les messages représentent les données échangées entre le client et le service Web lors d'une opération.
   - Ils définissent la structure des données à l'aide de types définis dans la section "Types de données".
4. Ports et liaisons :
   - Les ports et les liaisons définissent les protocoles utilisés pour accéder au service Web, tels que HTTP, SOAP, etc.
   - Ils spécifient également les adresses et les points d'extrémité du service Web.

Technologies associées

Introduction

Structure d'un fichier WSDL

Un fichier WSDL est généralement composé de plusieurs éléments :

- Types : définissant les types de données utilisés dans les messages.
- Messages : décrivant la structure des données échangées.
- PortTypes : spécifiant les opérations disponibles et les messages associés.
- Bindings : définissant les protocoles et les formats de données utilisés pour l'accès au service Web.
- Services : décrivant les points de terminaison du service Web.

Types de données dans WSDL

- La section "Types" de WSDL permet de définir les types de données utilisés dans les messages échangés entre le client et le service Web.

- Les types peuvent être des types de base XML (comme string, integer, etc.) ou des types complexes définis par l'utilisateur.

Technologies associées

Introduction

Messages dans WSDL

- La section "Messages" de WSDL définit la structure des données échangées entre le client et le service Web lors d'une opération.
- Elle spécifie les éléments de données et leur ordre dans le message.

Opérations dans WSDL

- La section "PortTypes" de WSDL spécifie les opérations disponibles dans le service Web.
- Chaque opération est associée à des messages d'entrée et de sortie, définis dans la section "Messages".

Bindings dans WSDL

- La section "Bindings" de WSDL définit les protocoles et les formats de données utilisés pour accéder au service Web.
- Elle spécifie comment les messages sont encodés pour la transmission, tels que SOAP, HTTP, etc.

Services dans WSDL

- La section "Services" de WSDL décrit les points de terminaison du service Web.
- Elle spécifie les informations nécessaires pour accéder au service, telles que l'URL, les protocoles supportés, etc.

Technologies associées

Introduction

Endpoint

- L'Endpoint définit l'adresse physique (URL) et le protocole d'accès spécifique à un service Web.

- Il indique comment accéder au service et les détails de communication, tels que l'adresse, le protocole et le format de données.

Technologies associées

Introduction

Avantages du WSDL

- Le WSDL permet une documentation précise et formelle des services Web, facilitant ainsi leur découverte et leur utilisation.
- Il favorise l'interopérabilité en permettant aux clients de comprendre et d'interagir avec les services Web de manière standardisée.
- Il facilite le développement d'outils et de bibliothèques logicielles pour générer automatiquement des clients ou des stubs de services Web.

Récapitulatif

- Le WSDL est un langage basé sur XML utilisé pour décrire les services Web.
- Il définit l'interface d'un service Web, y compris les opérations, les messages, les types de données et les protocoles utilisés.
- Le WSDL facilite la communication entre les fournisseurs de services Web et les clients en fournissant une description formelle et standardisée.

Technologies associées

Introduction

Exemple de WSDL :

```
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
                  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
                  targetNamespace="http://example.com/your-api">
  <wsdl:types>
    <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" targetNamespace="http://example.com/your-api">
      <xs:element name="GetMessageRequest">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="userId" type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>

      <xs:element name="GetMessageResponse">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="message" type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:schema>
  </wsdl:types>

  <wsdl:message name="GetMessageRequest">
    <wsdl:part name="parameters" element="tns:GetMessageRequest"/>
  </wsdl:message>

  <wsdl:message name="GetMessageResponse">
    <wsdl:part name="parameters" element="tns:GetMessageResponse"/>
  </wsdl:message>

  <wsdl:portType name="YourAPIPortType">
    <wsdl:operation name="GetMessage">
      <wsdl:input message="tns:GetMessageRequest"/>
      <wsdl:output message="tns:GetMessageResponse"/>
    </wsdl:operation>
  </wsdl:portType>
```

Technologies associées

Introduction

Exemple de WSDL :

```
  <wsdl:binding name="YourAPIBinding" type="tns:YourAPIPortType">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="GetMessage">
      <soap:operation soapAction="http://example.com/your-api/GetMessage"/>
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>

  <wsdl:service name="YourAPIService">
    <wsdl:port name="YourAPIPort" binding="tns:YourAPIBinding">
      <soap:address location="http://example.com/your-api/endpoint"/>
    </wsdl:port>
  </wsdl:service>

</wsdl:definitions>
```

Technologies associées

Introduction

Exercice WSDL :

Rédaction d'un WSDL pour une API de gestion de produitsObjectif : Créer un WSDL décrivant une API de gestion de produits permettant de récupérer les détails d'un produit par son identifiant.

Instructions :

1. Définissez un service Web avec un nom, une adresse, et un port.
2. Définissez un type complexe pour représenter une requête de récupération de détails de produit avec les éléments suivants :
    - Identifiant du produit (type : entier)
3. Définissez un type complexe pour représenter une réponse de détails de produit avec les éléments suivants :
    - Identifiant du produit (type : entier)
    - Nom du produit (type : chaîne de caractères)
    - Prix du produit (type : décimal)
4. Définissez une opération dans le service pour récupérer les détails d'un produit avec les éléments suivants :
    - Nom de l'opération : getProductDetails
    - Entrée : requête de récupération de détails de produit (type : complexe défini à l'étape 2)
    - Sortie : réponse de détails de produit (type : complexe défini à l'étape 3)
- Utilisez le protocole SOAP pour définir le style et l'encodage de l'opération.

Technologies associées

Introduction

Exercice WSDL :

Voici une explication détaillée de chaque partie de la correction :

1/ Définitions du WSDL et des espaces de noms :

```
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
                  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
                  xmlns:tns="http://www.example.com/product-service"
                  targetNamespace="http://www.example.com/product-service">
```

Dans cette partie, nous définissons les espaces de noms utilisés dans le WSDL. Le targetNamespace spécifie l'URI du service Web défini par ce WSDL. Dans cet exemple, nous avons utilisé l'URI "http://www.example.com/product-service".

Technologies associées

Introduction

Exercice WSDL :

2/ Types de données :

```
<wsdl:types>
    <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
               targetNamespace="http://www.example.com/product-service"
               elementFormDefault="qualified">
        <xs:element name="ProductDetailsRequest">
            <xs:complexType>
                <xs:sequence>
                    <xs:element name="ProductId" type="xs:int"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
        <xs:element name="ProductDetailsResponse">
            <xs:complexType>
                <xs:sequence>
                    <xs:element name="ProductId" type="xs:int"/>
                    <xs:element name="ProductName" type="xs:string"/>
                    <xs:element name="ProductPrice" type="xs:decimal"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
    </xs:schema>
</wsdl:types>
```

Ici, nous définissons les types de données utilisés dans les requêtes et les réponses de l'API. Nous avons deux types de complexe, ProductDetailsRequest et ProductDetailsResponse.
ProductDetailsRequest contient un seul élément, ProductId, qui est de type entier (int).
ProductDetailsResponse contient trois éléments, ProductId (entier), ProductName (chaîne de caractères) et ProductPrice (décimal).

Technologies associées

Introduction

Exercice WSDL :

3/ Messages :

```
<wsdl:message name="ProductDetailsRequestMessage">
    <wsdl:part name="parameters" element="tns:ProductDetailsRequest"/>
</wsdl:message>
<wsdl:message name="ProductDetailsResponseMessage">
<wsdl:part name="parameters" element="tns:ProductDetailsResponse"/>
</wsdl:message>
```

Nous définissons ici les messages qui seront utilisés dans les opérations. Les messages décrivent la structure des données envoyées et reçues par les opérations. Dans notre cas, nous avons deux messages : ProductDetailsRequestMessage avec un élément ProductDetailsRequest et ProductDetailsResponseMessage avec un élément ProductDetailsResponse.

Technologies associées

Introduction

Exercice WSDL :

4/ Port Type :

```
<wsdl:portType name="ProductServicePortType">
    <wsdl:operation name="getProductDetails">
        <wsdl:input message="tns:ProductDetailsRequestMessage"/>
        <wsdl:output message="tns:ProductDetailsResponseMessage"/>
    </wsdl:operation>
</wsdl:portType>
```

Le port type définit les opérations disponibles dans le service Web. Dans notre cas, nous avons une opération appelée getProductDetails. Cette opération prend le message ProductDetailsRequestMessage en entrée et renvoie le message ProductDetailsResponseMessage en sortie.

Technologies associées

Introduction

Exercice WSDL :

5/ Binding :

```
<wsdl:binding name="ProductServiceBinding" type="tns:ProductServicePortType">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="getProductDetails">
        <soap:operation soapAction="http://www.example.com/product-service/getProductDetails"/>
        <wsdl:input>
            <soap:body use="literal"/>
        </wsdl:input>
        <wsdl:output>
            <soap:body use="literal"/>
        </wsdl:output>
    </wsdl:operation>
</wsdl:binding>
```

Le binding connecte le port type à un protocole de transport spécifique, dans ce cas, SOAP sur HTTP. Nous avons utilisé le style "document" pour décrire que les paramètres de la requête et de la réponse sont encapsulés dans le corps de la requête SOAP. Le soapAction indique l'URI de l'opération, qui est utilisée pour l'identification lors de l'appel de l'opération.

Technologies associées

Introduction

Exercice WSDL :

5/ Service :

```
<wsdl:service name="ProductService">
    <wsdl:port name="ProductServicePort" binding="tns:ProductServiceBinding">
        <soap:address location="http://www.example.com/product-service"/>
    </wsdl:port>
</wsdl:service>
```

Enfin, nous définissons le service avec un nom et un port. Le port est associé au binding ProductServiceBinding que nous avons défini précédemment. L'adresse indique l'URL à laquelle le service est disponible, dans ce cas "http://www.example.com/product-service".

Ce WSDL décrit une API simple de gestion de produits avec une opération pour récupérer les détails d'un produit par son identifiant

XML-RPC

Introduction

Introduction à XML-RPC

- XML-RPC est un protocole de communication basé sur XML (eXtensible Markup Language) utilisé pour appeler des fonctions ou des procédures à distance.

- Il permet à des applications de s'échanger des données et d'appeler des méthodes sur des systèmes distants via des requêtes XML.

XML-RPC

Introduction

Principes de fonctionnement de XML-RPC

1. Encodage des données :
   - Les données sont encodées en XML pour être échangées entre le client et le serveur.
   - Les types de données courants, tels que les chaînes de caractères, les entiers, les booléens, etc., sont supportés.

2. Appel de procédure à distance :
   - Le client envoie une requête XML-RPC au serveur pour appeler une procédure distante.
   - La requête contient le nom de la méthode et les paramètres nécessaires à l'appel.

3. Traitement de la requête :
   - Le serveur reçoit la requête XML-RPC et extrait les informations nécessaires pour exécuter la procédure demandée.
   - Les paramètres sont extraits de la requête XML et la méthode est invoquée.

4. Réponse de la procédure distante :
   - Le serveur exécute la procédure distante et renvoie une réponse au client sous la forme d'une réponse XML-RPC.
   - La réponse peut inclure des résultats, des messages d'erreur ou d'autres informations.

XML-RPC

Introduction

Avantages de XML-RPC

- Simplicité : XML-RPC utilise XML comme format de données, ce qui le rend facile à comprendre et à manipuler.
- Interopérabilité : XML-RPC est basé sur des standards ouverts, ce qui le rend compatible avec différentes plateformes et langages.
- Extensibilité : De nouveaux types de données peuvent être ajoutés au protocole en étendant la spécification XML-RPC.

Limitations de XML-RPC

- Performance : Comparé à d'autres protocoles plus légers comme REST, XML-RPC peut être plus lourd en raison de l'encodage XML et de la manipulation des données.
- Complexité des structures de données : La manipulation de structures de données complexes peut être plus difficile dans XML-RPC en raison des contraintes XML.

XML-RPC

Introduction

Principe de l'appel de méthodes distantes avec XML-RPC

1. Définition des méthodes :
   - Les méthodes à appeler à distance sont définies sur le serveur.
   - Chaque méthode expose une fonctionnalité spécifique et accepte des paramètres en entrée.
2. Encodage des données :
   - Les paramètres de la méthode à distance sont encodés en XML.
   - Les types de données courants, tels que les chaînes de caractères, les entiers, les booléens, etc., sont supportés.
3. Création de la requête XML-RPC :
   - Le client construit une requête XML-RPC contenant le nom de la méthode à appeler et les paramètres associés.
   - La requête est envoyée au serveur via un protocole de transport, tel que HTTP.
4. Traitement de la requête :
   - Le serveur reçoit la requête XML-RPC et extrait les informations nécessaires, telles que le nom de la méthode et les paramètres.
   - La méthode est invoquée avec les paramètres fournis.
5. Renvoi de la réponse :
   - Le serveur exécute la méthode et renvoie une réponse XML-RPC contenant les résultats de l'appel.
   - La réponse peut inclure des valeurs de retour, des messages d'erreur ou d'autres informations.

XML-RPC

Introduction

Exemple d'appel de méthode distante avec XML-RPC :

Exemple de requête XML-RPC :

```xml
<?xml version="1.0"?>
<methodCall>
  <methodName>addNumbers</methodName>
  <params>
    <param>
      <value>
        <int>5</int>
      </value>
    </param>
    <param>
      <value>
        <int>10</int>
      </value>
    </param>
  </params>
</methodCall>
```

Exemple de réponse XML-RPC :

```xml
<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
      <value>
        <int>15</int>
      </value>
    </param>
  </params>
</methodResponse>
```

UDDI

Introduction

Qu'est-ce que UDDI ?

- UDDI (Universal Description, Discovery, and Integration) est un protocole et un registre basé sur XML (eXtensible Markup Language) utilisé pour la publication, la découverte et l'intégration de services Web.

- Il fournit un annuaire centralisé où les fournisseurs de services Web peuvent enregistrer leurs services et les clients peuvent les découvrir.

Rôle de UDDI dans la publication et la découverte de services Web

•Publication de services Web :

- Les fournisseurs de services Web utilisent UDDI pour enregistrer les détails de leurs services, tels que les informations sur l'interface, les fonctionnalités et les protocoles de communication.
- Ils décrivent leurs services en utilisant des métadonnées structurées conformes au format UDDI.

•Découverte de services Web :

- Les clients utilisent UDDI pour rechercher des services Web disponibles répondant à leurs besoins.
- Ils peuvent effectuer des recherches basées sur des critères tels que les mots-clés, les catégories, les emplacements, etc.

UDDI

Introduction

Utilisation de UDDI pour enregistrer et rechercher des services Web

- Enregistrement de services Web :
  - Les fournisseurs utilisent UDDI pour créer des descriptions de services Web et les publier dans le registre UDDI.
  - Les descriptions de services contiennent des informations telles que les noms, les descriptions, les URL, les types de données, les protocoles de communication, etc.

- Recherche de services Web :
  - Les clients utilisent UDDI pour rechercher des services Web en spécifiant des critères de recherche pertinents.
  - Le registre UDDI renvoie les informations sur les services Web correspondants, y compris les détails d'accès et les informations de contact.

UDDI

Introduction

Avantages de UDDI

- Centralisation : UDDI offre un annuaire centralisé pour enregistrer et découvrir des services Web, ce qui facilite la gestion et la recherche des services.
- Interopérabilité : UDDI est basé sur des standards ouverts et facilite l'interopérabilité entre différents fournisseurs et clients de services Web.
- Découverte facilitée : UDDI permet aux clients de rechercher des services Web en utilisant des critères spécifiques, ce qui simplifie la découverte de services appropriés.

Limitations de UDDI

- Complexité : L'utilisation de UDDI peut être complexe en raison de la nécessité de décrire et de publier des services Web conformément aux spécifications UDDI.
- Dépendance au registre : La disponibilité et la fiabilité du registre UDDI sont cruciales pour la découverte et l'accès aux services Web, ce qui peut poser des problèmes en cas de défaillance du registre.

UDDI

Introduction

Récapitulatif :

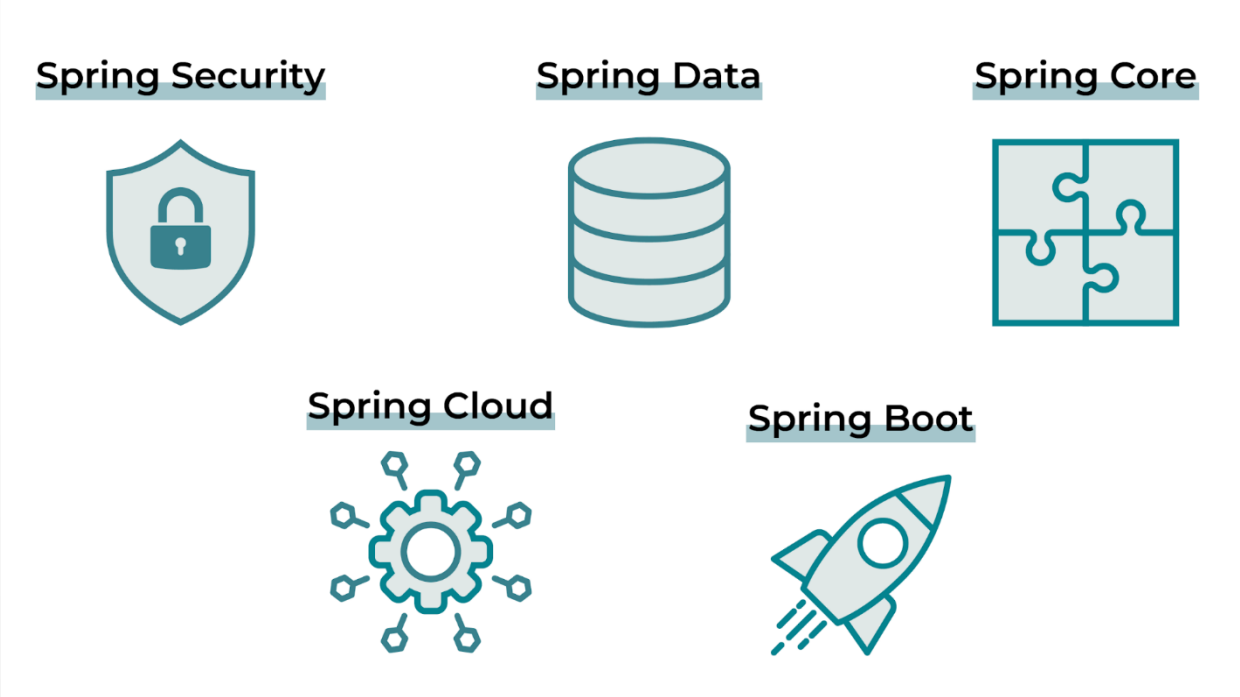
- UDDI est un protocole et un registre utilisé pour la publication, la découverte et l'intégration de services Web.
- Il permet aux fournisseurs de services Web d'enregistrer leurs services et aux clients de les découvrir en utilisant des critères de recherche.
- UDDI offre des avantages tels que la centralisation, l'interopérabilité et la facilitation de la découverte de services, mais peut présenter des limitations de complexité et de dépendance au registre.

Spring

Introduction

Introduction à Spring Boot :

- Spring Boot est un projet open-source développé par Pivotal Software (maintenant VMware).
- Il s'agit d'un framework Java qui simplifie le processus de configuration et de déploiement d'applications Spring.
- Spring Boot est basé sur Spring Framework, mais il fournit des fonctionnalités supplémentaires pour créer rapidement des applications autonomes et prêtes à l'emploi.



Spring

Introduction

Pourquoi utiliser Spring Boot ?

- Spring Boot facilite le démarrage rapide de nouveaux projets sans la nécessité d'une configuration fastidieuse.
- Il réduit le temps de développement en automatisant de nombreuses tâches courantes.
- Spring Boot intègre des serveurs embarqués comme Tomcat, Jetty ou Undertow, ce qui permet de créer des applications autonomes.

Principales fonctionnalités de Spring Boot :

1. Configuration automatique : Spring Boot configure automatiquement de nombreux composants en fonction des dépendances présentes dans le projet.
2. Serveurs embarqués : L'application peut être exécutée sans nécessiter de serveur d'application externe.
3. Actuator(Actionneur) : Fournit des fonctionnalités de supervision et de gestion de l'application.
4. Spring Data JPA : Intégration transparente avec la couche de persistance.
5. Annotations @RestController et @RequestMapping : Facilite la création de services RESTful.

Spring

Introduction

Création d'un projet Spring Boot

- Spring Initializr est un outil en ligne permettant de générer rapidement un projet Spring Boot.
- Vous pouvez sélectionner les dépendances souhaitées, comme Spring Web, Spring Data JPA, etc.
- Le projet généré contiendra une structure de base avec des classes principales préconfigurées.

Structure d'un projet Spring Boot

- Le dossier src/main/java contient les classes Java principales de l'application.
- Le dossier src/main/resources contient les fichiers de configuration, les propriétés et les ressources statiques.
- Le dossier src/test/java contient les classes de tests unitaires.
- Le fichier pom.xml contient les dépendances et la configuration du projet (si Maven est utilisé).

Classe principale de l'application

- Dans Spring Boot, l'application est lancée à partir d'une classe Java annotée avec @SpringBootApplication.
- Cette annotation regroupe plusieurs autres annotations, telles que @Configuration, @EnableAutoConfiguration et @ComponentScan.

Spring

Introduction

Création d'un contrôleur REST

- Un contrôleur REST traite les requêtes HTTP entrantes et renvoie des réponses au format JSON, XML, etc.
- Pour créer un contrôleur REST, il suffit de créer une classe annotée avec @RestController et de définir les méthodes pour les endpoints.

•**Exemple de contrôleur REST**

```java
@RestController
public class HelloWorldController {

    @GetMapping("/hello")
    public String sayHello() {
        return "Hello, World!";
    }
}
```

Exécution de l'application

- Pour exécuter l'application Spring Boot, vous pouvez utiliser la commande mvn spring-boot:run si vous utilisez Maven.
- Alternativement, vous pouvez également exécuter directement la classe principale de l'application à partir de votre IDE.

Synthèse :

- Spring Boot est un outil puissant pour simplifier le développement d'applications Java.
- Il facilite la création d'applications autonomes et prêtes à l'emploi.
- La configuration automatique, les serveurs embarqués et les fonctionnalités avancées font de Spring Boot un choix populaire pour le développement d'applications modernes.

Spring

Introduction

Exercice version TP :

Dans le cadre de cet exercice, vous allez devoir :

- Choisir un IDE
- Télécharger un SDK si ce n'est pas déjà fait minimum Java 11
- Installer Spring boot en utilisant Spring Initializr.

Le but de cet exercice est de créer une API en utilisant Spring Boot. Cette API devra répondre aux mêmes conditions que l'exercice 3 du premier cours.

Le livrable attendu est un zip contenant :

- Le projet de votre API en utilisant Spring boot
- Le projet de votre API avec JAX-RS
- Une capture d'écran de votre Postman pour chaque verbe du CRUD de votre API avec JAX-RS et en Spring boot.

Le livrable devra être rendu en septembre lors du prochain cours par mail à l'adresse : loic.dangla1@reseau-cd.net

Object du mail : Nom Prénom – Numéro classe
Nom du ZIP : NuméroClasseNomPrenom.zip

Fin du cours

EPSI l'École d'ingénierie informatique