

Module : Services Web

Version 1

Formateur :

DANGLA Loïc
loic.dangla1@reseau-cd.net

Planning du module

- 12 heures de cours :
 - **Séance 1** : 3 heures de cours et 1 heure d'exercice
 - **Séance 2** : 2 heures de cours et 2 heures de TP noté
 - **Séance 3** : 2,5 heures de cours et 1,5 heures de contrôle
- Deux notes + une note de participation
- Thèmes :
 - SOAP / REST / UDDI / WSDL / Authentification
 - SPRING
 - POSTMAN

Introduction

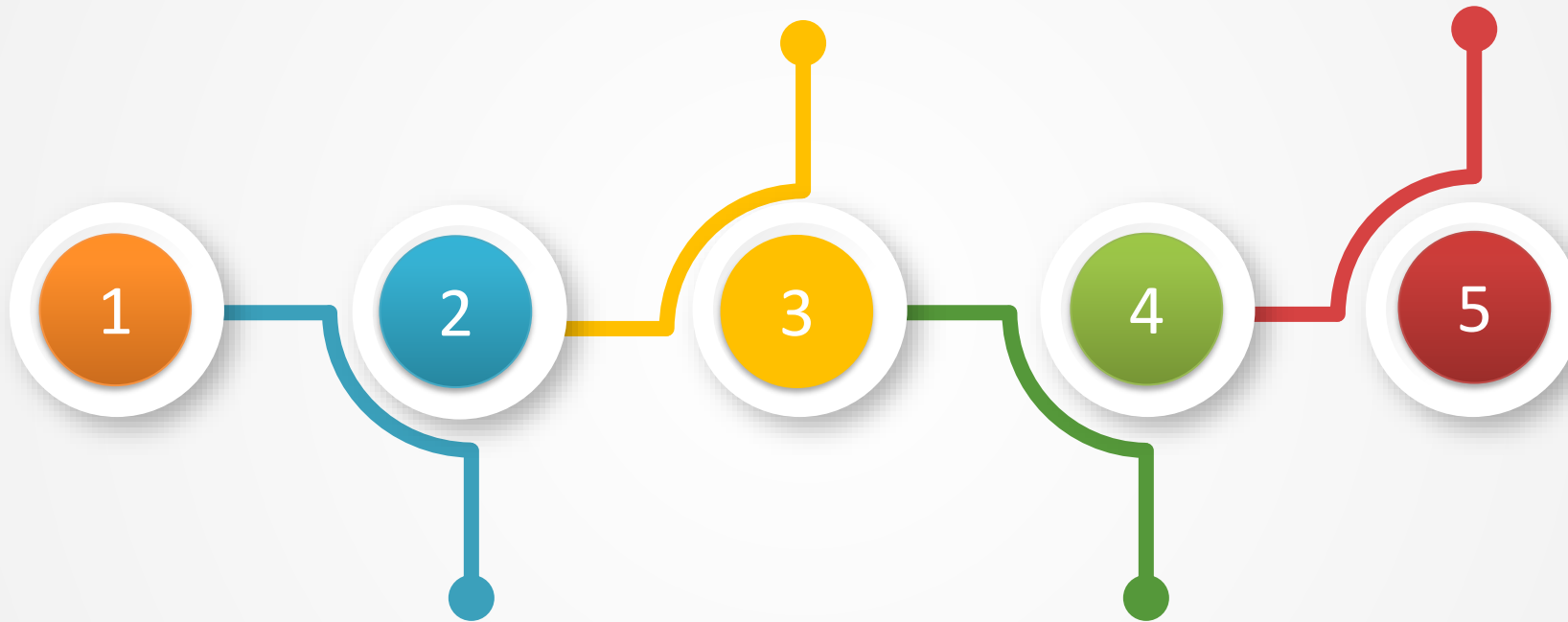
Histoire des API

REST

Type de format
Avantages

REST VS SOAP

Création d'un serveur REST



Fondamentaux

Règles générales
Formats

SOAP

Type de format
Avantages



Une API pour
vous, c'est ?



Un peu d'histoire

- API : Application Programming Interface
- Une API permet à un client d'exécuter une opération sur un serveur distant afin de garantir une interopérabilité entre le client et le serveur
- 1985 : Microsoft et la fédération d'une communauté technique autour des APIs avec les WinAPI
- 2000 : Salesforce et les premières APIs web (XML)
- Au début des années 2000, Roy Fielding a présenté l'architecture REST (Representational State Transfer) comme une alternative légère aux Services Web basés sur SOAP.
- REST a gagné en popularité grâce à sa simplicité, sa scalabilité et son adoption par des géants du Web comme Google et Twitter.

API

Introduction

- Les Services Web sont des composants logiciels permettant la communication entre applications distribuées.
- Ils facilitent l'échange de données et de fonctionnalités entre différentes plateformes et technologies.

Objectif :

- Fournir une architecture standardisée pour la communication entre applications.
- Promouvoir l'interopérabilité et l'intégration entre systèmes hétérogènes.
- Faciliter la réutilisation et la composition de services existants.

API

Introduction

1. Interopérabilité:

- Les Services Web permettent la communication entre différentes plateformes, langages de programmation et systèmes hétérogènes.
- Ils facilitent l'intégration de systèmes d'entreprise et la collaboration entre partenaires commerciaux.

2. Réutilisation:

- Les Services Web favorisent la réutilisation de fonctionnalités existantes.
- Ils permettent de composer des services pour créer de nouvelles applications ou fonctionnalités.

3. Découvrabilité:

- Les Services Web peuvent être découverts et utilisés dynamiquement grâce à des registres et des mécanismes de recherche.

4. Sécurité:

- Les Services Web offrent des mécanismes de sécurité pour protéger les données et les transactions.

API

Introduction

Caractéristiques :

- Les Services Web sont basés sur des standards ouverts tels que HTTP, XML, SOAP, etc.
- Ils sont indépendants du langage de programmation et de la plateforme.
- Ils sont accessibles via des protocoles réseau standard (HTTP, TCP/IP).
- Ils sont découvrables et peuvent être utilisés dynamiquement.

Principes :

1. Interface bien définie:

- Le contrat du service est spécifié via une description formelle (WSDL).
- Les opérations et les paramètres sont clairement définis.

2. Communication basée sur des protocoles standard:

- Utilisation de protocoles tels que HTTP, XML, SOAP pour la communication.

3. Découverte et annuaire:

- Les Services Web peuvent être découverts et localisés grâce à des registres (UDDI).

4. Interopérabilité:

- Les Services Web permettent la communication entre différentes plateformes et langages.

API

Introduction

Avantages

1. Intégration simplifiée:

- Les Services Web facilitent l'intégration de systèmes disparates.
- Ils permettent aux applications de communiquer et d'échanger des données de manière standardisée.

2. Plateforme indépendante:

- Les Services Web sont indépendants de la plateforme et du langage de programmation.
- Ils peuvent être utilisés avec n'importe quelle technologie qui supporte les protocoles Web standards.

3. Évolutivité:

- Les Services Web permettent de construire des systèmes évolutifs grâce à leur architecture distribuée.
- Ils peuvent être déployés et mis à l'échelle de manière flexible.

4. Facilité de maintenance:

- Les Services Web facilitent la maintenance et la mise à jour des applications.
- Les modifications apportées à un service Web n'affectent pas nécessairement les clients qui l'utilisent.

API

Introduction

Exemples d'utilisation des Services Web :

- Communication entre applications mobiles et serveurs.
- Échange de données entre partenaires commerciaux.
- Construction d'applications Web distribuées.

Récapitulatif :

- Les Services Web facilitent la communication entre applications distribuées.
- Ils sont basés sur des standards ouverts et permettent l'interopérabilité.
- Ils offrent une interface bien définie et sont découvrables via des registres.
- Ils sont utilisés dans divers scénarios, de l'intégration d'entreprise à l'échange de données entre partenaires.
- Les Services Web offrent une architecture standardisée pour la communication entre applications distribuées.
- Leurs objectifs incluent l'interopérabilité, la réutilisation, la découvrabilité et la sécurité.
- Les avantages des Services Web comprennent l'intégration simplifiée, la plateforme indépendante, l'évolutivité et la facilité de maintenance.

API

Fondamentaux communication

- La communication entre les applications est essentielle pour l'échange de données et de fonctionnalités entre différents systèmes.
- Les principes fondamentaux de la communication assurent une interaction efficace et fiable entre les applications.

Protocoles de communication :

- Les protocoles de communication sont des ensembles de règles et de conventions qui définissent le format et la séquence des messages échangés entre les applications.
- Exemples de protocoles : HTTP (HyperText Transfer Protocol), TCP/IP (Transmission Control Protocol/Internet Protocol).

Formats de données :

- Les formats de données définissent la structure et la représentation des informations échangées entre les applications.
- Exemples de formats : XML (eXtensible Markup Language), JSON (JavaScript Object Notation).

API

Fondamentaux communication

- Exemple de JSON :

```
{  
  "name": "discord-screenshot",  
  "version": "1.3.7",  
  "description": "A resource for FiveM that captures the screen.",  
  "scripts": {  
    "dev": "webpack --watch --mode development",  
    "build": "webpack"  
  },  
  "author": "Jaime Filho",  
  "license": "MIT"  
}
```

- les clés en violet avant les « : » (Ce sont toujours des chaînes de caractères)
- les valeurs en vert, de l'autre côté des « : » (Elles peuvent être de plusieurs types, des chaînes de caractères, mais aussi des nombres, des tableaux, des objets...)

À l'instar de XML, JSON est **indépendant des langages**, et peut se combiner avec nombre de ces derniers, dont C++, Java, Python.

API

Fondamentaux communication

Toutefois, contrairement à XML, JSON n'est qu'un **mode de représentation** des structures de données, par opposition à un **langage de marquage intégral**.

- Exemple de XML :

```
<NAME>discord-screenshot</NAME>  
<AUTHOR>Jaime Filho</AUTHOR>
```

Les balises `<NAME>` ou `<AUTHOR>` permettent de délimiter les informations correspondant au nom et à l'auteur.

- `<NAME>` annonce le début des informations concernant le nom.
- `</NAME>` annonce la fin des informations concernant le nom (grâce au `/`).

API

Fondamentaux communication

Exercice 1 :

Dans cet exercice, vous allez devoir créer un fichier JSON et un fichier XML en utilisant un scénario spécifique. Vous devrez utiliser les connaissances acquises sur la structure de données JSON et XML ainsi que sur la syntaxe correspondante.

Instructions :

Supposons que vous gérez une liste d'étudiants dans une université. Chaque étudiant a un nom, un numéro d'étudiant unique, une adresse e-mail et une liste de matières dans lesquelles il est inscrit.

- Partie 1 : Fichier JSON
 - Vous devez créer un fichier JSON représentant les informations d'un étudiant:
 - Nom : Votre nom
 - Prénom : Votre prénom
 - Numéro Etudiant : Votre numéro étudiant
 - Email : Votre email EPSI
 - Ecole : Le nom de votre école
 - Matières de la journée : Liste des matières de la journée
- Partie 2 : Fichier XML
 - Vous devez créer un fichier XML représentant les mêmes informations que le fichier JSON.

API

Fondamentaux communication

Exercice 1 :

```
{  
  "nom": "Dangla",  
  "prenom": "Loïc",  
  "numeroEtudiant": "12345678910",  
  "email": "loic.dangla1@reseau-cd.net",  
  "ecole": "EPSI",  
  "matieres": ["API", "JAVA"]  
}
```

```
<etudiant>  
  <nom>Dangla</nom>  
  <prenom>Loïc</prenom>  
  <numeroEtudiant>12345678910</numeroEtudiant>  
  <email>loic.dangla1@reseau-cd.net</email>  
  <ecole>EPSI</ecole>  
  <matieres>  
    <matiere>API</matiere>  
    <matiere>JAVA</matiere>  
  </matieres>  
</etudiant>
```

API

Fondamentaux communication

Architecture orientée services (SOA) :

- SOA est un paradigme qui favorise la communication entre applications via des services autonomes et réutilisables.
- Les services SOA sont des composants logiciels qui fournissent des fonctionnalités spécifiques et sont accessibles via des interfaces standardisées

Communication synchrone et asynchrone :

- La communication synchrone implique une attente active de la réponse de l'application distante avant de poursuivre l'exécution.
- La communication asynchrone permet à l'application d'envoyer une requête et de continuer son exécution sans attendre de réponse immédiate.

API

Fondamentaux communication

Protocoles basés sur les messages :

- Certains protocoles de communication, tels que SOAP (Simple Object Access Protocol), sont basés sur l'échange de messages structurés.
- Les messages SOAP encapsulent les données et les métadonnées nécessaires à la communication entre les applications.

Sécurité de la communication :

- La sécurité de la communication entre les applications est essentielle pour protéger les données sensibles et prévenir les attaques.
- Les mécanismes de sécurité incluent l'authentification, le chiffrement, les certificats numériques, etc.

API

Fondamentaux communication

Récapitulatif :

Les principes fondamentaux de la communication entre les applications reposent sur :

- les protocoles de communication
- les formats de données
- l'architecture SOA
- les modes de communication synchrone et asynchrone
- les protocoles basés sur les messages
- la sécurité.

API

REST

Introduction à REST (Representational State Transfer)

- REST est un style d'architecture pour la création de services Web.
- Il repose sur des principes fondamentaux pour la communication entre les clients et les serveurs.

Concepts clés de l'architecture REST

1. Ressources :

- Les ressources représentent les entités (données ou fonctionnalités) qui sont exposées via l'API REST.
- Chaque ressource est identifiée de manière unique par une URI (Uniform Resource Identifier).

2. Verbes HTTP :

- REST utilise les verbes HTTP (GET, POST, PUT, DELETE) pour définir les opérations à effectuer sur les ressources.
- Chaque verbe a une signification spécifique : GET pour récupérer, POST pour créer, PUT pour mettre à jour, DELETE pour supprimer.

3. Stateless (Sans état) :

- Le serveur ne garde pas d'informations sur l'état du client entre les requêtes.
- Chaque requête contient toutes les informations nécessaires pour la réponse.

API

REST

Exercice 2 :

Dans cet exercice, vous allez devoir télécharger un logiciel comme Postman pour pouvoir échanger avec API.

Le Swagger : <https://restful-booker.herokuapp.com/apidoc/index.html>

URI : <https://restful-booker.herokuapp.com/booking>

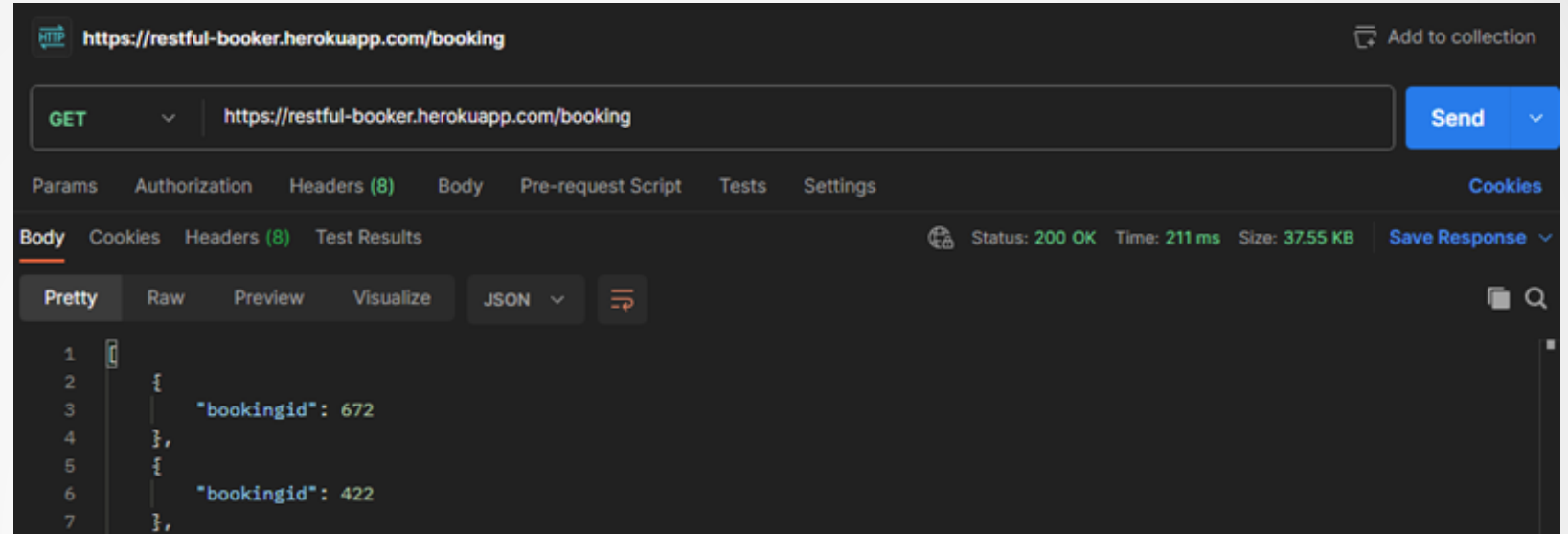
URI pour le DELETE : <https://restful-booker.herokuapp.com/booking/:idDuPost>

- Réaliser un appel à l'API ci-dessous en GET sans ID pour afficher toutes les réservations et ensuite avec un ID pour consulter le détail d'une réservation.
- Une fois l'action réalisée, vous allez devoir supprimer cette réservation en utilisant DELETE et l'ID de la réservation. Pour réaliser cette action, on va devoir renseigner dans le header le paramètre ci-dessous :
 - **Clé** : « Authorization »
 - **Valeur** : « Basic YWRtaW46cGFzc3dvcmQxMjM= »

API

REST

Exercise 2 :



https://restful-booker.herokuapp.com/booking

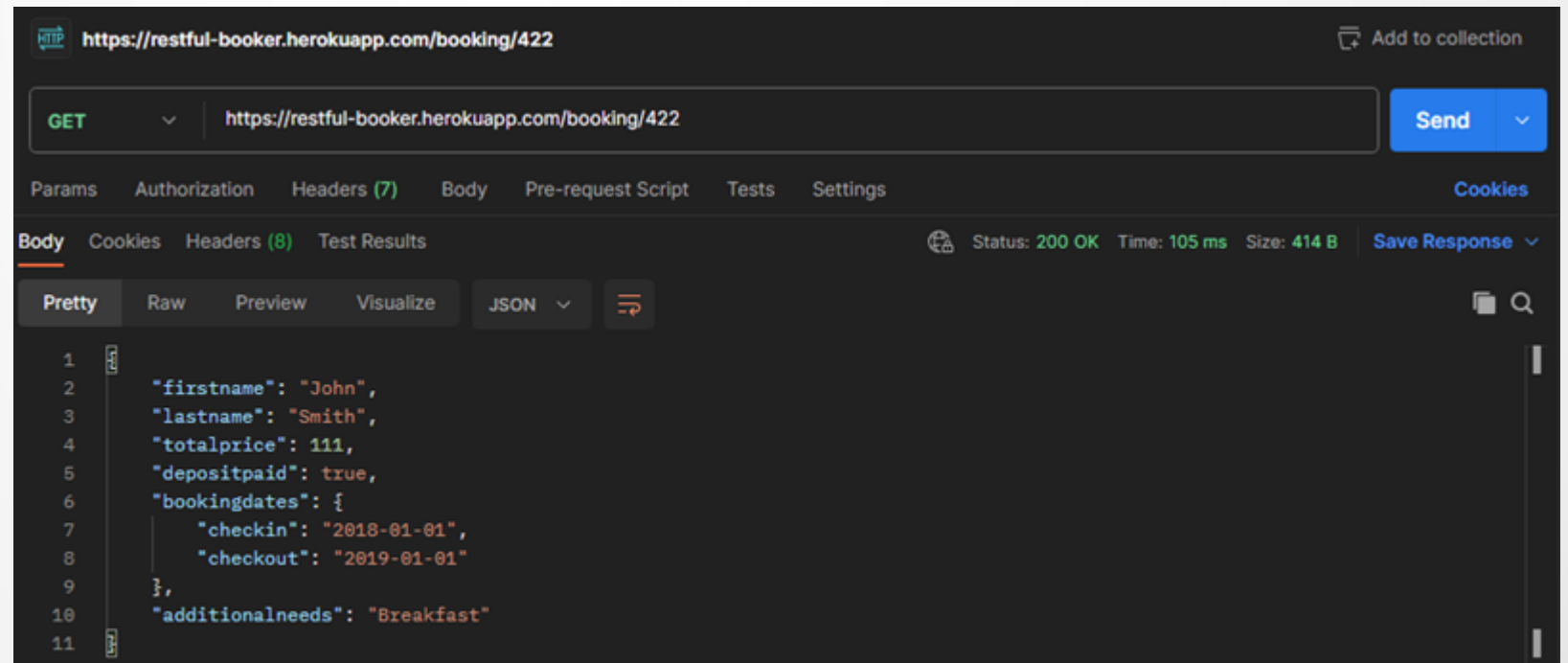
GET Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

Body Cookies Headers (8) Test Results Status: 200 OK Time: 211 ms Size: 37.55 KB Save Response

Pretty Raw Preview Visualize JSON

```
1 [
2   {
3     "bookingid": 672
4   },
5   {
6     "bookingid": 422
7   },
8 ]
```



https://restful-booker.herokuapp.com/booking/422

GET Send

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies

Body Cookies Headers (8) Test Results Status: 200 OK Time: 105 ms Size: 414 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "firstname": "John",
3   "lastname": "Smith",
4   "totalprice": 111,
5   "depositpaid": true,
6   "bookingdates": {
7     "checkin": "2018-01-01",
8     "checkout": "2019-01-01"
9   },
10  "additionalneeds": "Breakfast"
11 }
```

API

REST

Exercise 2 :

The screenshot shows a REST client interface with the following details:

- URL:** `https://restful-booker.herokuapp.com/booking/422`
- Method:** `DELETE`
- Headers (7):**
 - ☒ **Authorization**: `Basic YWRtaW46cGFzc3dvcmQxMjM=`
 - 6 hidden headers
- Body:** `Created`
- Status:** `201 Created`
- Time:** `408 ms`
- Size:** `248 B`

Key	Value	Bulk Edit
<input checked="" type="checkbox"/> Authorization	Basic YWRtaW46cGFzc3dvcmQxMjM=	

API

REST

Utilisation du format JSON pour les données :

- JSON (JavaScript Object Notation) est un format léger et lisible par les humains pour l'échange de données.
- REST utilise généralement JSON pour représenter et transférer les données entre les clients et les serveurs.
- JSON est basé sur des paires clé-valeur et prend en charge des structures de données complexes.

Mais pourquoi utiliser du JSON pour mon API REST ?

- XML est plus difficile à analyser que JSON : il est nécessaire d'avoir un analyseur XML. Alors que JSON peut être analysé dans un objet JavaScript standard.
- JSON est plus rapide car étant moins verbeux, son analyse est plus simple.

API

REST

Les codes de retour des verbes HTTP :

Lorsque vous envoyez une requête au serveur, le serveur vous renvoie une réponse avec un code d'état retour HTTP qui n'est pas visible dans votre navigateur. Cette réponse peut être positive ou non. C'est pourquoi le protocole HTTP définit quarante codes d'états standard pouvant être utilisés pour transmettre les résultats de la demande d'un client.

Chaque code est composé de 3 chiffres.

Ces codes sont répartis dans les cinq catégories suivantes :

- **1xx : informatif** : communique une information au client,
- **2xx : succès** : indique que la demande du client a été acceptée avec succès,
- **3xx : redirection** : indique que le client doit prendre des mesures supplémentaires pour terminer sa demande,
- **4xx : erreur du client** : indique une erreur de la part du client,
- **5xx : erreur du serveur** : indique une erreur de la part du serveur.

API

REST

Avantages de l'approche REST :

1.Simplicité :

- REST est simple à comprendre et à implémenter.
- Il utilise les fonctionnalités standard du protocole HTTP.

2.Scalabilité :

- L'architecture REST permet une évolutivité horizontale grâce à la séparation des ressources et l'absence d'état.

3.Interopérabilité :

- REST favorise l'interopérabilité entre différentes plateformes et langages.
- Il est basé sur des standards ouverts et largement adoptés.

4.Réutilisation :

- Les ressources REST peuvent être réutilisées dans différentes applications ou services.

API

REST

Limitations de l'approche REST

1. Complexité des opérations complexes :

- REST est plus adapté aux opérations de base (CRUD : Create, Read, Update, Delete).
- Les opérations plus complexes peuvent nécessiter des requêtes supplémentaires ou des conventions spécifiques.

2. Manque de standardisation :

- Bien qu'il soit basé sur des standards, REST n'impose pas de conventions strictes.
- Cela peut conduire à une variation d'implémentation entre différents services REST.

API

REST

Récapitulatif

- REST est un style d'architecture pour la création de services Web basé sur des principes fondamentaux.
- Il utilise les ressources, les verbes HTTP et l'absence d'état pour la communication entre les clients et les serveurs.
- REST utilise souvent le format JSON pour représenter les données.
- Ses avantages incluent la simplicité, la scalabilité, l'interopérabilité et la réutilisation, tandis que ses limitations résident dans la complexité des opérations complexes et le manque de standardisation.

API

SOAP

Introduction à SOAP (Simple Object Access Protocol)

- SOAP est un protocole de communication basé sur XML pour échanger des données structurées entre les applications distribuées.
- Il n'est pas un style d'architecture, c'est un protocole de communication basé exclusivement sur XML pour permettre aux applications de s'échanger des informations via HTTP.

Concepts clés de l'architecture SOAP

1. Enveloppe SOAP :

- L'enveloppe SOAP encapsule le message et fournit des informations sur son contenu et sa structure.
- Il définit les éléments tels que l'en-tête, le corps et les éventuelles informations de sécurité.

2. Utilisation de XML pour les données :

- SOAP utilise le format XML (eXtensible Markup Language) pour représenter les données échangées entre les applications.
- XML permet de structurer les données de manière hiérarchique et de les décrire à l'aide de balises.

3. Protocoles de transport :

- SOAP peut être utilisé avec différents protocoles de transport tels que HTTP, SMTP, FTP, etc.
- La plupart des implémentations SOAP utilisent le protocole HTTP pour la communication.

API

SOAP

Exemple :

Endpoint de l'API :

<https://api.example.com/users>

```
<Request>
  <Action>GetUsers</Action>
  <Parameters>
    <Param1>Value1</Param1>
    <Param2>Value2</Param2>
  </Parameters>
</Request>
```

```
<Response>
  <Status>Success</Status>
  <Users>
    <User>
      <Name>John Doe</Name>
      <Age>25</Age>
      <Email>john@example.com</Email>
    </User>
    <User>
      <Name>Jane Smith</Name>
      <Age>30</Age>
      <Email>jane@example.com</Email>
    </User>
  </Users>
</Response>
```

Dans cet exemple, l'appel API est effectué en envoyant une requête XML à l'endpoint <https://api.example.com/users>. La requête contient une action spécifique (ici, "GetUsers") et des paramètres optionnels (Param1 et Param2).

La réponse de l'API est également en format XML. Elle contient un élément <Response> avec un statut de réussite, suivi d'une liste d'utilisateurs <User>. Chaque utilisateur est représenté par un ensemble d'informations telles que le nom, l'âge et l'email.

API

SOAP

Avantages de l'approche SOAP

1. Interopérabilité :

- SOAP favorise l'interopérabilité entre différentes plateformes et langages.
- Il est basé sur des standards ouverts et largement adoptés.

2. Flexibilité :

- SOAP permet de définir des structures de données complexes grâce à XML.
- Il prend en charge les types de données personnalisés et la description des services via WSDL (Web Services Description Language).

3. Sécurité :

- SOAP offre des fonctionnalités de sécurité avancées, telles que la confidentialité et l'authentification des messages.
- Il permet le chiffrement et la signature numérique des données échangées.

API

SOAP

Limitations de l'approche SOAP

1.Complexité :

- L'utilisation de XML et la structure complexe des messages SOAP peuvent rendre la mise en œuvre et le traitement plus complexes.

2.Performance :

- Comparé à d'autres approches légères telles que REST, SOAP peut être plus gourmand en ressources et avoir des temps de latence plus élevés.

3.Coût de développement :

- La mise en place et la maintenance d'une infrastructure SOAP peuvent nécessiter des ressources et des compétences plus importantes.

Récapitulatif :

- SOAP est un protocole de communication basé sur XML pour l'échange de données entre les applications distribuées.
- Il utilise une enveloppe SOAP pour encapsuler les messages et permet une communication interopérable et sécurisée.
- L'utilisation d'XML offre une grande flexibilité, mais peut entraîner une complexité et des coûts de performance plus élevée.

API

SOAP vs REST

Ok mais on doit choisir SOAP ou REST ?

- Les API SOAP (Simple Object Access Protocol) et REST (Representational State Transfer) sont deux approches pour la création d'API (Application Programming Interface) dans les services Web.
- Chaque approche a ses propres caractéristiques et convient à des cas d'utilisation spécifiques.

API SOAP :

- API SOAP utilise XML (eXtensible Markup Language) comme format de données pour les échanges entre le client et le serveur.
- Il repose sur des normes et des contrats stricts définis dans le WSDL (Web Services Description Language).
- Les appels API SOAP sont généralement effectués via des protocoles tels que HTTP, SMTP ou JMS.

API

SOAP vs REST

API REST :

- API REST utilise des verbes HTTP (GET, POST, PUT, DELETE) pour définir les opérations sur les ressources.
- Les échanges de données se font généralement au format JSON (JavaScript Object Notation) ou XML.
- REST suit les principes d'architecture RESTful, tels que l'utilisation d'URLs pour identifier les ressources et le respect du protocole HTTP.

Différences clés :

- API SOAP utilise XML pour les données, tandis que REST permet l'utilisation de différents formats de données, comme JSON ou XML.
- SOAP repose sur des contrats WSDL pour définir les services, tandis que REST utilise une approche plus légère avec des conventions d'URL.
- SOAP est plus lourd et complexe en termes de traitement et de configuration, tandis que REST est plus simple et plus flexible.

API

SOAP vs REST

Choix approprié :

- API SOAP convient aux scénarios nécessitant une sécurité élevée, une transactionnalité complexe et une gestion fine des erreurs.
- REST est recommandé pour des scénarios où la simplicité, la scalabilité et l'interopérabilité sont essentielles, tels que les applications mobiles et les API publiques.

Autres facteurs à considérer :

- Écosystème et compatibilité avec les technologies existantes.
- Niveau de compétence et de familiarité de l'équipe de développement.
- Besoins spécifiques de l'application, tels que la performance, la flexibilité ou la sécurité.

API

SOAP vs REST

Récapitulatif :

- API SOAP et REST sont deux approches différentes pour créer des API dans les services Web.
- SOAP utilise XML, WSDL et des protocoles divers, tandis que REST se base sur HTTP, des verbes et des formats de données variés.
- Le choix approprié dépend des exigences spécifiques de l'application et des facteurs tels que la sécurité, la simplicité et l'interopérabilité.

API

JAX-RS

Comment créer une API REST de base :

Dans un premier temps, il faut créer un projet JAVA avec :

- Une classe Main qui va exécuter le code ci-joint par exemple, ce code est générique et peut varier en fonction des IDE utilisés, il permet d'initialiser le serveur.



Main.java

- Une classe contenant vos API REST :

1/ La librairie « jakarta.ws.rs » qui permet de fournir des annotations, des interfaces et des classes pour simplifier le développement de services web RESTful en utilisant Java. Elle permet aux développeurs de définir des points de terminaison RESTful, de gérer les requêtes et les réponses, de manipuler les données JSON ou XML, de gérer les exceptions, d'effectuer des validations et bien plus encore

```
import jakarta.ws.rs.*;  
import jakarta.ws.rs.core.MediaType;  
import jakarta.ws.rs.core.Response;
```

API

JAX-RS

2/ Indiquer le path et créer la classe pour ajouter les différentes fonctionnalités

```
no usages
@Path("/messages")
public class MessageResource {
```

3/ Création de la fonction GET

- **@GET** indique le type de verbe
- **@Produces** : Cette annotation est utilisée pour spécifier le type de média que le point de terminaison peut produire

```
@GET
@Produces(MediaType.TEXT_PLAIN)
public String getHelloMessage() {
    // Ajouter le code que l'on souhaite réaliser
    return "Bonjour, bienvenue dans notre API REST !";
}
```

API

JAX-RS

4/ En complément des éléments déjà analysés précédemment :

- **@Path** : Cette annotation est utilisée pour définir le chemin relatif du point de terminaison dans l'API. Elle indique l'URL à laquelle le point de terminaison sera accessible

- **@Consumes** : Cette annotation est utilisée pour spécifier le type de média que le point de terminaison peut consommer

```
no usages
@POST
@Consumes(MediaType.TEXT_PLAIN)
public Response addMessage(String message) {
    // Logique pour ajouter le message
    // Ajouter le code que l'on souhaite réaliser
    return Response.status(Response.Status.CREATED).build();
}

no usages
@PUT
@Path("/{id}")
@Consumes(MediaType.TEXT_PLAIN)
public Response updateMessage(@PathParam("id") String id, String message) {
    // Logique pour mettre à jour le message avec l'ID spécifié
    // Ajouter le code que l'on souhaite réaliser
    return Response.ok().build();
}

no usages
@DELETE
@Path("/{id}")
public Response deleteMessage(@PathParam("id") String id) {
    // Logique pour supprimer le message avec l'ID spécifié
    // Ajouter le code que l'on souhaite réaliser
    return Response.noContent().build();
}
```

API

JAX-RS

Félicitation, vous venez de créer votre première API :

Pour tester votre API, vous pouvez utiliser un client REST comme Postman.

Les URLs correspondantes aux différentes opérations sont :

- GET : `http://localhost:8080/messages`
- POST : `http://localhost:8080/messages` (avec le corps de la requête contenant le message)
- PUT : `http://localhost:8080/messages/{id}` (remplacez {id} par l'ID du message à mettre à jour)
- DELETE : `http://localhost:8080/messages/{id}` (remplacez {id} par l'ID du message à supprimer)

API

JAX-RS

Exercice 3 :

Dans cet exercice, vous allez créer une API RESTful en utilisant JAX-RS en Java. Votre API devra prendre en charge les opérations GET, POST, PUT et DELETE pour manipuler des données d'une ressource spécifique.

Instructions :

- Créez un projet Java vide dans votre environnement de développement préféré (Eclipse, IntelliJ, etc.).
- Ajoutez les dépendances nécessaires à votre projet pour utiliser JAX-RS. Vous pouvez les spécifier manuellement ou utiliser un gestionnaire de dépendances tel que Maven ou Gradle.
- Définissez une classe principale pour démarrer votre application et configurer le serveur HTTP.
- Créez une classe de ressource pour représenter votre entité ou votre ressource (par exemple, "Product", "User", "Book", etc.).

Dans la classe de ressource, utilisez les annotations JAX-RS pour définir les méthodes et les chemins d'accès correspondant aux opérations REST :

- Pour l'opération GET, utilisez l'annotation @GET avec le chemin d'accès approprié.
- Pour l'opération POST, utilisez l'annotation @POST avec le chemin d'accès approprié avec 2 éléments en entrée.
- Pour l'opération PUT, utilisez l'annotation @PUT avec le chemin d'accès approprié.
- Pour l'opération DELETE, utilisez l'annotation @DELETE avec le chemin d'accès approprié.

Implémentez le corps des méthodes pour réaliser les actions correspondantes :

- Pour l'opération GET, retournez le message « GET ».
- Pour l'opération POST, retournez le message « POST » si toutes les données sont renseignées .
- Pour l'opération PUT, retournez le message « PUT » si au moins une donnée est transmise.
- Pour l'opération DELETE, retournez le message « DELETE » si la donnée est présente pour une suppression.

Testez votre API en utilisant un client REST tel que Postman. Vérifiez que les différentes opérations fonctionnent correctement en renvoyant les réponses attendues.



EPSSI

l'École
d'ingénierie
informatique

Fin du cours