UNIVERSITÉ
CAEN
NORMANDIE

Introduction aux API

avec Symfony

Introduction

- **Qu'est-ce qu'une API ?**
  - Interface de programmation applicative
  - Pont entre différents logiciels
- **Importance des API**
  - Communication entre applications
  - Fondamentaux du développement web moderne

Pourquoi une API ?

- **Flexibilité et évolutivité**
  - Facilite l'intégration avec d'autres services et applications
- **Accessibilité**
  - Permet l'accès aux fonctionnalités et aux données à partir de divers clients
- **Séparation des préoccupations**
  - Sépare la logique métier de l'interface utilisateur

Modèle de Maturité de Richardson

- **Niveau 0 : POX (Plain Old XML)**
  - Un seul URI, une seule méthode
- **Niveau 1 : Ressources**
  - Plusieurs URI, une seule méthode
- **Niveau 2 : HTTP Verbs**
  - Utilisation des méthodes HTTP
- **Niveau 3 : HATEOAS (Hypermedia As The Engine Of Application State)**
  - Interaction avec l'API guidée par hypermedia

Types d'API

- **API REST**
  - Basée sur HTTP, utilise des standards web
- **API SOAP**
  - Protocole standard, sécurité élevée
- **API GraphQL**
  - Requêtes flexibles, obtention de données spécifiques

Les Fondamentaux des API

- **Définition d'une API**
  - Interface pour accéder à un ensemble de fonctions
- **Types d'API**
  - REST, SOAP, GraphQL
- **Méthodes HTTP**
  - GET, POST, PUT, DELETE

Fonctionnement d'une API

- **Interface entre systèmes**
  - Permet aux applications de communiquer entre elles
- **Échange de données**
  - Envoi et réception de requêtes et réponses
- **Formats standards**
  - JSON, XML pour la structuration des données

Architecture RESTful

- **Représentation de l'état des ressources**
  - Les ressources sont des entités clés
- **Sans état**
  - Chaque requête contient toutes les informations nécessaires
- **Cacheable**
  - Les réponses peuvent être mises en cache pour améliorer la performance

Principe clé de REST

- **Uniform Interface**
  - Interface standardisée pour interagir avec les ressources
- **Client-Server**
  - Séparation des responsabilités entre client et serveur
- **Layered System**
  - Architecture en couches pour la scalabilité

Méthodes HTTP/HTTPS

- **GET**
  - Récupérer des informations
- **POST**
  - Créer une nouvelle ressource
- **PUT/PATCH**
  - Mettre à jour des ressources
- **DELETE**
  - Supprimer des ressources

HTTP et Codes de Retour

- **200 OK**
  - Réponse standard pour les requêtes réussies
- **201 Created**
  - Ressource créée avec succès
- **400 Bad Request**
  - Erreur dans la requête du client
- **404 Not Found**
  - Ressource non trouvée
- **500 Internal Server Error**

Formats de Données

- **XML**
  - Extensible Markup Language, structuré comme HTML
- **JSON**
  - JavaScript Object Notation, léger et facile à lire
- **YAML**
  - YAML Ain't Markup Language, utilisé pour la configuration

Structure Visuelle des Endpoints

- **Endpoint**
  - Point d'accès unique à une ressource
- **URI Structure**
  - `/api/resource`
  - `/api/resource/{id}`
- **Consistance**
  - Même structure pour toutes les ressources

Créer une API Simple avec Symfony

- **Installation et configuration**
  - Composer, Symfony CLI
- **Création d'une API REST simple**
  - Définir les routes dans `config/routes.yaml`
  - Créer un contrôleur dans `src/Controller/`
  - Retourner des données JSON
- **Doctrine pour la base de données**
  - Entités, repository, ORM

Exemple: API REST Simple

```php
// src/Controller/Api/PostController.php
#[Route('/api/posts')]
class PostController extends AbstractController {
    #[Route('/', name: 'api_post_index', methods: ['GET'])]
    public function index(): JsonResponse {
        // Récupérer et retourner les données
    }
}
```

Sécurisation de l'API

- **Authentification et autorisation**
  - Concepts clés
- **JWT (JSON Web Tokens)**
  - Création et validation de JWT
- **Exemple de mise en place**
  - LexikJWTAuthenticationBundle

JWT (JSON Web Tokens)

- **Authentification et autorisation**
  - Token qui contient des claims (informations de l'utilisateur)
- **Structure**
  - En-tête, charge utile, signature

Exemple : JWT

```
{
  "alg": "HS256",
  "typ": "JWT"
}
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret
)
```

Exemple : JWT

- **Utilisation**
  - Envoyé dans l'en-tête Authorization pour les requêtes sécurisées

Bonnes Pratiques et Maintenance

- **Documentation de l'API**
  - Swagger, NelmioApiDoc
- **Tests unitaires et fonctionnels**
  - PHPUnit, Behat
- **Déploiement et maintenance**
  - Environnements, CI/CD

Cas Pratique

- **Projet API avec Symfony**
  - Analyse, conception, implémentation

Conclusion

- **Récapitulatif**
  - Points clés abordés
- **Importance des API**
  - Dans le paysage web actuel

Ressources et Références

- **Documentation Symfony**
  - Symfony Official Website