Formation Module 6 Partie - 2

Ihab ABADI / UTOPIOS

Flask – Flask-SQLAlchemy

- Flask-SQLAlchemy est une librairies qui permet de simplifier les interactions entre notre application flask et les base de données.
- Flask-SQLAchemy permet une utilisation et une intégration plus facile de SQLAlchemy
- SQLAchemy permet de créer un objet pour interagir avec la base de données à partir de notre application.
- Lien doc :
- https://flask-sqlalchemy.palletsprojects.com/en/2.x/config/

```
from flask import Flask
from flask_sqlalchemy import SQLAlchemy
app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] =
'sqlite:////tmp/test.db'
db = SQLAlchemy(app)
```

Flask – Flask-SQLAlchemy

- Flask-SQLAchemy offre :
  - Une classe Model.
  - Une classe Column
  - Une session
  - Des méthodes pour les jointures (relationship)
  - Des classe pour les clés
  - Démo

Flask – Flask-SQLAlchemy

- Flask-SQLAchemy offre :
  - Un objet query pour exécuter des requêtes
  - La possibilité d'appliquer des filtres
  - …
  - Démo

Flask – Flask-SQLAlchemy – Démo – initialisation

- L'initialisation de la db se fait à l'aide de l'objet SQLAlchemy
- La création des tables peut se faire à l'aide de la méthode create_all à la première requête à l'aide du hook flask app.before_first_request

```
from flask_sqlalchemy import SQLAlchemy

db = SQLAlchemy()



@app.before_first_request
def initialisation():
    db.init_app(app)
    ##Création des tables
    db.create_all()
```

Flask – Flask-SQLAlchemy – Démo – Model

- La création du model se fait par héritage de classe Model de l'objet db.
- L'objet db permet d'utiliser la session de SQLAlchemy ORM

```
from database import db


##création model
class SimpleUser(db.Model):
    id =db.Column(db.Integer, primary_key=True)
    email = db.Column(db.String(200), nullable=False)
    __tablename__ = 'simple_user'

    def __init__(self, email):
        self.email = email

    ##Méthode save
    def save_to_db(self):
        db.session.add(self)
        db.session.commit()

    def json(self):
        return {'email':self.email, 'id': self.id}

    ##Methode
    @classmethod
    def find_by_email(cls, email):
        return cls.query.filter_by(email=email).first()
```

Suite api Gestion de commandes

TP

- Nous souhaitons modifier le système de stockage de notre api commandes pour passer d’un système en liste vers une base de données relationnelles avec SQLAlchemy comme ORM.
- Modifier les models en conséquence.
- Modifier la logiques métier des services pour enregistrer dans notre base de données.

Flask – Flask-injector

- Flask-injector est une bibliothèque qui permet de mettre en place le design pattern d'injection de dépendance et d'inversion de contrôle.
- Flask-injector permet de configurer les différents types quand nous souhaitons injecter.
- Flask-injector automatise la création d'instance et l'injection dans :
  - Class ressource
  - Hook de flask
  - …
- Flask-injector permet de définir plusieurs scope (Request, Singleton)
- Démo

Flask – Flask-injector – démo – configuration

- La configuration des éléments à injecter se fait à l'aide d'un objet de type FlaskInjector et une ou plusieurs fonctions de configurations.

```
def configure(binder):
    binder.bind(Service, to=Service, scope=request)
    binder.bind(Repository, to=Repository, scope=request)

FlaskInjector(app=app, modules=[configure])
```

Flask – Flask-injector – démo – inject

- Pour injecter un élément à l'intérieur d'un autre, nous pouvons utiliser le décorateur inject

```
class Service:
    @inject
    def __init__(self, repository:Repository):
        self.respository = repository

    def save(self,element):
        return self.respository.save(element)
```

```
class SimpleUserResource(Resource):
    @inject
    def __init__(self, service:Service):
        self.service = service
    def get(self, email):
        return SimpleUser.find_by_email(email).json()

    def post(self):
        email = request.json.get("email")
        u = SimpleUser(email)
        return self.service.save(u).json()
```

Flask – Injection de dépendance et inversion de contrôle

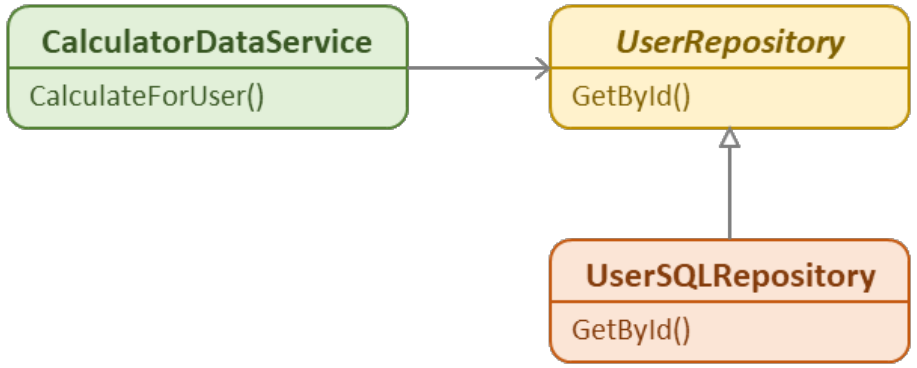
- Lorsqu'une classe (A) a besoin d'une autre classe (B) pour fonctionner, on dit que A a une dépendance vers B et que B est une dépendance pour A. Le cas le plus basique se produit lorsque la classe A instancie elle-même la classe B : elle crée in situ la dépendance dont elle a besoin.

Flask – Injection de dépendance et inversion de contrôle

- L'injection de dépendances, *Dependency Injection (DI),* a pour but de séparer la création d'un objet de son usage. Cette création est faite à l'extérieur de la classe, par le code appelant. L'instance de la dépendance ainsi créée est injectée dans la classe qui en fait usage
  - Par constructeur : la dépendance est passée en paramètre au constructeur.
  - Par méthode : la dépendance est passée en paramètre à la méthode qui l'utilise.
  - Par propriété : la dépendance est injectée dans la classe au moyen d'un *setter*, d'une propriété en écriture (et éventuellement en lecture).

Flask – Injection de dépendance et inversion de contrôle

- Le principe d'inversion des dépendances, *Dependency Inversion Principle (DIP)*, est l'un des 5 principes SOLID : c'en est le "D"
- Les modules de haut niveau ne doivent pas dépendre des modules de bas niveau. Les deux doivent dépendre d'abstractions.
- Les abstractions ne doivent pas dépendre des détails. Les détails doivent dépendre des abstractions.



Flask – Injection de dépendance et inversion de contrôle

- L'inversion de contrôle, *Inversion of control (IoC),* consiste à déléguer une partie du "contrôle" à l'extérieur de l'élément courant dans un système.
- Au départ, CalculatorDataService contrôlait l'instanciation de ses dépendances.
- Après le premier refactoring (mettant en place l'injection de dépendances), CalculatorDataService a perdu ce contrôle. Il y a eu une inversion du flux de contrôle. On constate donc que l'injection de dépendances est une forme d'IoC.

Suite api Gestion de commandes

TP

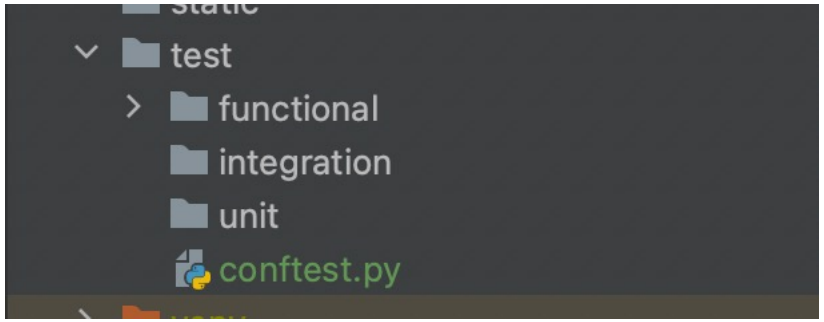
- En utilisant les principes d’injection de dépendance et Inversion de contrôles :
  - Ajouter des classes repository pour gérer les interactions avec la base de données.
  - Faire en sorte que les classes repository et services soit générer automatiquement.

Flask – Ecriture des tests

- Pour s'assurer que notre application fonctionne correctement même après de modifications, nous pouvons :
  - Automatiser les tests manuels répétitifs en réduisant les erreurs humaines
- Intégrer les tests dans un process CI/CD.
- Pour écrire les différents tests de notre api, nous pouvons :
  - Utiliser les objets tests de Flask pour simuler les différentes requêtes vers notre api.
  - Utiliser notre framework de test pytest
  - Démo

Flask – Structure dossiers tests

- Nous pouvons regrouper les tests par types
- Fonctionnels
- Intégrations
- Unitaires



Flask – Ecriture des tests

- L’application flask fournit un client pour tester notre application.
- Le client de test peut être utiliser avec les fonctionnalités des frameworks de tests Pytest ou inittest.
- Le client test permet d’exécuter des requêtes get, post, put,… avec des données en json, form,…
- Le client test permet de récupérer les résultats des requêtes pour appliquer des assertions.

Flask – exemple de test get et post

```
def test_get_simple(app):
    with app.test_client() as test_client:
        response = test_client.get('/simple/ihab@utopios.net')
        assert response.status_code == 200

def test_add_simple(app):
    with app.test_client() as test_client:
        response = test_client.post('/simple', json={'email': 'test'})
        assert response.status_code == 200
```

Suite api Gestion de commandes

TP

- Ecrire des tests pour les différents endpoints de notre api commandes

Sécurisation d’une API Rest

Les 4 principaux concepts de la sécurité sont :

- Confidentialité : rejet des accès non autorisés.
- Intégrité : rejet des modifications non autorisées.
- Disponibilité : lutte contre les dénis de service.
- Non répudiation : capacité à fournir des preuves

Sécurisation d'une API Rest – Identification, Authentification et Autorisation

- Identification : On identifie une entité sans pouvoir en vérifier l'authenticité.
- Authentification : Il est possible de vérifier l'authenticité d'un message, d'une action etc...
  Cela n'implique pas forcément une identification.
  Ex. : Enregistrement d'un pseudo sur IRC.
  Ex. : Facebook's Anonymous Login.
  Ex. : Clés SSH.
- Autorisation : Détermine si une entité a accès à une ressource en fonction des règles définis dans les A.C.L. (Access Control Lists).
  Ex. : Accès autorisé / refusé à une ressource sur une API.
  Ex. : Accès autorisé / masqué / refusé à une propriété d'une ressource.
- Les règles A.C.L. ne sont pas forcément associées à une entité.
  Le porteur d'un "token" n'est donc pas forcément identifié.
  Ex. : Un "token" temporaire partagé avec plusieurs utilisateurs pour accéder à un document

Sécurisation d’une API Rest – Identification, Authentification et Autorisation – Vulnérabilités

- Authentification et autorisation absentes ou insuffisantes.
- Authentification sans autorisation.
- Autorisations trop permissives sur les propriétés en lecture ou en écriture

Sécurisation d'une API Rest – Identification, Authentification et Autorisation – Vulnérabilités

- "Origin" : scheme + FQDN + port.
  https://www.wishtack.com(:443)

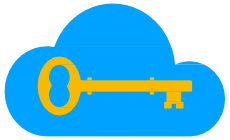
- Le client envoie une preflight request (méthode OPTIONS) indiquant l'"origin", la méthode, les headers **si nécessaire** :
  - Méthode autre que GET / HEAD / POST.
  - POSTavec un media type autre que text/plainou application/x-www-form-urlencodedou multipart/ form-data.
  - "Headers" modifiés autres que Accept / Accept-Language / Content-Type (Cf. condition précédente) / Content-Language.

- Le serveur répond avec les headers :
  Access-Control-Allow-Origin Access-Control-Allow-Methods Access-Control-Allow-Headers
  Access-Control-Allow-Credentials

J.W.T. JSON Web Token

- J.O.S.E. : Un framework pour échanger des “claims” de manière sécurisée.
  - J.W.K. : JSON Web Key.
  - J.W.E. : JSON Web Encryption.
  - J.W.S. : JSON Web Signature.
  - J.W.T. : JSON Web Token.

J.W.T. Exemple d'utilisation



identity-provider.py

6 - Get Public Key
if not in cache

smurf.com

7 - Verify J.W.T.
with public key

4 - Create J.W.T.
and sign with private key

2 - Redirect

3 - Authenticate
& Authorize

1 - Signin
request

5 - Redirect with
J.W.T.

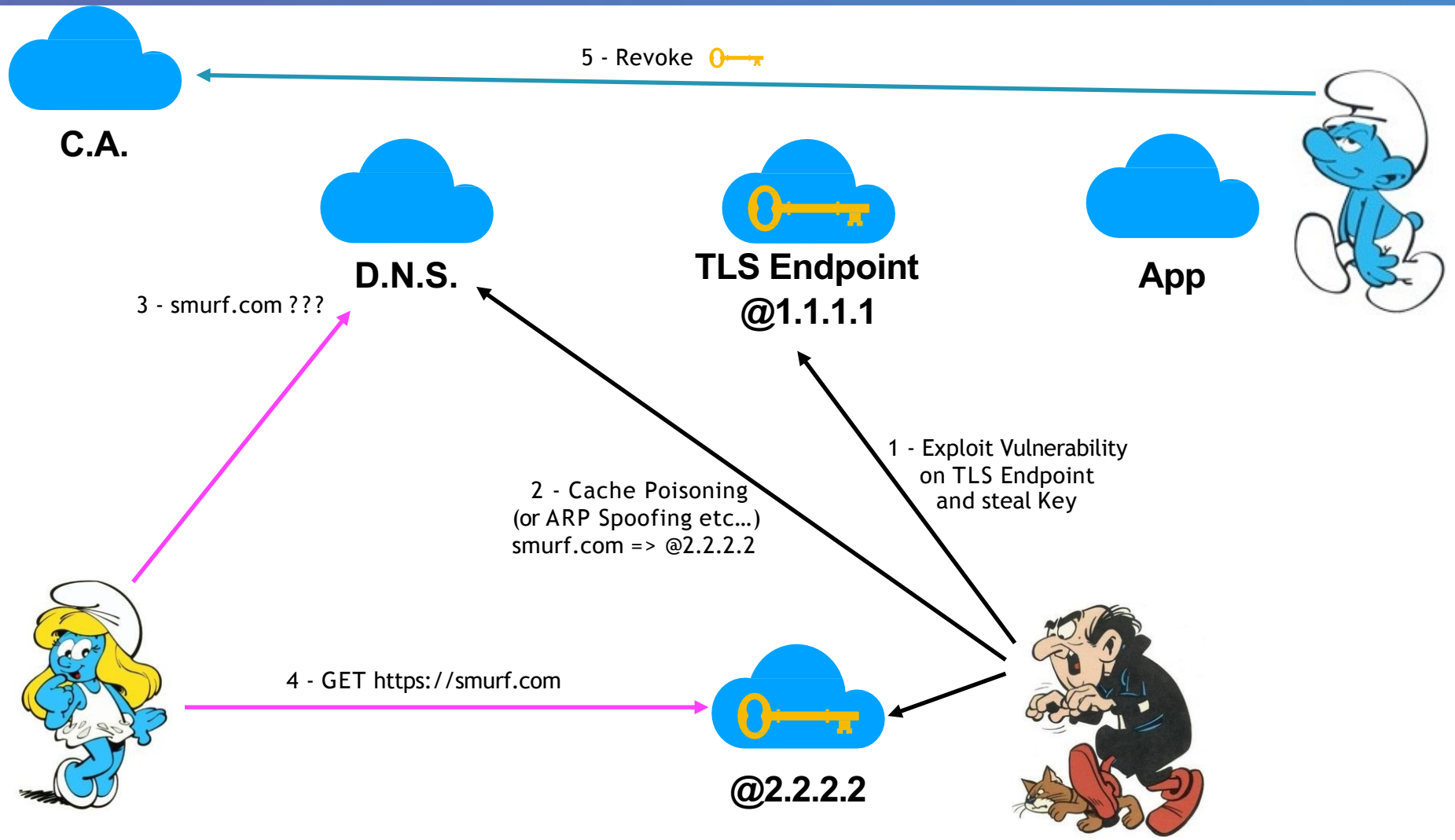
8 - Send flowers

J.W.T. ou cryptographie sans “Key Management”

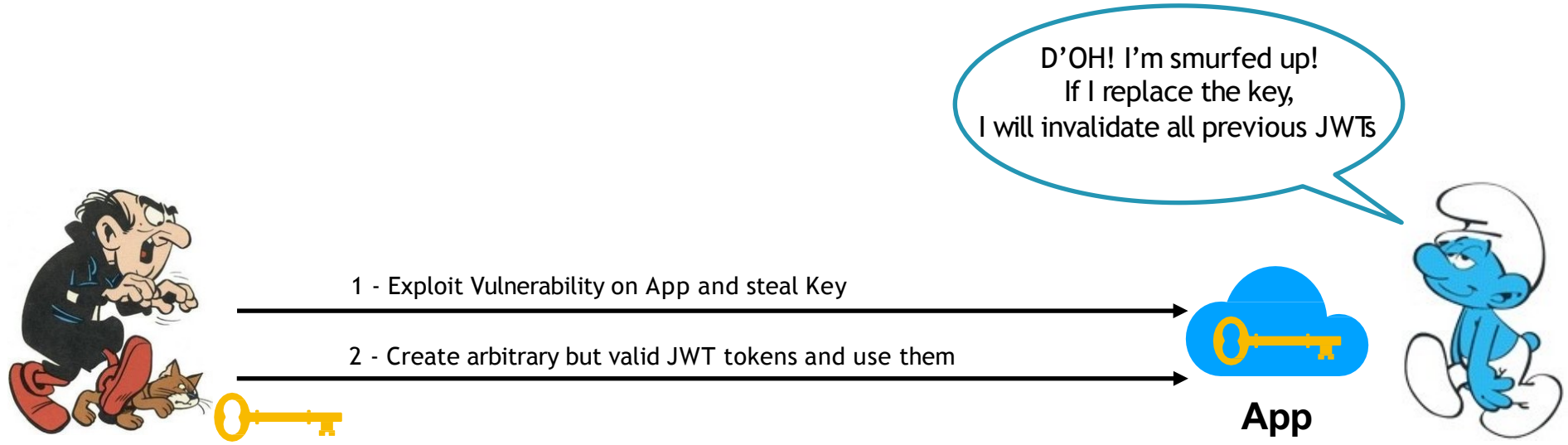
- La cryptographie sans “Key Management”, c’est comme une voiture sans freins, tant qu’on ne cherche qu’à rouler en ligne droite sans s’arrêter, ça répond plutôt bien au besoin…

- Analysons d’abord la sécurité de nos clés TLS.

Quand on vole une clé T.L.S.



Quand on vole une clé J.W.T.



J.W.T. Vulnérabilités classiques

- Stockage non sécurisé.
- `data = jwt.decode(token, key)`
  vs.
  `data = jwt.verify(token, key)`
- Signature stripping: `{alg: 'none'}`
- ~~a~~symétrique ?
- HS256 nécessite une clé de longueur supérieure ou égale à :
  256 bits / 32 bytes / 64 caractères hexadécimaux.
- HS256 vs RS256.
- Selon NIST :
  1024 : RIP 2006
  2048 : RIP 2030
  4096 : ???

J.W.T. Exemple RS256

- Et hop, 1/2 kilo de token :
  eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdWIiOiJqb2
  huZG9lIiwiaWF0IjoxMzk3MzgzOTA4LCJrZXlpZCI6IjEyMyJ9.
  b0cBt45NzbdAi21VNv_mVtGwYNWRJBkkWNyk_IQDTt6las
  PvXUmbPU-6ou1Yj9FElRCDr7aqjvm7IaQHs0cx0aU5CmBY
  EcvbTo7kNxJkhOtWl2XRU7Mk2zCNJgNK2eEEOHDTT48_U
  MCkHcCGADYzJ5H9mPySeMGTYq4cHGHVbw5v6LRjXYaB
  Xa1jgDfqjTqy5RL2hS19YYaKsoCm5Vsk1tsHAyz4TdqM-
  Ctbuk6AnA57TL_-
  zcx2XbXqv_ztTpdZSA9hLzXVJyFQOj-8hDmNHpZTTFLKfeC
  Ha7HSZfQ1kUGD6AX-
  Kn5Gk3nmaRv7Ox0Dtl-2v15BELiFCLAOFp1acw

J.W.T. Key Rotation

- Les clés doivent être générées et remplacées dynamiquement et régulièrement.
- Les clés ne peuvent pas avoir une durée de vie inférieure à celles des tokens générés.
- Chez Google, une durée moyenne de 3 jours : https://www.googleapis.com/oauth2/v3/certs

Flask – flask-jwt-extended

- flask-jwt-extended permet de créer un JWT à partir d'une secret key.
- flask-jwt-extended permet de protéger des routes ou des ressources.
- flask-jwt-extended fournit un ensemble de fonction pour implémenter notre jwt:
  - JWTManager
  - create_access_token
  - get_jwt_identity
  - jwt_required
- Démo

Suite api Gestion de commandes

TP

- Nous souhaitons rendre l'endpoint d'ajout de produit accessible uniquement pour des utilisateurs connectés.
  - Pour implémenter ce mécanisme, nous souhaitons implémenter une authentification et autorisation par JWT.
  - Ajouter un Model user avec un email et passord.
  - Ajouter une ressource user pour enregister un user et se connecter et générer un JWT
- Protéger l'endpoint d'ajout de produit.

Formalisation des APIs avec OpenAPI

spécification d'API : fichier OpenAPI json/yaml

OpenAPI est un métalangage standardisé de description des APIs REST, issu du projet **Swagger**. Un fichier OpenAPI (en JSON ou en YAML) contient la description complète d'une API :

- informations générales sur l'API : version, titre, description
- les protocoles autorisés (HTTP, HTTPS...)
- les représentations des ressources autorisées (JSON, XML...)
- les URLs des ressources, avec pour chacune d'elles :
  - les verbes autorisés
  - une description de l'opération (son but, son usage...)
  - les paramètres d'entrée
  - les réponses possibles avec leurs codes de statut
- la définition des ressources elles-mêmes (objets échangés).

Un ensemble d'outils Swagger s'appuyant sur OpenAPI permettent de générer la documentation, des tests, le squelette de code client/serveur...

Formalisation des APIs avec OpenAPI

écriture d'API : Swagger Editor

Swagger Editor est un outil accessible sur le web. Il est également possible de l'installer en local. Il permet :

- d'écrire directement le fichier OpenAPI (description de l'API en JSON ou YAML)
- de vérifier sa validité syntaxique
- d'avoir un affichage très lisible et en temps réel, en miroir du code OpenAPI.

Note : un fichier OpenAPI étant un fichier texte, il est possible d'utiliser tout éditeur de texte, mais vous n'aurez pas de validation syntaxique ni l'affichage miroir temps réel.

Formalisation des APIs avec OpenAPI

consultation et test d'API : Swagger UI

Swagger UI est un outil accessible sur le web.

Il permet d'afficher la définition d'une API OpenAPI sous une forme graphique, claire et conviviale. Il suffit de renseigner l'URL d'accès au fichier OpenAPI déployé préalablement sur un serveur. L'outil **Swagger uploader** du programme API permet de le faire facilement.

Le but est de faciliter la compréhension et l'adoption de l'API par les clients.

Swagger UI sert d'**interface entre le fournisseur et le client** de l'API :

- le fournisseur y **publie** son API (fichier OpenAPI) et la fait pointer sur une plateforme de test ou un serveur de bouchons (exemples : Mock-server, SoapUI)
- le client la **consulte**, et peut **tester** son comportement (paramètres d'entrée, données échangées, statuts de retour...).

Formalisation des APIs avec OpenAPI

inspection d'API : Swagger Inspector

Swagger Inspector est un outil accessible sur le web. Il a 2

fonctionnalités principales :

- servir de client HTTP pour tester une API (comme POSTMAN, RESTer, RESTClient...)
- générer un fichier OpenAPI à partir des API testées :
  - lancer les requêtes HTTP souhaitées pour tester l'API
  - ces requêtes sont stockées dans l'historique de l'outil et sont facilement accessibles
  - sélectionner dans cet historique les requêtes qui seront prises en compte pour la génération du fichier OpenAPI.

Formalisation des APIs avec OpenAPI

génération de code client/serveur : Swagger Codegen

Swagger Codegen permet de générer du code client et/ou serveur (comme WSDL2Java pour les webservices Soap) à partir du fichier OpenAPI. Cet outil est utilisable :

- via Swagger Editor, menu **Generate Server** ou **Generate Client**
- via un plugin Maven.

Swagger Codegen supporte plusieurs frameworks Java de gestion d'API, tels que Jersey, Spring Web MVC, CXF. D'autres langages sont également disponibles (PHP, Node.js, Scala, Ruby, Python, Typescript Angular 2...).

La suite de la présentation ne s'appuiera pas sur Swagger Codegen, afin de montrer comment écrire du code Spring MVC pour exposer des services REST.