# Document technique

Table des matières

[Document technique 1](#_Toc33983846)

[1) Introduction 2](#_Toc33983847)

[2) Schéma d’architecture 2](#_Toc33983848)

[3) Choix technique 4](#_Toc33983849)

[4) Explication des microservices 4](#_Toc33983850)

[Utilisateurs 4](#_Toc33983851)

[Authentification 5](#_Toc33983852)

[Gestion train et bus 6](#_Toc33983853)

[Réservation 6](#_Toc33983854)

[Spring Eureka 6](#_Toc33983855)

[Api gateway 8](#_Toc33983856)

[Base de données 8](#_Toc33983857)

[5) Tester l’API sur Postman 9](#_Toc33983858)

## Introduction

Ce TP a pour but de nous apprendre les concepts d’une architecture microservice, en quoi il est important pour les entreprises de passer en microservices.

Savoir concevoir une Api REST en sachant maitriser le protocole http ainsi que la compréhension et l’implémentation d’une Api Gateway.

## Schéma d’architecture

Schéma technique

Une image contenant texte, carte

Description générée automatiquement

Dans le cadre d’une application de production nous pouvons proposer un schéma de ce type. Communication asynchrone basé sur les événements.

Schéma utilisé pour ce TP :

Communication de client à microservice HTTP via des passerelles d’API.

Une image contenant texte, carte

Description générée automatiquement

## Choix technique

Pour ce projet j’ai choisi de travailler sur la technologie Spring Boot en JAVA,

En effet cette technologie est davantage pratique pour la construction d’environnement microservices.

J’ai ainsi utilisé des dépendances Maven telles que spring Zuul, Spring Eureka, ConfigClient, Spring Security ou encore Feign.

Pour ce projet nous ne possédons pas de base de données pour le moment, nous travaillons donc avec des listes de données.

Nous pourrions faire aussi une base de données ou « fichier plat » ce qui nous permettront de générer de la data au lancement de chaque MS.

En effet je n’ai pas fait de base de données, j’ai travaillé avec des listes.

Les micro-services doivent fonctionner en mode Stateless c’est-à-dire qu’ils se suffisent à eux-même.

Le principe est de ne pas faire de requêtes entre les micro-services directement.

En effet si un micro-services attend des réponses d’un autre micro-services et que ce dernier vient à tomber cela pourrait compromettre son état.

Si nous devons faire des appels entre micro-services cela reviendrait à faire du WebService.

C’est pourquoi j’ai des objet et dao d’autres microservices étant donné que je travaille avec des listes.

## Explication des microservices

Pour ce projet nous avons besoin de 4 micro-services ou chacun aura des routes api rest différentes.

Nous pouvons identifier 4 micro-services

### Utilisateurs

Ce micro-services aura la tâche de créer les utilisateurs lors de leur première venue ainsi que pour la modification de leurs comptes.

Pour la création d’un compte : POST /register

Pour modifier son compte : PUT /account

Il va pouvoir voir son compte avec : GET /account

Il va aussi servir aux autres microservices pour utiliser les infos clients, ce microservice ira lire et écrire sur la base de données distantes, la base va pouvoir être ainsi approvisionné pour les autres microservices.

J’ai aussi utilisé le pattern Data Access Object pour avoir un accès simplifié pour la lecture et l’écriture des données de ma base.

### Authentification

Ce micro-services aura la tâche de sécurisé et d’authentifier les clients.

Dès lors qu’il voudra s’identifier grâce à la route (POST) /login,

Nous ferons une identification en allant faire une requête a notre base de données pour vérifier que l’utilisateur existe et que ses identifiants correspondent bien.

Pour une authentification sécurisée nous utiliserons le JSON Web Token (JWT) que nous enverrons au client.

Son principe est le suivant :

Le client fait une requête post au serveur ou il y aura dans le corps : le login et mdp;

Le serveur répond dans le corps de la requête http le jwt généré.

A chaque requête du client il va envoyer dans le header de la requête son token (que le serveur lui a donné auparavant).

Le serveur va donc analyser ce token pour savoir s’il peut répondre à sa demande, et il va vérifier s’il est toujours authentifié.

Pour vérifier, il faut comparer la signature du token

-> un qui va être en clair que je vais devoir comparer, avec celui en hash

Décoder en base64 (token)

Grâce à ça je récupère le header et le payload

Puis j'encode le header et le payload que je concatène.

Et je hash le tout en dans l’algorithme de hachage présent dans le header de la requête.

Et je compare le hash obtenu avec celui de la signature de la requête.

Pour chaque requête a un service il va vérifier à chaque fois si le token est toujours valable.

Il y aura donc une vérification a chaque requête pour vérifier le token

Nous pourrions aussi nous tourner vers une authentification avec **Keycloack,**

Son principe est le suivant :

1- l'utilisateur s'authentifie

2- Après être bien identifier, Keycloak va envoyer au front, une clé d’accès à l’application appelé access\_token

3- L’utilisateur va pour utiliser l’application front, avec les différentes pages et les différents droits qu’il a (admin / user). Le front va aussi faire des appels REST pour récupérer les informations nécessaires au bon fonctionnement de l’application.

4- L’API, en recevant la requête REST du front-end, va vérifier s’il y a un access\_token et demander à Keycloak si le token est bien valide.

5- Keycloak va confirmer ou non la bonne validité de l’access\_token.

Et l’API, construira, en fonctionne des droits que dispose l’utilisateur, la response REST à la requête demandée.

### Gestion train et bus

Ce microservice sera la base du site car il va permettre au client de choisir un train ou bus et de le réserver par la suite.

Nous aurons donc une route /all qui montrera tous les trains ou bus.

Il y aura aussi une route qui permettra au personnel d’ajouter des trains.

Ainsi qu’une route pour que le client puisse voir un train en particulier.

Ce microservice va donc faire principalement des requêtes a la base de données pour pouvoir avoir toujours sa liste de train ou bus à jour.

### Réservation

Ce microservices servira à réserver des trains ou bus.

Il y aura donc une création de réservation, nous identifierons dans le corps de la requête si c’est un train ou bus, s’il est impossible de réservé car il est plein ou encore de mettre à jour le fichier client avec ses points de fidélité.

Nous avons simulé aussi un payement avec la route /reservation/{id}/payment

Ce microservices aura donc la tâche d’identifier les clients et de sécurisé leurs sessions.

### Spring Eureka ­­­

Quand votre application répond à une montée en charge et que vous avez plusieurs instances de chaque microservice, il est vital de pouvoir garder un **registre de toutes les instances disponibles** afin de distribuer la charge entre celles-ci.

Une image contenant capture d’écran

Description générée automatiquement

C’est le principe de Eureka il va tenir un registre. Et tous les autre Edge micro-services tel que Zuul ou encore Ribbon (load Balancer) vont se baser sur Eureka.

Interface d’Eureka :

Une image contenant capture d’écran

Description générée automatiquement

### Api gateway

Concernant l’api gateway, j’ai choisi d’utiliser un proxy Zuul, qui est une dépendance Maven.

Zuul fonctionne en correspondance avec Eureka, en effet dès lors qu’un microservice sera en état de marche il ira directement s’enregistrer sur le Server d’Eureka grâce à sa configuration, Eureka saura quel est le nom de ce microservice, ainsi à chaque requête du client vers l’api gateway le proxy Zuul saura vers quel microservice il doit rediriger sa requête.

Si la requête du client est en destination du micro-services réservation

La requête venant du client sera alors de ce type /reservation/allreservation

--- le premier /réservation sera ici pour Zuul pour qu’il identifie le microservice.

Il exclura ensuite pour ne laisser place qu’à la requête /allreservation.

Nous pouvons aussi mettre en place des filtres pour filtrer les requêtes entrantes ou sortantes

Les filtres seront soit en « pre » soit en « post ».

Zuul va alors être le seul point d’entrée (frontal d’accès) pour l’application.

### Base de données

Concernant la partie data il n’était pas demandé de réaliser une base de données.

Mais pour faciliter la gestion des données entre les micro-services il faut implémenter une base de données.

Dans un premier temps pour simplifier ma gestion d’un SGBD j’ai créé un docker MySQL

J’ai créé un docker-compose avec l’accès depuis l’hôte grâce au port 3306.

Pour simplifier et avoir une interface de gestion j’ai mis un adminer sous localhost:8080

Il faudra bien penser à changer les ports des micro-services pour ne pas avoir de conflit par la suite.

Dans un contexte de production nous aurons bien évidement pas de docker pour une base de données mais un ou des serveurs de base(s) de donnée(s) dédiés selon les besoins.

Pour que mes micro-services puissent interagir avec ma BDD il faut utiliser JPA qui est le connecteur pour faire le mapping des entités de la base avec les entités de nos micro-services.

L’utilisation de JPA nécessite une certaine compréhension.

Mais dans un autre contexte que ce tp il faut bien évidemment fonctionner avec une base de données pour que tous nos micro-services puissent être indépendant.

Nous pouvons aussi utiliser des bus d’évènement tel que Kafka ou encore des solutions d’ordonnanceur comme Rabbit MQ.

## Tester l’API sur Postman

Pour faire des requetes au microservice authentification : localhost:9004/auth/XXXX

Pour faire des requetes au microservice gestionbustrain : localhost:9004/ybus/XXXX

Pour faire des requetes au microservice utilisateurs : localhost:9004/users/XXXX

Pour faire des requetes au microservice reservation : localhost:9004/reservation/XXXX

{

"id": 5,

"id\_train": 3,

"name": "Warren",

"surname": "Buffet",

"email": "stringerbell@gmail.com",

"birth\_date": "01/08/77",

"tel\_number": "0629338108",

"type":"train",

"reservation\_number": 67548,

"prix\_HT": "29$",

"tva": "20%",

"gare\_depart": "Paris",

"heure\_depart\_prevue": "7H",

"gare\_arrivee\_prevue": "Lyon",

"heure\_arrivee\_prevue": "9H",

"seat\_number": 986

}