

# 133 - Développer des applications WEB

## Rapport Personnel

Date de création 21.03.2024

Version 1

---

**Dufour Johan**

---



Module du 21.03.2024 à 19.04.2024

## Table des matières

1	Introduction et contexte du project .....	4
2	Tests technologiques selon les exercices .....	5
2.1	Installation et Hello World .....	5
2.1.1	Observez la console pour comprendre comment le projet est lancé et comment il tourne ? .....	5
2.1.2	C'est quoi le build et le run de Java ? Quel outil a-t-on utiliser pour build le projet ? .....	5
2.1.3	Y a-t-il un serveur web ? .....	5
2.1.4	Quelle version de java est utilisée ? .....	5
2.1.5	S'il y a un serveur web,quelle version utilise-t-il ?.....	5
2.2	Conteneurisation .....	6
2.2.1	Pourquoi faire un container pour une application Java ? .....	6
2.2.2	Y a-t-il un serveur web ? Ou se trouve-t-il ? .....	6
2.2.3	A quoi faut-il faire attention ? .....	6
2.3	Création d'un projet Spring Boot.....	7
2.4	Connexion à la DB JDBC.....	8
2.5	Connexion à la DB JPA.....	9
2.6	Connexion à la DB JPA avec DTO .....	9
2.7	Gestion des sessions .....	10
2.8	Documentation API avec Swagger .....	13
3	Analyse à faire complètement avec EA -> à rendre uniquement le fichier EA .....	16
3.1	Description de projet .....	16
3.1.1	Site de Johan .....	16
3.2	Use case client et use case Rest.....	16
3.3	Activity Diagram d'un cas complet navigant dans les applications avec les explications .....	17
3.4	Sequence System global entre les applications .....	19
4	Conception à faire complètement avec EA -> à rendre uniquement le fichier EA .....	21
4.1	Class Diagram complet avec les explications de chaque application.....	21
5	Bases de données .....	23
5.1	Modèles WorkBench MySQL.....	23
6	Implémentation des applications <Le client Ap1> et <Le client Ap2> .....	24
6.1	Une descente de code client.....	24
7	Implémentation de l'application <API Gateway> .....	25
7.1	Une descente de code APIGateway .....	25
7.1.1	Controlleur.....	25
7.1.2	ServiceApiRest2.....	25

8	Implémentation des applications <API élève1> et <API élève2> .....	27
8.1	Une descente de code de l'API REST .....	27
9	Hébergement .....	28
10	Installation du projet complet avec les 5 applications .....	29
11	Tests de fonctionnement du projet .....	30
12	Auto-évaluations et conclusions .....	33
12.1	Auto-évaluation et conclusion de Johan .....	33

# 1 Introduction et contexte du project

1 Analyser la donnée, projeter la fonctionnalité et déterminer le concept de la réalisation.

2 Réaliser une fonctionnalité spécifique d'une application Web par Session-Handling, authentification et vérification de formulaire.

3 Programmer une application Web à l'aide d'un langage de programmation compte tenu des exigences liées à la sécurité.

4 Vérifier la fonctionnalité et la sécurité de l'application Web à l'aide du plan tests, verbaliser les résultats et, le cas échéant, corriger les erreurs.

## 2 Tests technologiques selon les exercices

### 2.1 Installation et Hello World

#### 2.1.1 Observez la console pour comprendre comment le projet est lancé et comment il tourne ?

L'application est lancée par la commande « Starting estServiceApplication », l'application tourne en utilisant le Framework Spring Boot et le serveur web Tomcat.

#### 2.1.2 C'est quoi le build et le run de Java ? Quel outil a-t-on utiliser pour build le projet ?

Build : Cette étape consiste à compiler le code source Java en bytecode, qui peut être exécuté par la machine virtuelle Java (JVM).

Run : Cette étape consiste à exécuter l'application Java. Cela se fait en lançant la JVM et en lui donnant le bytecode à exécuter.

#### 2.1.3 Y a-t-il un serveur web ?

Oui, Tomcat est actif.

#### 2.1.4 Quelle version de java est utilisée ?

Java en version 17.0.6

```
Starting RestServiceApplication using Java 17.0.6 with PID 9597
```

#### 2.1.5 S'il y a un serveur web, quelle version utilise-t-il ?

Tomcat en version 10.1.16

```
Starting Servlet engine: [Apache Tomcat/10.1.16]
```

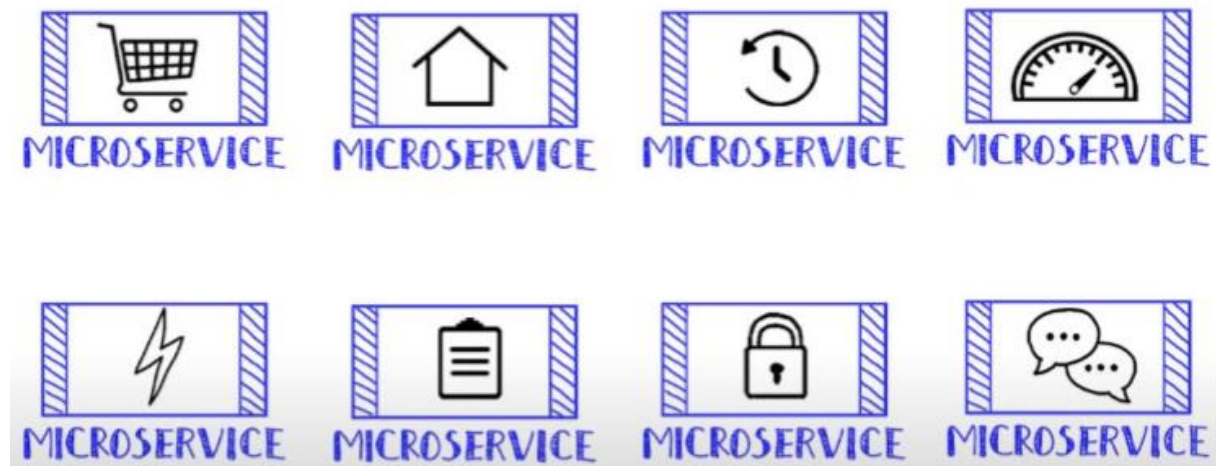
## 2.2 Conteneurisation

### 2.2.1 Pourquoi faire un container pour une application Java ?

Les conteneurs Docker offrent plusieurs avantages pour le déploiement d'applications Java.

Ils permettent de créer des environnements isolés et reproductibles pour les applications, ce qui facilite le déploiement et la mise à l'échelle.

De plus, comme les conteneurs Docker encapsulent toutes les dépendances de l'application, ils garantissent que l'application fonctionnera de la même manière sur toutes les machines.



### 2.2.2 Y a-t-il un serveur web ? Ou se trouve-t-il ?

Oui, c'est Spring Boot. Il est directement inclus dans l'application

### 2.2.3 A quoi faut-il faire attention ?

Faire attention à la version du jdk

```
dufourj@WSTEMFA39-12:~$ sudo apt-get install openjdk-17-jdk
```

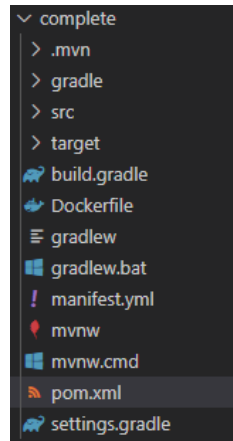
Faire attention à la version du Dockerfile

```
FROM openjdk:17-jdk-alpine
```

Faire attention à la version du pom.xml

```
<properties>
  <java.version>17</java.version>
</properties>
```

Faire attention que le Dockerfile soit au même endroit que le pom.xml



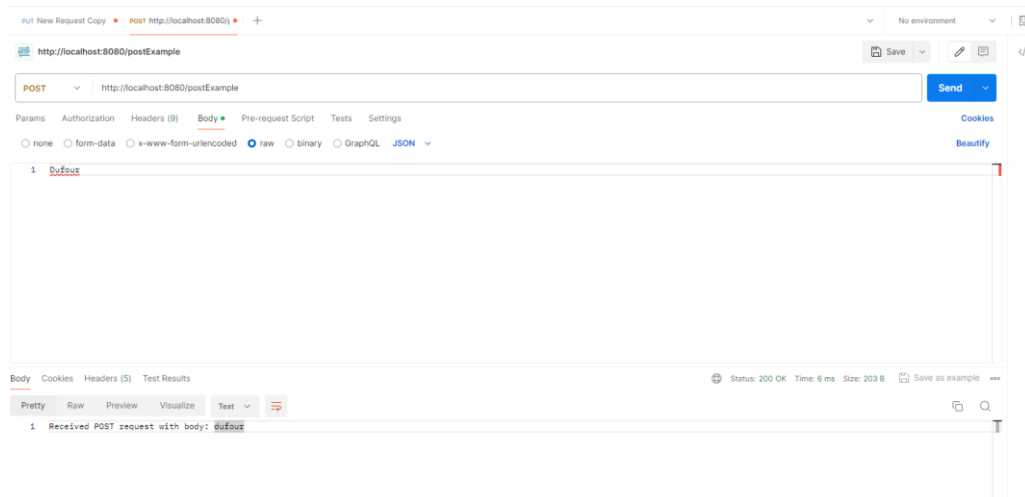
## 2.3Création d'un projet Spring Boot

**@RestController** : Cette annotation indique que la classe est un contrôleur REST. Cela signifie que les méthodes de cette classe sont prêtes à traiter les requêtes HTTP entrantes.

**@GetMapping("/getExample")** : Cette annotation indique que la méthode `getExample` doit être appelée lorsqu'une requête HTTP GET est envoyée à l'URL `"/getExample"`. La méthode prend un paramètre `name` qui a une valeur par défaut de `"World"` si aucun autre nom n'est fourni dans la requête. Elle renvoie ensuite une chaîne formatée qui dit `"Hello, [name]!"`.

**@PostMapping("/postExample")** : Cette annotation indique que la méthode `postExample` doit être appelée lorsqu'une requête HTTP POST est envoyée à l'URL `"/postExample"`. La méthode prend le corps de la requête HTTP comme paramètre et renvoie une chaîne qui dit `"Received POST request with body: "` suivie du contenu du corps de la requête.

**@PutMapping("/putExample")** : Cette annotation indique que la méthode `putExample` doit être appelée lorsqu'une requête HTTP PUT est envoyée à l'URL `"/putExample"`. Comme pour la méthode `postExample`, cette méthode prend le corps de la requête HTTP comme paramètre et renvoie une chaîne qui dit `"Received PUT request with body: "` suivie du contenu du corps de la requête.



## 2.4 Connexion à la DB JDBC

Pour pouvoir se connecter, il faut commencer par ajouter ceci dans pom.xml

```
<dependency>
  <groupId>com.mysql</groupId>
  <artifactId>mysql-connector-j</artifactId>
  <scope>runtime</scope>
</dependency>
```

Voilà ma méthode pour la connexion, attention de mettre  
« host.docker.internal » pour éviter des erreurs.

```
public boolean openConnexion(){
    final String url =
        "jdbc:mysql://host.docker.internal:3306/bd_kitzbuehl?serverTimezone=CET";
    final String user = "root";
    final String pw = "emf123";
    boolean result = false;
    try{
        //nécessaire pour fonctionnement en web
        Class.forName("com.mysql.jdbc.Driver");
    } catch ( ClassNotFoundException ex ) {
        System.out.println("Connexion au driver JDBC à échoué!\n" + ex.getMessage());
    }
    try {
        dbConnexion = DriverManager.getConnection( url, user, pw );
        //System.out.println("Connection successfull");
        result = true;
    } catch ( SQLException ex ) {
        System.out.println("Connexion à la BD a échouée!\n" + ex.getMessage());
    }
    return result;
}
```



Exemple de la méthode depuis le contrôleur pour encoder en JSON et renvoyer un string

```
@GetMapping("/getPays")
public String getPays() {
    ArrayList<String> lstPays = wrkDB.getPays();
    ObjectMapper objectMapper = new ObjectMapper();
    String paysJson = null;
    try {
        paysJson = objectMapper.writeValueAsString(lstPays);
    } catch (Exception e) {
        e.printStackTrace(); //Gérer l'exception dans votre application
    }
    return paysJson;
}
```

## 2.5 Connexion à la DB JPA

L'annotation `@Autowired` dans Spring est utilisée pour l'injection de dépendances. Elle permet à Spring de résoudre et d'injecter automatiquement les beans collaborateurs dans notre bean. Par exemple, dans le contrôleur, `@Autowired` est utilisé pour injecter automatiquement les instances de `SkieurRepository` et `PaysRepository`.

```
@Autowired
private PaysRepository paysRepository;
```

L'annotation `@ManyToOne` est utilisée pour mapper une relation de plusieurs à un entre deux entités. Dans l'entité `Skieur`, `@ManyToOne` indique qu'un `Skieur` peut être associé à un seul `Pays`.

Le `FetchType.EAGER` signifie que l'entité associée sera chargée en même temps que l'entité principale lors de la récupération de l'entité principale à partir de la base de données. Cela signifie que lorsque l'on récupère un `Skieur` de la base de données, le `Pays` associé sera également chargé.

```
@ManyToOne(fetch = FetchType.EAGER)
@JoinColumn(name = "fk_pays")
private Pays pays;
```

## 2.6 Connexion à la DB JPA avec DTO

On a utilisé un `SkieurDTO` dans notre application pour encapsuler les données de `Skieur` et les transférer entre différentes parties de l'application, ou pour adapter les données de `Skieur` à d'autres systèmes ou services externes.

**Model** : Les modèles représentent les entités dans votre application. Dans votre cas, Skieur et Pays sont des modèles qui représentent respectivement un skieur et un pays dans votre application. Ils sont utilisés pour structurer les données qui seront ensuite stockées dans la base de données.

**Repository** : Les repositories sont des interfaces qui permettent d'interagir avec la base de données. Ils fournissent des méthodes pour effectuer des opérations CRUD (Create, Read, Update, Delete) sur les entités.

**DTO (Data Transfer Object)** : Les DTO sont des objets qui sont utilisés pour encapsuler les données et les transmettre entre les processus. Dans votre cas, SkieurDTO est un DTO qui contient les données d'un skieur. Il est utilisé pour envoyer les données du skieur au client.

**Service** : Les services contiennent la logique métier de l'application. Ils utilisent les repositories pour interagir avec la base de données et effectuer des opérations sur les entités. Dans votre cas, SkieurService et PaysService sont des services qui effectuent des opérations sur les skieurs et les pays respectivement.

**Controller** : Les contrôleurs gèrent les requêtes HTTP entrantes et renvoient des réponses HTTP. Ils utilisent les services pour effectuer des opérations et renvoyer les données requises. Dans votre cas, Controller est un contrôleur qui contient des méthodes pour gérer différentes requêtes HTTP.

## 2.7 Gestion des sessions

HttpSession est une interface fournie par Java Servlet qui fournit un moyen d'identifier un utilisateur à travers plusieurs requêtes HTTP. Une session est associée à un utilisateur spécifique et peut contenir des informations sur l'utilisateur à travers plusieurs connexions HTTP.

Dans l'application, on utilise HttpSession pour gérer les sessions utilisateur. Lorsqu'un utilisateur se connecte avec succès, on crée une nouvelle session et stocke le nom d'utilisateur et le nombre de visites dans la session. Le nombre de visites est incrémenté chaque fois qu'un utilisateur se connecte.

**login** : Cette méthode est appelée lorsque l'utilisateur tente de se connecter. Si les identifiants de l'utilisateur sont valides, vous créez une nouvelle session et stockez le nom d'utilisateur et le nombre de visites dans la session.

```
@PostMapping("/login")
public ResponseEntity<String> login(@RequestParam String username,
    @RequestParam String password, HttpSession session) {
    // Vérifiez les identifiants de l'utilisateur ici
    if ("Jojo".equals(username) && "test".equals(password)) {
        // Si les identifiants sont valides :
        // if (session.getAttribute("username") != null) {
        //     return ResponseEntity.status(403).body("Already logged in");
        // }
        session.setAttribute("username", username);
        Integer visites = (Integer) session.getAttribute("visites");
        if (visites == null) {
            visites = 0;
        }
        session.setAttribute("visites", visites + 1);
        return ResponseEntity.ok("Logged in with " + username);
    } else {
        // Sinon, retournez une erreur
        return ResponseEntity.status(401).body("Invalid username or
password");
    }
}
```

**logout** : Cette méthode est appelée lorsque l'utilisateur souhaite se déconnecter. Vous invalidez la session actuelle, ce qui efface toutes les informations stockées dans la session.

```
@PostMapping("/logout")
public ResponseEntity<String> logout(HttpSession session) {
    session.invalidate();
    return ResponseEntity.ok("Logged out ");
}
```

**visites** : Cette méthode est appelée pour obtenir le nombre de visites de l'utilisateur. Vous récupérez le nombre de visites de la session et le renvoyez à l'utilisateur.

```
@GetMapping("/visites")
public ResponseEntity<Integer> getVisites(HttpSession session) {
    Integer visites = (Integer) session.getAttribute("visites");
    if (visites == null) {
        return ResponseEntity.status(401).body(null); // Non autorisé
    }
    // session.setAttribute("visites", visites + 1);
    return ResponseEntity.ok(visites);
}
```

## Log in pour la première fois :

POST http://localhost:8080/user/login Send

Params Authorization Headers (9) Body Pre-request Script Tests Settings Cookies

☐ none ☐ form-data ☒ x-www-form-urlencoded ☐ raw ☐ binary ☐ GraphQL

Key	Value	Description	...	Bulk Edit
<input checked="" type="checkbox"/> username	Jojo			
<input checked="" type="checkbox"/> password	test			
Key	Value	Description		

Body Cookies (1) Headers (5) Test Results Status: 200 OK Time: 6 ms Size: 183 B Save as example

Pretty Raw Preview Visualize Text 1 Logged in with Jojo

## Get le nombre de visite

GET http://localhost:8080/user/visites Send

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

Query Params

Key	Value	Description	...	Bulk Edit
Key	Value	Description		

Body Cookies (1) Headers (5) Test Results Status: 200 OK Time: 3 ms Size: 165 B Save as example

Pretty Raw Preview Visualize JSON 1 1

## Et si je me connecte plusieurs fois

GET http://localhost:8080/user/visites Send

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

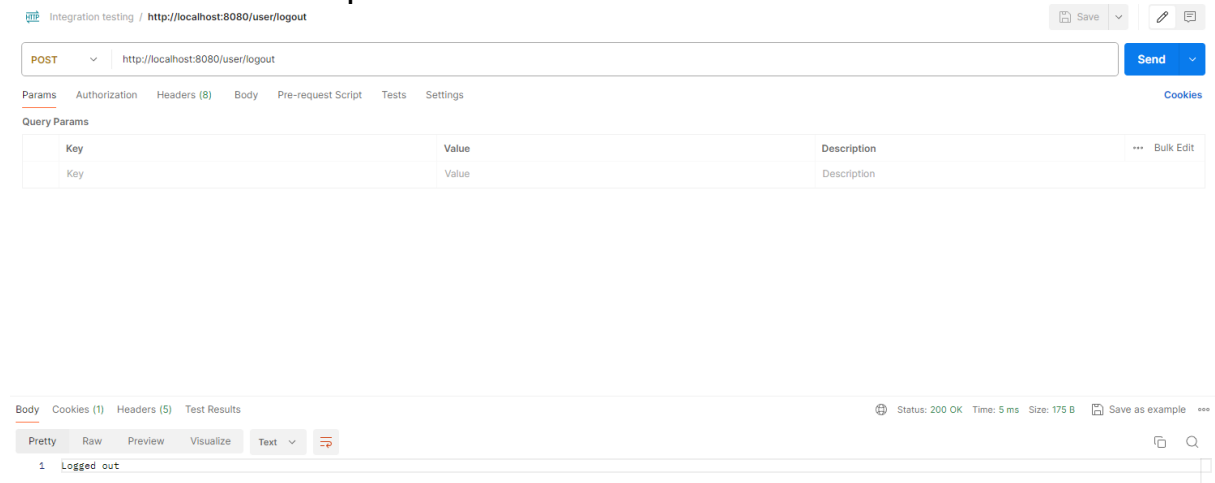
Query Params

Key	Value	Description	...	Bulk Edit
Key	Value	Description		

Body Cookies (1) Headers (5) Test Results Status: 200 OK Time: 3 ms Size: 165 B Save as example

Pretty Raw Preview Visualize JSON 1 6

## Voici le résultat lorsqu'on se déconnecte



Integration testing / http://localhost:8080/user/logout

POST http://localhost:8080/user/logout Send

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

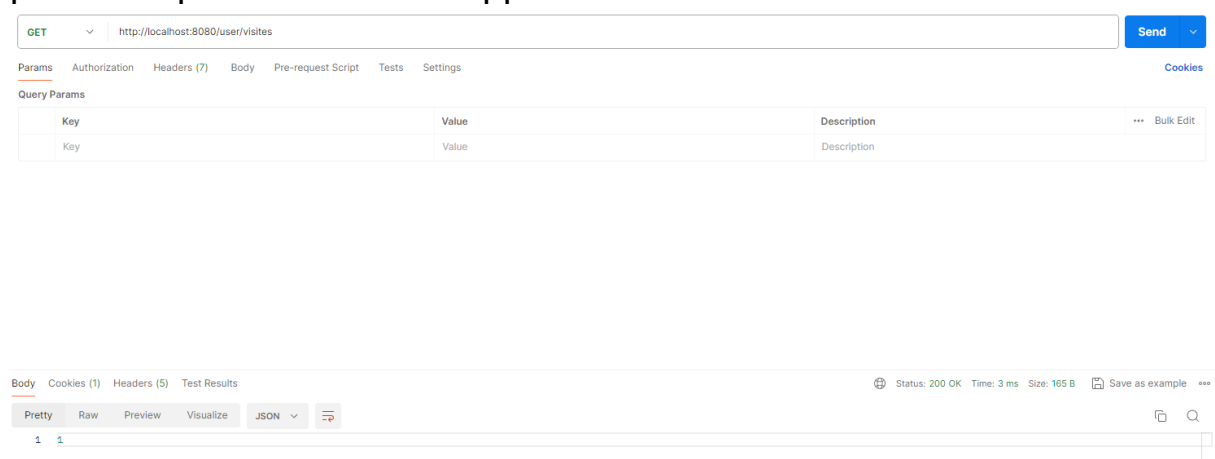
Query Params

Key	Value	Description	...	Bulk Edit
Key	Value	Description		

Body Cookies (1) Headers (5) Test Results Status: 200 OK Time: 5 ms Size: 175 B Save as example

Pretty Raw Preview Visualize Text 1 Logged out

Et voici le résultat après avoir fermé la session et l'avoir réouverte, on peut voir que tous ont été supprimer.



GET http://localhost:8080/user/visites Send

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

Query Params

Key	Value	Description	...	Bulk Edit
Key	Value	Description		

Body Cookies (1) Headers (5) Test Results Status: 200 OK Time: 3 ms Size: 165 B Save as example

Pretty Raw Preview Visualize JSON 1

## 2.8 Documentation API avec Swagger

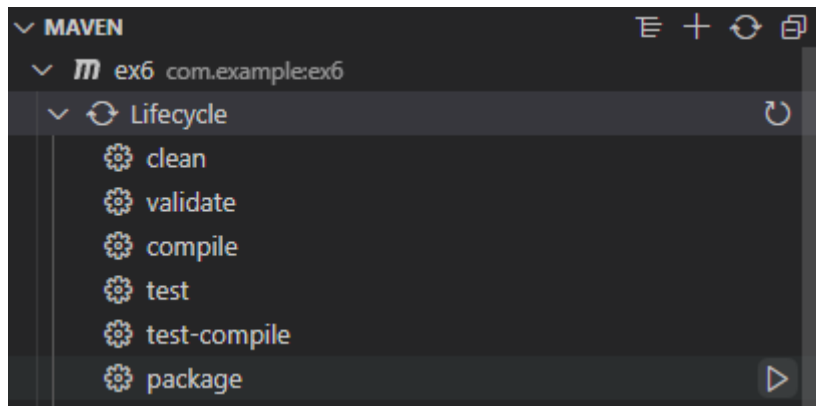
Swagger est un outil pour faire la description de notre API plus simplement. Il

Pour pouvoir utiliser Swagger, il faut rajouter ce bout de code dans les dependency de pom.xml

```
<dependency>
  <groupId>org.springdoc</groupId>
  <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
  <version>2.3.0</version>
</dependency>
```

Après avoir fait ça il suffit de lancé l'application et d'ouvrir dans le navigateur le liens suivant : <http://localhost:8080/swagger-ui/index.html>

Une erreur peut survenir lors du lancement de l'application, pour la régler il suffit de lancer « package » dans le lifecycle de maven.



Voila ce que vous devriez voir depuis votre navigateur



Tests technologiques selon les exercices

POST

/user/logout

^

Parameters

Cancel

No parameters

Execute

Clear

Responses

Curl

```
curl -X 'POST' \
  'http://localhost:8080/user/logout' \
  -H 'accept: */*' \
  -d ''
```

Request URL

http://localhost:8080/user/logout

Server response

Code

Details

200

Response body

Logged out

Download

Response headers

connection: keep-alive  
content-length: 11  
content-type: text/plain;charset=UTF-8  
date: Thu, 29 Mar 2024 14:17:06 GMT  
keep-alive: timeout=60

Responses

Code

Description

Links

200

OK

No links

Media type

\*/\*

Controls

Accept header

Example Value

Schema

string

## 3 Analyse à faire complètement avec EA - > à rendre uniquement le fichier EA

### 3.1 Description de projet

#### 3.1.1 Site de Johan

Lorsque que l'on arrive sur le site, on arrive sur une page de connexion, si on rentre les identifiants d'un compte qui est admin, on est amené à la page principale.

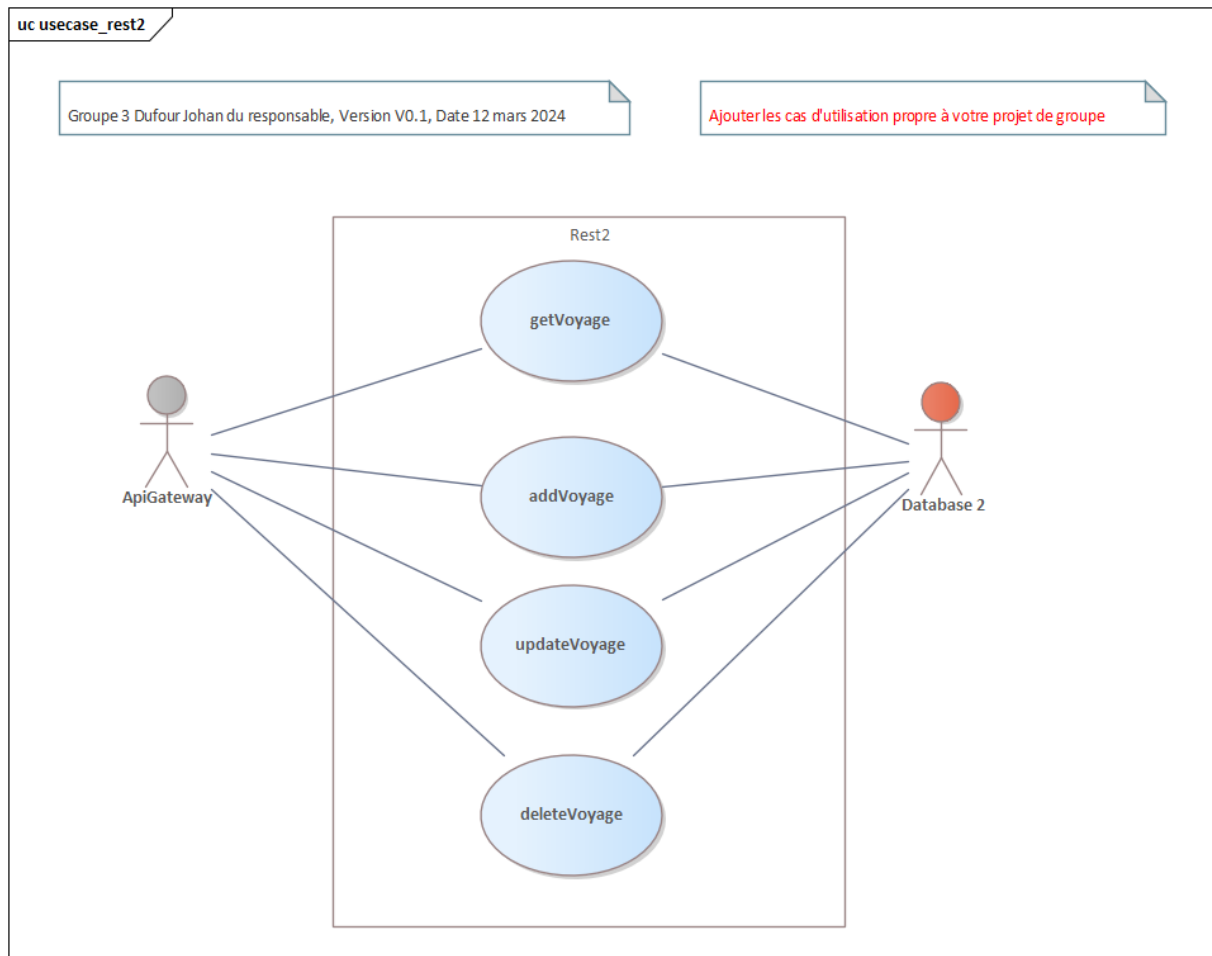
Sur la page principale, on verrait la liste de tous les voyages.

On pourrait aussi ajouter ou supprimer des voyages. Ou bien modifier un voyage existant.

### 3.2 Use case client et use case Rest

L'API peut soit, récupérer la liste des voyages, en ajouter un nouveaux, en modifier un, en supprimer un.





### 3.3 Activity Diagram d'un cas complet navigant dans les applications avec les explications

Lorsque que l'utilisateur arrive sur la page, il tombera sur la page de login. Si ses identifiants sont juste alors il accédera à la page principale.

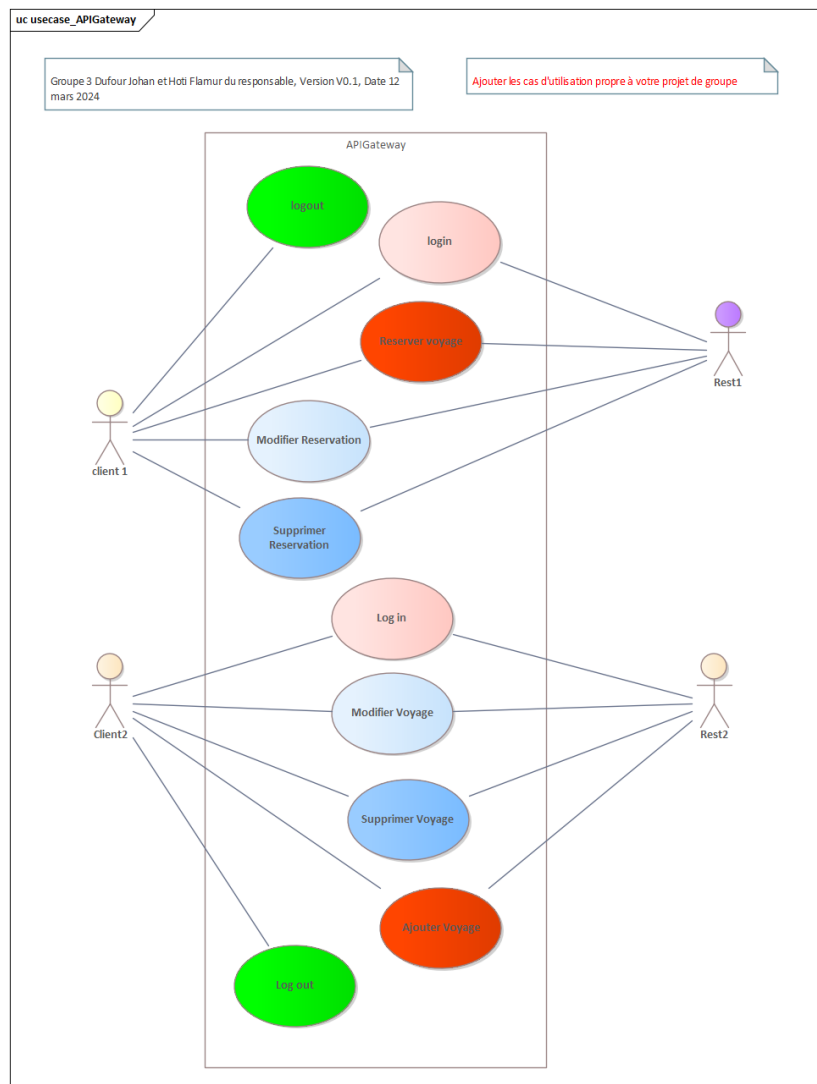
Sur la page principale, s'il s'est connecté avec un compte administrateur, il pourra soit : ajouter un voyage, modifier un voyage, supprimer un voyage ou se déconnecter.

Analyse à faire complètement avec EA -> à rendre uniquement le fichier EA



Pour le client 2, il peut soit se connecter, ajouter un voyage, modifier un voyage, supprimer un voyage ou se déconnecter.

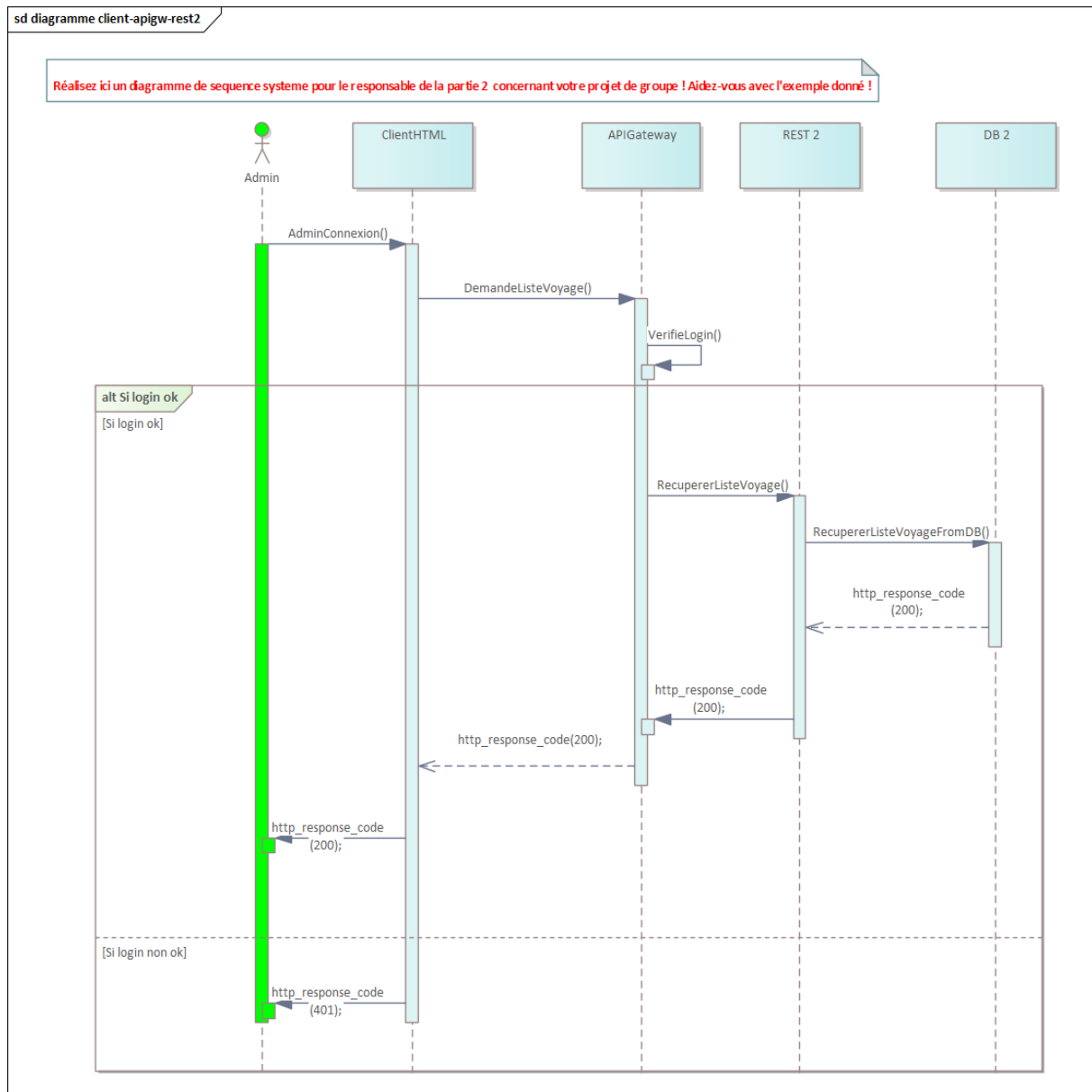
Analyse à faire complètement avec EA -> à rendre uniquement le fichier EA



### 3.4 Sequence System global entre les applications

Voici le diagramme de sequence system pour le chargement de la liste de voyage a la connexion d'un administrateur.

Analyse à faire complètement avec EA -> à rendre uniquement le fichier EA



AdminConnection : L'administrateur se connecte au système.

DemandeListeVoyage : Une fois connecté, l'administrateur demande la liste des voyages.

VerifLogin : Le système vérifie les informations de connexion de l'administrateur.

RecupererListeVoyage : Si la vérification est réussie, le système récupère la liste des voyages depuis la base de données (DB 2).

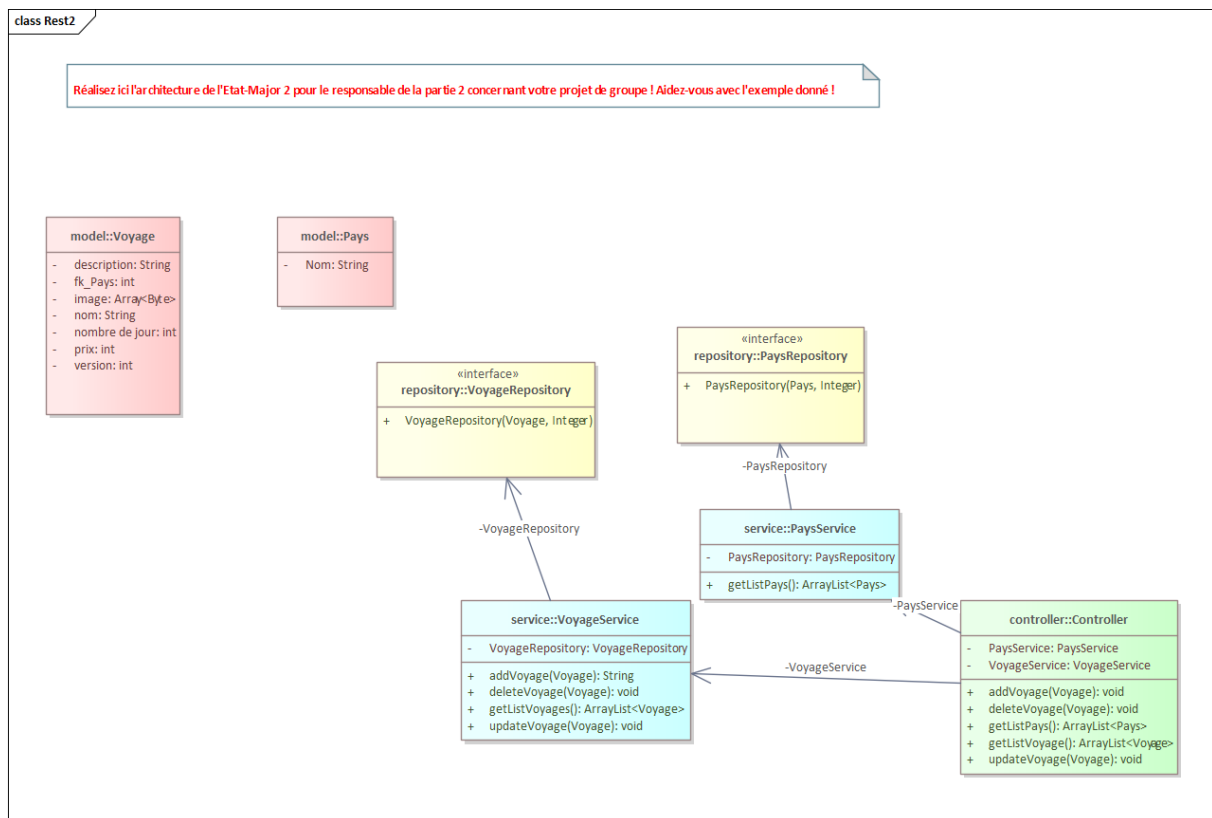
Si le login est valable, alors le liste va être renvoyer avec une réponse http 200

Sinon, si il n'est pas valable alors ça sera une erreur 400.

## 4 Conception à faire complètement avec EA -> à rendre uniquement le fichier EA

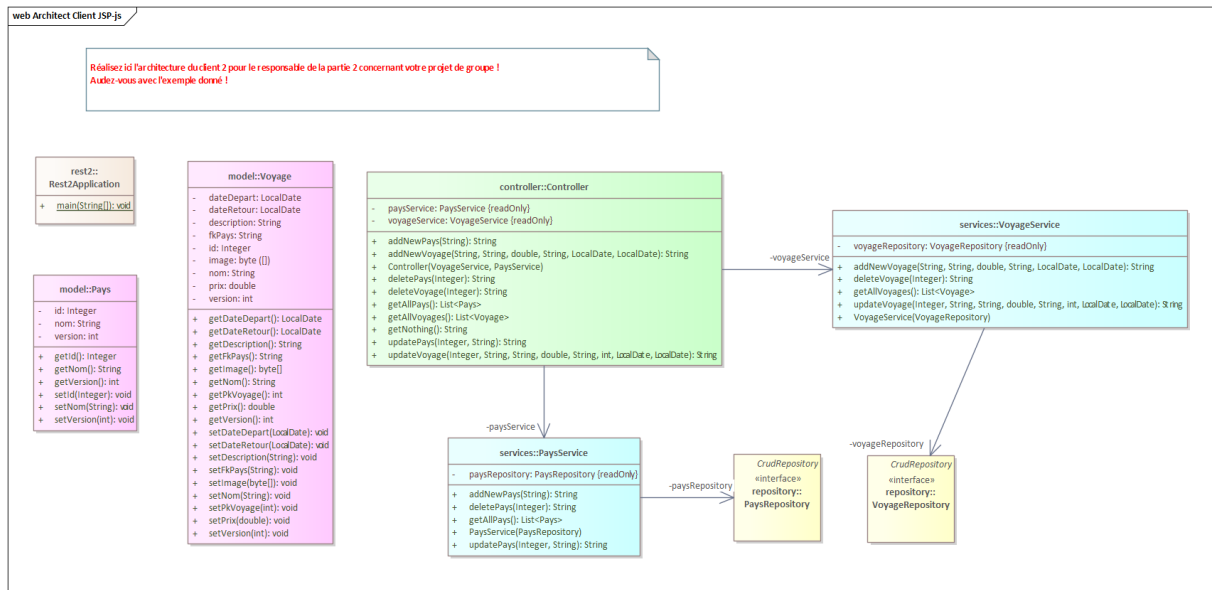
### 4.1 Class Diagram complet avec les explications de chaque application

Voici mon diagramme de classe initial.



Et voici le reverse après avoir fini le back end

## Conception à faire complètement avec EA -> à rendre uniquement le fichier EA



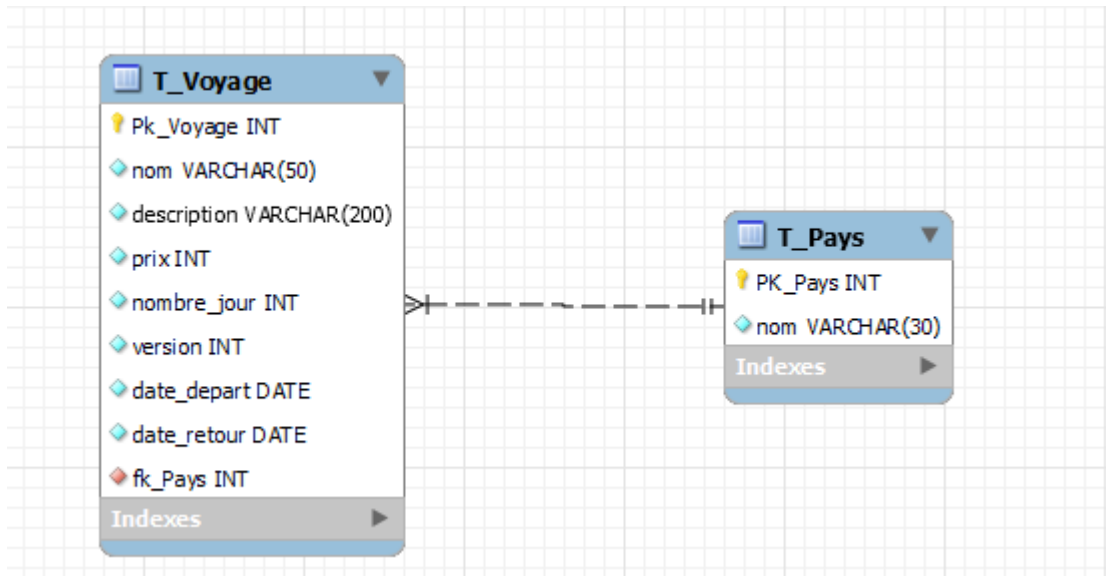
Voici les différences :

- Il y a les getters setters pour les models
- Le model voyage n'a plus d'attribut « nombre de jour » mais a la place « date de départ » et « date de retour »
- Le contrôleur a désormais aussi les méthodes CRUD pour les pays
- Le voyageService est resté le même, le constructeur a simplement été rajouté
- Les cruds on été ajouté pour PaysService

## 5 Bases de données

### 5.1 Modèles WorkBench MySQL

La DB contient 2 tables, un table qui s'occupe des voyages (l'image n'est pas à jour mais je n'ai actuellement pas accès à la DB) et une autre table qui s'occupe des pays.



## 6 Implémentation des applications <Le client Ap1> et <Le client Ap2>

### 6.1 Une descente de code client

Il n'y a pas encore de code sur le client, seulement de l'html

The screenshot shows a web application interface titled "DH Destinations Gestion" in the top left corner and a "Log out" button in the top right corner. The interface is divided into two main sections. On the left, there is a form for selecting a type and an element. The "Sélectionnez un type :" dropdown menu is set to "Voyage". Below it, the "Sélectionnez un élément :" section shows a list of "Voyage 1", "Voyage 2", and "Voyage 3". At the bottom of this section are three buttons: "Ajouter" (blue), "Modifier" (orange), and "Supprimer" (red). On the right, there is a form for entering details. It includes fields for "Nom :", "Description :", "Prix :", "Pays :", "Nombre de jour :", "Date de départ :", "Date de retour :", and "Pays:". Each field has a placeholder text "Entrez du texte ici".



## 7 Implémentation de l'application <API Gateway>

### 7.1 Une descente de code APIGateway

#### 7.1.1 Contrôleur

```
@GetMapping("/getAllVoyages")
public ResponseEntity<String> getAllVoyages() {
    try {
        // Appelle la méthode du service
        ResponseEntity<String> response = serviceApiRest2.getAllVoyages();

        // Vérifie si la réponse est réussie (code d'état 200)
        if (response.getStatusCode().is2xxSuccessful()) {
            // Retourne HTTP 200 avec le corps de la réponse en cas de succès
            return ResponseEntity.ok(response.getBody());
        } else {
            // Retourne HTTP 400 avec un message d'erreur en cas d'échec
            return ResponseEntity.badRequest().body("Échec de la récupération des données");
        }
    } catch (Exception e) {
        // Retourne HTTP 400 avec un message d'erreur en cas d'exception
        return ResponseEntity.badRequest().body("Erreur : " + e.getMessage());
    }
}
```

#### 7.1.2 ServiceApiRest2

```
public ResponseEntity<String> getAllVoyages() {
    String url = Rest2Url + "/getAllVoyages";

    try {
        // Effectuer une requête GET
        ResponseEntity<String> response = restTemplate.getForEntity(url, String.class);

        // Vérifier si la réponse est réussie (code d'état 200)
        if (response.getStatusCode().is2xxSuccessful()) {
            // Traiter la réponse si nécessaire
            // Pour l'instant, renvoyons simplement le corps de la réponse
            return ResponseEntity.ok(response.getBody());
        } else {
            // Gérer les erreurs si nécessaire
            return ResponseEntity.badRequest().body("Échec de la récupération des données");
        }
    } catch (Exception e) {
        // Gérer les exceptions (par exemple, erreurs réseau)
        return ResponseEntity.badRequest().body("Erreur : " + e.getMessage());
    }
}
```

## Implémentation de l'application <API Gateway>

```
}  
}
```

## 8 Implémentation des applications <API élève1> et <API élève2>

### 8.1 Une descente de code de l'API REST

```
public List<Voyage> getAllVoyages() {  
    // Récupère tous les voyages de la base de données  
    Iterable<Voyage> voyageIterable = voyageRepository.findAll();  
  
    // Convertit l'itérable en une liste en utilisant StreamSupport et Collectors  
    List<Voyage> voyageList = StreamSupport.stream(voyageIterable.splitterator(),  
false)  
                                .collect(Collectors.toList());  
  
    // Retourne la liste de voyages  
    return voyageList;  
}
```

## 9 Hébergement

Malheureusement, nous n'avons pas eu le temps de faire cette partie.

## 10 Installation du projet complet avec les 5 applications

Malheureusement nous n'avons pas eu le temps de finaliser cette partie.

# 11 Tests de fonctionnement du projet

Voici ce que j'ai utilisé pour tester mon application depuis des conteneurs locaux. Chaque méthode a un retour normalement.

[HTTP](#) Integration testing / **addPays**

**POST** ▼ http://localhost:8080/addNewPays

Params Authorization Headers (9) **Body ●** Scripts Settings

☐ none ☐ form-data ☒ x-www-form-urlencoded ☐ raw ☐ binary ☐ GraphQL

	Key	Value
<input checked="" type="checkbox"/>	name	VIVI
<input type="checkbox"/>	password	test
	Key	Value

[HTTP](#) Integration testing / **getPays**

**GET** ▼ http://localhost:8080/getAllPays

Params Authorization Headers (6) **Body ●** Scripts Settings

☐ none ☐ form-data ☒ x-www-form-urlencoded ☐ raw ☐ binary ☐ GraphQL

**DELETE** ▼ http://localhost:8080/deletePays/19

Params Authorization Headers (6) **Body ●** Scripts Settings

☐ none ☐ form-data ☒ x-www-form-urlencoded ☐ raw ☐ binary ☐ GraphQL


## Tests de fonctionnement du projet

**PUT** ▼ http://localhost:8080/updatePays/16

Params Authorization Headers (8) **Body** ● Scripts Settings

☐ none ☐ form-data ☒ x-www-form-urlencoded ☐ raw ☐ binary ☐ GraphQL

	Key	Value
<input checked="" type="checkbox"/>	name	AlbaniePASLOL
<input type="checkbox"/>	password	test
	Key	Value

 Integration testing / **addVoyage**

**POST** ▼ http://localhost:8080/addNewVoyage

Params Authorization Headers (9) **Body** ● Scripts Settings

☐ none ☐ form-data ☒ x-www-form-urlencoded ☐ raw ☐ binary ☐ GraphQL

	Key	Value
<input checked="" type="checkbox"/>	name	test 3
<input checked="" type="checkbox"/>	description	Un voyage de 10 jours pour explorer les magnifiques paysages des Rocheuse...
<input checked="" type="checkbox"/>	prix	900
<input checked="" type="checkbox"/>	fkPays	9
<input checked="" type="checkbox"/>	dateDepart	15-07-2024
<input checked="" type="checkbox"/>	dateRetour	20-07-2024

 Integration testing / **updateVoyage**

**PUT** ▼ http://localhost:8080/updateVoyage/17

Params Authorization Headers (9) **Body** ● Scripts Settings

☐ none ☐ form-data ☒ x-www-form-urlencoded ☐ raw ☐ binary ☐ GraphQL


	Key	Value
<input checked="" type="checkbox"/>	name	Voyage à Paris
<input checked="" type="checkbox"/>	description	n voyage de 7 jours dans la ville lumière, avec des visites à la Tour Eiffel, au L...
<input checked="" type="checkbox"/>	prix	21000
<input checked="" type="checkbox"/>	fkPays	2
<input checked="" type="checkbox"/>	dateDepart	01-07-2024
<input checked="" type="checkbox"/>	dateRetour	07-07-2024


## Tests de fonctionnement du projet

 Integration testing / **deleteVoyage**

**DELETE**  `http://localhost:8080/deleteVoyage/17`

Params   Authorization   Headers (9)   Body    Scripts   Settings

 Integration testing / **getVoyage**

**GET**  `http://localhost:8080/getAllVoyages`

Params   Authorization   Headers (9)   Body    Scripts   Settings



## 12 Auto-évaluations et conclusions

### 12.1 Auto-évaluation et conclusion de Johan

Ce module m'a permis de développer mes compétences techniques et conceptuelles dans le domaine du développement d'applications web.

J'ai aussi appris à utiliser docker, qui était, avant ce module, un concept éloigné.

Pour conclure, j'ai beaucoup aimé ce module, malgré la quantité de travail élevé. La façon de travailler en groupe était bien aussi, tout comme le fait d'avoir eu quelques pauses libres, même si je n'en ai pas beaucoup profité.