



# 133 - Réaliser des applications Web en Session-Handling

## Rapport Personnel

**David Morisetti**

**Module du 21.03.2024 au 03.05.2024**

**Date de création : 21.03.2024**

**Version 1 du 01.05.2024**

# Table des matières

1	Introduction et contexte du projet .....	3
2	Tests technologiques selon les exercices .....	4
2.1	Installation et Hello World .....	4
2.2	Conteneurisation .....	4
2.3	Création d'un projet Spring Boot.....	4
2.4	Connexion à la DB JDBC .....	5
2.5	Connexion à la DB JPA .....	5
2.6	Connexion à la DB JPA avec DTO .....	5
2.7	Gestion des sessions .....	6
2.8	Documentation API avec Swagger.....	7
2.9	Hébergement .....	8
3	Analyse à faire complètement avec EA -> à rendre uniquement le fichier EA .....	9
3.1.1	Use case client et use case Rest .....	9
3.1.2	Activity Diagram d'un cas complet navigant dans les applications avec les explications <b>Error! Bookmark not defined.</b>	
3.1.3	Sequence System global entre les applications.....	15
4	Conception à faire complètement avec EA -> à rendre uniquement le fichier EA .....	17
4.1	Class Diagram complet avec les explications de chaque application .....	17
5	Bases de données.....	19
5.1	Modèles WorkBench MySQL.....	19
6	Implémentation des applications <Le client Ap1> et <Le client Ap2> .....	20
6.1	Une descente de code client.....	20
7	Implémentation de l'application <API Gateway> .....	21
7.1	Une descente de code APIGateway.....	21
8	Implémentation des applications <API élève1> et <API élève2>.....	22
8.1	Une descente de code de l'API REST .....	22
9	Hébergement.....	23
10	Installation du projet complet avec les 5 applications .....	24
11	Tests de fonctionnement du projet .....	25
12	Auto-évaluations et conclusions .....	26
12.1	Auto-évaluation et conclusion de ... ..	26
12.2	Auto-évaluation et conclusion de ... ..	26

# 1 Introduction et contexte du projet

Le projet se nomme **YouQuiz**. Il permettra à des utilisateurs connectés de remplir des quiz ou d'en créer pour les administrateurs.

Premièrement il y aura la partie qui permettra de créer des quiz pour les utilisateurs connectés sur celui-ci. Ensuite il y aura la deuxième partie qui permet à un utilisateur connecté de remplir des quiz. Un utilisateur peut aussi liker les quiz, ceux-ci seront ainsi triés par leur nombre de likes.

Les quiz consistent en plusieurs questions qui ont plusieurs réponses possibles. Le choix de réponse peut être unique, multiples ou vrai ou faux.

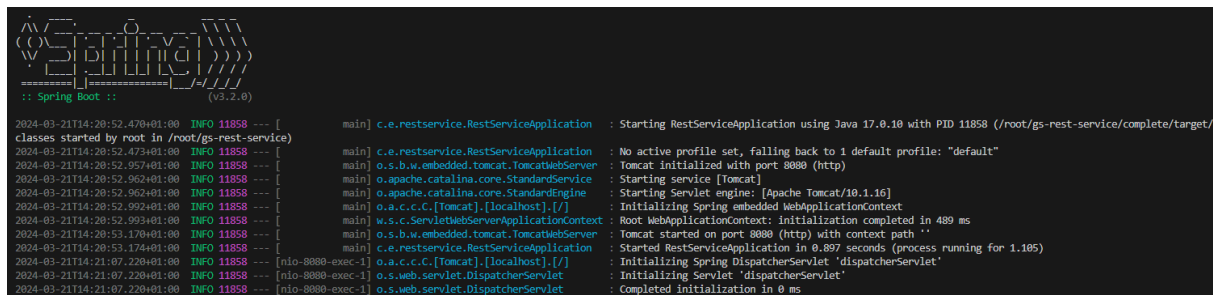
Je vais m'occuper de la partie permettant la création de quiz et Matteo de la partie permettant de remplir les sites.

## 2 Tests technologiques selon les exercices

### 2.1 Installation et Hello World

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

Le projet se lance avec Spring Boot sur le port 8080.



```
2024-03-21T14:20:52.470+01:00 INFO 11858 --- [main] c.e.restservice.RestServiceApplication : Starting RestServiceApplication using Java 17.0.10 with PID 11858 (/root/gs-rest-service/complete/target/
Classes started by root in /root/gs-rest-service)
2024-03-21T14:20:52.473+01:00 INFO 11858 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port 8080 (http)
2024-03-21T14:20:52.957+01:00 INFO 11858 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2024-03-21T14:20:52.962+01:00 INFO 11858 --- [main] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.16]
2024-03-21T14:20:52.992+01:00 INFO 11858 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2024-03-21T14:20:52.993+01:00 INFO 11858 --- [main] w.a.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 489 ms
2024-03-21T14:20:53.170+01:00 INFO 11858 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8080 (http) with context path ''
2024-03-21T14:20:53.174+01:00 INFO 11858 --- [main] c.e.restservice.RestServiceApplication : Started RestServiceApplication in 0.897 seconds (process running for 1.105)
2024-03-21T14:21:07.220+01:00 INFO 11858 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2024-03-21T14:21:07.220+01:00 INFO 11858 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2024-03-21T14:21:07.220+01:00 INFO 11858 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 0 ms
```

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

C'est Java17, Maven

Y a-t-il un serveur web, quelle version utilise-t-il ?

Oui, Apache Tomcat/10.1.16

### 2.2 Conteneurisation

Pourquoi faire un container pour une application Java ?

Pour avoir une application qui marche partout.

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

Oui, Tomcat, compilé dans le jar.

A quoi faut-il faire attention (pensez aux versions!) ?

Changé la version de java dans le pom.xml et dans le Dockerfile.

### 2.3 Création d'un projet Spring Boot

Quelles sont les annotations utilisée (commencent par @) dans votre controller ?

- **GetMapping** : Cette annotation est utilisée pour mapper les requêtes HTTP GET sur des méthodes de gestion spécifiques. Par exemple, `@GetMapping("/home")` mappe les requêtes GET à l'URL `"/home"` à la méthode de gestion correspondante.
- **PostMapping** : Cette annotation est utilisée pour mapper les requêtes HTTP POST sur des méthodes de gestion spécifiques. Par exemple, `@PostMapping("/submit")` mappe les requêtes POST à l'URL `"/submit"` à la méthode de gestion correspondante.

- **PutMapping** : Cette annotation est utilisée pour mapper les requêtes HTTP PUT sur des méthodes de gestion spécifiques. Par exemple, `@PutMapping("/update")` mappe les requêtes PUT à l'URL `"/update"` à la méthode de gestion correspondante.
- **RequestBody** : Cette annotation est utilisée pour lier le corps de la requête HTTP à un objet de domaine dans une méthode de gestion. Le corps de la requête HTTP est passé dans une forme appropriée à la méthode de gestion.
- **RequestParam** : Cette annotation est utilisée pour lier les paramètres de requête à une méthode de gestion. Par exemple, `@RequestParam("id") String id` lie le paramètre de requête `"id"` à la variable `String "id"`.
- **RestController** : Cette annotation est utilisée au niveau de la classe pour indiquer qu'une classe est un contrôleur où les méthodes de gestion renvoient le corps de la réponse directement, et non une vue. Elle est une combinaison de `@Controller` et `@ResponseBody`.

## 2.4 Connexion à la DB JDBC

Comment transformer en JSON :

```
ArrayList<String> pays = wrk.getPays();
ObjectMapper mapper = new ObjectMapper();

try {
    String json = mapper.writeValueAsString(pays);
    return json;
} catch (IOException e) {
    e.printStackTrace();
    return "Erreur";
}
```

## 2.5 Connexion à la DB JPA

**À quoi sert l'annotation `@Autowired` dans vos contrôleur pour les Repository ?**

L'annotation `@Autowired` dans Spring est utilisée pour l'injection automatique des dépendances. Cela signifie que Spring va chercher et instancier automatiquement le bean qui correspond à la dépendance déclarée.

**A quoi sert l'annotation `@ManyToOne` dans l'entité skieur ?**

L'annotation `@ManyToOne` est utilisée pour établir une relation de plusieurs à un entre deux entités dans votre modèle JPA.

**Sur la même ligne, quel `FetchType` est utilisé et pourquoi, réessayer avec le `FetchType LAZY` et faites un `getSkieur`.**

Le `FetchType` est `EAGER`, pour aller chercher le pays directement. Si j'utilise le `FetchType Lazy` j'obtiens une erreur en transformant en JSON les skieurs car le pays n'est pas défini.

## 2.6 Connexion à la DB JPA avec DTO

**Pourquoi dans ce cas, on retrouve un `SkierDTO` et pas de `PaysDTO` ?**

Car **Pays** n'a pas de fk et donc pas besoin de fetch d'autre objets.

### **À quoi servent les model, les repository, les dto, les services et les controleurs ?**

- **Model** : Les modèles représentent les entités de votre domaine d'application. Dans ce cas, Skieur et Pays sont des modèles qui représentent les skieurs et les pays dans votre application.
- **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. Par exemple, SkieurRepository est une interface qui fournit des méthodes pour interagir avec les données de Skieur dans la base de données.
- **DTO (Data Transfer Object)** : Les DTO sont des objets qui encapsulent les données qui doivent être transférées entre les différentes couches de l'application. Par exemple, SkieurDTO est un objet qui contient les données d'un Skieur qui doivent être transférées de la couche de service à la couche de contrôleur.
- **Service** : Les services contiennent la logique métier de l'application. Ils coordonnent les opérations entre les différentes parties de l'application, comme l'interaction avec les repositories pour récupérer ou enregistrer des données.
- **Controller** : Les contrôleurs gèrent les interactions avec l'utilisateur. Ils reçoivent les requêtes de l'utilisateur, appellent les services appropriés pour traiter ces requêtes, et renvoient les réponses appropriées. Par exemple, dans votre code, Controller est une classe qui gère les requêtes HTTP et renvoie les réponses appropriées.

## **2.7 Gestion des sessions**

### **HttpSession**

HttpSession est une interface dans le package **javax.servlet.http** de Java Servlet API. Elle fournit un moyen de suivre l'état d'un utilisateur entre plusieurs requêtes HTTP, ce qui est essentiel pour créer des applications web interactives.

### **Création d'une session**

Lorsqu'un client fait une première requête à un serveur qui utilise **HttpSession**, une session est automatiquement créée. Le serveur génère un identifiant de session unique pour cette nouvelle session et l'envoie au client, généralement sous la forme d'un cookie.

### **Accès à la session**

Une fois la session créée, vous pouvez y accéder dans vos méthodes de contrôleur en utilisant l'injection de dépendances de Spring. Vous pouvez injecter **HttpSession** comme paramètre de méthode :

```
public ResponseEntity<String> maMethode(HttpSession session) {  
    // votre code ici  
}
```

### **Utilisation de la session**

Vous pouvez utiliser la session pour stocker et récupérer des attributs. Par exemple, vous pouvez stocker le nom d'utilisateur de l'utilisateur connecté et un compteur de visites :

```
session.setAttribute("username", username);
```

```
session.setAttribute("visites", 0);
```

Et vous pouvez récupérer ces attributs plus tard :

```
String username = (String) session.getAttribute("username");  
Integer visites = (Integer) session.getAttribute("visites");
```

### Destruction de la session

Il est possible de détruire la session en appelant **session.invalidate()** :

```
@PostMapping(value = "/logout")  
public ResponseEntity<String> logout(HttpSession session) {  
    session.invalidate();  
    return ResponseEntity.ok("Logout réussi");  
}
```

Dans cet exemple, **session.invalidate()** détruit la session HTTP, supprimant toutes les informations stockées dans la session.

## 2.8 Documentation API avec Swagger

Swagger est un ensemble d'outils open-source construits autour de la spécification OpenAPI qui peut vous aider à concevoir, construire, documenter et consommer des API RESTful. Springdoc est une bibliothèque qui simplifie la génération de documentation OpenAPI pour les applications Spring Boot.

### Dépendance

Pour utiliser Springdoc dans votre projet, vous devez ajouter la dépendance suivante à votre fichier **pom.xml** :

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

Cette dépendance inclut Springdoc OpenAPI et l'interface utilisateur Swagger, qui vous permet de visualiser et d'interagir avec votre API directement depuis votre navigateur.

### Utilisation

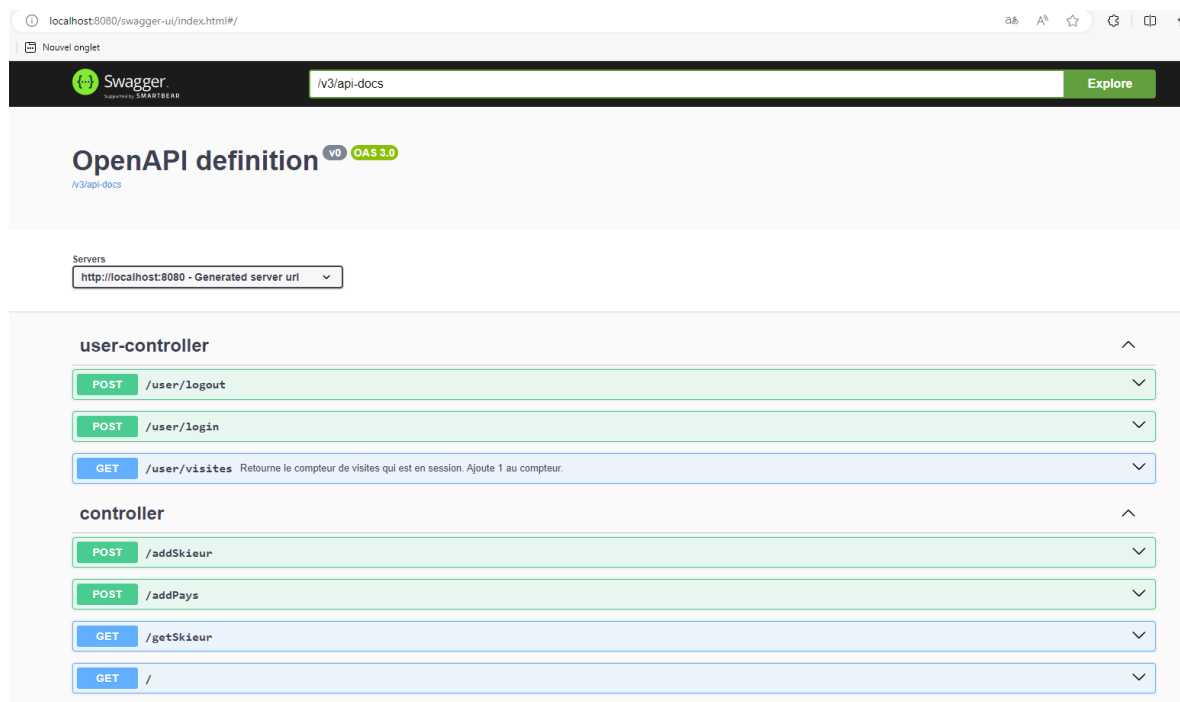
Une fois que vous avez ajouté la dépendance Springdoc à votre projet, la documentation OpenAPI est automatiquement générée à partir de votre code source. Vous pouvez accéder à cette documentation en visitant l'URL **/swagger-ui.html** de votre application.

Vous pouvez personnaliser la documentation générée en utilisant diverses annotations de Springdoc et de Swagger dans votre code. Par exemple, vous pouvez utiliser l'annotation **@Operation** pour décrire une opération d'API, et l'annotation **@ApiResponse** pour décrire une réponse possible de l'API.

Voici un exemple de comment vous pouvez utiliser ces annotations :

```
@GetMapping("/mon-api")
@Operation(summary = "Ceci est un résumé de mon API")
@ApiResponses(value = {
    @ApiResponse(responseCode = "200", description = "Succès"),
    @ApiResponse(responseCode = "404", description = "Non trouvé")
})
public ResponseEntity<String> monApi() {
    // votre code ici
}
```

Dans cet exemple, **@Operation(summary = "Ceci est un résumé de mon API")** ajoute une description à l'opération d'API, et **@ApiResponse(responseCode = "404", description = "Non trouvé")** documente qu'une réponse avec le code de statut HTTP 404 est possible.



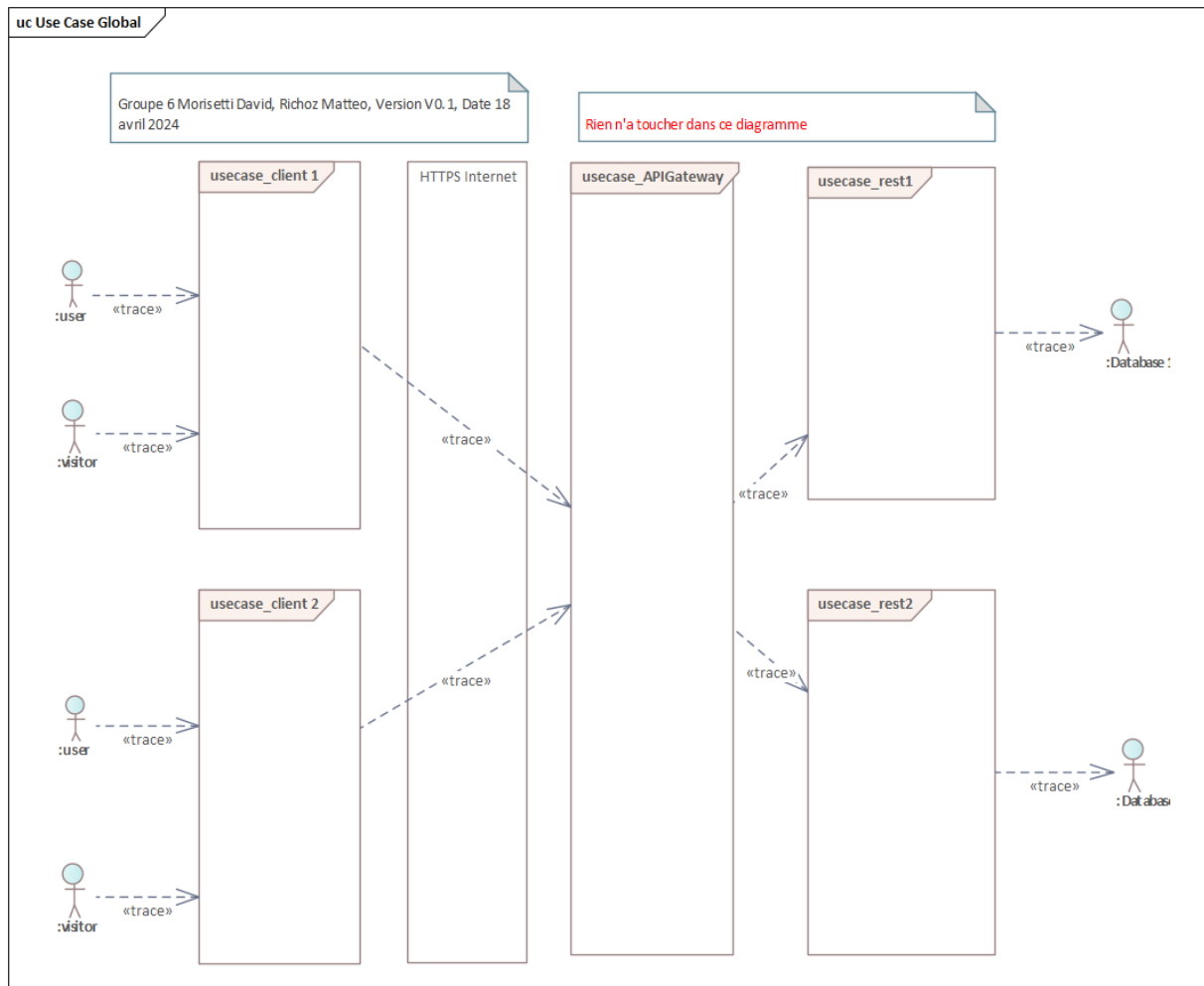
## 2.9 Hébergement



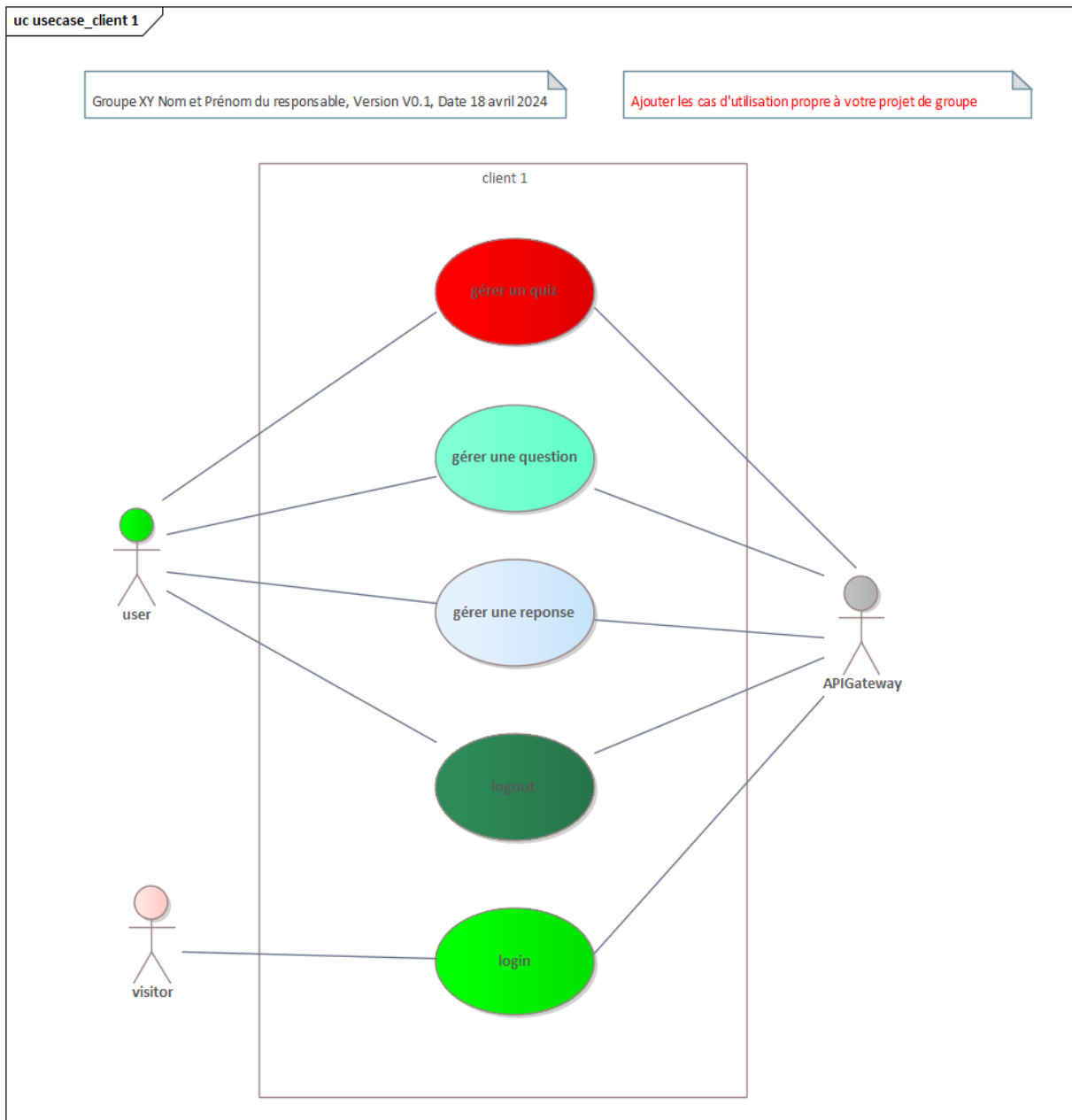
### 3 Analyse à faire complètement avec EA -> à rendre uniquement le fichier EA

#### 3.1.1 Use case client et use case Rest

Use case global :



Use case client 1 :



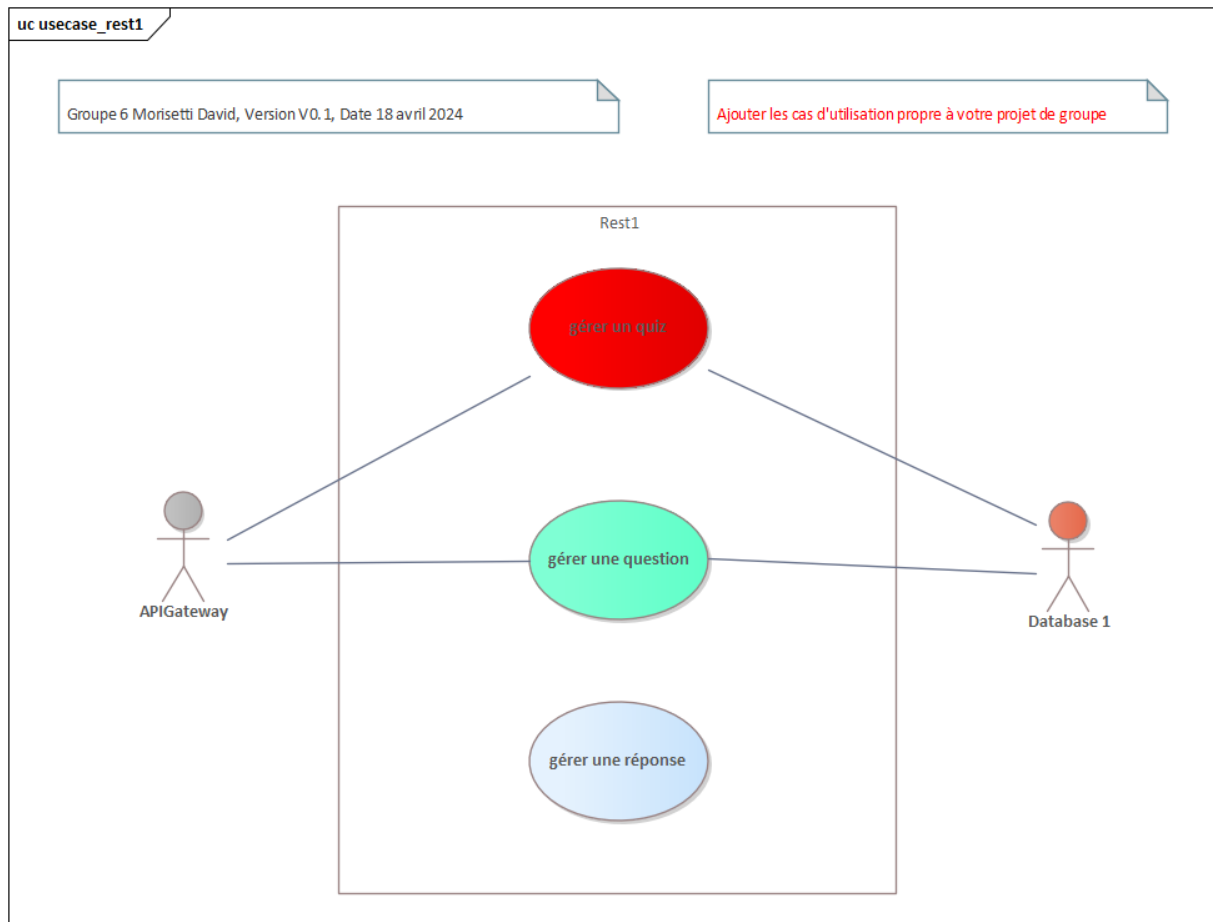
Use case client 2 :



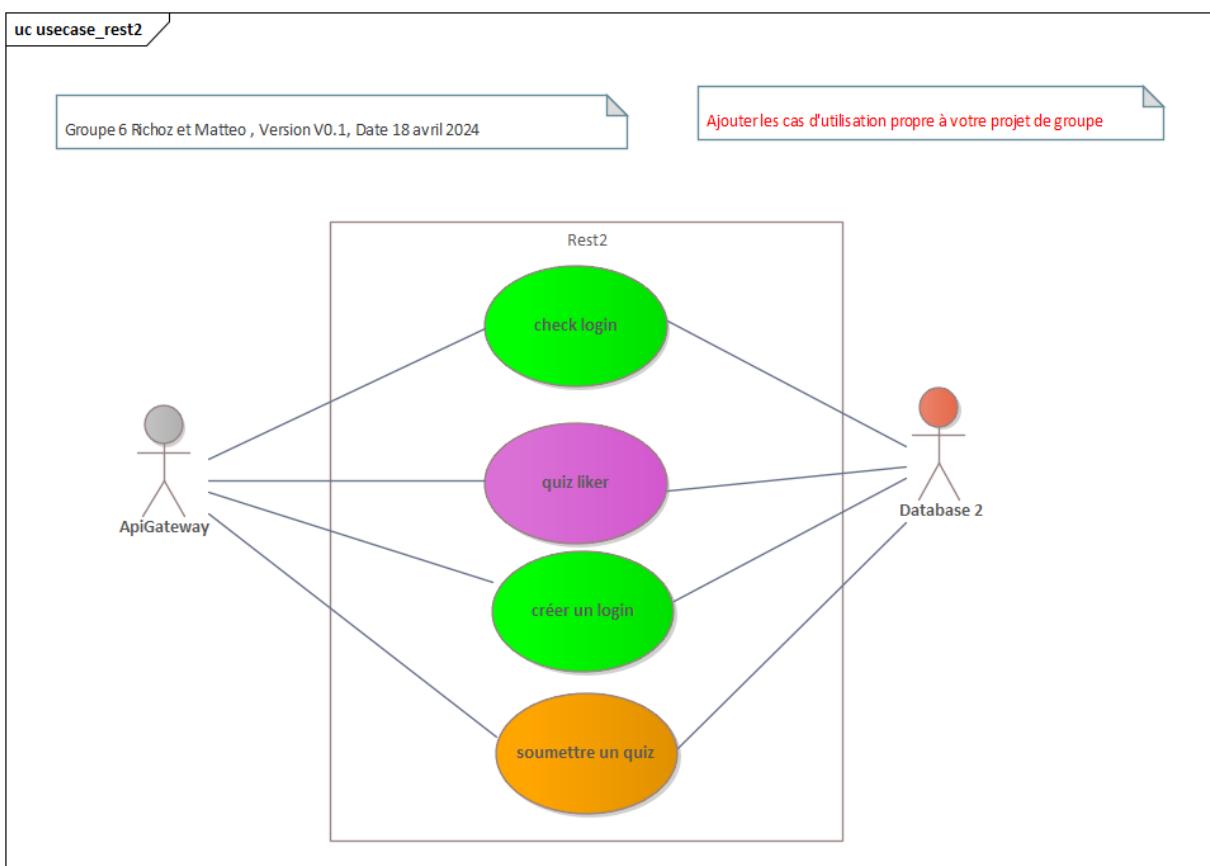
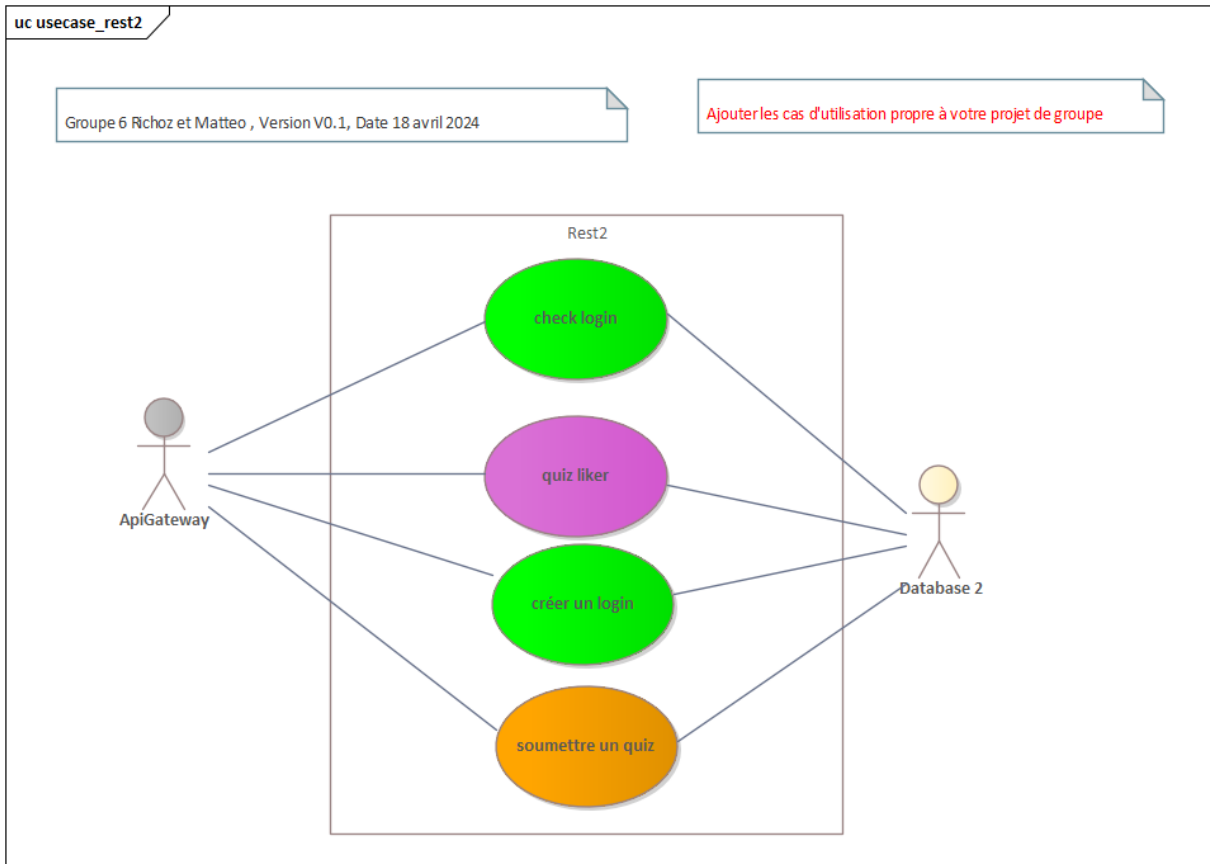
Use case API gateway :



Use case REST 1 :

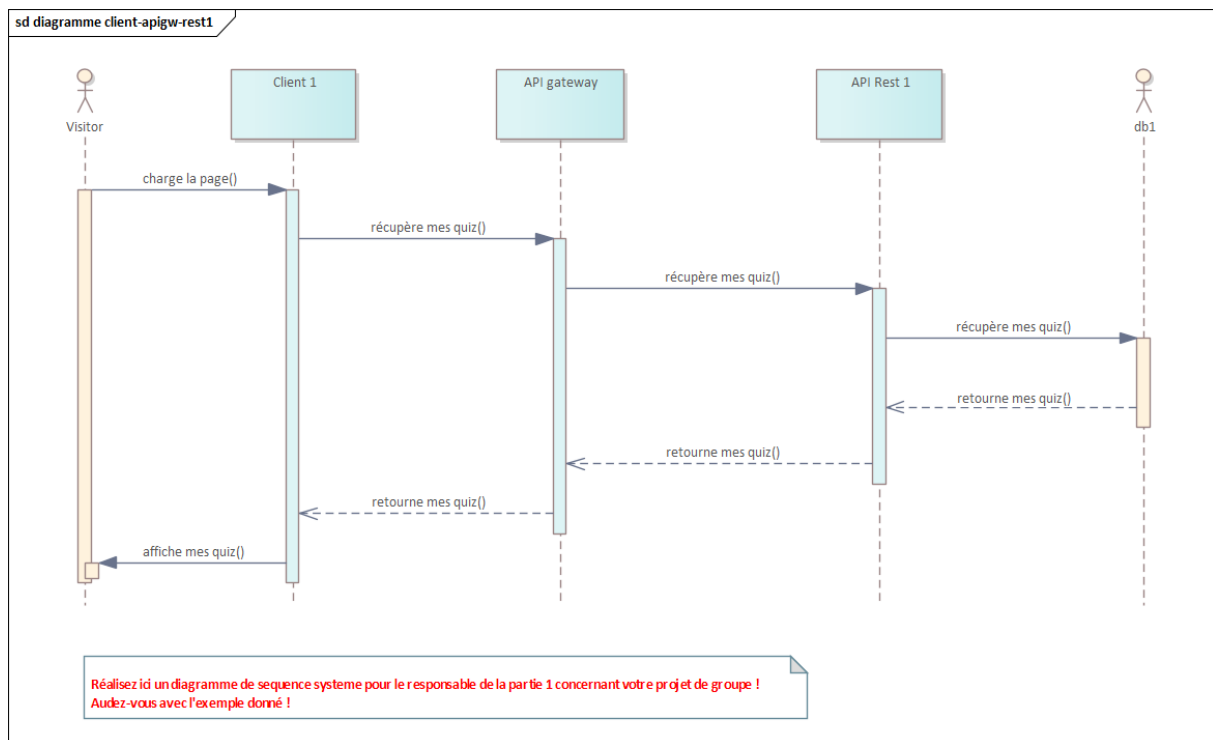


Use case REST 2 :

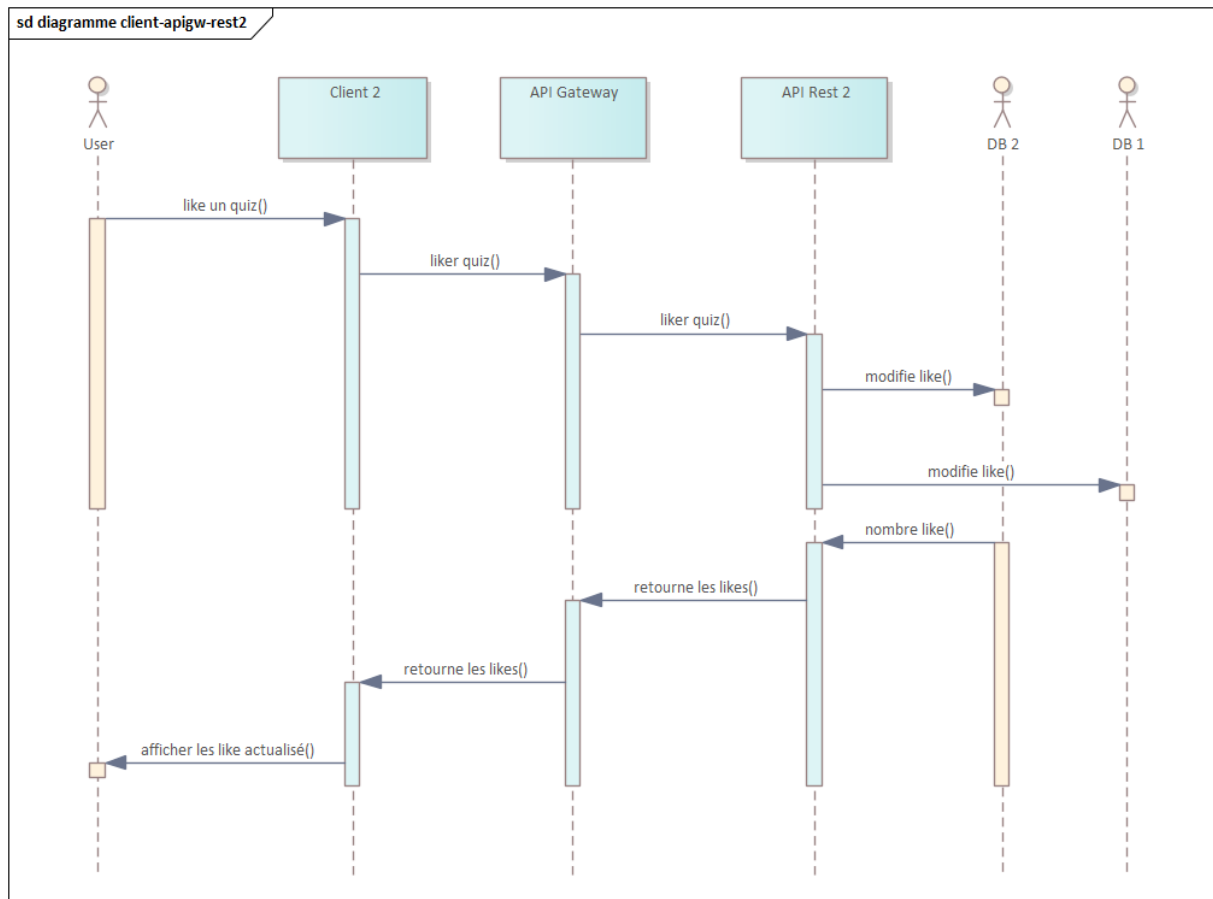


### 3.1.2 Sequence System

Rest 1 :



Rest 2 :

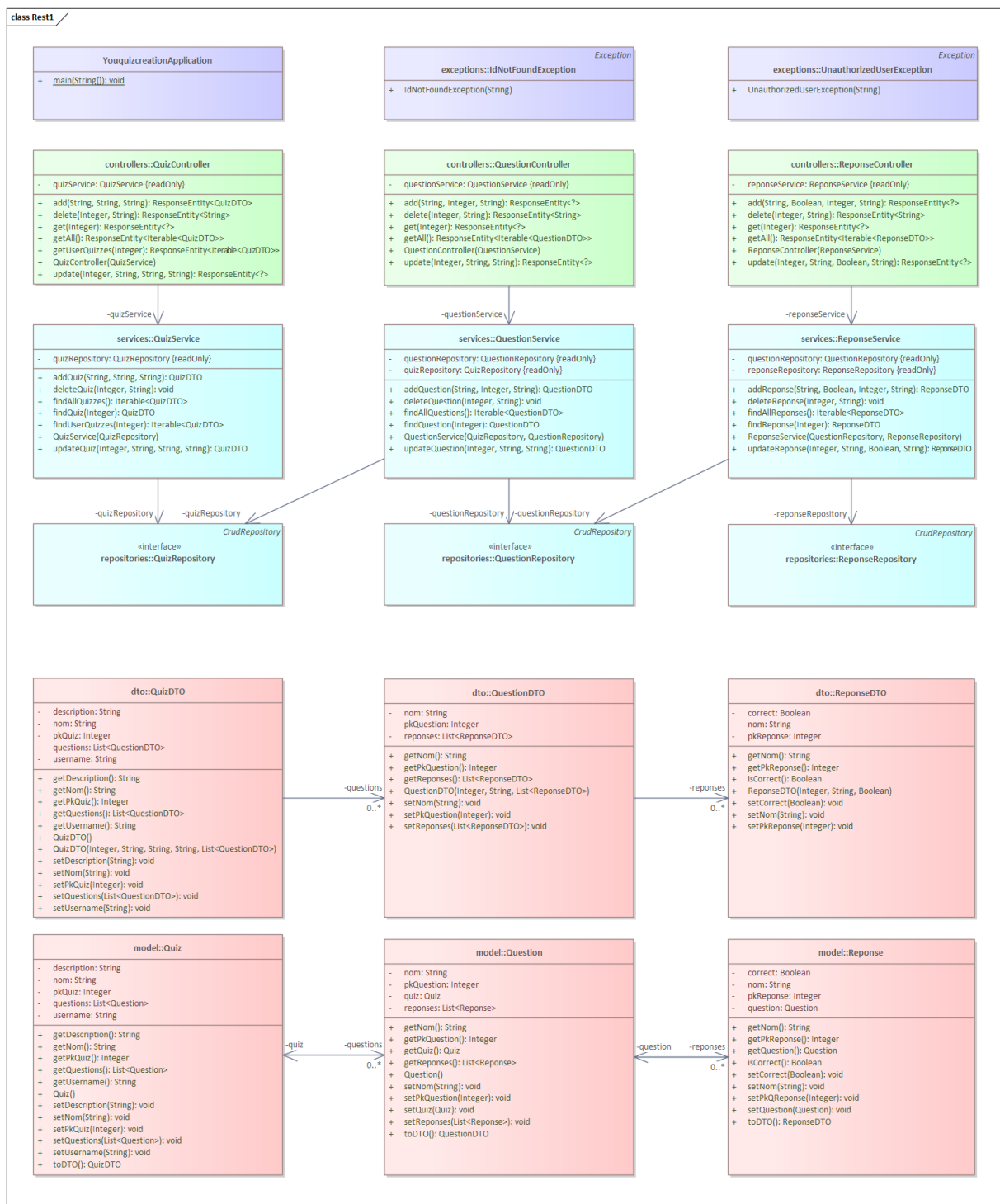




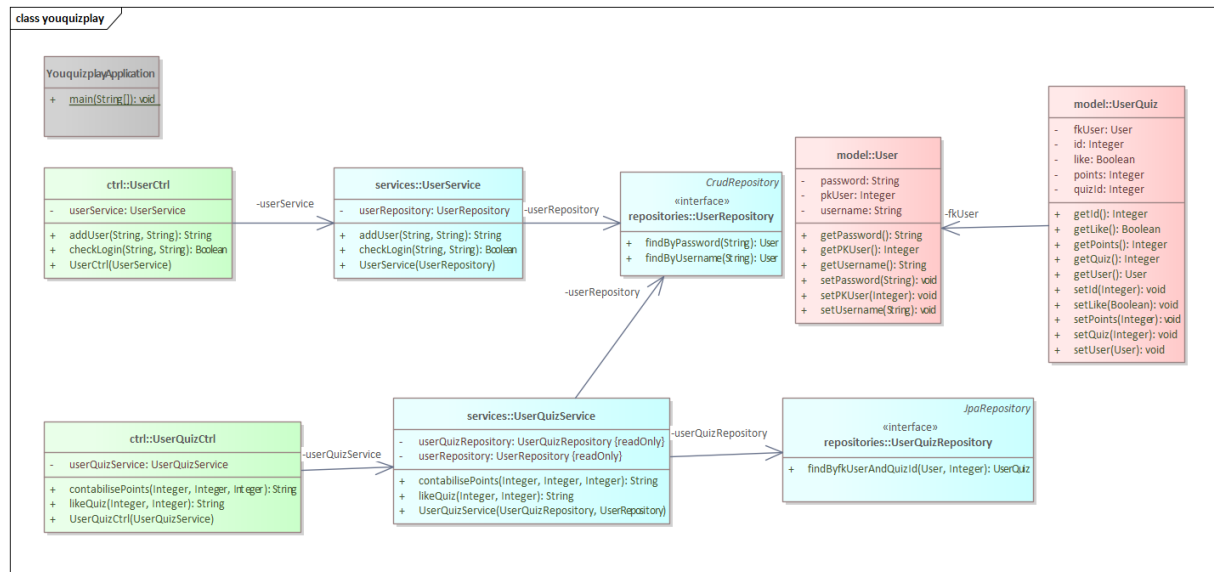
## 4 Conception à faire complètement avec EA -> à rendre uniquement le fichier EA

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

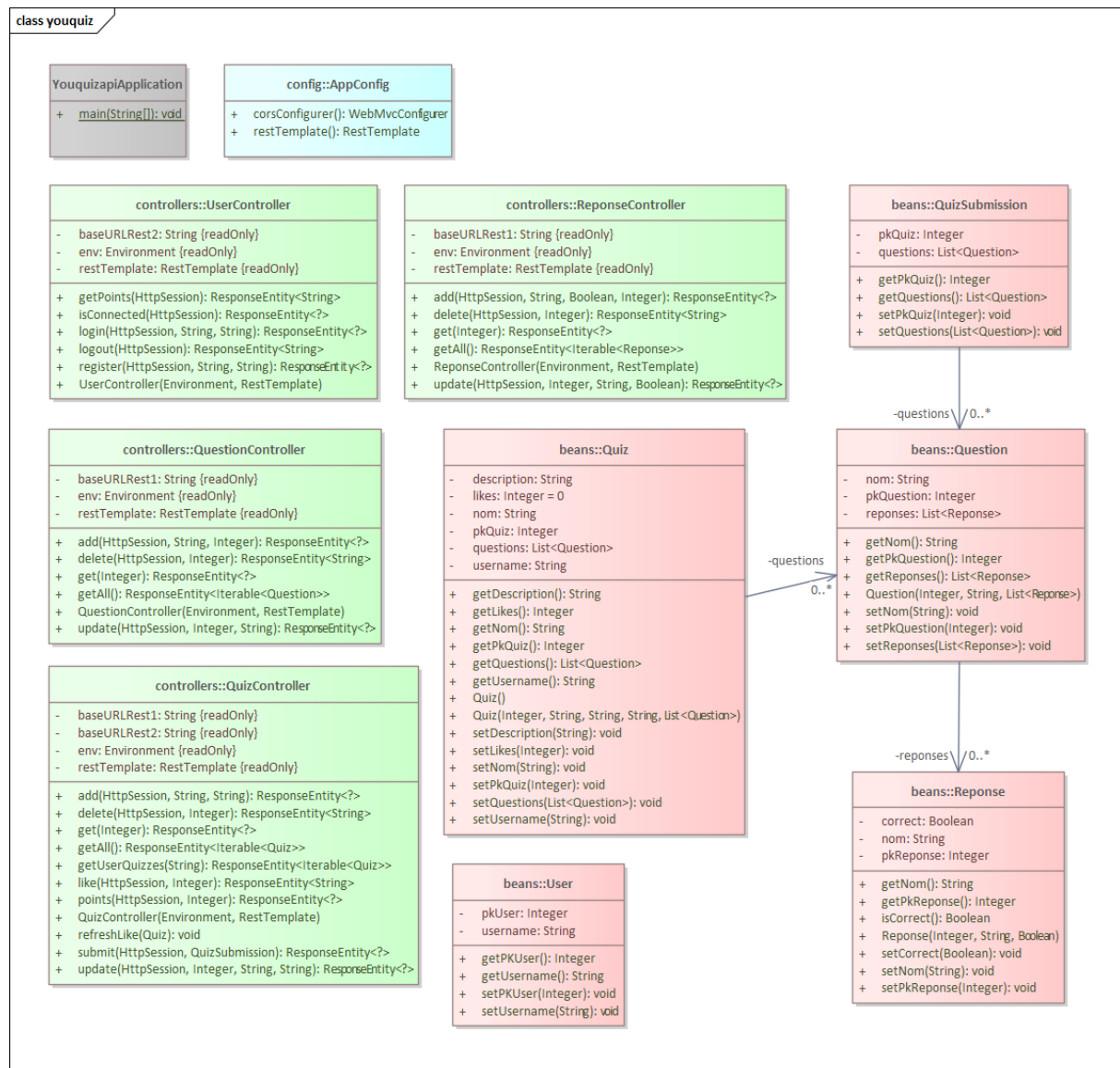
Rest 1 :



Rest 2 :



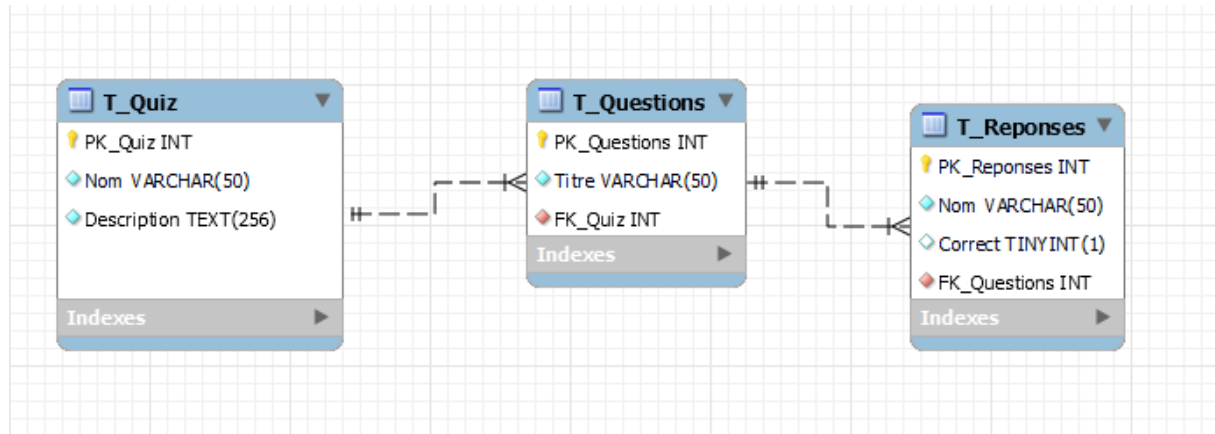
## API Gateway :



## 5 Bases de données

### 5.1 Modèles WorkBench MySQL

DB 1 :



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

### **6.1 Une descente de code client**

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

### **7.1 Une descente de code APIGateway**

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

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

## 9 Hébergement

## **10 Installation du projet complet avec les 5 applications**



## 11 Tests de fonctionnement du projet

## **12 Auto-évaluations et conclusions**

### **12.1 Auto-évaluation et conclusion de ...**

### **12.2 Auto-évaluation et conclusion de ...**