133 - Réaliser des applications web en session-handling

Richoz Matteo

Rapport personnel

Version 1 du 01.05.2024 09:08:00

Module du : 21.03.2024		Au : 03.05.2024

Date de création : 21.03.2024 14:56:00

Table des matières

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.

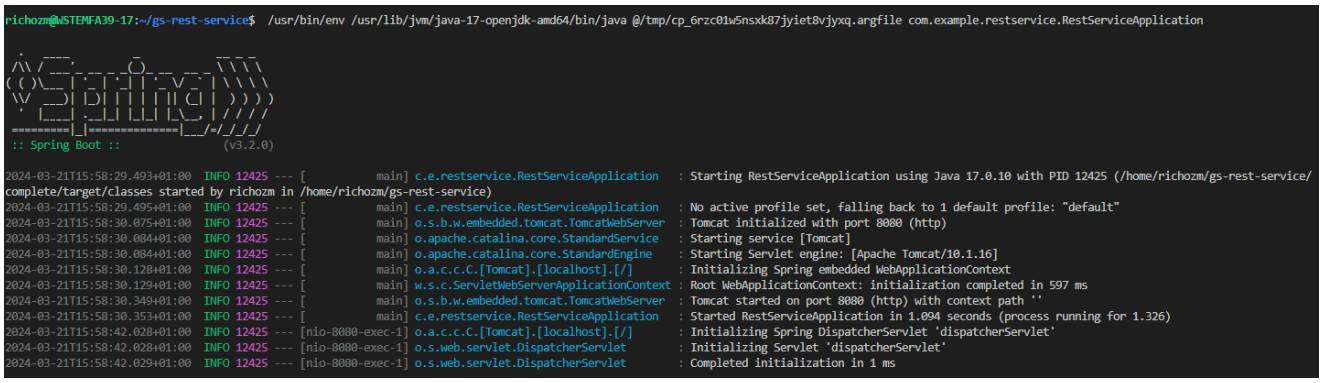
Les quiz consistent en plusieurs questions qui ont plusieurs réponses possibles. Le choix de réponse peut être unique, multiples ou vrai ou faux.

David va s'occuper de la partie permettant la création de quiz et moi de la partie permettant de remplir les sites.

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 ?



Pour lancer le projet, on utilise Spring Boot par le port 8080

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

On utilise Maven pour build notre projet

2.1.3 Y a-t-il un serveur web ?

Oui il y a un serveur Apache Tomcat

2.1.4 Quelle version de java est utilisée ?

On utilise la version 17 de java

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

Apache Tomcat version 10.1.16

2.2 Conteneurisation

2.2.1 Pourquoi faire un container pour une application Java ?

- Portabilité
- Facilité de déploiement
- Gestion des dépendances
- Isolation et sécurité

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

Oui, Tomcat, compilé dans le jar.

2.2.3 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

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

- @RestController : Cette annotation indique que la classe est un contrôleur REST, ce qui signifie qu'elle est responsable de la gestion des requêtes HTTP et des réponses en utilisant les principes RESTful.
- @GetMapping : Cette annotation est utilisée pour mapper les requêtes HTTP GET aux méthodes de gestion appropriées dans le contrôleur. Elle spécifie que la méthode annotée répondra uniquement aux requêtes HTTP GET.
- @PostMapping : Cette annotation est utilisée pour mapper les requêtes HTTP POST aux méthodes de gestion appropriées dans le contrôleur. Elle spécifie que la méthode annotée répondra uniquement aux requêtes HTTP POST.
- @PutMapping : Cette annotation est utilisée pour mapper les requêtes HTTP PUT aux méthodes de gestion appropriées dans le contrôleur. Elle spécifie que la méthode annotée répondra uniquement aux requêtes HTTP PUT.
- @RequestParam : Cette annotation est utilisée pour extraire les paramètres de requête de l'URL et les associer aux paramètres de la méthode dans le contrôleur. Elle permet de récupérer les valeurs des paramètres de requête HTTP dans le contrôleur.
- @RequestBody : Cette annotation est utilisée pour mapper le corps de la requête HTTP à un objet Java dans le contrôleur. Elle permet de désérialiser automatiquement le corps de la requête JSON (ou XML) en un objet Java correspondant.

2.4 Connexion à la DB JDBC

2.4.1 Créer une base de données dans un container

Pour créer une nouvelle db dans un container, il faut faire ces commandes :

```
#Création du répertoire sur la machine locale qui contiendra les données de MySQL

mkdir -p /opt/mysql


#Démarrage du container MySQL
```

```
docker run --name mysql -d -p 3306:3306 -e MYSQL_ROOT_HOST=% -e
MYSQL_ROOT_PASSWORD=emf123 -v /opt/mysql:/var/lib/mysql mysql/mysql-server:8.0
```

2.5 Connexion à la DB JPA

2.5.1 À quoi sert l'annotation @Autowired dans vos controlleur 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.

2.5.2 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.

2.5.3 Sur la même ligne, quel FetchType est utilisé et pourquoi, rééssayer 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

2.6.1 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.

2.6.2 Expliquez dans votre rapport à quoi servent les model, les repository, les dto, les services et les controlleurs en vous basant sur le code donné.

- 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

Pour effectuer cet exercice, j'ai dû créer un nouveau controller avec 3 méthodes à l'intérieur. Une pour se login, une pour se logout et une pour compter le nombre de visites. Voici le contenu du controller :

```
@RestController

@RequestMapping("/user")

public class UserController {



    @PostMapping("/login")

    public ResponseEntity<String> login(@RequestParam String username,
@RequestParam String password, HttpSession httpSession) {

        httpSession.setAttribute("username", username);

        httpSession.setAttribute("visites", 0);

        return ResponseEntity.ok("Login successful");

    }



    @PostMapping("/logout")

    public ResponseEntity<String> logout(HttpSession httpSession) {

        httpSession.invalidate();

        return ResponseEntity.ok("Logout successful");

    }



    @GetMapping("/visites")

    public ResponseEntity<String> getVisites(HttpSession httpSession) {

        Integer visites = (Integer) httpSession.getAttribute("visites");

        if (visites == null) {

            return ResponseEntity.status(HttpStatus.I_AM_A_TEAPOT).body("NON !!!");

        }
```

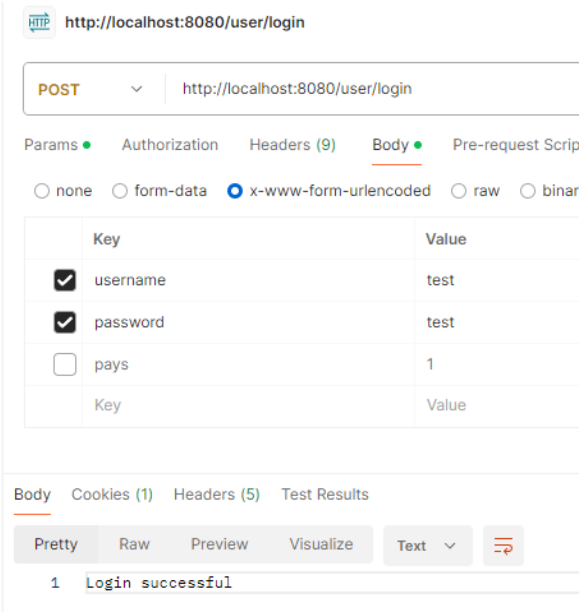
```
        visites++;
        httpSession.setAttribute("visites", visites);
        return ResponseEntity.ok("Visites : " + visites);
    }
}
```

Il ne faut pas oublier d'ajouter l'import pour httpSession

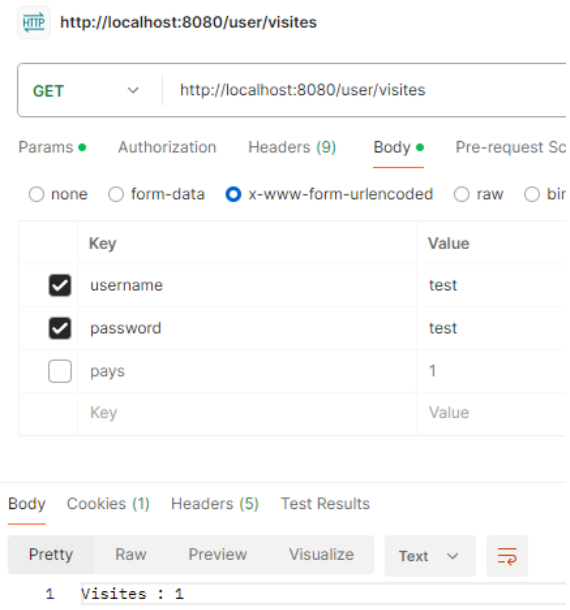
```
import jakarta.servlet.http.HttpSession;
```

Ensuite pour tester je me suis rendu sur Postman et j'ai tester mes méthodes comme ceci :

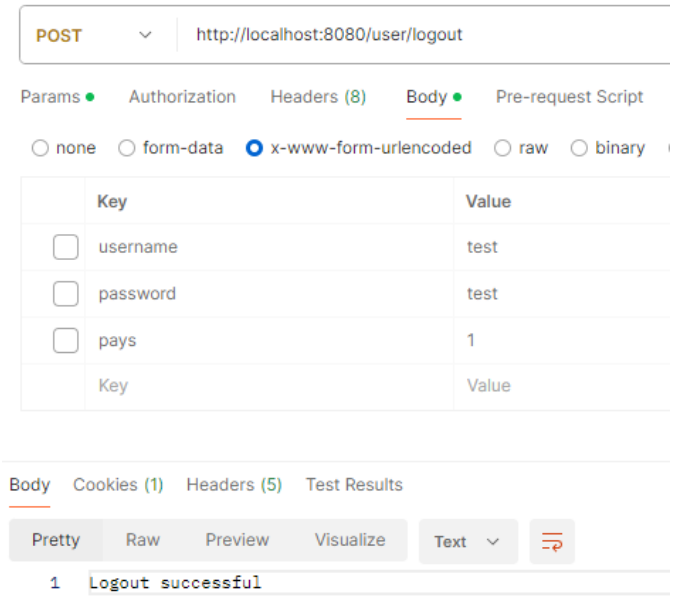
Login :



Pour le comptage de visites :



Pour le logout :



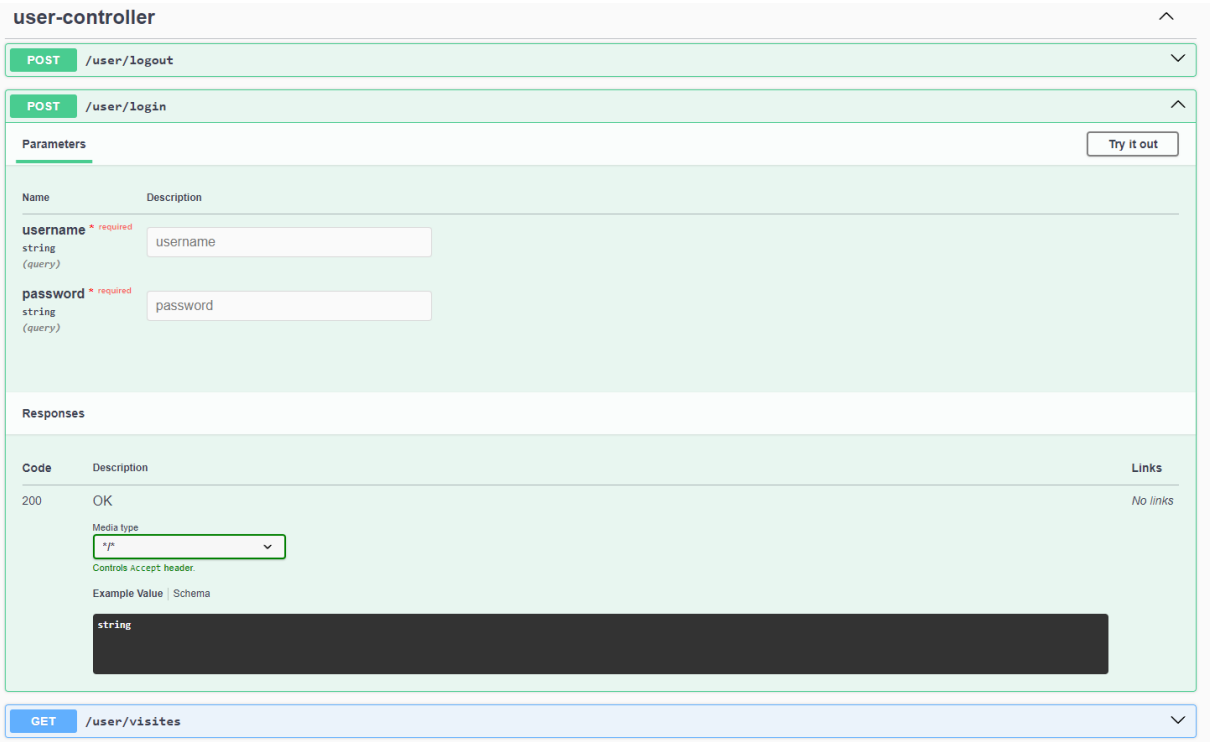
2.8 Documentation API avec Swagger

Pour faire notre documentation avec Swagger, j'ai dû ajouter une dépendance dans mon pom.xml :

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

```
    </dependency>
```

Une fois que c'est ajouté j'ai relancé mon projet et je suis allé sur ce lien pour voir ma doc : http://localhost:8080/swagger-ui/index.html

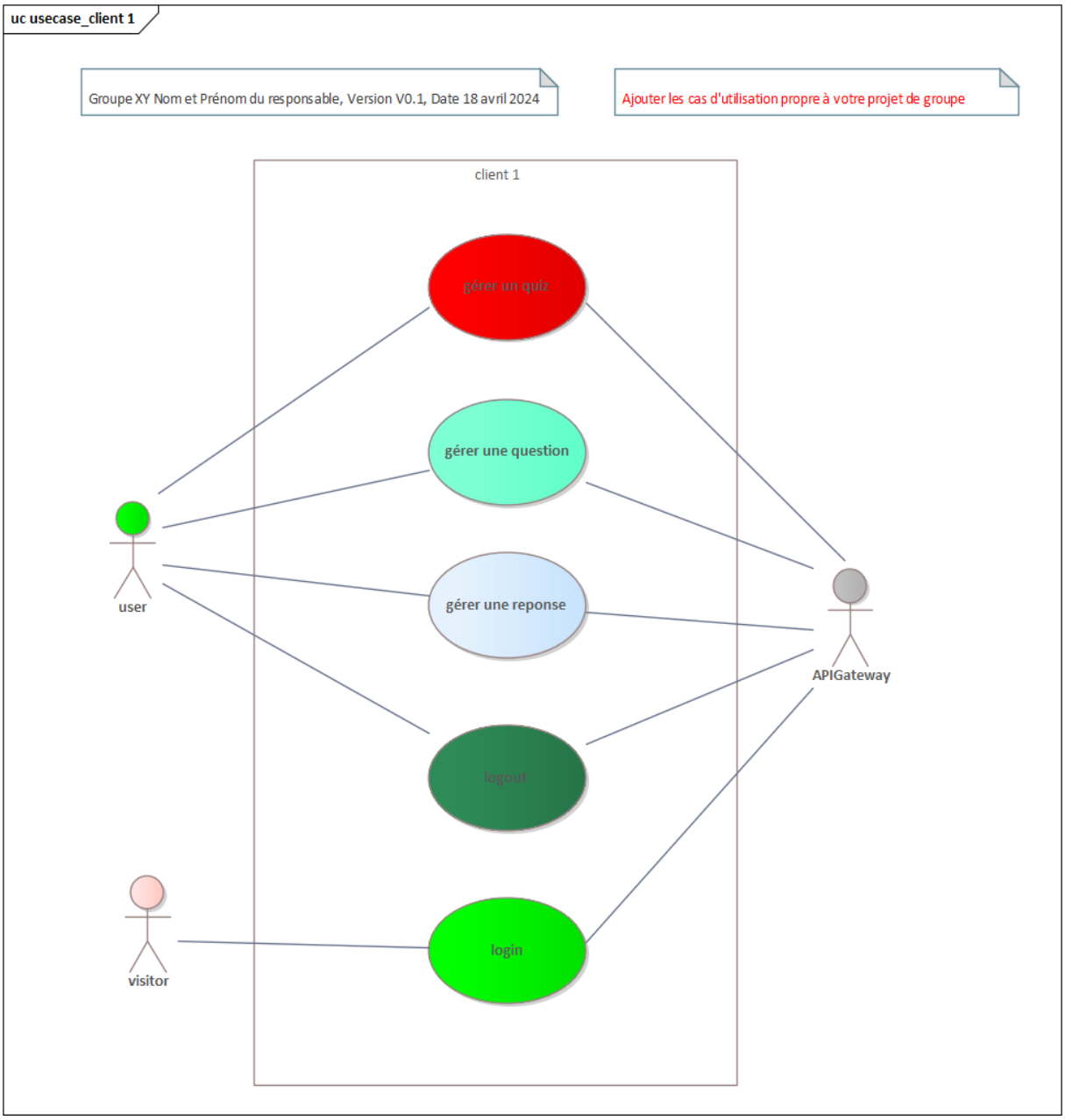


2.9 Hébergement

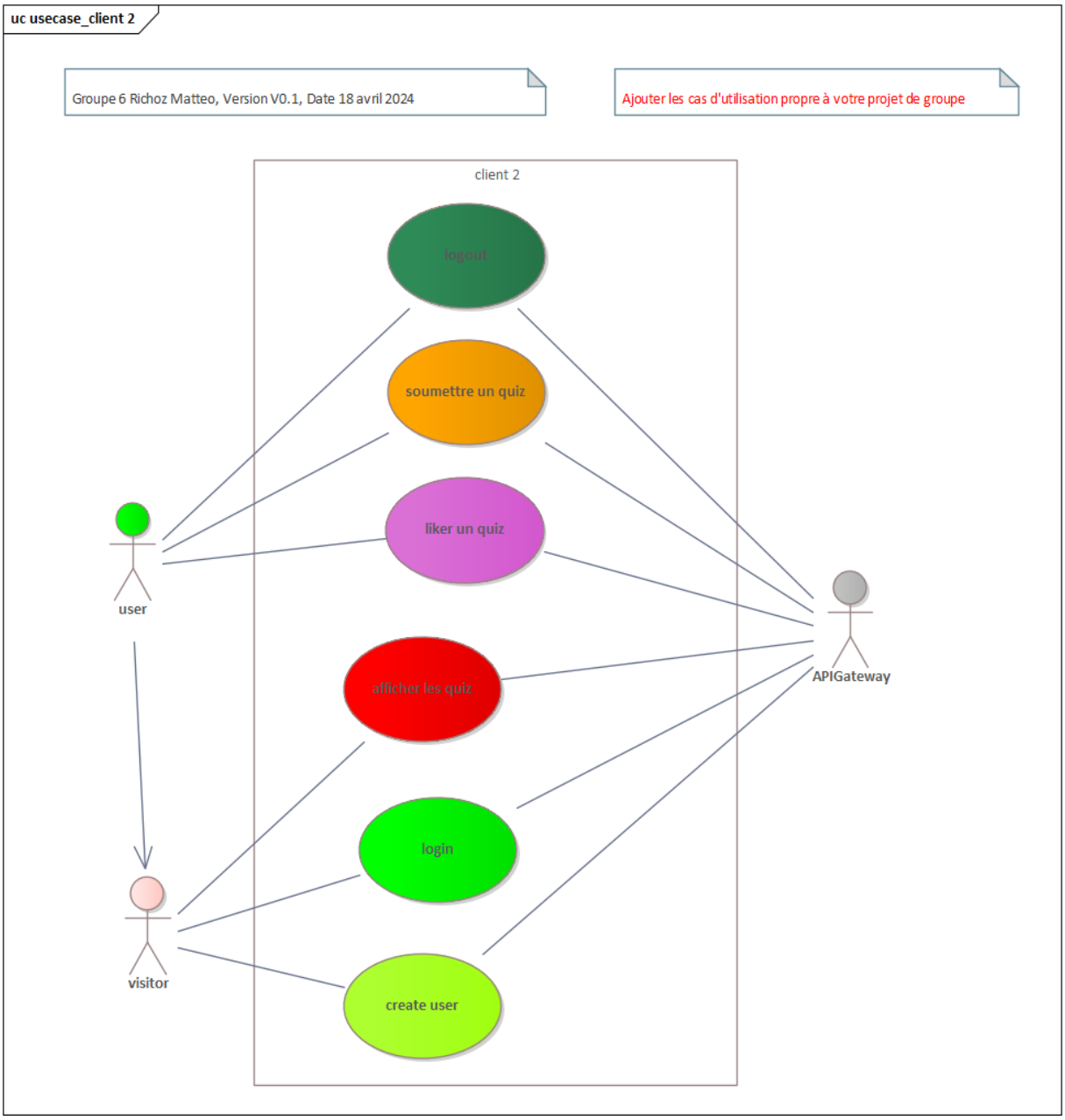
3 Analyse à faire complètement avec EA -> à rendre uniquement le fichier EA

3.1 Use case client et use case Rest

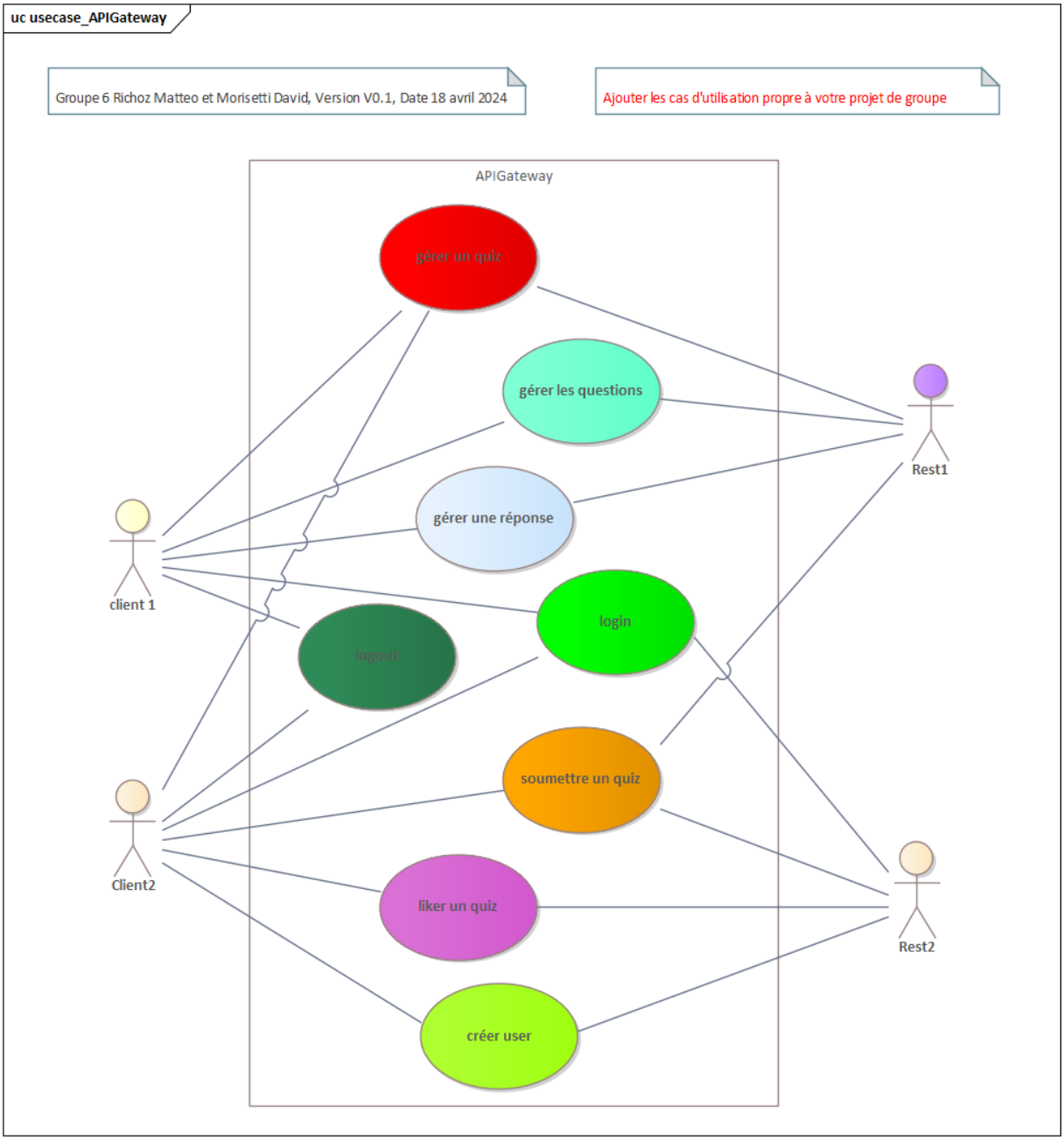
3.1.1 Use case client 1



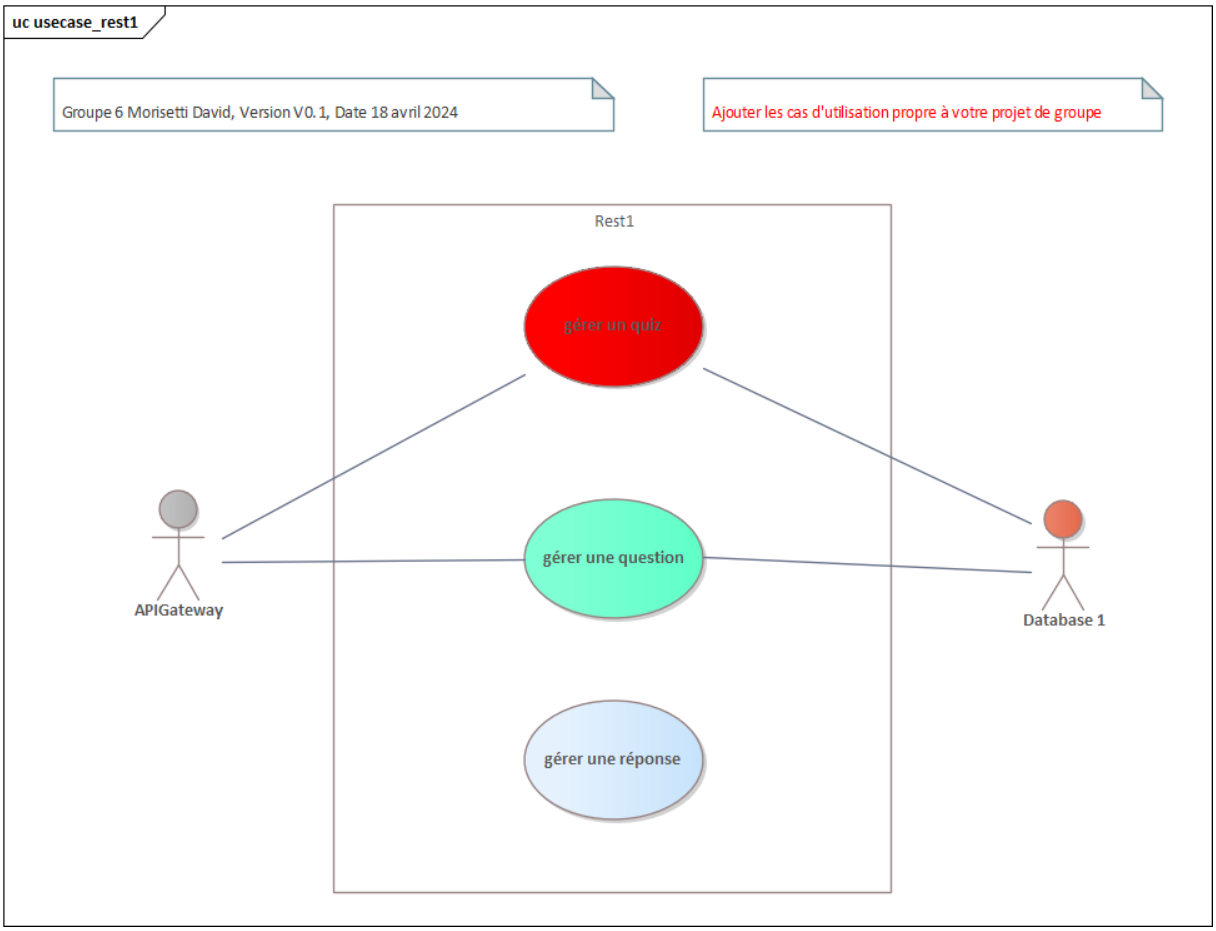
3.1.2 Use case client 2



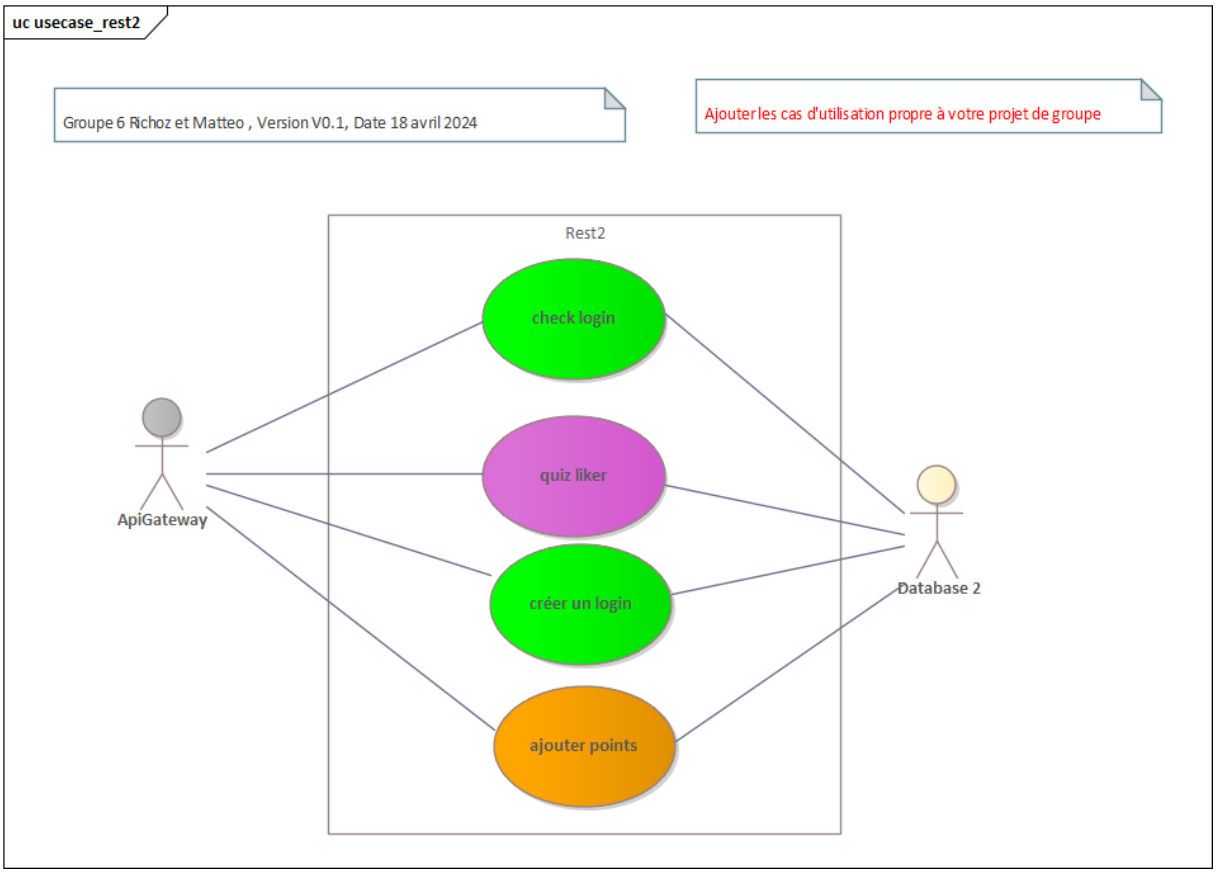
3.1.3 Use case API Gateway



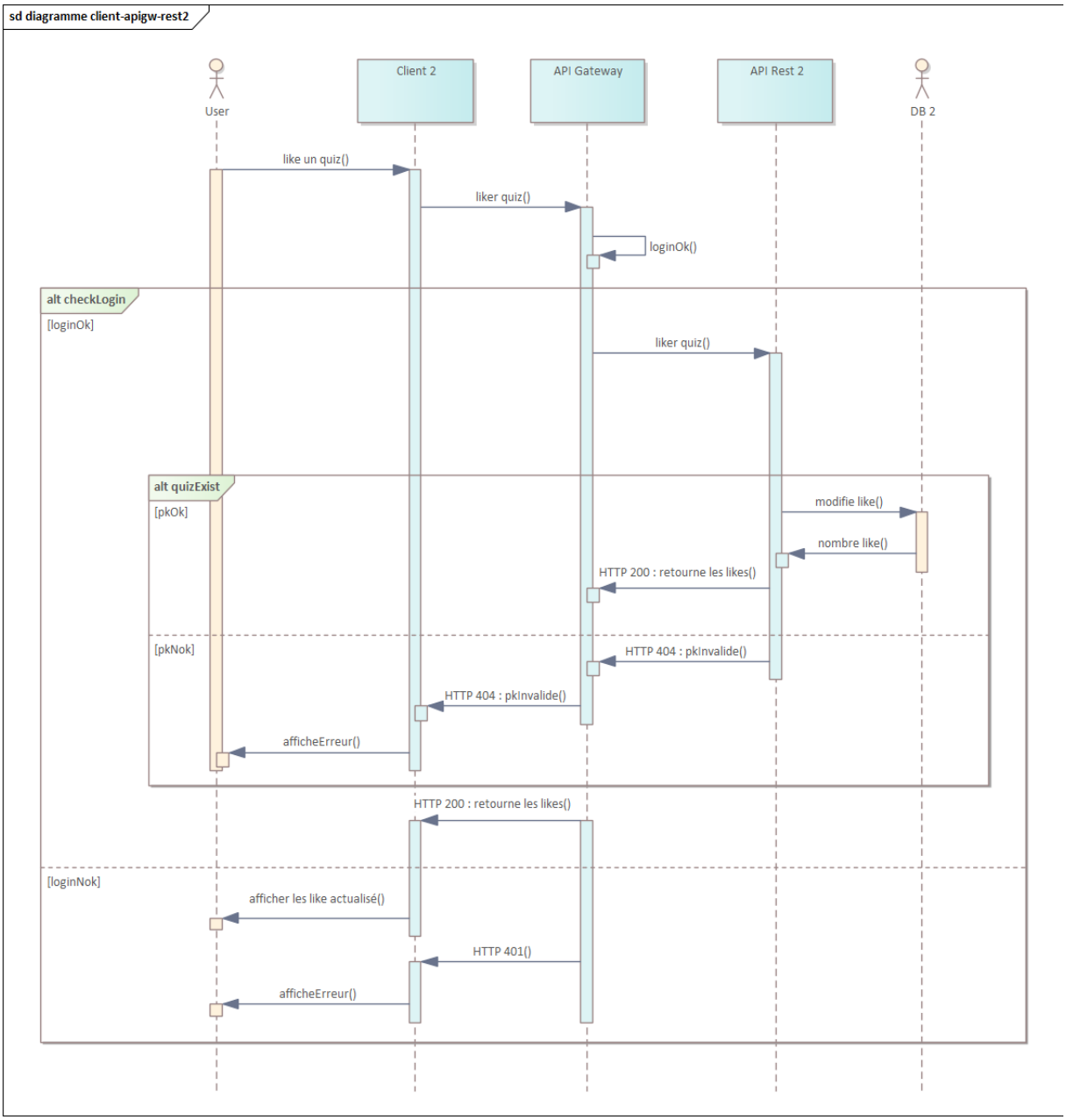
3.1.4 Use case API Rest 1



3.1.5 Use case API Rest 2

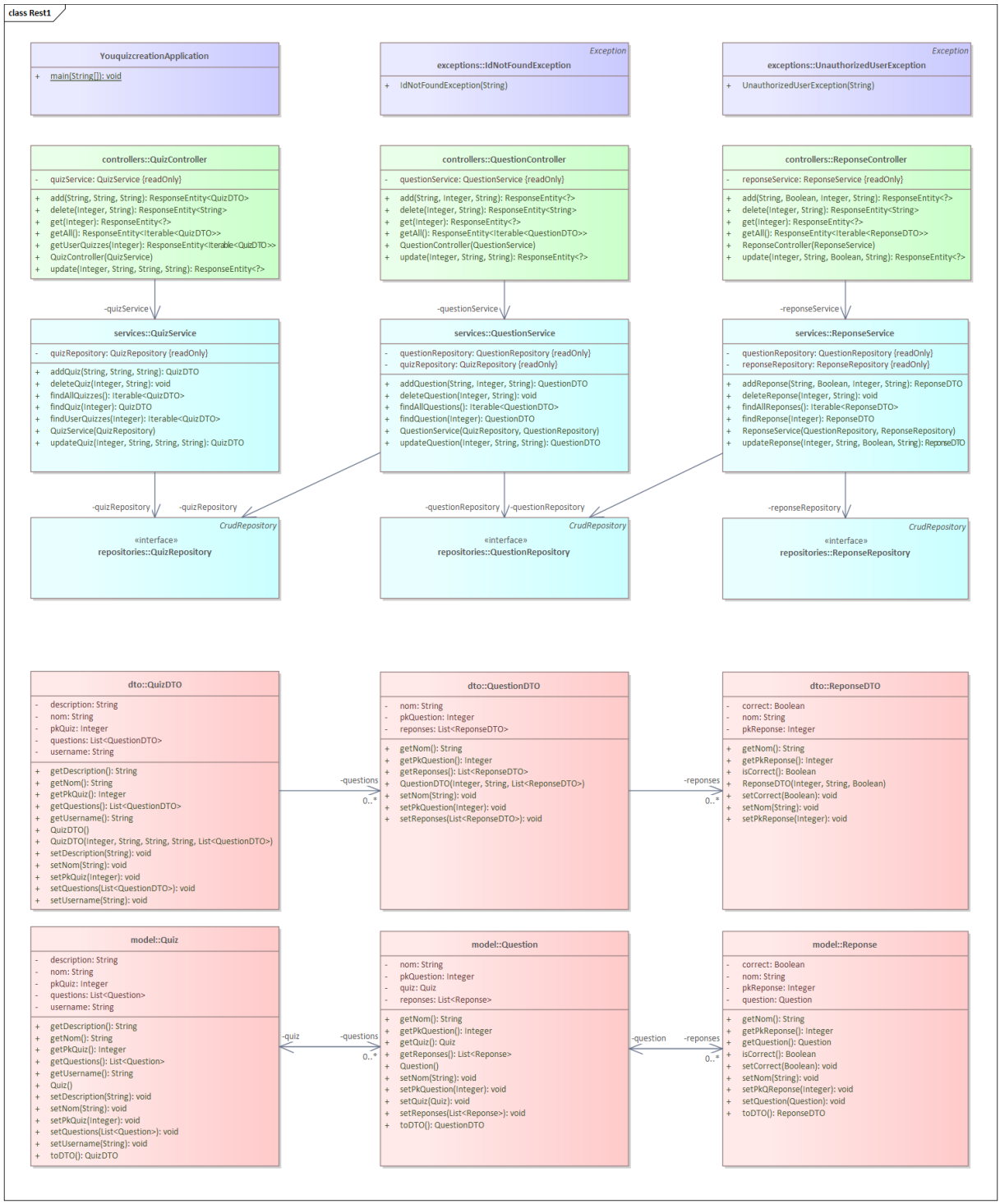


3.2 Sequence System global entre les applications

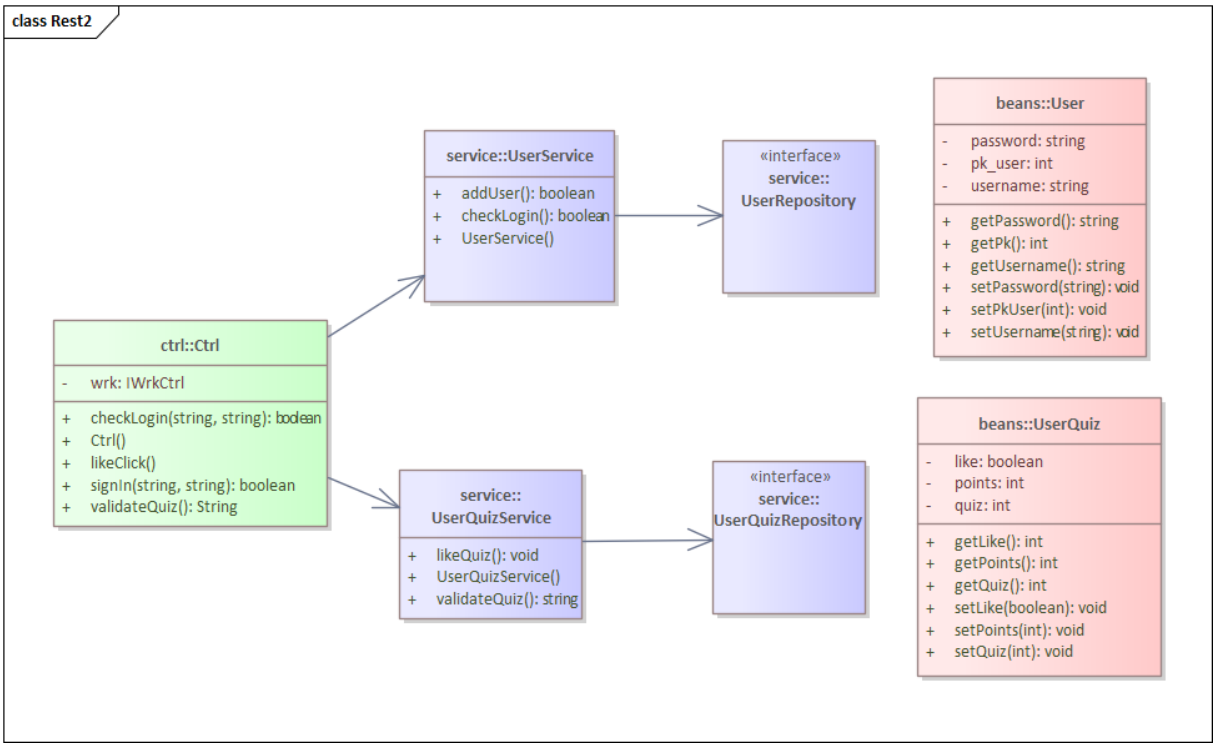


4 Conception à faire complétement avec EA -> à rendre uniquement le fichier EA

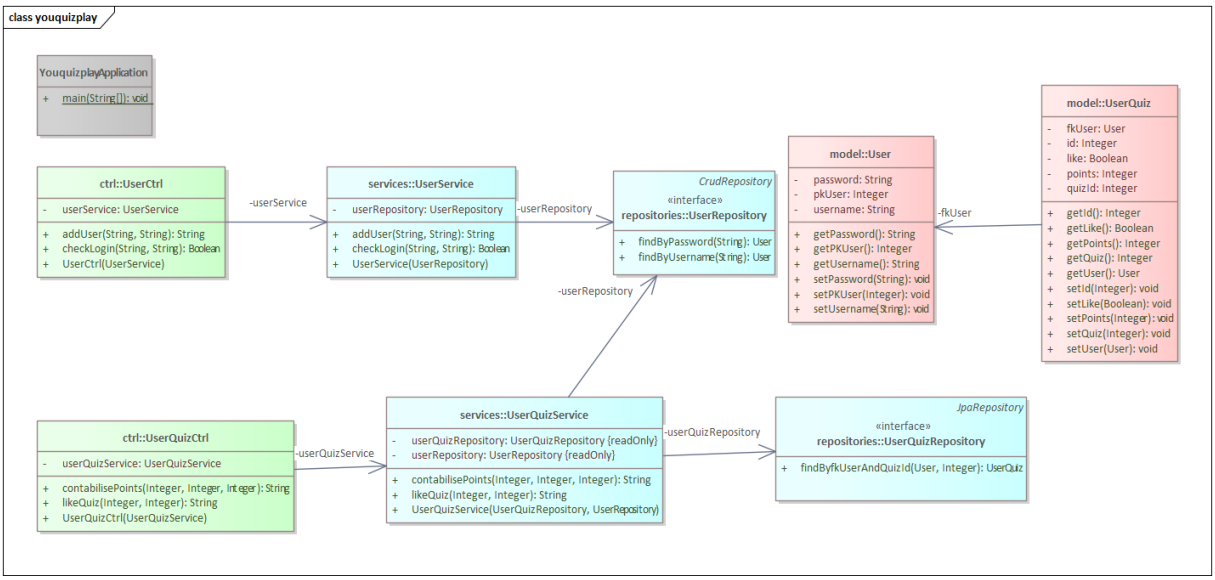
4.1 Diagramme class Rest 1



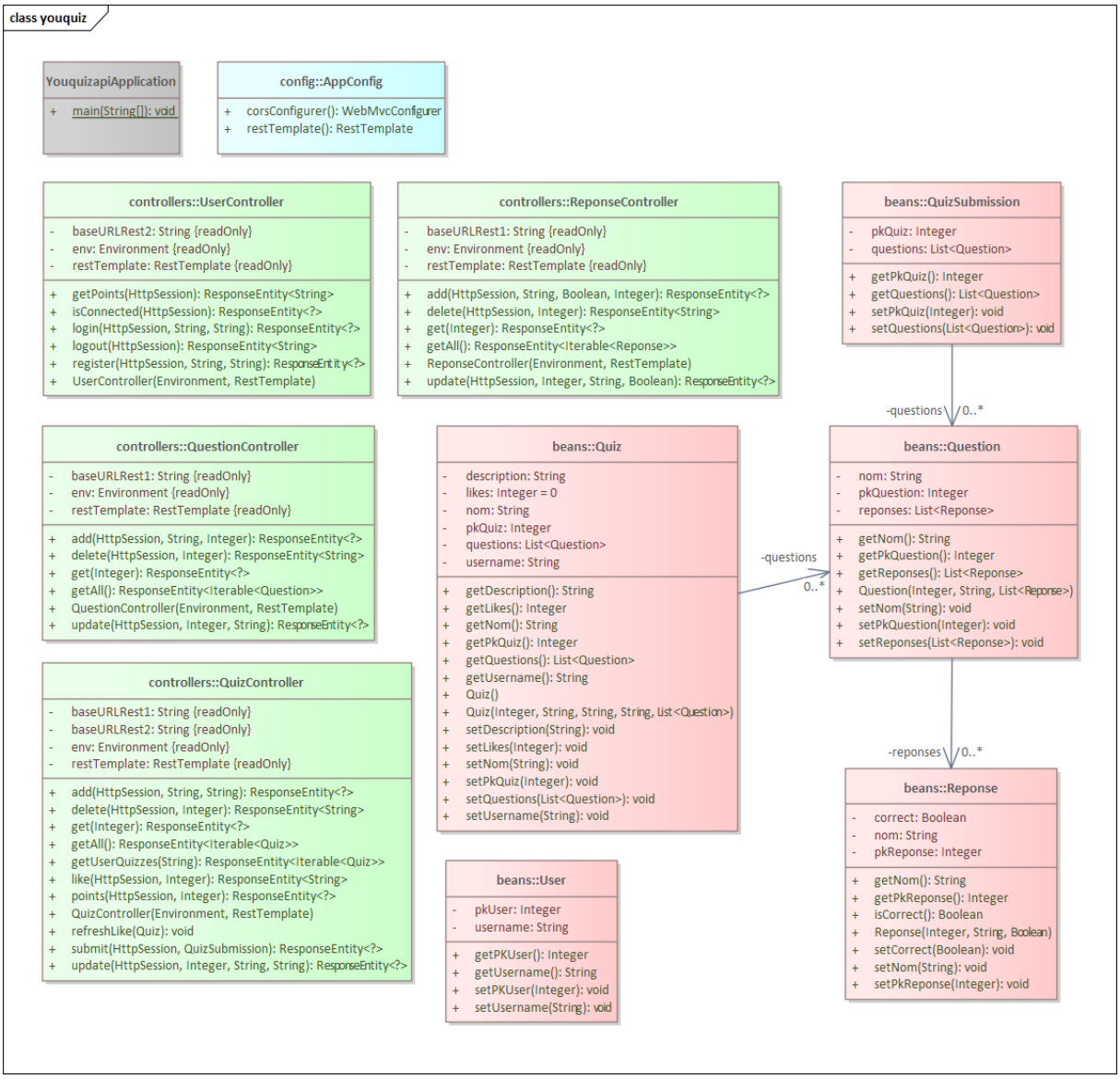
4.2 Diagramme class Rest 2



4.3 Diagramme class Rest 2 après implémentation

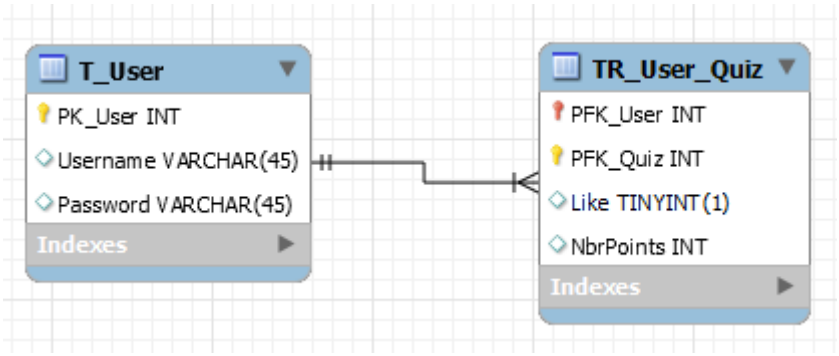


4.4 Diagramme class API Gateway



5 Bases de données

5.1 Modèles WorkBench MySQL



Dans ma base de données, je vais stocker toutes les données liées aux utilisateurs comme leur nom d'utilisateur, le mot de passe, le nombre de points qu'il ont faits aux quiz et s'ils ont liké le quiz ou pas.

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

12.2 Conclusion