



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 06.05.2024

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 Erreur ! Signet non défini.	
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	29
12.1	Auto-évaluation et conclusion de	29
12.2	Auto-évaluation et conclusion de	29

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.

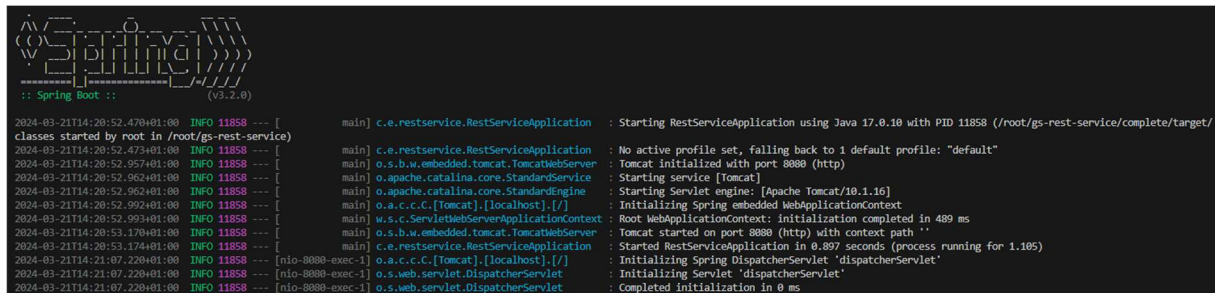
Je vais m'occuper de la partie permettant la création de quiz et Matteo de la partie permettant de remplir les sites.

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 --- [
classes started by root in /root/gs-rest-service)
2024-03-21T14:20:52.473+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/
2024-03-21T14:20:52.957+01:00 INFO 11858 --- [
main] o.s.b.w.embedded.tomcat.TomcatWebServer : No active profile set, falling back to 1 default profile: "default"
2024-03-21T14:20:52.962+01:00 INFO 11858 --- [
main] o.apache.catalina.core.StandardService : Tomcat initialized with port 8080 (http)
2024-03-21T14:20:52.962+01:00 INFO 11858 --- [
main] o.apache.catalina.core.StandardEngine : Starting service [Tomcat]
2024-03-21T14:20:52.992+01:00 INFO 11858 --- [
main] o.a.c.c.C.[Tomcat].[localhost].[/] : Starting Servlet engine: [Apache Tomcat/10.1.16]
2024-03-21T14:20:52.993+01:00 INFO 11858 --- [
main] w.s.c.ServletWebServerApplicationContext : Initializing Spring embedded WebApplicationContext
2024-03-21T14:20:53.170+01:00 INFO 11858 --- [
main] o.s.b.w.embedded.tomcat.TomcatWebServer : Root WebApplicationContext: initialization completed in 489 ms
2024-03-21T14:20:53.174+01:00 INFO 11858 --- [
main] c.e.restservice.RestServiceApplication : Tomcat started on port 8080 (http) with context path ''
2024-03-21T14:21:07.220+01:00 INFO 11858 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : 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.s.web.servlet.DispatcherServlet : 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'autres objets.

À quoi servent les model, les repository, les dto, les services et les contrôleurs ?

- **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

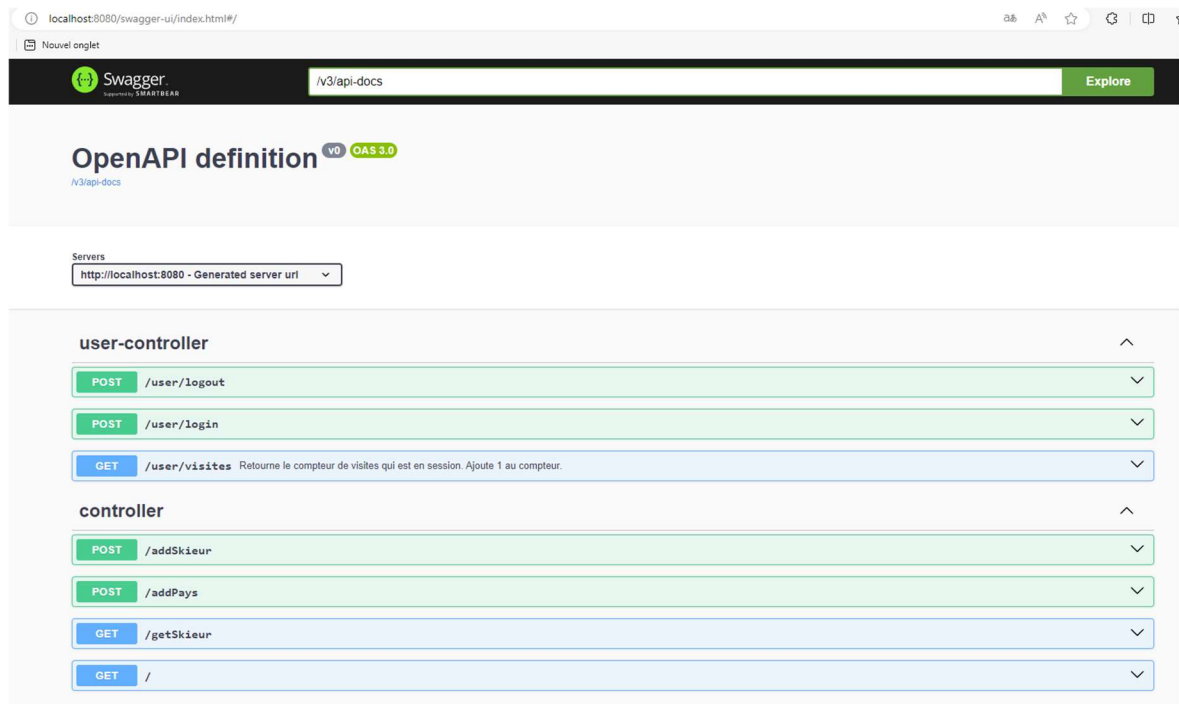
Une fois que vous avez ajouté la dépendance Springdoc à votre projet, la documentation OpenAPI est automatiquement générée à partir de votre code source. Vous pouvez accéder à cette documentation en visitant l'URL **/swagger-ui.html** de votre application.

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.

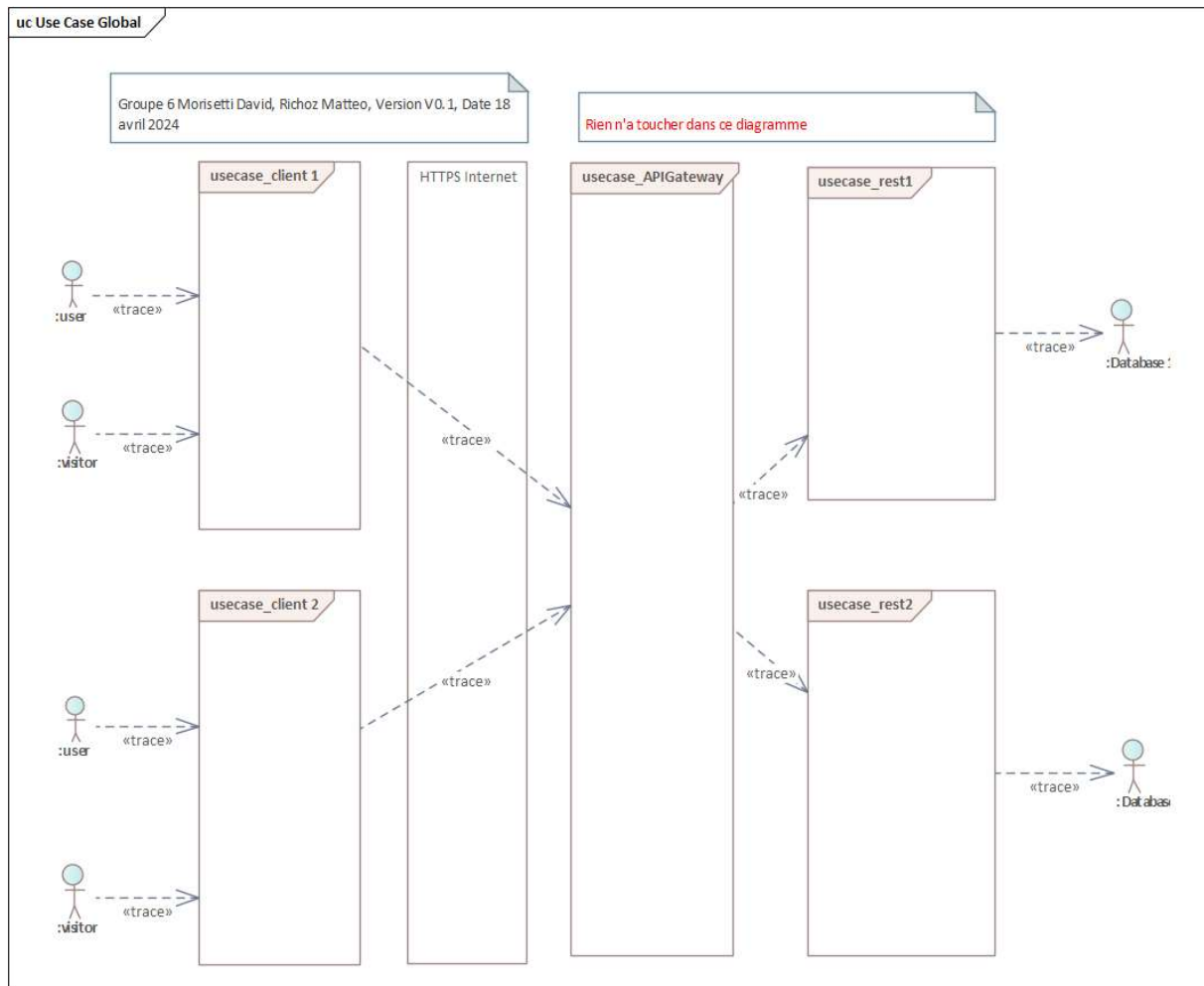


2.9 Hébergement

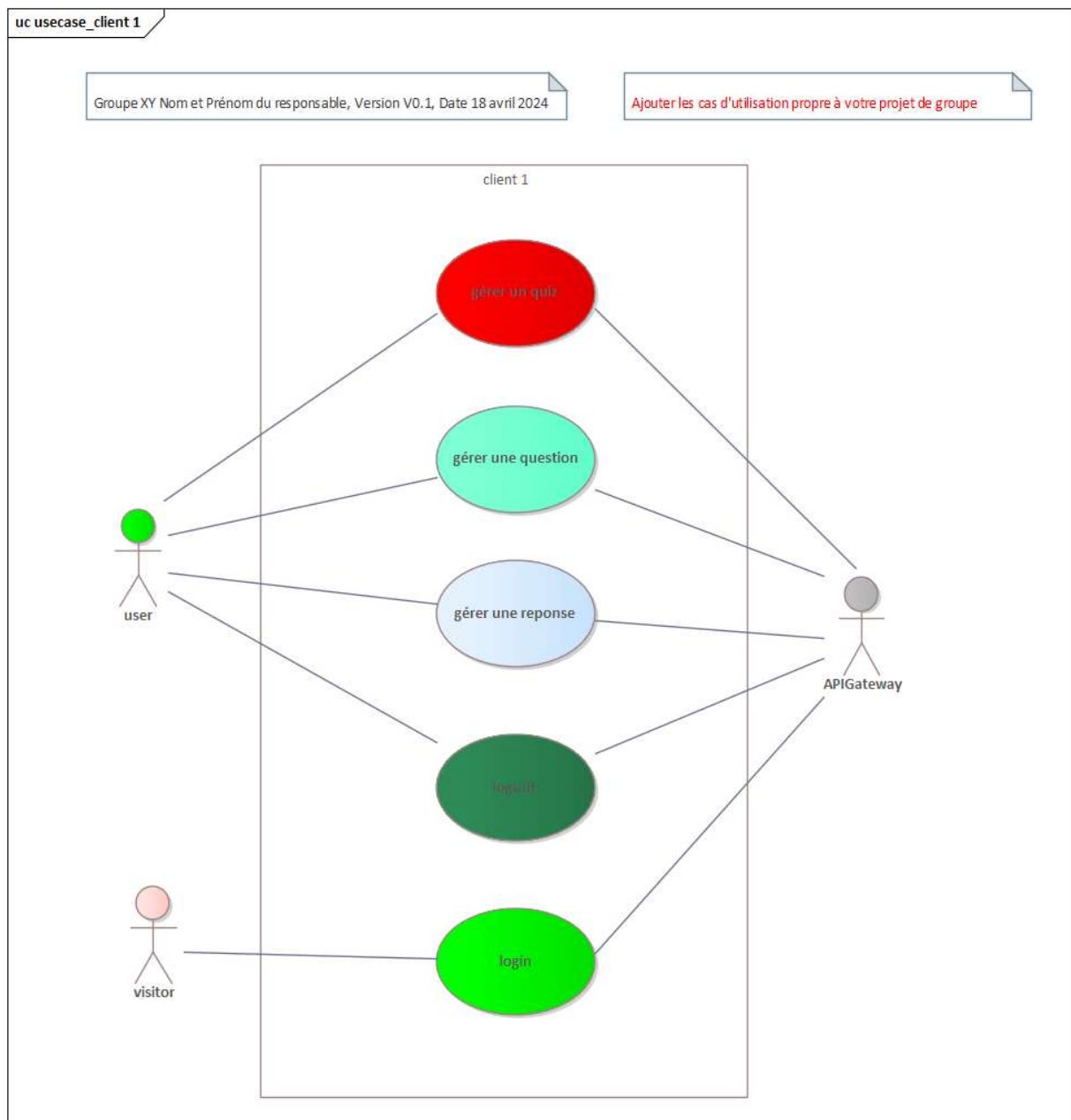
3 Analyse à faire complètement avec EA -> à rendre uniquement le fichier EA

3.1.1 Use case client et use case Rest

Use case global :



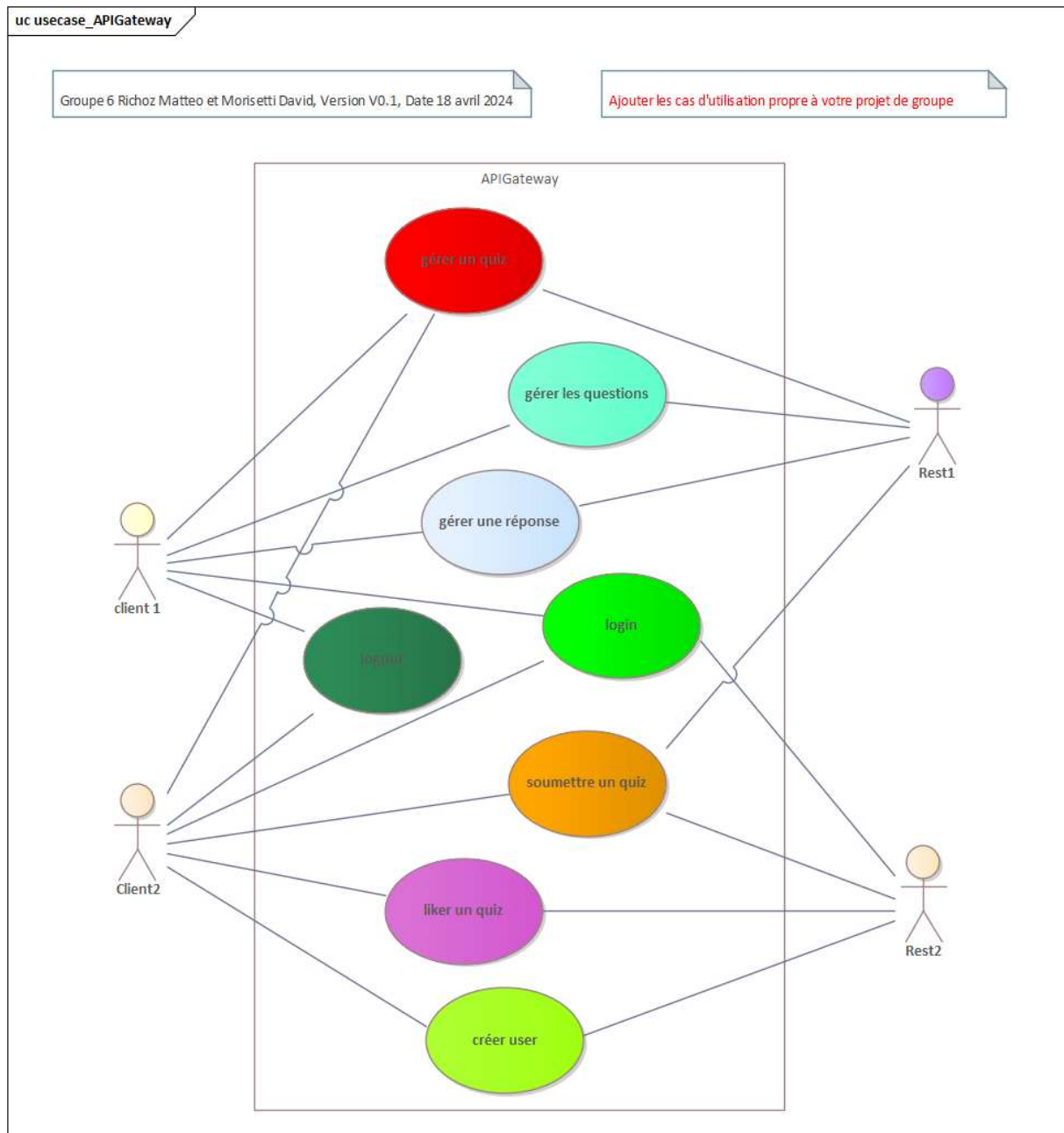
Use case client 1 :



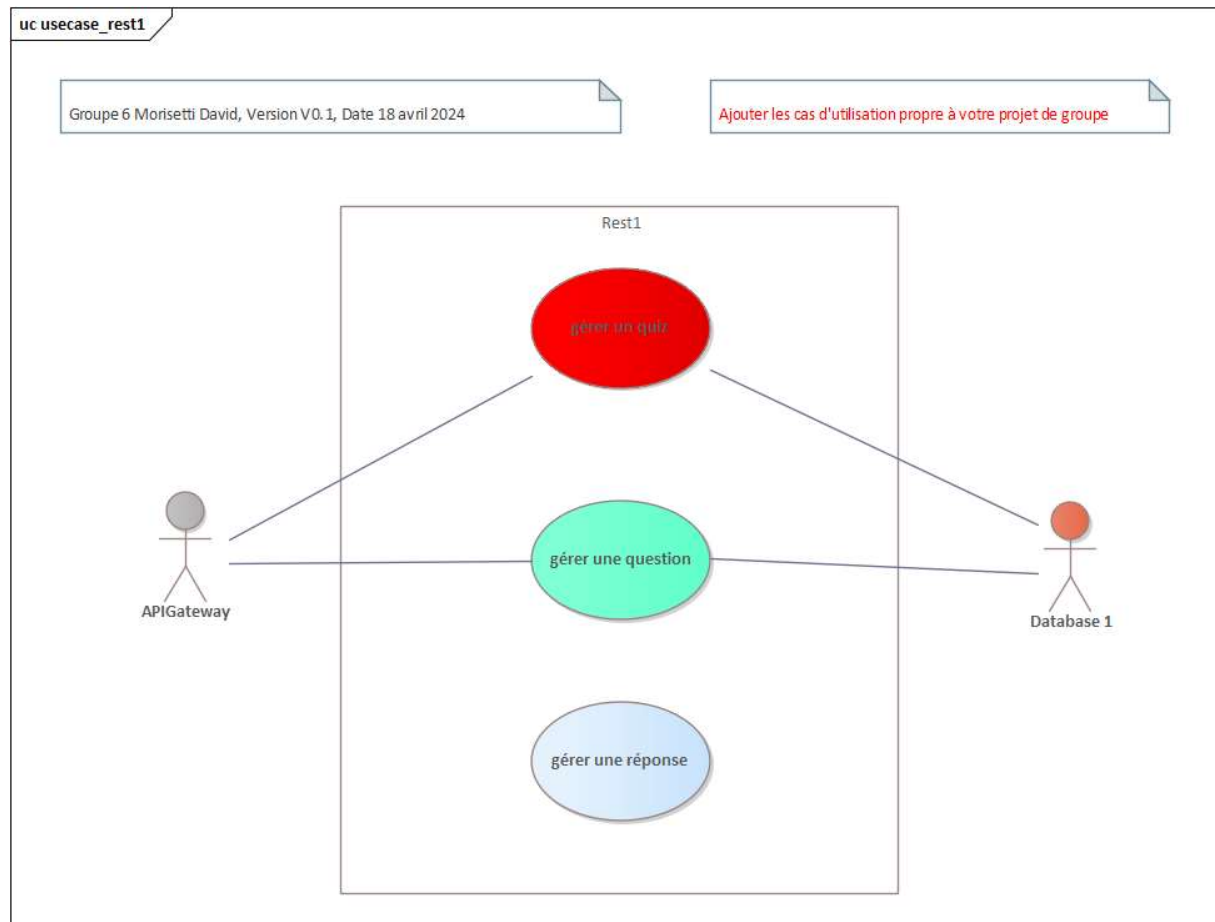
Use case client 2 :



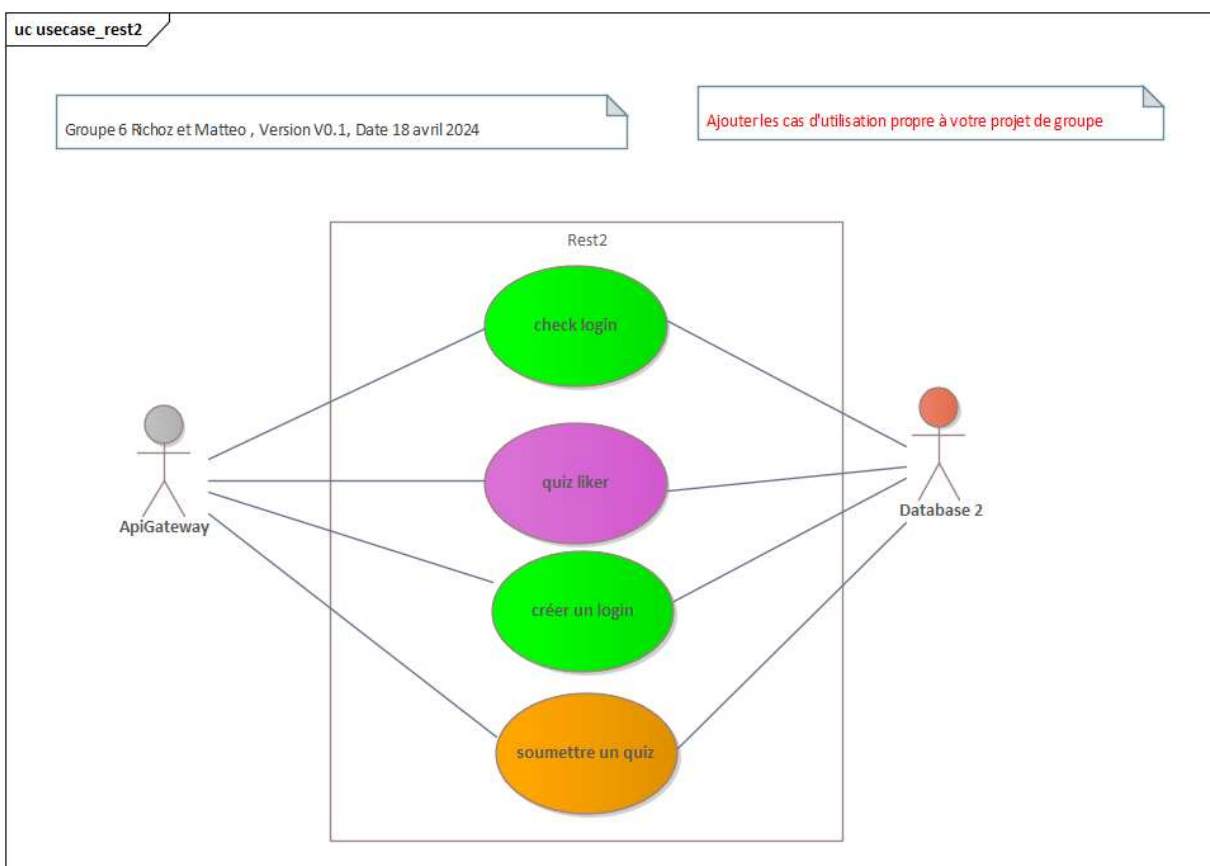
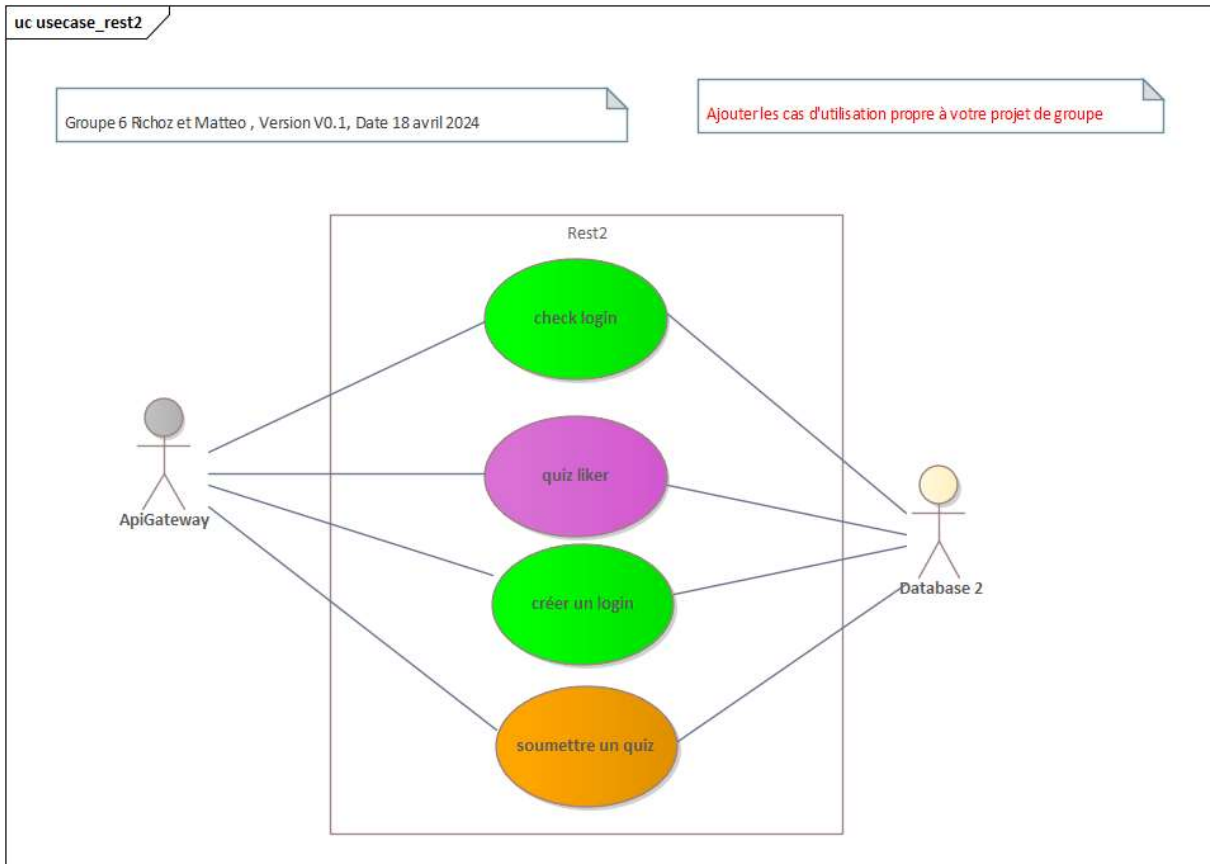
Use case API gateway :



Use case REST 1 :

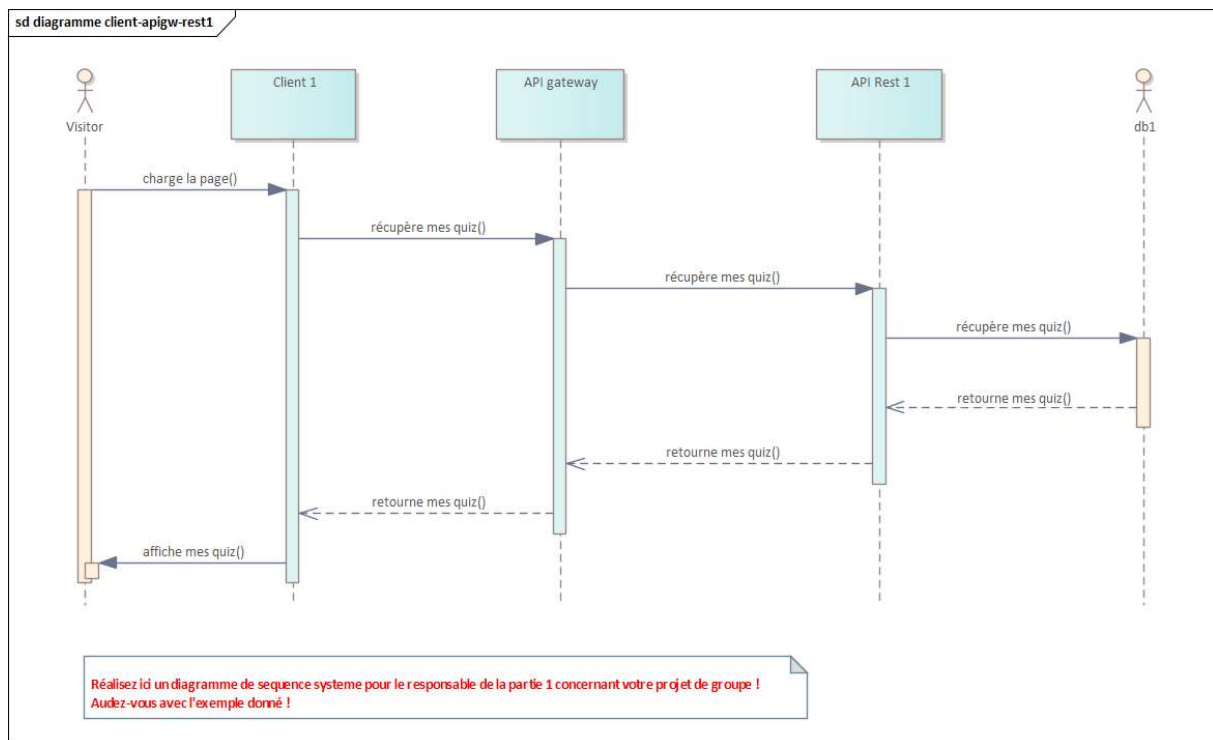


Use case REST 2 :

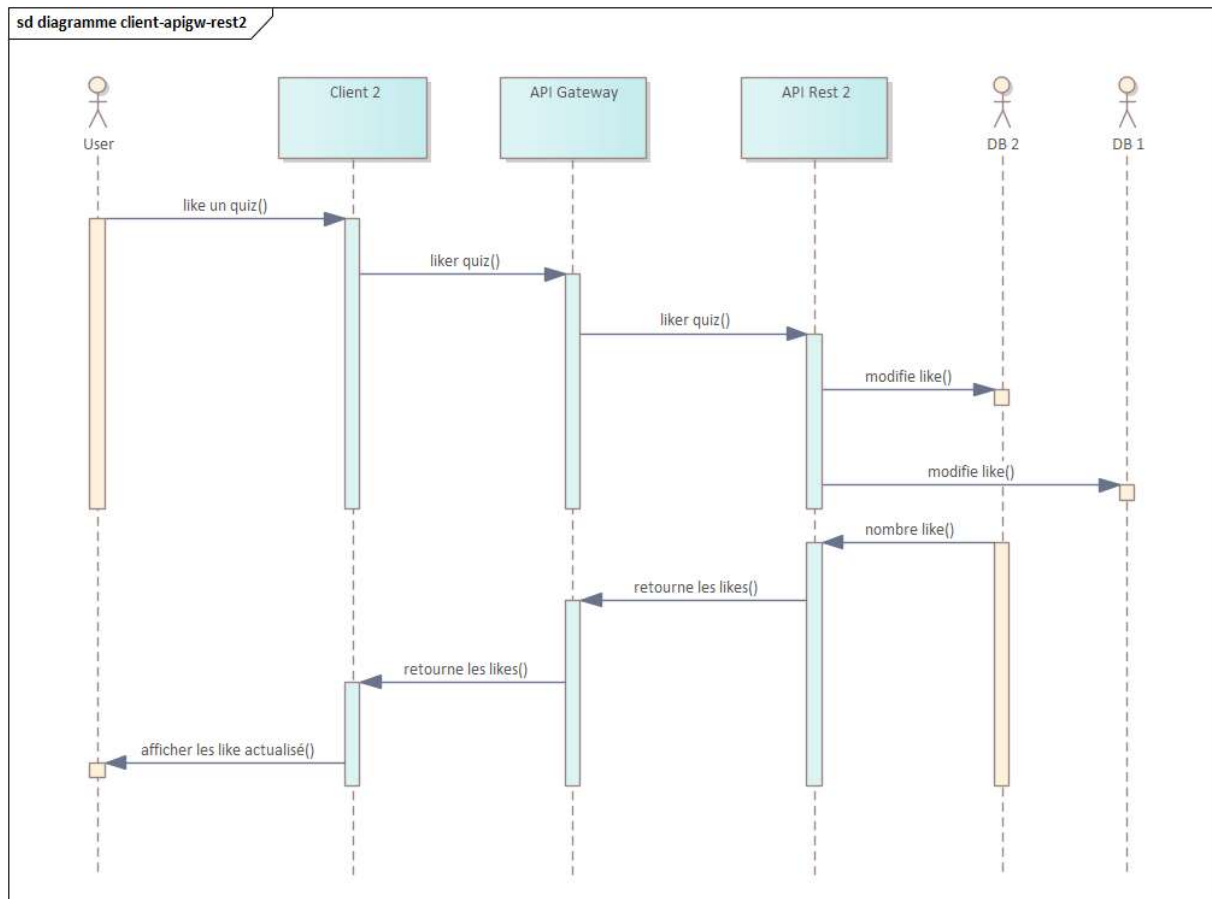


3.1.2 Sequence System

Rest 1 :



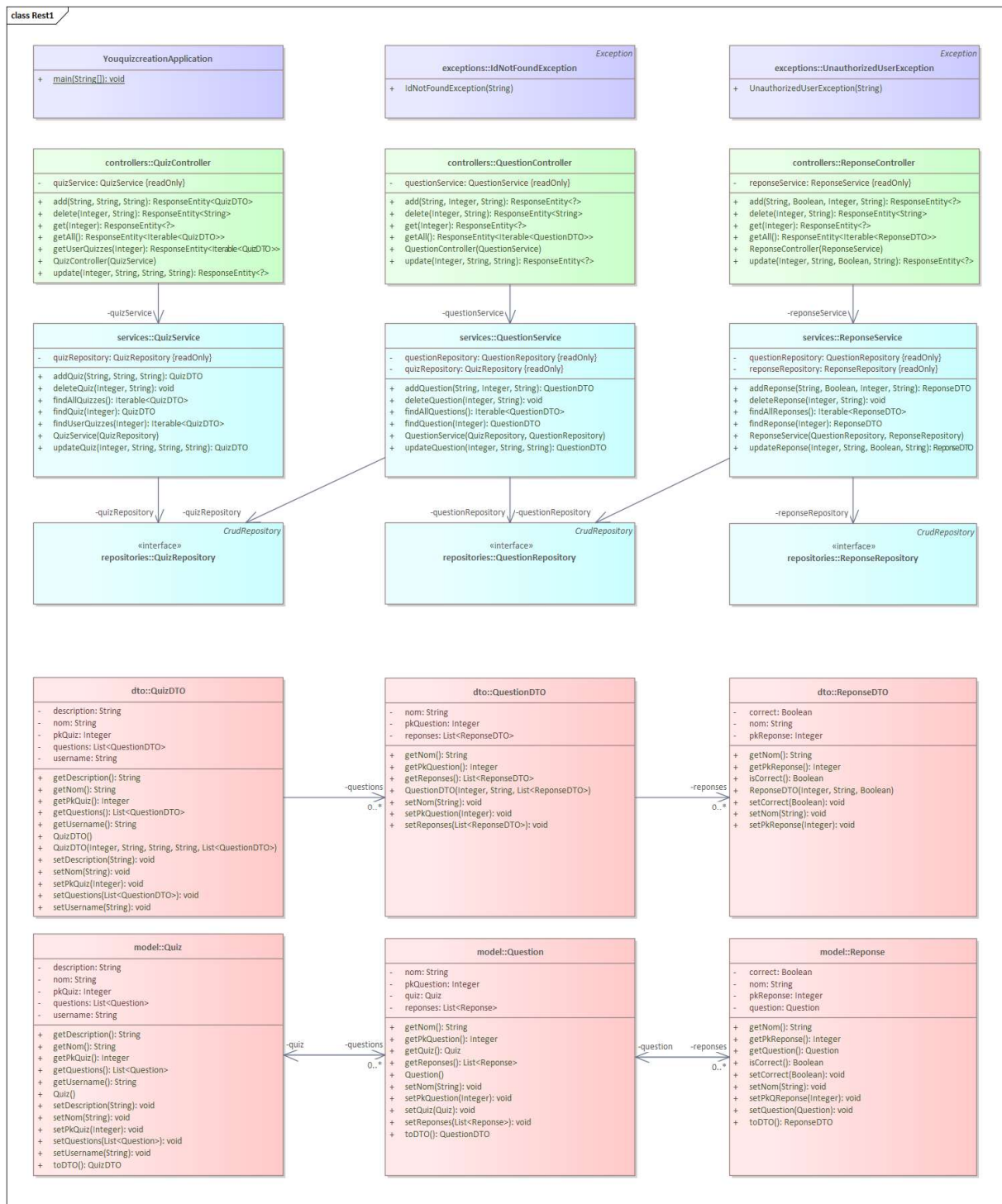
Rest 2 :



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

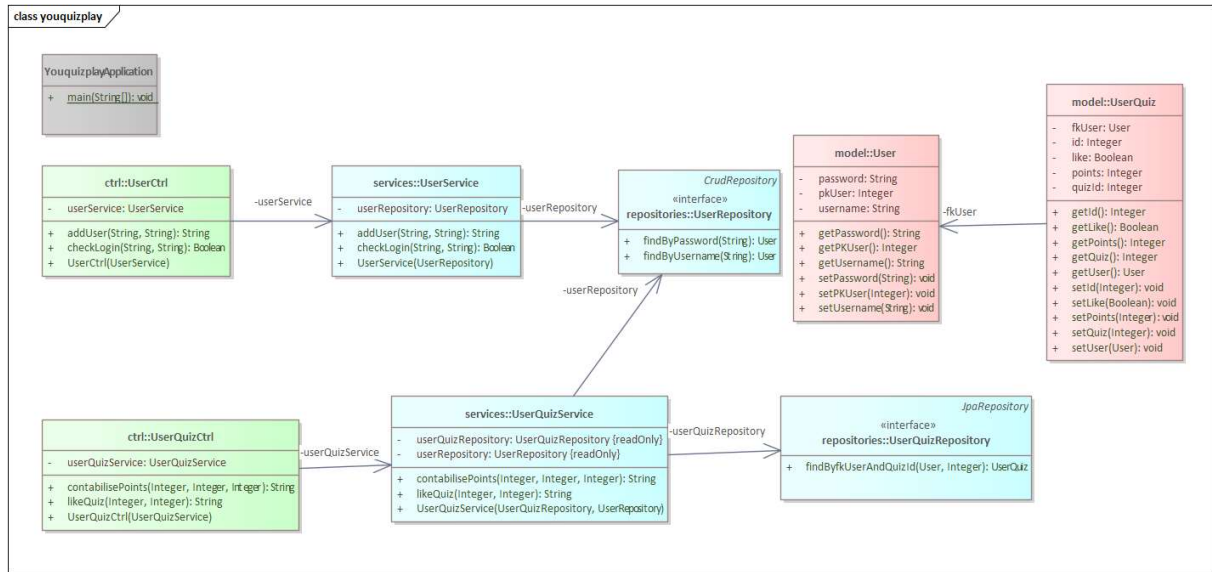
4.1 Class Diagram complet avec les explications de chaque application

Rest 1 :

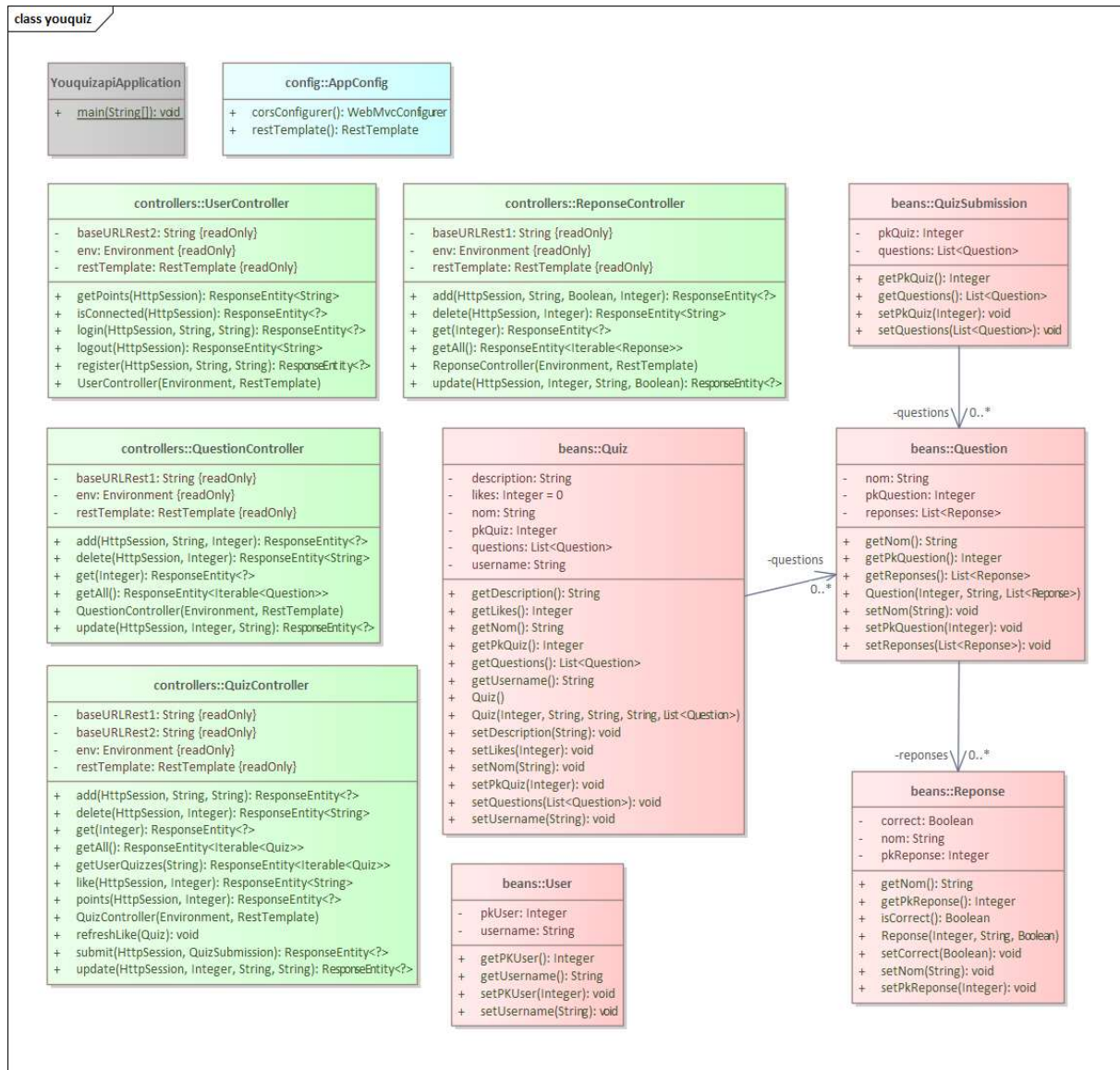


Rest 2 :

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



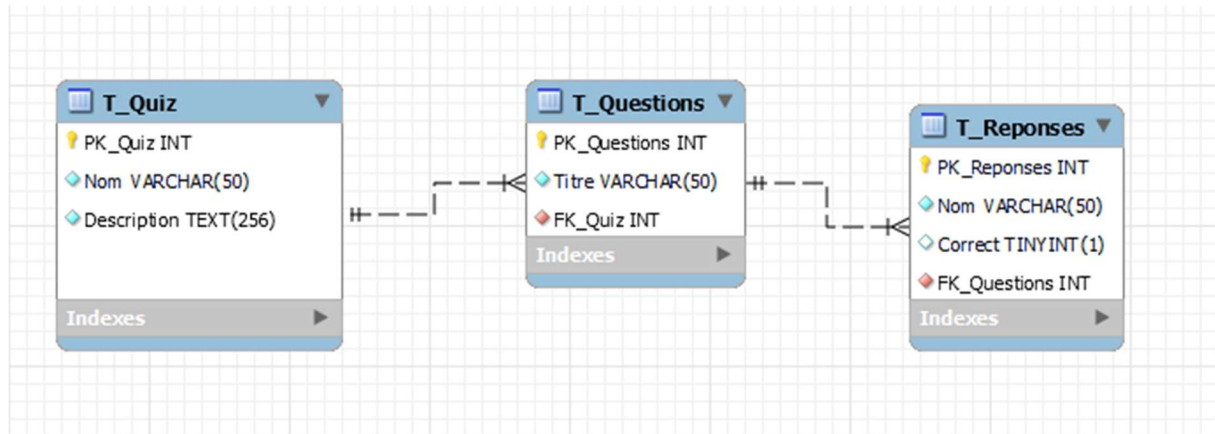
API Gateway :



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

Quand un utilisateur connecté charge la page, donc la vue home par défaut, ses quiz sont chargés au chargement de celle-ci :

```
if (this.vueService.indexCtrl.username !== null) {
  chargerQuizzes(this.vueService.indexCtrl.username, (data) => {
    let boutonNouveauQuiz = $(".bouttonNouveauQuiz");
    boutonNouveauQuiz.show();
    boutonNouveauQuiz.click(() => {
      $("#content").data("currentQuiz", null);
      this.vueService.changerVue("creation");
    });

    $("#quizzes").html("");
    if (data.length > 0) {
      data.forEach((elementQuiz) => {
        let quiz = Quiz.fromJSON(elementQuiz);
        this.nouveauElementQuiz(quiz);
      });
    } else {
      $("#quizzes").html(`<div class="text-gray-500 text-center py-8 mt-16 text-2xl font-bold">Vous n'avez pas encore de quiz</div>`);
    }

    $(".quiz").click((event) => {
      $("#content").data("currentQuiz", $(event.currentTarget).data("quiz"));
      this.vueService.changerVue("creation");
    });

    $(".bouttonDelete").click((event) => {
      event.stopPropagation();
      deleteElement($(event.currentTarget).parent().parent().parent().attr("id"), "quiz",
() => {
      this.load()
    }, () => {
      this.load()
    });
  });
} else {
  $(".bouttonNouveauQuiz").hide();
  $("#quizzes").html(`<div class="text-gray-500 text-center py-8 mt-16 text-2xl font-bold">Vous devez vous connecter pour créer ou voir vos quiz</div>`);
}
```

Les quiz sont pris à partir de la méthode « chargerQuizzes » dans servicesHttp :

```
function chargerQuizzes(username, successCallback, errorCallback) {
  $.ajax({
    type: "GET",
    dataType: "json",
    url: BASE_URL + "quiz/user/" + username,
    success: successCallback,
    error: errorCallback,
  });
}
```

7 Implémentation de l'application <API Gateway>

7.1 Une descente de code APIGateway

Lorsque que l'api gateway reçoit une demande des quiz d'un utilisateur :

```
@GetMapping("/user/{username}")
public ResponseEntity<Iterable<Quiz>> getUserQuizzes(@PathVariable("username") String
username) {
    ResponseEntity<Quiz[]> response = restTemplate.getForEntity(baseUrlRest1 + "/user/" +
username, Quiz[].class);
    List<Quiz> quizList = new ArrayList<>(Arrays.asList(response.getBody()));

    for (Quiz quiz : quizList) {
        refreshLike(quiz);
    }

    return ResponseEntity.ok(quizList);
}
```

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

8.1 Une descente de code de l'API REST

Lorsque que la rest 2 reçoit une demande des quiz d'un utilisateur :

```
@GetMapping("/user/{username}")
public ResponseEntity<Iterable<QuizDTO>> getUserQuizzes(@PathVariable("username") String
username) {
    return new ResponseEntity<>(quizService.findUserQuizzes(username), HttpStatus.OK);
}
```

Qui appelle la méthode de service :

```
public Iterable<QuizDTO> findUserQuizzes(String username) {
    Iterable<Quiz> quizzes = quizRepository.findAllByUsername(username);
    List<QuizDTO> quizDTOS = new ArrayList<>();

    for (Quiz quiz : quizzes) {
        quizDTOS.add(quiz.toDTO());
    }

    return quizDTOS;
}
```

9 Hébergement

Le client 1 permettant de la création de quiz est hébergé sur :

[YouQuizCreation \(emf-informatique.ch\)](http://YouQuizCreation(emf-informatique.ch))

Le client 2 permettant de remplir les quiz est hébergé sur :

richozm.emf-informatique.ch/133

L'api gateway est accessible sur :

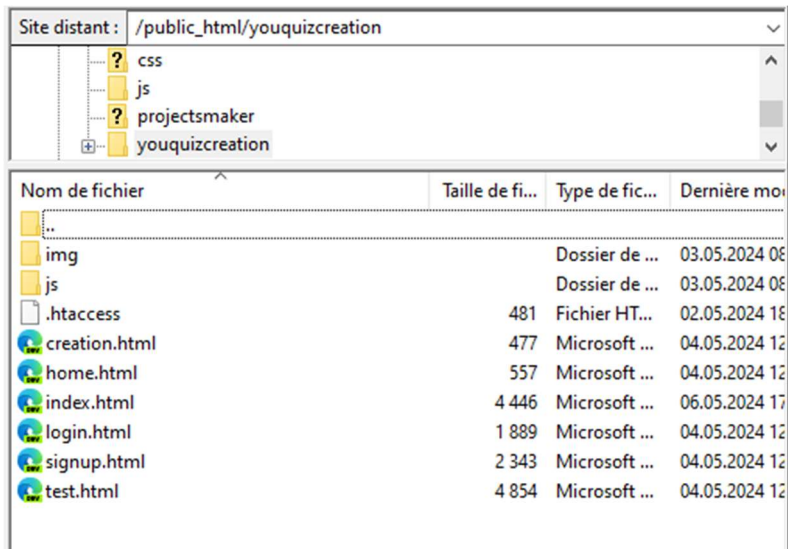
<https://backend-6.emf4you.ch/>

Et sa documentation sur :

[Swagger UI \(emf4you.ch\)](http://SwaggerUI(emf4you.ch))

10 Installation du projet complet avec les 5 applications

Les clients sont déposés sur nos serveurs de l'école en ftp :



Pour les rests api et l'api gateway, ils ont été installés à l'aide d'un docker compose sur un serveur de docker derrière un proxy, seul le container de l'api gateway à un port exposé sur le serveur docker, voici le docker compose :

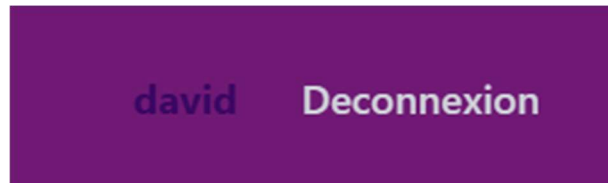
```
version: '3.9'

services:
  apigateway:
    image: dadamomo/youquiz-apigateway:latest
    container_name: youquiz-apigateway
    environment:
      - REST1_URL=http://youquiz-rest1:8080
      - REST2_URL=http://youquiz-rest2:8080
    ports:
      - "8080:8080"
    depends_on:
      - youquiz-rest1
      - youquiz-rest2
    restart: unless-stopped
  youquiz-rest1:
    image: dadamomo/youquiz-rest1
    container_name: youquiz-rest1
    depends_on:
      - db
    environment:
      - DATABASE_URL=jdbc:mariadb://db:3306/youquizcreation
    restart: unless-stopped
  youquiz-rest2:
    image: ricooz/youquiz-rest2
    container_name: youquiz-rest2
    depends_on:
      - db
    environment:
      - DATABASE_URL=jdbc:mariadb://db:3306/youquizplay
    restart: unless-stopped
  db:
    image: mariadb:latest
    environment:
      - MARIADB_ROOT_PASSWORD=emf123
    volumes:
      - ./data:/var/lib/mysql
      - ./docker-entrypoint-initdb.d:/docker-entrypoint-initdb.d
    restart: unless-stopped
```


11 Tests de fonctionnement du projet

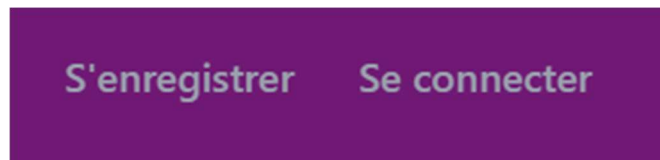
11.1 Login

Lorsque je me connecte avec un utilisateur qui existe déjà, dans mon cas, le nom d'utilisateur est david et le mot de passe emf, en haut à droite doit s'afficher le nom d'utilisateur ainsi que le bouton de déconnexion :



11.2 Log out

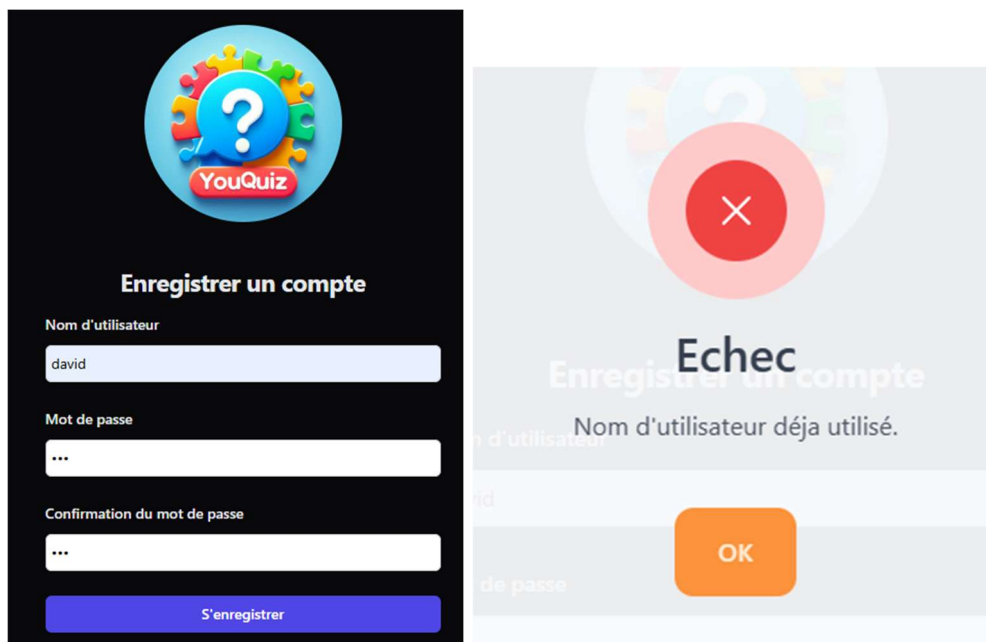
Lorsque l'utilisateur clique sur le bouton de déconnexion, en haut à droite doit s'afficher le bouton d'enregistrement et de login :



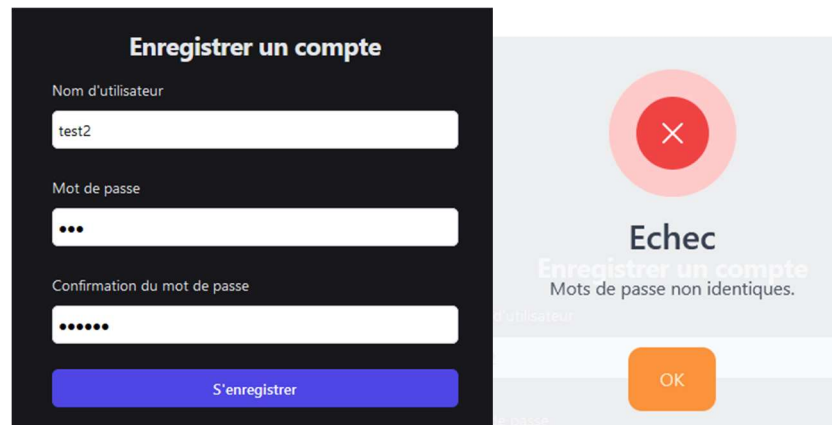
11.3 Enregistrement

L'utilisateur peut créer un nouvel utilisateur. Si le nom d'utilisateur existe déjà, la création est refusée :

Création invalide car conflit de nom :

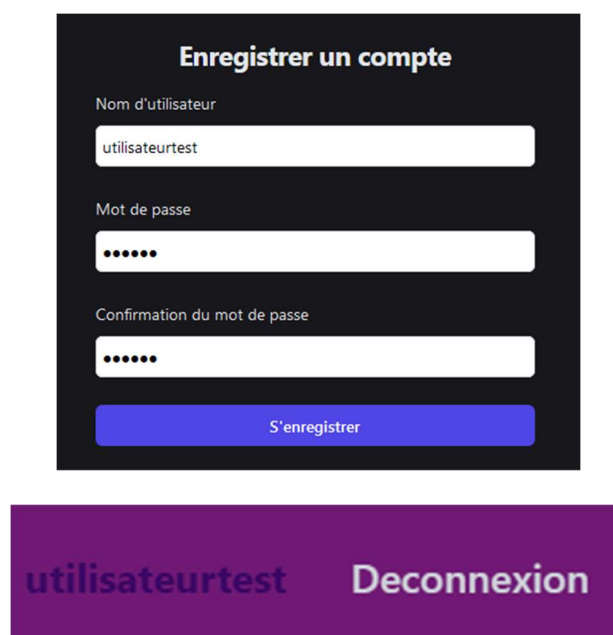


Création invalide car mot de passe et confirmation invalide :



The image shows a registration form titled "Enregistrer un compte" on a dark background. The form has three input fields: "Nom d'utilisateur" with the value "test2", "Mot de passe" with masked characters "...", and "Confirmation du mot de passe" with masked characters ".....". A blue button labeled "S'enregistrer" is at the bottom. To the right, a light gray box displays an error message: "Echec Enregistrer un compte" with a red circle containing a white 'X'. Below the title, it says "Mots de passe non identiques." and "utilisateur". At the bottom of the error box is an orange button labeled "OK".

Création valide :



The image shows the registration form after a successful login. The "Nom d'utilisateur" field now contains "utilisateurtest". Below the form, a purple bar contains the text "utilisateurtest" and a button labeled "Deconnexion".

On voit que l'utilisateur a pu être créer.

11.4 Créer un quiz

Quand l'utilisateur clique sur bouton « + Quiz », la page d'un quiz vierge apparaît :

Création

Titre
titre

Description
description

Questions
+

🔒 ✕

11.5 Sélection un quiz

Quand l'utilisateur clique sur un de ces quiz crée précédemment dans la page d'accueil, la page de modification s'ouvrira :

Astronomie david

Un ensemble de question sur l'astronomie

4 Questions ★ 0 🗑️

Création

Titre
Astronomie

Description
Un ensemble de question sur l'astronomie

Questions
+

Quelle est la planète la plus proche du soleil ?
Mars
Mercure
Vénus
Terre


Quelle planète est connue comme la planète rouge ?
Vénus
Mars



11.6 Sauvegarder un quiz



Quand un utilisateur clique sur le bouton de sauvegarde, le quiz est sauvegardé :



Astronomiedavid



Un ensemble de question sur l'astronomie

4 Questions★ 0










Astronomiedavid

Un ensemble de question sur l'astronomie

5 Questions★ 0

12 Auto-évaluations et conclusions

12.1 Auto-évaluation

Je pense avoir bien travaillé durant ce module, toutes les fonctionnalités ont été implémentées comme prévu.

12.2 Conclusion

Je pense que ce module c'est très bien passé, j'ai personnellement pris du plaisir à réaliser mon projet et L'idée trouvé avec Matteo était assez riche pour nous motivé à travailler agréablement.