133 - Réaliser des applications Web en Session-Handling

Rapport Personnel

Date de création : 21.03.2024
Version 1 du 18.04.2024

Gustavo Neves

Module du 21.03.2024
au date fin…

Table des Matières

1 Introduction

En quelques phrases ce que vous en comprenez (objectifs du module). Objectif de l'IM sous Sharepoint.

2 Tests technologiques selon les exercices

2.1 Installation et Hello World

2.1.1 C'est quoi le build et le run de Java ?

Le build est le processus de conversion du code source Java (fichiers .java) en bytecode Java (fichiers .class) compréhensible par la machine virtuelle Java (JVM)

Le run est le processus de démarrage d'une application Java pour qu'elle s'exécute sur la machine virtuelle Java (JVM)

2.1.2 Quel outil a-t-on utiliser pour build le projet ?

J'ai utilisé Maven

2.1.3 Y a-t-il un serveur web ?

Oui

2.1.4 Version de java utilisée

Java 17.0.6

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

Apache Tomcat/10.1.16

2.2 Conteneurisation

2.2.1 Pourquoi faire un container pour une application Java ?

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

Oui et il se situe en local.

2.2.3 A quoi faut-il faire attention?

Faut faire très attention aux versions java utilisées et donc la cohérence entre les différents fichiers (les 3 doivent avoir la même version) :

DockerFile –

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

pom.xml –

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

Linux –

```
gugu@WSTEMFA39-13:~$ java -version
openjdk version "17.0.10" 2024-01-16
OpenJDK Runtime Environment (build 17.0.10+7-Ubuntu-122.04.1)
OpenJDK 64-Bit Server VM (build 17.0.10+7-Ubuntu-122.04.1, mixed mode, sharing)
```

2.3 Création d'un projet Spring Boot

2.3.1 Annotations utilisées

2.3.1.1 @RestController

Indique un contrôleur qui écoute les requêtes entrantes sur des chemins (ou endpoints) spécifiques et retourne des réponses.

2.3.1.2 @GetMapping

Indique que la méthode gère les requêtes GET.

2.3.1.3 @PostMapping

Indique que la méthode gère les requêtes POST.

2.3.1.4 @PutMapping

Indique que la méthode gère les requêtes PUT.

2.3.1.5 @RequestParam

Extrait les paramètres de la requêtes http/s.

2.3.1.6 @RequestBody

Extrait le body de la requête http/s.

2.4 Connexion à la DB JDBC

Avec cet exercice on interagit avec un DB pour aller chercher des informations. La méthode qu'on utilise pour faire les requêtes c'est JDBC.

Mettre à la place de localhost vu l'existance de deux container :

```
final String url = "jdbc:mysql://host.docker.internal:" + port + "/" + dbName
+ "?serverTimezone=CET";
```

Utiliser la ligne suivante sur la méthode de connexion pour que ça marche sur le web :

```
Class.forName("com.mysql.jdbc.Driver");
```

2.5 Connexion à la DB JPA

2.5.1 À quoi sert l'annotation @Autowired dans vos controlleur pour les Repository ?

Dans le contexte des contrôleurs Spring utilisant des repositories, `@Autowired` est utilisé pour injecter les instances des repositories dans les contrôleurs.

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

L'annotation `@ManyToOne` dans l'entité Skieur définit une relation Many-to-One avec l'entité Pays, indiquant que chaque skieur est associé à un seul pays, et spécifiant comment charger les données du pays associé lorsque le skieur est récupéré.

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

Dans l'extrait de code donné, le FetchType utilisé est `FetchType.EAGER`. Cela signifie que lorsqu'une instance de l'entité Skieur est récupérée (par exemple, lors d'une requête pour obtenir un skieur à partir de la base de données), l'entité associée Pays sera également récupérée immédiatement.

2.5.4 FetchType LAZY

Avec `LAZY`, les données de l'entité associée ne sont pas chargées immédiatement lors de la récupération de l'entité principale.
Au lieu de cela, les données de l'entité associée ne sont chargées qu'au moment où elles sont explicitement demandées

2.5.5 Code important

Récupérer pays correspondant à l'id :

```
Optional<Pays> objPays;
        try {
            objPays = paysRepository.findById(paysID);
        } catch (Exception e) {
            return "Error not found";
        }
        newSkieur.setPays(objPays.get());
```

2.6 Connexion à la DB JPA avec DTO

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

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.

Repo :
Les repository servent de pont entre votre application Spring et la base de données pour effectuer des opérations CRUD dans la base de données. Il utilise la convention de nommage de méthodes de Spring Data JPA pour générer automatiquement les requêtes SQL nécessaires en fonction des méthodes définies dans l'interface.

Service :
Les services permettent d'isoler les différentes parties du modèle pour avoir seulement la logique métier, facilitant ainsi la maintenance, la réutilisabilité et l'évolutivité de l'application. Les services agissent comme une couche intermédiaire qui traite les données, effectue des opérations métier et coordonne les interactions entre les différentes parties de l'application.

DTO :
Les DTO sont des objets utilisés pour transférer des données entre les différentes parties d'une application, souvent entre la couche de présentation (par exemple, les contrôleurs dans une application Spring MVC) et la couche de services ou la couche d'accès aux données.

Controller :
Le Controller permet de faire le pont entre la vue et les différents services de l'application.

2.7 Gestion des sessions

J'ai créé une nouvelle classe appelée ControllerSession et juste au-dessus de sa déclaration j'ai mis le suivant :

```
@RestController
@RequestMapping(path = "/user")
```

Ensuite j'ai fait 3 méthode. Login, qui a comme paramètres l'username et le mot-de-passe, logout et visites qui retourne le nombre de fois que la session a été visitée.

Login :

```
    @PostMapping(path = "/login")
    public ResponseEntity<String> login(@RequestParam String username,
@RequestParam String pwd, HttpSession session) {
        if ((username.equals("test")) && (pwd.equals("test"))) {
            session.setAttribute("username", username);
            session.setAttribute("visites", 0);
            return ResponseEntity.ok("Login successful");
        } else {
            return
ResponseEntity.status(HttpStatus.UNAUTHORIZED).body("Invalid credentials");
        }
    }
```

Logout :

```
    @PostMapping(path = "/logout")
    public ResponseEntity<String> logout(HttpSession session) {
        if (session.getAttribute("username") == null) {
            return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body("No one
is logged");
        } else {
            session.invalidate();
            return ResponseEntity.ok("Logout successful");
        }
    }
```

Visites :

```
    @GetMapping(path = "/visites")
    public ResponseEntity<String> getVisites(HttpSession session){
        if(session.getAttribute("username") != null){
            int nbrSessions = (int) session.getAttribute("visites");
            nbrSessions += 1;
            session.setAttribute("visites", nbrSessions);
            return ResponseEntity.ok("Visites: " + nbrSessions);
        } else {
            return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body("No
session available");
```

```
        }
    }
```

Il faudra aussi mettre le suivant dans le fichier « application.properties » :

```
spring.datasource.url=${DATABASE_URL:jdbc:mysql://localhost:3306/133ex5}
```

Et ceci dans le docker :

```
ENV DATABASE_URL=jdbc:mysql://host.docker.internal:3306/133ex5
```

Sans ces lignes on ne pourra pas lancer le container sur docker.

Important !!

À chaque fois qu'on veut utiliser la session dans une méthode il faudra qu'on mette en paramètre :

```
HttpSession session
```

2.8 Documentation API avec Swagger

Swagger sert à générer la documentation de l'API automatiquement et donc économiser du temps.

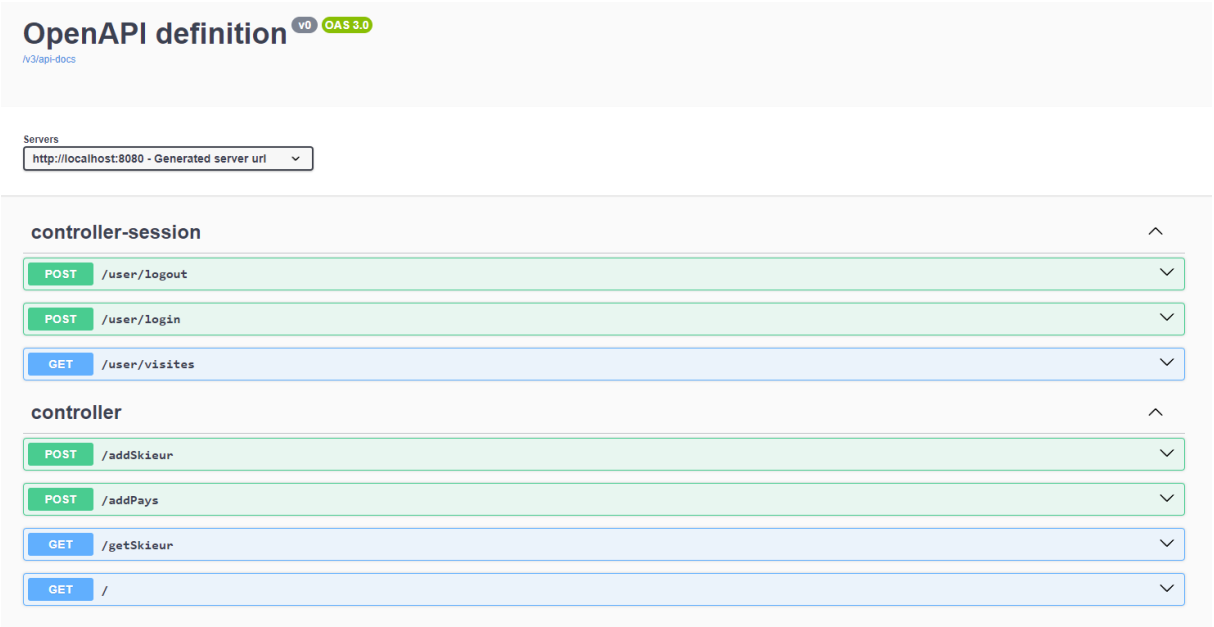
Il faudra mettre la dépendance suivant sur « pom.xml » :

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

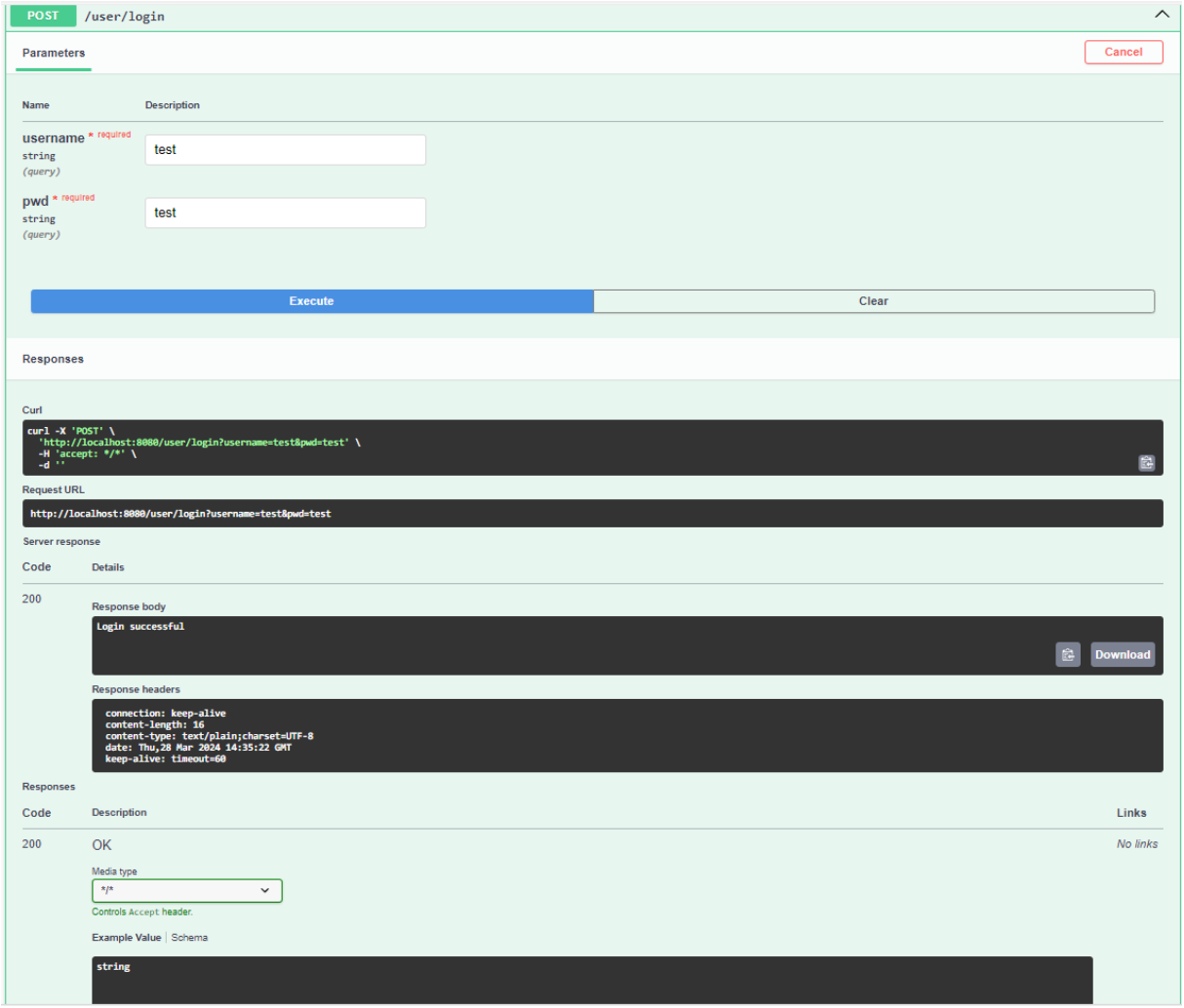
Et ensuite sur l'adresse, après avoir re-build l'application :

http://localhost:8080/swagger-ui/index.html#/

On verra toutes nos méthodes et on pourra donc les exécuter pour avoir leur doc :



Exemple d'exécution :



Attention !!

Il faut faire en sorte de générer une erreur pour qu'il documente aussi l'erreur.

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

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

3.3 Sequence System global entre les applications

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

4.1 Class Diagram complet avec les explications de chaque application

5 Bases de données

5.1 Modèles WorkBench MySQL

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 …

13 Conclusion

Ce que j'ai appris
Ce que j'ai aimé
Mes axes d'amélioration
1)