133 – Réaliser des applications web en session-handing

Rapport personnel

Version 1 du 18 mars 2025

Crée le 17.03.25

Bernardo Simões

|  |  |
| --- | --- |
| Une image contenant texte  Description générée automatiquement | Module du 17.03.25 au 15.04.25 |

Table des matières

[1 Introduction 4](#_Toc193097089)

[1.1 Objectifs du module/compétences 4](#_Toc193097090)

[1.2 Enjeux dans mon métier 4](#_Toc193097091)

[2 Tests technologiques exercices 5](#_Toc193097092)

[2.1 Installation et Hello World 5](#_Toc193097093)

[2.2 Conteneurisation 5](#_Toc193097094)

[2.3 Création d'un projet Spring Boot 5](#_Toc193097095)

[2.4 Connexion à la DB JDBC 5](#_Toc193097096)

[2.5 Connexion à la DB JPA 5](#_Toc193097097)

[2.6 Connexion à la DB JPA avec DTO 5](#_Toc193097098)

[2.7 Gestion des sessions 5](#_Toc193097099)

[2.8 Documentation API avec Swagger 5](#_Toc193097100)

[2.9 Hébergement 5](#_Toc193097101)

[3 Conclusion 6](#_Toc193097102)

[3.1 Ce que j’ai appris, ce qui est important 6](#_Toc193097103)

[3.2 Ce que j’ai apprécié 6](#_Toc193097104)

[3.3 Mes propositions d’amélioration 6](#_Toc193097105)

[3.4 Conclusion personnelle 6](#_Toc193097106)

[3.4.1 Points forts 6](#_Toc193097107)

[3.4.2 Points faibles 6](#_Toc193097108)

# Introduction

## Objectifs du module/compétences

## Enjeux dans mon métier

# Tests technologiques exercices

## Installation et Hello World

### Prérequis

Il faut installer WSL ainsi que docker via les liens suivants :

* <https://learn.microsoft.com/fr-fr/windows/wsl/install>
* <https://docs.docker.com/desktop/wsl/#download>

### Extensions VS code

Pour que tous fonctionnent correctement il faut que les extensions suivantes soit installé et fonctionnelle.

Une image contenant texte, capture d’écran, Police

Le contenu généré par l’IA peut être incorrect.Une image contenant texte, capture d’écran, Police

Le contenu généré par l’IA peut être incorrect.Une image contenant texte, capture d’écran, Police, logo

Le contenu généré par l’IA peut être incorrect.Une image contenant texte, capture d’écran, Police, Graphique

Le contenu généré par l’IA peut être incorrect.

### Test Hello word

Faire le clonage avec la commande suivante et ouvrir le dossier :

git clone <https://github.com/spring-guides/gs-rest-service.git>

Utiliser la commande suivante pour installer le jdk 17.

sudo apt install openjdk-17-jdk -y

echo 'export JAVA\_HOME=/usr/lib/jvm/java-17-openjdk-amd64' >> ~/.bashrc

source ~/.bashrc

echo $JAVA\_HOME

Si vous avez des problèmes avec l’installation poursuivez avec la commande **apt update** qui permettra de mettre a jour et ressayez la commande au-dessus.

### Questions

Démarrage de l'application :

L'application Spring Boot démarre avec le JDK 17, ce qui est indiqué dans le premier log. Cela signifie que l'environnement Java est configuré et prêt à exécuter l'application.

Profil de configuration :

L'application essaie de charger un profil de configuration spécifique. Si aucun profil n'est défini, elle utilise le profil par défaut (ici "default"), ce qui est souvent le cas dans les environnements de développement.

Initialisation du serveur web (Tomcat) :

Spring Boot utilise Tomcat, un serveur web embarqué, pour héberger l'application. Il est configuré pour écouter sur le port 8080 (par défaut).

Tomcat est initialisé pour recevoir des requêtes HTTP.

Initialisation du contexte Spring :

Spring Boot crée le contexte d'application (le cœur de Spring) et le contexte web (qui gère les interactions avec le serveur web).

Cela inclut la configuration des beans, des contrôleurs, des services, etc. qui composent ton application.

Démarrage du serveur sur le port 8080 :

Tomcat démarre effectivement, et le serveur web est maintenant en fonctionnement. Tu peux maintenant accéder à l'application via http://localhost:8080.

Initialisation du DispatcherServlet :

Le DispatcherServlet de Spring est le composant principal qui gère les requêtes HTTP dans une application Spring MVC. Il prend une requête et la dirige vers le bon contrôleur.

Une fois initialisé, le DispatcherServlet est prêt à recevoir les requêtes HTTP et à les traiter.

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

Build : C'est la préparation du projet pour le rendre exécutable (compilation des fichiers .java en un fichier .jar ou .war).

Run : C'est l'exécution du projet après le build.

Outil utilisé : Il semble que Maven a été utilisé pour construire le projet (si le fichier pom.xml est présent).

* **Y a-t-il un serveur web ?**

Oui, il y a un serveur web **Tomcat** utilisé dans ce projet.

* **Quelle version de java est utilisée ?**

La version de Java utilisée est **Java 17.0.14**.

* **S’il y a un serveur web, quelle version utilise-t-il ?**

Le serveur web **Tomcat** utilise la version **10.1.24**.

## Conteneurisation

### Création DockerFile

Créer le fichier suivant comme écrit dans l’exercice

FROM openjdk:17-jdk-alpine

VOLUME /tmp

ARG JAVA\_OPTS

ENV JAVA\_OPTS=$JAVA\_OPTS

COPY complete/target/rest-service-complete-0.0.1-SNAPSHOT.jar complete.jar

EXPOSE 8080

ENTRYPOINT exec java $JAVA\_OPTS -jar complete.jar

Faire les commandes suivantes qui permettent de build l’image

cd complete

sudo apt install maven

mvn clean package

Voici ce qui m’est rendu dans l’onglet docker :

Une image contenant texte, capture d’écran, Police, logiciel

Le contenu généré par l’IA peut être incorrect.

### Questions

* Pourquoi faire un container pour une application Java ?

La conteneurisation d'une application Java (Spring Boot) avec Docker facilite grandement son déploiement et améliore sa portabilité. Le conteneur inclut toutes les dépendances nécessaires, ce qui garantit que l'application fonctionnera exactement de la même manière sur n’importe quel environnement (développement, test, production), évitant ainsi les problèmes liés aux différences de configuration ou d’environnement. Le container est également léger, rapide à démarrer, et facilement reproductible.

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

Oui, un serveur web intégré est utilisé. Avec Spring Boot, c'est généralement un serveur embarqué (Tomcat par défaut), directement inclus dans l’application elle-même. Il est lancé automatiquement lorsqu'on exécute l'application via le .jar généré. Le serveur web se trouve donc à l'intérieur du fichier .jar qui est exécuté par le conteneur Docker.

## Création d'un projet Spring Boot

### Observations

Annotations utilisées dans le controller :

* **@RestController**

Permet à Spring Boot de reconnaître la classe comme un contrôleur REST, capable de traiter les requêtes HTTP et renvoyer directement les réponses au format JSON ou texte brut.

* **@GetMapping("/getExample")**

Méthode permettant de gérer les requêtes HTTP GET vers l'endpoint /getExample.

* **@PostMapping("/postExample")**

Méthode permettant de gérer les requêtes HTTP POST vers l'endpoint /postExample.

* **@PutMapping("/putExample")**

Méthode permettant de gérer les requêtes HTTP PUT vers l'endpoint /putExample.

* **@RequestParam (pour GET)**

Récupère un paramètre de requête depuis l'URL (exemple : ?name=Bernardo).

* **@RequestBody (pour POST et PUT)**

Permet de récupérer le corps des requêtes POST et PUT envoyées par l'utilisateur.

À ajouter dans le main de l’exercice :

@ComponentScan("ctrl")

## Connexion à la DB JDBC

### Container mySQL

Pour créer la DB, on commence par démarrer un nouveau container qui contiendra le serveur MySQL avec ces commandes depuis le terminal du WSL :

#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 3308:3306 -e MYSQL\_ROOT\_HOST=% -e MYSQL\_ROOT\_PASSWORD=emf123 -v /opt/mysql:/var/lib/mysql mysql/mysql-server:8.0

docker stop mysql  # Arrêter le conteneur existant  
docker rm mysql    # Supprimer le conteneur existant  
docker run --name mysql -d -p 3308:3306 -e MYSQL\_ROOT\_HOST=% -e MYSQL\_ROOT\_PASSWORD=emf123 -v /opt/mysql:/var/lib/mysql mysql/mysql-server:8.0

### Connexion JDBC

Sur la base de l'exercice 3, on ajoute une fonctionnalité getPays à notre contrôleur.

Cette fonctionnalité doit aller rechercher dans la DB (à disposition dans le OneNote) la liste des pays.

On utilise le WrkDB à disposition dans le OneNote pour la connexion jdbc.

package ex4.ex4.ctrl;

import ex4.ex4.wrk.WrkDB;

import org.springframework.web.bind.annotation.\*;

import java.util.List;

@RestController

public class Controller {

   private final WrkDB wrkDB = new WrkDB("3308", "bd\_kitzbuehl"); // Mets le bon nom de ta DB

    @GetMapping("/getPays")

   public List<String> getPays() {

       return wrkDB.getPays(); // Appelle WrkDB pour récupérer la liste des pays

   }

   // Handler pour GET

   @GetMapping("/getExample")

   public String getExample(@RequestParam(value = "name", defaultValue = "World") String name) {

       return String.format("Hello, %s!", name);

   }

   // Handler pour POST

   @PostMapping("/postExample")

   public String postExample(@RequestBody String body) {

       return "Received POST request with body: " + body;

   }

   // Handler pour PUT

   @PutMapping("/putExample")

   public String putExample(@RequestBody String body) {

       return "Received PUT request with body: " + body;

   }

}

On ajoute ensuite les drivers JDBC dans le pom.xml : faire un clic droit sur pom.xml et "Add starter". Choisissez d'ajouter le driver MySQL :

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Le contenu généré par l’IA peut être incorrect.

Ceci a dû être ajouté à votre fichier pom.xml :

<dependency>

<groupId>com.mysql</groupId>

<artifactId>mysql-connector-j</artifactId>

<scope>runtime</scope>

</dependency>

### Test avec un Run en local dans VSC

Dans le backend, vous pouvez maintenant vous connecter à la base de données en utilisant les credentials :

'mysql:host=localhost;port=3308;dbname=nomDB;charset=utf8', 'root', 'emf123'

### Test avec un container pour l’application et la MySQL

Afin d'être capable de se connecter depuis le container backend sur la base de données MySQL contenue dans le container MYSQL, il faut créer un nouveau réseau Docker et connecter ces deux containeurs sur ce réseau. Voici les commandes à exécuter sur le CLI de WSL :

docker network create asgard

docker network connect asgard mysql

docker network connect asgard nomContainerBackend

Puis dans le backend, on change localhost en host.docker.internal pour l'accès à la DB :

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

## Connexion à la DB JPA

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

* L'annotation @Autowired permet à Spring d'effectuer automatiquement l'injection de dépendance. Concrètement, Spring injecte automatiquement une instance de la classe implémentant l'interface SkieurRepository.
* Cela évite d'avoir à créer manuellement un objet via new ou une autre instanciation explicite, facilitant ainsi la gestion des objets et rendant le code plus clair et plus propre.

@Autowired

private SkieurRepository skieurRepository;

**À quoi sert l'annotation @ManyToOne dans l'entité Skieur ?**

* L'annotation @ManyToOne indique une relation plusieurs-à-un entre deux entités JPA.
* Dans ce cas, plusieurs skieurs peuvent être liés à un seul pays.
* Cela crée automatiquement une clé étrangère dans la table des skieurs (ici fk\_pays), reliant chaque skieur à son pays.

@ManyToOne(fetch = FetchType.EAGER)

@JoinColumn(name = "fk\_pays")

private Pays pays;

**Quel FetchType est utilisé et pourquoi ?**

* Ici, le type utilisé est FetchType.EAGER. Ce mode charge immédiatement les données du pays associé lorsqu'un skieur est récupéré depuis la base de données.

L'avantage :

* Simplicité d'utilisation : les données associées (ici le pays) sont directement disponibles, sans nécessiter une requête supplémentaire.

L'inconvénient potentiel :

* Peut réduire les performances si beaucoup de données associées doivent être chargées alors qu'on n’en a pas toujours besoin.

## Connexion à la DB JPA avec DTO

### Utilisation de DTO

L'utilisation de DTO permet de :

* Contrôler les données exposées par l'API.
* Réduire les problèmes de Lazy Loading.
* Améliorer les performances en ne chargeant que les données essentielles.

Exemple de DTO (SkieurDTO.java) :

public class SkieurDTO {

private Long id;

private String nom;

private String prenom;

private String pays;

// Getters et Setters

}

### Utilisation de service

L’utilisation de services permet de séparer la logique métier du contrôleur, rendant l’architecture plus modulaire et maintenable.

* Les services agissent comme une couche intermédiaire entre le Controller et le Repository.
* Ils permettent de centraliser la logique métier et d’améliorer la réutilisabilité du code.

Exemple d’un service (SkieurService.java) :

@Service

public class SkieurService {

private final SkieurRepository skieurRepository;

public SkieurService(SkieurRepository skieurRepository) {

this.skieurRepository = skieurRepository;

}

public List<SkieurDTO> getAllSkieurs() {

return skieurRepository.findAll()

.stream()

.map(skieur -> new SkieurDTO(skieur.getId(), skieur.getNom(), skieur.getPrenom(), skieur.getPays().getNom()))

.collect(Collectors.toList());

}

}

### Lancement de l’application

Configuration de la base de données

Modifiez le fichier application.properties pour définir l'URL de connexion MySQL :

spring.datasource.url=${DATABASE\_URL:jdbc:mysql://localhost:3306/133ex6}

spring.datasource.username=root

spring.datasource.password=1234

spring.jpa.hibernate.ddl-auto=update

### Observations

Pourquoi y a-t-il un SkieurDTO mais pas de PaysDTO ?

* Dans cette implémentation, les données du pays sont intégrées dans SkieurDTO, ce qui évite la création d’un DTO spécifique pour Pays. Cela permet d'éviter des relations complexes et de simplifier les appels API.

Rôle des différentes couches :

* Model : Définit les entités de la base de données.
* Repository : Gère l’accès aux données.
* DTO : Sert d’interface entre l’API et les entités.
* Service : Contient la logique métier.
* Controller : Expose les endpoints de l’API.

## Gestion des sessions

### Code source du UserController

package com.example.ex7.controller;

import jakarta.servlet.http.HttpSession;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.\*;

@RestController

@RequestMapping("/user")

public class UserController {

@PostMapping("/login")

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

if ("user".equals(username) && "pass".equals(password)) {

session.setAttribute("username", username);

session.setAttribute("visites", 0);

return ResponseEntity.ok("Connecté");

} else {

return ResponseEntity.badRequest().body("Échec de la connexion");

}

}

@GetMapping("/visites")

public ResponseEntity<String> visites(HttpSession session) {

if (session.getAttribute("username") != null) {

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

if (visites == null) {

visites = 0;

}

visites++;

session.setAttribute("visites", visites);

return ResponseEntity.ok("Nombre de visites: " + visites);

} else {

return ResponseEntity.badRequest().body("Non connecté");

}

}

@PostMapping("/logout")

public ResponseEntity<String> logout(HttpSession session) {

session.invalidate();

return ResponseEntity.ok("Déconnecté");

}

}

### Test Postman

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Le contenu généré par l’IA peut être incorrect.

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Le contenu généré par l’IA peut être incorrect.

## Documentation API avec Swagger

### Dépendances Maven

Ajoutez ces dépendances à votre fichier pom.xml :

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

</dependencies>

### Configuration de la session dans application.properties

server.servlet.session.timeout=30m

server.servlet.session.persistent=true

Cela permet de conserver la session pendant 30 minutes.

### Test

Une image contenant texte, capture d’écran, logiciel, Icône d’ordinateur

Le contenu généré par l’IA peut être incorrect.

## Hébergement

Création du repository public dans docker hub

Une image contenant texte, capture d’écran, logiciel, Police

Le contenu généré par l’IA peut être incorrect.

Création du token de notre compte dans docker :

Une image contenant texte, capture d’écran, Police, logiciel

Le contenu généré par l’IA peut être incorrect.

Une image contenant texte, capture d’écran, Police, logiciel

Le contenu généré par l’IA peut être incorrect.

Faire le login à l’aide de la commande suivante dans le terminal :

docker login -u username

Mettre le mot de passe qui est donnée lors de la création du token.

Commandes qui permettent de tag et de push l’image sur dockerhub :

docker tag nomImageLocale:latest username/nomRepo:latest

docker push username/nomRepo:latest

Une image contenant texte, capture d’écran, Police, logiciel

Le contenu généré par l’IA peut être incorrect.

# Conclusion

## Ce que j’ai appris, ce qui est important

Durant ce module, j’ai appris à développer une application Spring Boot en suivant une architecture propre avec les bonnes pratiques de séparation des couches (Controller, Service, DTO, Repository). J’ai découvert l’importance de l’injection de dépendance avec @Autowired, ainsi que le fonctionnement de JPA et des relations entre entités.

J’ai également compris comment gérer une session HTTP en Java avec HttpSession sans utiliser d’outils comme JWT ou Spring Security. Cette gestion basique m’a permis de mieux saisir comment fonctionne une session utilisateur côté serveur.

Un autre point clé a été la conteneurisation avec Docker. J’ai appris à créer une image Docker pour mon application Java, à configurer la connexion entre conteneurs, et à héberger une image sur Docker Hub.

Enfin, j’ai découvert Swagger, un outil très utile pour documenter automatiquement mes APIs, les tester et les présenter proprement.

## Ce que j’ai apprécié

J’ai particulièrement apprécié la clarté des exercices et le côté progressif du module. Chaque exercice s’appuyait sur le précédent, ce qui m’a permis de construire une application complète étape par étape.

J’ai aussi aimé travailler avec Swagger, car cela m’a permis de tester mes endpoints directement via une interface web, ce qui est très pratique pour le développement et les démonstrations.

Le côté concret avec la mise en place de Docker, la configuration réseau entre conteneurs, et l'hébergement de l'image m’a permis de me projeter dans un contexte réel de déploiement.

## Conclusion personnelle

Ce module m’a permis de renforcer mes compétences sur Spring Boot et de consolider mes bases en développement web backend. J’ai également amélioré ma compréhension de l’architecture des applications modernes, et je me sens plus à l’aise pour déployer une application dans un environnement réel.

### Points forts

* Bonne structure du module
* Exercices progressifs et concrets
* Intégration de Docker et Swagger
* Mise en situation proche du réel

### Points faibles

* Quelques difficultés au début avec la configuration de Docker
* La partie session aurait pu être comparée avec d'autres solutions modernes (comme JWT ou Spring Security)