



---

# Rapport personnel du module 133 - Réaliser des applications Web en Session-Handling

Langenegger Max

Début du module le 21.03.2024 , fin de module le 03.05.2023

Création du document le 21.03.2024 12:52:00  
Version 1 du 06.05.2024

## Table des matières

1	Introduction.....	<b>Erreur ! Aucun nom n'a été donné au signet.</b>
2	Tests technologiques selon les exercices ....	<b>Erreur ! Aucun nom n'a été donné au signet.</b>
2.1	Installation et Hello World .....	<b>Erreur ! Aucun nom n'a été donné au signet.</b>
2.1.1	Observez la console pour comprendre comment le projet est lancé et comment il tourne ?	<b>Erreur ! Aucun nom n'a été donné au signet.</b>
2.1.2	C'est quoi le build et le run de Java ? Quel outil a-t-on utiliser pour build le projet ?	<b>Erreur ! Aucun nom n'a été donné au signet.</b>
2.1.3	Quelle version de java est utilisée ? .....	<b>Erreur ! Aucun nom n'a été donné au signet.</b>
2.1.4	S'il y a un serveur web, quelle version utilise-t-il ? ..	<b>Erreur ! Aucun nom n'a été donné au signet.</b>
2.2	Conteneurisation.....	<b>Erreur ! Aucun nom n'a été donné au signet.</b>
2.2.1	Pourquoi faire un container pour une application Java ? .....	<b>Erreur ! Aucun nom n'a été donné au signet.</b>
2.2.2	Y a-t-il un serveur web ? Ou se trouve-t-il ?	<b>Erreur ! Aucun nom n'a été donné au signet.</b>
2.2.3	A quoi faut-il faire attention ? .....	<b>Erreur ! Aucun nom n'a été donné au signet.</b>
2.3	Création d'un projet Spring Boot .....	<b>Erreur ! Aucun nom n'a été donné au signet.</b>
2.4	Connexion à la DB JDBC.....	<b>Erreur ! Aucun nom n'a été donné au signet.</b>
2.5	Connexion à la DB JPA .....	<b>Erreur ! Aucun nom n'a été donné au signet.</b>
2.5.1	À quoi sert l'annotation @Autowired dans vos controlleur pour les Repository ? ..	<b>Erreur ! Aucun nom n'a été donné au signet.</b>
2.5.2	A quoi sert l'annotation @ManyToOne dans l'entité skieur ? ..	<b>Erreur ! Aucun nom n'a été donné au signet.</b>
2.6	Connexion à la DB JPA avec DTO.....	<b>Erreur ! Aucun nom n'a été donné au signet.</b>
2.7	Gestion des sessions.....	<b>Erreur ! Aucun nom n'a été donné au signet.</b>
2.8	Documentation API avec Swagger .....	<b>Erreur ! Aucun nom n'a été donné au signet.</b>
2.9	Hébergement .....	<b>Erreur ! Aucun nom n'a été donné au signet.</b>
3	Bases de données.....	<b>Erreur ! Aucun nom n'a été donné au signet.</b>
3.1	Modèles WorkBench MySQL.....	<b>Erreur ! Aucun nom n'a été donné au signet.</b>
3.2	Implémentation des applications <Le client Ap2>	<b>Erreur ! Aucun nom n'a été donné au signet.</b>
3.3	Une descente de code client .....	<b>Erreur ! Aucun nom n'a été donné au signet.</b>
4	Implémentation de l'application <API Gateway> ...	<b>Erreur ! Aucun nom n'a été donné au signet.</b>
4.1	Une descente de code APIGateway .....	<b>Erreur ! Aucun nom n'a été donné au signet.</b>
5	Implémentation des applications <API élève2>.....	<b>Erreur ! Aucun nom n'a été donné au signet.</b>

5.1	Une descente de code de l'API REST .....	<b>Erreur ! Aucun nom n'a été donné au signet.</b>
6	Hébergement .....	<b>Erreur ! Aucun nom n'a été donné au signet.</b>
7	Installation du projet complet avec les 5 applications .....	<b>Erreur ! Aucun nom n'a été donné au signet.</b>
8	Tests de fonctionnement du projet .....	<b>Erreur ! Aucun nom n'a été donné au signet.</b>
9	Auto-évaluations et conclusions .....	<b>Erreur ! Aucun nom n'a été donné au signet.</b>
9.1	Auto-évaluation et conclusion de .....	<b>Erreur ! Aucun nom n'a été donné au signet.</b>

# 1 Introduction

- 1 Analyser la donnée, projeter la fonctionnalité et déterminer le concept de la réalisation.
- 2 Réaliser une fonctionnalité spécifique d'une application Web par Session-Handling, authentification et vérification de formulaire.
- 3 Programmer une application Web à l'aide d'un langage de programmation compte tenu des exigences liées à la sécurité.
- 4 Vérifier la fonctionnalité et la sécurité de l'application Web à l'aide du plan tests, verbaliser les résultats et, le cas échéant, corriger les erreurs.

## 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 ?

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

The screenshot shows a terminal window with the following output:

```

langeneggerm@STEFA39-14:~/gs-rest-service$ /usr/bin/env /home/langeneggerm/.vscode-server/extensions/redhat.java-1.16.0-linux-x64/jre/17.0.6-linux-x86_64/bin/java @/tmp/cp_c01ajoxna99nnk
j6mlw60ml5w.argfile com.example.restservice.RestServiceApplication

:: Spring Boot ::
(v3.2.0)

2024-03-21T14:11:15.691+01:00 INFO 6679 --- [main] c.e.restservice.RestServiceApplication : Starting RestServiceApplication using Java 17.0.6 with PID 6679 (/home/langeneggerm
/gs-rest-service/complete/target/classes started by langeneggerm in /home/langeneggerm/gs-rest-service)
2024-03-21T14:11:15.695+01:00 INFO 6679 --- [main] c.e.restservice.RestServiceApplication : No active profile set, falling back to 1 default profile: "default"
2024-03-21T14:11:16.173+01:00 INFO 6679 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port 8080 (http)
2024-03-21T14:11:16.178+01:00 INFO 6679 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2024-03-21T14:11:16.178+01:00 INFO 6679 --- [main] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.16]
2024-03-21T14:11:16.210+01:00 INFO 6679 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2024-03-21T14:11:16.211+01:00 INFO 6679 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 485 ms
2024-03-21T14:11:16.481+01:00 INFO 6679 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8080 (http) with context path ''
2024-03-21T14:11:16.486+01:00 INFO 6679 --- [main] c.e.restservice.RestServiceApplication : Started RestServiceApplication in 0.996 seconds (process running for 1.118)
2024-03-21T14:14:57.727+01:00 INFO 6679 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2024-03-21T14:14:57.727+01:00 INFO 6679 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2024-03-21T14:14:57.728+01:00 INFO 6679 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 1 ms
  
```

Below the terminal, a web browser window is open at `localhost:8080/greeting`. The response is a JSON object:

```

{
  "id": 1,
  "content": "Hello, World!"
}
  
```

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

Build :

Le build consiste à transformer les fichiers source de l'application en un format exécutable ou déployable. Cela implique généralement la compilation du code et la génération de fichiers JAR ou WAR, ainsi que d'autres tâches telles que la minification, l'optimisation, etc.

Run (Exécution) :

Une fois que le processus de build est terminé et que notre application est sous forme exécutable, on peut la lancer pour qu'elle accomplisse ses fonctions prévues. Cela implique de démarrer l'application ou le service correspondant et de laisser le programme s'exécuter pour répondre aux requêtes des utilisateurs ou effectuer les tâches prévues.

Pour run et build l'outil

### 2.1.3 Quelle version de java est utilisée ?

Dans les exercices on utilise OpenJdk17

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

Oui, Apache Tomcat/10.1.16

## 2.2 Conteneurisation

### 2.2.1 Pourquoi faire un container pour une application Java ?

Les conteneurs offrent un environnement isolé qui encapsule l'application ainsi que toutes ses dépendances, y compris les bibliothèques, les dépendances système, etc. Cela garantit que l'application fonctionnera de la même manière quel que soit l'environnement dans lequel elle est déployée, qu'il s'agisse d'un ordinateur de développement, d'un serveur de test ou d'un environnement de production.

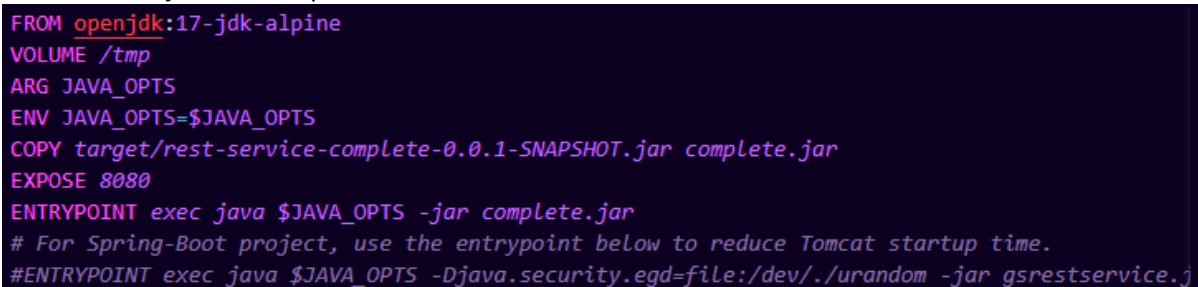
### 2.2.2 Y a-t-il un serveur web ? Ou se trouve-t-il ?

Il y a un serveur nommé Tomcat qui se trouve compilé dans le jar.

### 2.2.3 A quoi faut-il faire attention ?

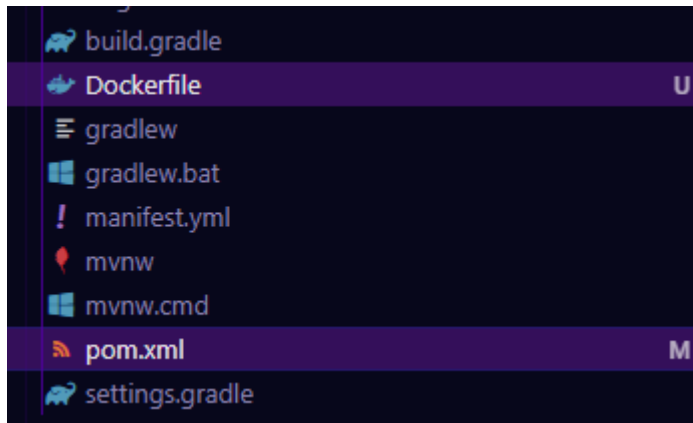
Il y a différents points à faire attention notamment :

La version du jdk installé qui doit être la version 17

A screenshot of a Dockerfile with a dark background and light-colored text. The text is as follows:

```
FROM openjdk:17-jdk-alpine
VOLUME /tmp
ARG JAVA_OPTS
ENV JAVA_OPTS=$JAVA_OPTS
COPY target/rest-service-complete-0.0.1-SNAPSHOT.jar complete.jar
EXPOSE 8080
ENTRYPOINT exec java $JAVA_OPTS -jar complete.jar
# For Spring-Boot project, use the entrypoint below to reduce Tomcat startup time.
#ENTRYPOINT exec java $JAVA_OPTS -Djava.security.egd=file:/dev/./urandom -jar gsrestservice.j
```

Mais aussi l'emplacement du fichier Docker qui doit se trouver au même niveau que le fichier pom.xml.



## 2.3 Création d'un projet Spring Boot

Quelles sont les annotations utilisées (commencent par @) dans votre contrôleur ?

- **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

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'existence de deux containers :

```
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 contrôleur pour les Repository ?

L'annotation @Autowired est utilisée dans les contrôleurs Spring pour injecter automatiquement une instance de Repository. Les Repository sont des composants utilisés dans Spring Data JPA pour interagir avec la base de données. L'utilisation de @Autowired permet de demander à Spring d'injecter automatiquement une instance du Repository requis dans le contrôleur, évitant ainsi la nécessité de créer manuellement l'instance du Repository.

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

Quant à l'annotation @ManyToOne, elle est utilisée pour spécifier une relation many-to-one (plusieurs vers un) entre deux entités dans JPA (Java Persistence API). Elle indique qu'un seul objet de l'entité annotée peut être associé à plusieurs instances d'une autre entité. Cette annotation est généralement utilisée du côté de l'entité qui possède la clé étrangère vers une autre entité.

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

## 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 contrôleurs 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 :

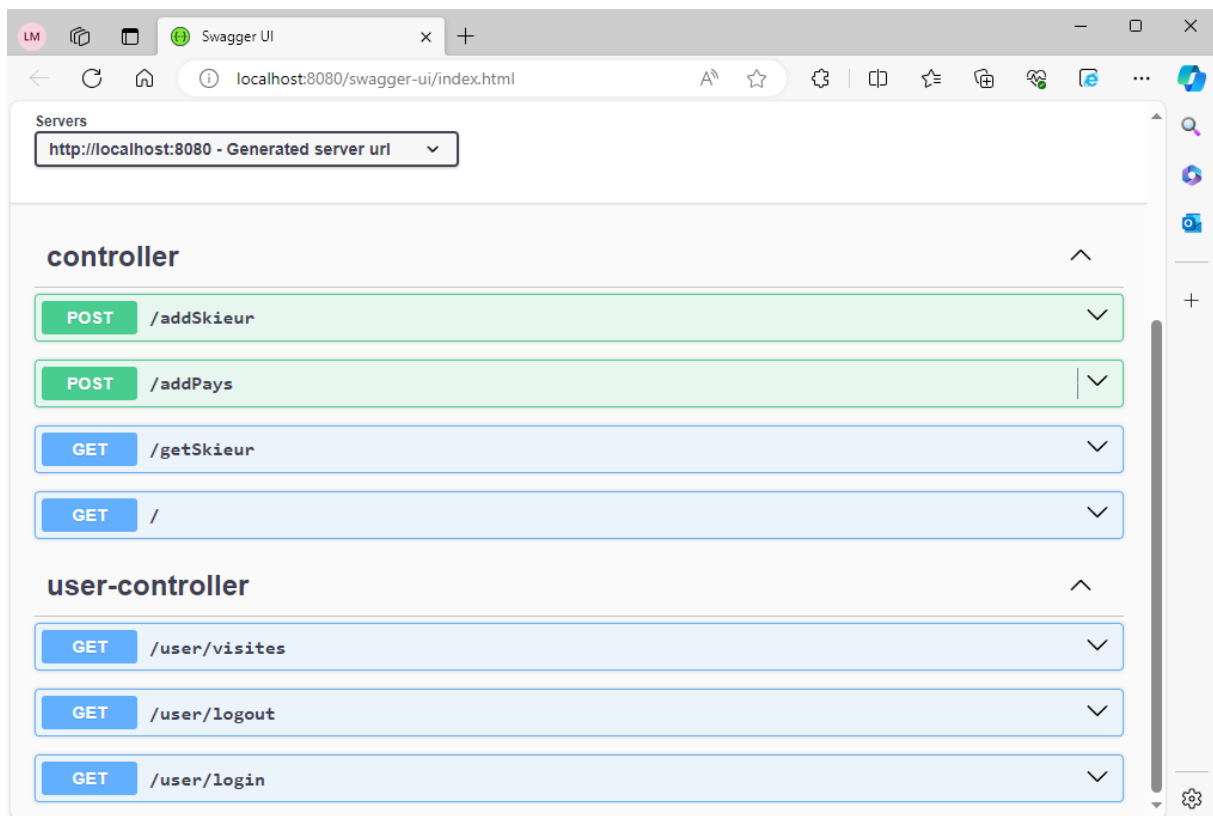
```
HttpSession session
```

## 2.8 Documentation API avec Swagger

Pour afficher la documentation de notre API, il nous suffit d'ajouter le code ci-dessous dans le fichier pom.xml.

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

Grâce à cela on peut accéder à la page de la doc :



## 2.9 Hébergement

## 3 Bases de données

### 3.1 Modèles WorkBench MySQL

## 4 Implémentation des applications <Le client Ap2>

### Magasin de Nerf

[Accueil](#) [Produits](#) [login](#)

#### Connexion

Nom d'utilisateur :

Mot de passe :



Se connecter

© 2024 Magasin de Nerf. Tous droits réservés.

### 4.1 Une descente de code client

```
//LoginCtrl.js
$("#btnConnect").on('click', this.login);
//Ensuite
static login(){
  username = $("#username").val();
  pwd = $("#password").val();
  http.login(pk, LoginCtrl.loginSuccess, LoginCtrl.error);
}
//HttpService.js
login(login, mdp, successCallback, errorCallback) {
  console.log("connect");
  $.ajax({
    type: "POST",
    url: this.BASE_URL + "login",
```

```
data: {
  username: login,
  pwd: mdp
},
success: successCallback,
error: errorCallback
})
}
//LoginCtrl.js
static loginSuccess(data){
  prompt("Vous êtes désormais connecté!\nProfitez à présent de toutes les fonctionnalités de manager");
}
//or
static error(data){
  console.log("problème " + data);
  alert("erreur: " + data);
}
```

## 5 Implémentation de l'application <API Gateway>

### 5.1 Une descente de code APIGateway

```
@RestController
public class Controller {

    private final Rest1Service rest1;
    private final Rest2Service rest2;

    @Autowired
    public Controller() {
        rest1 = new Rest1Service();
        rest2 = new Rest2Service();
    };

    @GetMapping("/")
    public @ResponseBody String base() {
        return "test";
    }

    @GetMapping("/getAllNerf")
    public @ResponseBody String getAllNerf() {
        return rest1.getAllNerf();
    }

    @GetMapping("/getNerf")
    public @ResponseBody ResponseEntity<String> getNerf(@RequestParam int id) {
        return rest1.getNerf(id);
    }

    @PutMapping("/newStock")
    public @ResponseBody ResponseEntity<String> fullStock(HttpSession session, @RequestParam int id,
        @RequestParam int addedQty) {
        ResponseEntity<String> result = null;
        if (session.getAttribute("login") != null) {
            if (session.getAttribute("isAdm").equals(1)) {
                result = rest1.fullNerfStock(id, addedQty);
            } else {
                HttpStatus httpCode = HttpStatus.UNAUTHORIZED;
                result = new ResponseEntity<>("Only administrators are authorized to change the store.", httpCode);
            }
        } else {
            HttpStatus httpCode = HttpStatus.UNAUTHORIZED;
            result = new ResponseEntity<>("You are not logged in. Try again when logged!", httpCode);
        }
        return result;
    }
}
```

```
}

@PostMapping("/addNerf")
public @ResponseBody ResponseEntity<String> addNerf(HttpSession session, @RequestParam String
nom,
    @RequestParam String description, @RequestParam String typeTir, @RequestParam double
prix,
    @RequestParam int quantite, @RequestParam byte[] img) {
    ResponseEntity<String> result = null;
    if (session.getAttribute("login") != null) {
        if (session.getAttribute("isAdm").equals(1)) {
            result = rest1.addNewNerf(nom, description, typeTir, prix, quantite, img);
        } else {
            HttpStatus httpCode = HttpStatus.UNAUTHORIZED;
            result = new ResponseEntity<>("Only administrators are authorized to change the
store.", httpCode);
        }
    } else {
        HttpStatus httpCode = HttpStatus.UNAUTHORIZED;
        result = new ResponseEntity<>("You are not logged in. Try again when logged!",
httpCode);
    }
    return result;
}

@PutMapping("/newPrice")
public @ResponseBody ResponseEntity<String> changerPrix(HttpSession session, @RequestParam int
id,
    @RequestParam double prix) {
    ResponseEntity<String> result = null;
    if (session.getAttribute("login") != null) {
        if (session.getAttribute("isAdm").equals(1)) {
            result = rest1.changerPrixNerf(id, prix);
        } else {
            HttpStatus httpCode = HttpStatus.UNAUTHORIZED;
            result = new ResponseEntity<>("Only administrators are authorized to change the
store.", httpCode);
        }
    } else {
        HttpStatus httpCode = HttpStatus.UNAUTHORIZED;
        result = new ResponseEntity<>("You are not logged in. Try again when logged!",
httpCode);
    }
    return result;
}

@PostMapping("/achatNerf")
public @ResponseBody ResponseEntity<String> achatNerf(HttpSession session, @RequestParam Date
date,
    @RequestParam int idNerf, @RequestParam int idUser, @RequestParam double prix) {
    ResponseEntity<String> result = null;
    ResponseEntity<String> temp = null;
    if (session.getAttribute("login") != null) {
        double prixNeg = -1 * prix;
        temp = rest2.changerSolde(idUser, prixNeg);
        if (temp.getBody().equals("true")) {
            temp = rest1.sellOne(idNerf);
            if (!(temp.getStatusCode() != HttpStatus.valueOf(200))) {
                Date today = new Date();
                temp = rest2.addNewCommande(today, idNerf, idUser, prixNeg);
            }
        }
    } else {
        HttpStatus httpCode = HttpStatus.UNAUTHORIZED;
        temp = new ResponseEntity<>("You are not logged in. Try again when logged!", httpCode);
    }
    if (temp.getStatusCode() != HttpStatus.valueOf(200)) {
        result = temp;
    } else {
        result = new ResponseEntity<>(HttpStatus.valueOf(200));
    }
    return result;
}
```

```

@GetMapping("/getCommandes")
public @ResponseBody ResponseEntity<String> getAllCommandes(HttpSession session) {
    ResponseEntity<String> result = null;
    if (session.getAttribute("login") != null) {
        int idUser = Integer.parseInt(session.getAttribute("idUser").toString());
        result = rest2.findAllCommandes(idUser);
    }
    return result;
}

@PostMapping("/login")
public ResponseEntity<String> login( @RequestBody Login credentials, HttpSession session) {
    // MicroserviceUtil service = new MicroserviceUtil(null);

    ResponseEntity<String> result = null;
    if (session.getAttribute("login") == null) {
        result = rest2.login(credentials.getUsername(), credentials.getPassword());
        if (result.getStatusCode().equals(HttpStatus.ACCEPTED)) {
            session.setAttribute("login", "true");
            int bodyToInt = Integer.parseInt(result.getBody());
            ResponseEntity<String> user = rest2.getUser(bodyToInt);
            ObjectMapper map = new ObjectMapper();
            try {
                JsonNode json = map.readTree(user.getBody());
                int isAdmin = json.get("admin").asInt();
                int idUser = json.get("id").asInt();
                session.setAttribute("idUser", idUser);
                session.setAttribute("isAdmin", isAdmin);
            } catch (JsonProcessingException e) {
                e.printStackTrace();
            }
        }
    } else {
        result = new ResponseEntity<>("You already are logged in. Please logout.",
        HttpStatus.CONFLICT);
    }
    return result;
}

@PostMapping("/logout")
public @ResponseBody ResponseEntity<String> logout(HttpSession session) {
    ResponseEntity<String> result = null;
    if(session.getAttribute("login") != null){
        session.invalidate();
        result = new ResponseEntity<>("You logged out succesfully.", HttpStatus.ACCEPTED);
    } else {
        result = new ResponseEntity<>("You need to be logged in in order to logout.",
        HttpStatus.BAD_REQUEST);
    }
    return result;
}
}

```

## 6 Implémentation des applications <API élève2>

### 6.1 Une descente de code de l'API REST

```

@RestController
public class Controller {

    private final UserService userService;
    private final CommandeService commandeService;

    @Autowired
    public Controller(UserService userService, CommandeService commandeService) {
        this.userService = userService;
        this.commandeService = commandeService;
    }
}

```

```
}

@GetMapping("/")
public String getNothing() {
    return "Page par défaut";
}

@PostMapping(path = "/acheterNerf")
public ResponseEntity<String> addNewCommande(@RequestParam Date date, @RequestParam int fk_nerf,
    @RequestParam int fk_user, @RequestParam double montant) {
    if (userService.setSoldeUser(fk_user, montant)) {
        return ResponseEntity.ok(new Gson().toJson(commandeService.addNewCommande(date, fk_nerf,
fk_user)));
    } else {
        return ResponseEntity.badRequest().body(" erreur lors de l'achat");
    }
}

@GetMapping(path = "/getCommandes")
public ResponseEntity<String> getAllCommandesByUser(@RequestParam int pk_user) {
    return ResponseEntity.ok(new Gson().toJson(commandeService.findAllCommandes(pk_user)));
}

@GetMapping(path = "/getUser")
public ResponseEntity<String> getUser(@RequestParam int pk_user) {
    return ResponseEntity.ok(new Gson().toJson(userService.getUser(pk_user)));
}

@PostMapping(path = "/changeSolde")
public ResponseEntity<String> changerSolde(@RequestParam int pk_user, @RequestParam Double
montant) {
    if (userService.setSoldeUser(pk_user, pk_user)) {
        return ResponseEntity.ok(new Gson().toJson(true));
    } else {
        return ResponseEntity.badRequest().body("Le montant est trop élevé");
    }
}

@PostMapping(path = "/login")
public ResponseEntity<String> login(@RequestBody Login credentials) {
    int verif = userService.login(credentials.getUsername(), credentials.getPassword());
    if (verif > 0) {
        return ResponseEntity.ok(new Gson().toJson(verif));
    } else {
        return ResponseEntity.badRequest().body(null);
    }
}
}
```

## 7 Hébergement

### 7.1 Info serveur

Ip: 91.92.200.252

user: g4admin

mdp: emf123

### 7.2 Commandes

Pousser image docker sur dockerhub :

```
docker tag apigw:latest mrpepinou/133projet:gw2
```

```
docker push mrpepinou/133projet:gw2
```

```
docker tag rest2:latest mrpepinou/133projet-rest2:latest2
```

```
docker push mrpepinou/133projet-rest2:latest2
```

```
docker tag rest1:latest mrpepinou/133projet-rest1:latest3
```

```
docker push mrpepinou/133projet-rest1:latest4
```

Faire un run des containers sur le serveur :

```
docker run --network reseauDocker --name rest-1 -p 8081:8080/tcp -d  
mrpepinou/133projet-rest1:latest4
```

```
docker run --network reseauDocker --name rest-2 -p 8082:8080/tcp -d  
mrpepinou/133projet-rest2:latest2
```

```
docker run --network reseauDocker --name api_gw -p 8080:8080/tcp -d  
mrpepinou/133projet:gw2
```

### 7.3 Site perso

La partie client de mon application se trouve sur mon site personnel de mon [Projet](#).

## 8 Installation du projet complet avec les 5 applications

## 9 Tests de fonctionnement du projet

Lors de la réalisation des tests, plusieurs problèmes sont apparus ce qui nous a fortement freiner lors de la fin du projet.

## 10 Auto-évaluations et conclusions

Je trouve que j'ai bien avancé mon projet malgré les nombreux problèmes que nous avons rencontrer dans le groupe. A cause de cela, nous n'avons pas pu finir à temps notre projet.