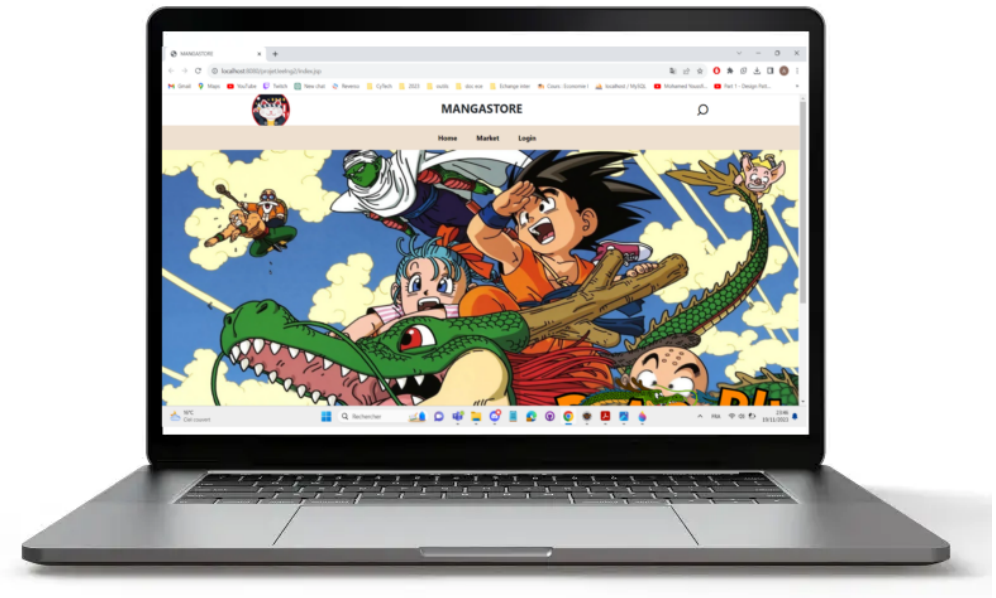
HADDACHE Noaman
CAI Christophe
LEMBA Mohamed
OUAALI Abd-Ennour

ING2 GSI groupe 1
2023-2024

Rapport

# Projet JEE Spring Boot



# Sommaire

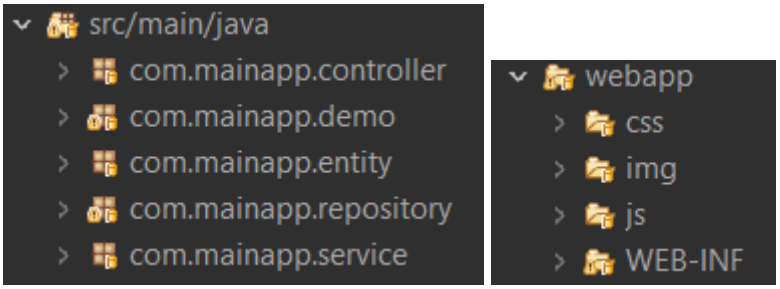
# Introduction

L'objectif de ce second rapport est de détailler le fonctionnement de Spring Boot; qui a été utilisé pour réaliser la seconde partie du projet de e-commerce.

La seconde partie du projet utilise à peu près les mêmes outils que la première, on y retrouve donc un projet Maven qui a une architecture MVC. Ce projet Spring Boot réutilise l'ensemble des entités Hibernate/JPA pour les bases de données(JDBC+MySQL), ainsi que les JSP pour le frontend du site.

Pour ce qui est des nouveautés de cette seconde partie, nous avons l'utilisation des controller de Spring, puis l'utilisation des repository et service propre à Spring, qui viennent respectivement remplacer les servlets et les DAO du site de la première partie.

**Annexe 1: architecture générale du projet**




# I- Définition de Spring Boot

Spring Boot est un framework qui se base sur l'utilisation d'un autre framework nommé Spring. L'objectif principal de Spring Boot est de faciliter l'utilisation des fonctionnalités de Spring, permettant de simplifier la création et la réalisation de projets autour de ce framework.

Spring Boot propose donc des fonctionnalités comme SpringInializer, qui permet facilement de créer un projet autour de Spring en facilitant l'implémentation des dépendances nécessaires selon les besoins du développeur. C'est notamment cet outil là que nous avons utilisé pour générer le projet de cette seconde partie.
Un autre exemple d'outil proposé par Spring Boot et que nous avons beaucoup utilisé aussi est le CLI proposé par Spring Boot, que nous avons utilisé afin d'exécuter et de lancer facilement tous les services nécessaires au fonctionnement du site.

De manière plus générale, l'utilisation de Spring Boot va aussi se traduire par l'utilisation de nombreuses annotations, qui va permettre à Spring Boot de reconnaître les classes et les fichiers qu'il doit intégrer dans son architecture MVC.

Ces annotations permettent aussi de définir des fonctionnalités propres à certaines composantes de l'architecture MVC pour Spring Boot, que nous verrons plus en détail dans la suite du rapport.

**Annexe 2: exemple d'annotations Spring Boot**

```
@SpringBootApplication(scanBasePackages = "com.mainapp")
@EnableJpaRepositories(basePackages = "com.mainapp")
@EntityScan(basePackages = "com.mainapp.entity")
```

# II- Partie Model/BDD de MVC avec Spring Boot

Pour ce qui concerne la partie Model de l'application Spring Boot (BDD), les nouvelles fonctionnalités propre à Spring Boot que nous avons utilisées sont les Repository et les Service (en plus des entités JPA que nous avions déjà).

Les repository sont des interfaces qui correspondent à l'ensemble des intéractions définies avec une table de la BDD. Ces interfaces héritent de la classe de base JpaRepository, qui permet de lier une entité qui va correspondre au type de données lues et écrites dans la base de données à un ensemble de requêtes. Dans l'organisation de notre projet, chaque table du modèle possède son propre Repository.

JpaRepository offre déjà un certain nombre de requêtes de base comme l'écriture ou la suppression d'un n-uplet, mais les requêtes plus précises sur la BDD ont dû être implémentées avec l'annotation '@Query', qui nous a permis d'exécuter nativement des requêtes SQL.

En dehors de '@Query', il y a aussi d'autres annotations que nous avons utilisées pour des cas plus spécifiques

**Annexe 2: exemple d'un Repository**

```
@Repository
public interface AdministratorRepository extends JpaRepository<Administrator, Integer>{

    @Query(value = "SELECT * FROM Administrator a JOIN User u ON a.id = u.id WHERE u.id = :id", nativeQuery = true)
    Administrator getAdministrator(@Param("id") int id);
```

La seconde fonctionnalité de Spring Boot que nous avons utilisée pour implémenter le modèle a été celle des services.

Les services dans Spring Boot sont chargés de gérer l'ensemble de la logique métier de l'application. Dans le cadre du projet, nous utilisons surtout des intéractions avec la base de données, donc l'ensemble des services permettent d'encapsuler les requêtes des Repository pour pouvoir en tirer une utilisation conforme selon les normes de l'application. Cela permet par exemple de rajouter du code java (cryptage de mot de passe et autres) sur les résultats des requêtes SQL, ou bien de définir clairement le return attendu pour une requête SQL.

De ce fait, nous avons un service par Repository, et donc un Service par table de la base de données du modèle. Les services sont des classes Java classiques, ce sont donc ces objets qui sont directement appelés dans les controllers lorsqu'une requête SQL doit être effectuée.

**Annexe 3: exemple d'un Service**

```java
@Service
public class AdministratorService {

    private AdministratorRepository ar;

    @Autowired
    public void setDependencies(AdministratorRepository ar) {
        this.ar = ar;
    }

    public Administrator getAdministrator(String email) {
        return ar.getAdministrator(email);
    }

    public Administrator getAdministrator(int id) {
        return ar.getAdministrator(id);
    }
```

# III- Partie Controller/Vue de MVC avec Spring Boot

En ce qui concerne la gestion de la partie Controller et Vue, nous avons utilisé les controllers de Spring Boot. Ces controller nous permettent de gérer les requêtes HTTP du site, et de rediriger vers les bonnes pages en exécutant les fonctionnalités correspondant aux actions de l'utilisateur. Dans notre projet, chaque controller utilise un ensemble de services différents, qui va correspondre aux tables de la base de données appelées par le controller. Pour ce qui est de la construction générale de ces controller, nous avons repris la structure des servlets de la partie une. Les controllers possèdent une méthode pour les requêtes GET, et une méthode pour les requêtes POST.

La plupart des éléments cités sont déterminés une nouvelle fois grâce aux annotations de Spring Boot, qui sont cette fois assez diverses dans le cadre de controller pour déterminer par exemple les objets passés en sessions (@SessionAttributes), ou encore le nom de la requete complet vers le controller(@RequestMapping(/adresse)).

C'est aussi à travers ces controller là que la redirection vers les autres adresses HTTP du site seront effectuées. Nous avons paramétré le site de sorte à ce que seuls les noms des fichiers soient donnés pour naviguer à travers les différents JSP du site (cela correspond au String return par les méthodes des controller).

**Annexe 4: exemple d'un Controller**

```java
@Controller
@RequestMapping("/Login")
@SessionAttributes({"email", "password", "user", "showAlert"})
public class LoginController {

    private UserService userService;

    public LoginController(UserService us) {
        this.userService = us;
    }

    @GetMapping
    public String doGet( Model model) {
        return "login";
    }

    @PostMapping
    public String doPost(@RequestParam("email") String email,
```

En ce qui concerne les JSP, nous avons dû réadapter le code Java qui gérait l'ensemble des variables de sessions selon les nouveaux controller, car les sessions étaient gérées différemment avec les Servlets de la première partie du projet.

**Annexe 5: exemple de code d'un JSP**

```jsp
<%
User loginUser = (User) session.getAttribute("user");
boolean isLogged = loginUser != null && loginUser.getId() != 0;
%>
<head>
<title>MANGASTORE</title>
<meta charset="UTF-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

# Conclusion

Ce rapport vient clore l'explication de notre travail sur Spring Boot. Pour résumer le travail que nous avons fait à surtout été un travail d'adaptation de la première partie du projet, ce qui nous a permis de découvrir le fonctionnement de Spring Boot, ainsi que les atouts qu'il permet d'offrir en tant que framework pour le web.