Formation Spring

Horaires:
Lundi - Jeudi: 9h - 17h30
Vendredi: 9h - 12h

Lien GIT:

https://github.com/MouradDawan75/FormationSpringMvc

Lien Teams:

https://teams.microsoft.com/l/meetup-join/19%3ameeting_OTRlNzVkZGEtYjBlNS00ZTg4LTljNzEtNWZkMmU0NzI1OTYw%40thread.v2/0?context=%7b%22Tid%22%3a%2243df35b4-026a-4006-b482-40a00d253e13%22%2c%22Oid%22%3a%229af4790c-c21f-4f79-99f2-277189517954%22%7d

Mourad MAHRANE: Dév/Formateur java/.net Python Android
émail: mmahrane@dawan.fr

*************Participants:**
David:
Arthur:
Victor:
Eroudini:
Sylvain:
Kilian:

********** **Spring** **********
Java SE: application console, application graphique
Java EE: applications web client/serveur (MVC, Api Rest)
Java ME: application mobile (remplacée par Android)

Galaxy Spring: est un ensemble de framework (modules)
- Spring Core (Spring Framework)
  **** Concept IOC: Inversion Of Control:
    - Instanciation des classes
    - Gestion des dépendances
    Spring conserve ses objets dans une zone mémoire qui porte e non de l'application (context)
- Spring web: créer des applications web (MVC - Api Rest)
- Spring Data: gérer la persistence des données
- Spring Security: sécuriser les apllication web
- Spring Boot: permet de gérer la configuration de l'application

Gestion du déploiement de l'application et génération du livrable:
Spring Boot embaque un serveur Tomcat
Plus besoin de générer un .war (il suffit de générer un .jar qui sera déployé par Spring Boot)
Facilité la gestion des dépendances: il propose des starters (un ensemble de dépendance) préconfigurée

********** **REST API** **********
c'est une application sans IHM qui renvoie principalement du contenu JSON.
API SOAP:
c'est un protocole d'échange de messages codés en XML (extension du protocole HTTP)

API REST:
est un style d'architecture qui utilise le protocole HTTP (GET, PUT,DELETE,POST)
REST: Representational State Transfert
une api REST n'est pas limitée au format JSON: elle peut renvoyée du xml, texte, fichiers.....

Il s'agit d'un ensemble de ressources (images, produits, article d'un journal....), où chaque possède un id (URI: Uniform Resource Identifier - End Point), une méthode d'accès (GET,POST,PUT,DELETE) et elle peut être publique ou privée

Toutes ces informations sont fournies dans la documentation de l 'api

********** **Gestion des logs Java** **********
permet l'écriture des traces dans un support de persistence (fichier, BD....)
Le JDK fournit une api permettant d'écrire des logs: java.util.logging
On peut aussi utiliser des librairies externes: Log4j, Logback, commons-logging

Spring utilise slf4j qui une sorte de façade pour pouvoir écrire sur plusieurs systèmes de logs.

Spring Boot: utilise slf4j + logback (succésseur de log4j) car Log4j est vulnérable.
Vocabulaire:
- Logger: objet permettant d'écrire des logs
- Appender: support des logs (console, fichier, BD....)
- Pattenr: format du message: %date%message
- Level: niveau de gravité du message (debug,info,warning,error)
Si level=warning: tous les de niveau inférieur seront ignorés

Mise en place:
- Slf4j et Logback sont chagé par le starter de Spring Boot -> aucune dépendance à ajouter dans le pom.xml
- Configuration des loggers à mettre soit dans application.properties ou extraire la conf. dans un fichier.xml à part.

********** **Documentation d'une API**
****
Open API Specification (Swagger): est la spécification d'une API.
Elle décrit l'api rest au travers d'un fichier yml/json qu'on peut visualiser dans Swagger Editor

Mise en place: ajout de la dépendance Maven suivante:
Lien doc: https://www.baeldung.com/spring-rest-openapi-documentation

```
<dependency>
  <groupId>org.springdoc</groupId>
  <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
  <version>2.8.4</version></dependency>
```

2 end points:
Swagger Editor: server:port/swagger-ui.html
Doc Json: server:port/v3/api-docs

Possibilité de modifier ces 2 ends points dans application.properties

On peut compléter la doc générée par cette dépendance, soit dans Swagger Editor, soit via une classe de conf.
Voir la classe SwaggerConfig

********** **Profiles Spring** **********
On peut définir plusieurs configurations pour une application Spring:
- 1 conf. pour la prod.
- 1 conf. pour le dév.
- 1 conf. pour les tests

Il suffit de créer un fichier application.properties pour chaque conf:
application-dev.properties
application-test.properties
application-prod.properties

Dans le fichier principal application.properties choisir la conf à mettre en place via la propriété:
spring.profiles.active=dev (ou prod ou test)

********** **Gestion des upload et download**
****
voir UploadDownloadController

Pour le upload: la méthode du controller consomme du multipart/form-data
Pour le download: la méthode du controller produit en sortie du contenu de type application/octet-stream.
Elle renvoie une Resource fournie dans le header de la réponse HTTP en injectant la param suivant:
.header(HttpHeaders.CONTENT_DISPOSITION,"attachment;filename="+nom_fichier)
.header(HttpHeaders.CONTENT_TYPE,Files.probeContentType(path_de_la_ressource))

********** **Spring Data JPA** **********
Spring Data est l'implémentation de Spring de JPA/Hibernate

Il propose un ensemble d'annotations pour mappey les Entity, les relations et l'héritage.
Des JpaRepository contenant des méthodes implémentées, qu'on peut compléter avec des méthodes personnalisées via les commandes JP-QL ou Sql natives, ou bien en utilisant les finders méthodes de Spring Data.

2 types de commandes SQL
Commandes LMD: Langage de manipulation de données: INSERT,DELETE,UPDATE,SELECT
Commandes LDD: Langage de définition des données (concerne la structure de la BD): CREATE,ALTER,DROP

Gestion des accès conccurrentiels à une ressource partagée:
@Version:
mécanisme qui permet aux dév. de gérer les modifications apportées à leur entity.
Avantages:
- Contrôle de versions: permet d'éviter la perte de maj
- Suivi des modifications: peut êttre utile à des fins d'audit
- Capacité de faire des restaurations: en cas d'erreurs, on peut revenir à une version précedente de l'entity.

---------- **Annotations pour mapper les Entity:**
@Entity: pour mapper l'entity à la table
@Id @GeneratedValue
@Version: versionner les modifs
@Column
@Enumerated: pour les enums
@ElementCOllection: pour les collections
@Transient: pour ignorer des champs

Voir classe Product.

---------- **Annotations pour mapper les relations:**

@ManyToOne - @OneToMany: voir Product et Category
@ManyToMany: voir Product et Supplier

*** @OneToOne:
1 Player ******* est encadré par ******** 1 Manager

3 représentations possibles:
1) une seule table:
Player(id,name, managerName) @Embedable Manager(id,name)
@Embedded
private Manager manager;

2) Association par clé primaire:
2 tables séparées avec @OneToOne que d'un côté. Les 2 objets possèdent le mm id.

Voir: Player2 et Manager2

3) Association par clé étrangère:
@OneToOne dans les 2 classes avec 1 seule mappedBy pour générer la clé de jointure
Voir Player3 et Manager3

---------- **Annotations pour mapper l'héritage:**

@Inheritance(strategy= choisir une stratégie)

- Single Table:
l'héritage est représenté par une seule table (Entity parente). Pour chaque Entity enfant une colonne par attribut est générée dans la table. Ele contient aussi la colonne pour determiner le type effectif de l'entity enfant (@DescriminatorColumn)

- Joined:
l'héritage est représenté par une jointure entre la table de l'entity parente et les entity enfant
Classes: (Vehicule - Voiture - Moto)

- Table Per class:
l'héritage est représenté par une par Entity (sauf pour l'entity parente), les attributs hérités seront répétés dans chaque table de classe fille.
La stratégie la plus complèxe à gérer à cause de l'Id qui ne peut pas en autoincrement (@GeneratedValue)
Classes(Employe - Ingenieur - Technicien)

- Cas particulier de l'héritage: fusion de la classe mère @MappedSuperClass (Ex: BaseEntity)
La classe mère n'est pas une Entity, contient somplement les attributs communs aux classes filles.

---------- **JpaRepository:**

Pour convertir Entity en DTO:
Option1:
Definir ses propres méthodes de conversion

Option2:
Utiliser des bibliothèques de conversion: ModelMapper - MapStruct
doc ModelMapper: https://modelmapper.org/getting-started/

********** Exo:
- CatergoryRepository - CategoryServiceImpl - CategoryController + Test dans postman
- SupplierRepository - SupplierServiceImpl - SupplierController + Test dans postman

********** **Appel d'API externes**

****

Pour faire appel à des API externes à partir d'un projet Spring, il faut utiliser la classe RestTamplate permettant d'exécuter des requêtes HTTP.
Voir: LocationDto - LocationServiceImpl - LocationController

Lien GIT du wrokspace:
https://github.com/MouradDawan75/SpringWorkspace

********** **Envoi de mails**********

- Soit utiliser l'api standard de Java: JavaMail: api de bas niveau
- Soit utiliser des bibliothèques externes: Commons Mail, Spring Mail

Mise en place:

1- Ajoutez lé dependance spring-boot-starter-mail dans pom.xml

2- Configuration du server SMTP dans application.properties

3- Définir les classes: Service + Controller