1. Tp sur spring-boot (2jours)

Récupérer une copie de https://github.com/didier-tp/spring_2025.git (via ***git clone*** ou via **code/download-zip** + extraction du zip dans c:\tp ou ailleurs)

1.1. timing approximatif du cours SpringBoot (2 jours)

Journée 1
- pres grandes lignes (SpringFramework vs SpringBoot , ...) : pages 1 à 13 et 17-18
- hyper rapide pres/aperçu "config_explicite" (sans spring_boot) : page 41
- vérification ou installation "jdk , ide(eclipse ou intellij, …)"
- chapitre "spring-boot" : pages 55 à 65 et rapidement 66-70(profils)
- éventuelle proposition de QCMs (https://www.d-defrance.fr/qcm-app , training , springBoot)
- début TP "sbapp" avec Spring-Initializer (JPA, Web , h2 et mySql_ou_postgres , openApidoc , ...)
- config/code minimaliste (hello world, page62) et run (:8080 ou:8181)
- config log + re-run
- éventuelle demo mystarter
- pres "typologie de projets" : page 98
- acces_données (p71) , config data_source (p79) et Spring-Data (p82-90,...) , rappel_profils (pages 66,70)
- suite Tp : mise en place des profils "dev_h2" et "dev_sgbd" ou autres + config_ds (p79)
- @Entity (p77) + DAO (JpaRepository p85) et test d'intégration partielle (base h2 ou ...) p89 et ...

J1 ou J2:
- notion de @Service , @Transactional et DTO , p92,91,97 (page 7)
- mise en pratique simple/rapide en TP (pour respecter bonnes pratiques couches logicielles)

Journée 2 :
- pres essentiel springMvc_REST pages 108, 109,...
- mise en pratique debut Api_rest en TP et run (:8080)
- tests d'api rest (page 139,140,...) et mise en pratique (suite Tp)
- pres essentiel "springSecurity" p153-155,165,178 + démo (sur projet préparé "appliSpringWeb" ou autre)
- pres "spring_boot" et "war" page 104
- Tp "mvn package" + "démarrage via .bat" d'un .jar auto_executable .
- exemple "Dockerfile" (dans tp/appliSpringWeb) p69

1.2. TP "mise en place du projet"

Créer un noveau projet "my**sbapp**" (pour **s**pring**b**oot**a**pp)

via l'assistant en ligne spring-initializr (https://start.spring.io)

group : tp , artifcatId : mysbapp , maven , jar , java21

dependencies : web , jpa , lombok , devTools , h2 , mysql et/ou postgres

pas "security" car n'importe quoi sans config

Générer/télécharger "sbapp.zip"

Extraire le contenu de l'archive et charger cela dans un IDE (eclipse ou itelliJ ou VSCode+pluginJava) .

S'inspirer de la page 62 (structure minimaliste)

Configurer un numéro de port (ex : **8080** ou **8181**) dans **application.properties**

Ajouter la page **index.html** dans **static** avec un message de type welcome

Lancer l'application et tester via une url de ce genre :

http://localhost:8181/mysbapp

En cas de problème , vérifier la configuration du jdk (dans l'IDE) via par exemple le menu "windows/preferences , java/install JRE " d'eclipse ou bien le menu "files/project structure" d'intelliJ .

NB : pour que lombok fonctionne bien dans l'IDE eclipse il faut installer le plugin lombok via le menu Help/install news sofware avec le chemin https://projectlombok.org/p2 .

Dans **MysbappApplication.main()** , ajouter la génération d'une ligne de log de niveau **debug**() et paramétrer les niveaux de logs dans **application.properties** de manière cohérente pour que cette ligne de log s'affiche dans la console au démarrage de l'application

Exemple :

```
...
import org.slf4j.LoggerFactory;
import org.slf4j.Logger;
@SpringBootApplication
public class MysbappApplication {
	private static Logger logger = LoggerFactory.getLogger(MysbappApplication.class);
	public static void main(String[] args) {
		SpringApplication.run(MysbappApplication.class, args);
		//logger.info("http://localhost:8181/mysbapp");
		logger.debug("http://localhost:8181/mysbapp");
	}
}
```

1.3. TP "accès au données" et test

`Mise en place des profils :`

dev_h2 : base de données "mysbdb" h2 en mode embedded sans serveur
dev_sgbd : base de données "mysbdb" gérée par un serveur (ex: MySql/MariaDB , Postgres ou Oracle)
ddl_auto : pour recréation automatique des tables (en mode dev)
reinit : pour ajouter quelques donnees en mode dev/main (mais pas avec tests)
prod : production (à définir ultérieurement)

Exemple de configuration à adapter :

```
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.url=jdbc:h2:~/mysbdb
#spring.datasource.url=jdbc:h2:mem:mysbdb
spring.datasource.username=sa
spring.datasource.password=
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
```

```
#maria_db = version simplifiee de mySql (compatible)
spring.datasource.driverClassName=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/mysbdb?
createDatabaseIfNotExist=true&serverTimezone=UTC
spring.datasource.username=root
spring.datasource.password=root
spring.jpa.database-platform=org.hibernate.dialect.MySQL8Dialect
```

```
spring.jpa.hibernate.ddl-auto=create
#spring.jpa.hibernate.ddl-auto=none
```

Exemple partiel d'entité persistante :

tp.mysbapp.entity.**ProductEntity**

```
...
@Entity
@Table(name="product")
@NoArgsConstructor @AllArgsConstructor
@Getter @Setter @ToString
public class ProductEntity {

        @Id
        @GeneratedValue(strategy = GenerationType.IDENTITY)
        private Long id;

        @Column(length = 64)
        private String label;

        private Double price;

   ...
}
```

Coder l'interface **tp.mysbapp.dao.ProductDao** héritant de **JpaRepository**<ProductEntity,Long>

Exemple de données pour le test :

src/test/resources/import_products.sql

```
DELETE FROM product;
INSERT INTO product (label,price) VALUES ('styloA',2.2);
INSERT INTO product (label,price) VALUES ('cahierXy',3.8);
INSERT INTO product (label,price) VALUES ('trousseZz',4.8);
INSERT INTO product (label,price) VALUES ('regle 20cm',2.6);
```

Exemple de classe de test :

src/test/java/*tp.mysbapp.dao.TestProductDaoDevH2*

```
package tp.sbapp.dao;
import static org.junit.jupiter.api.Assertions.assertTrue;
import java.util.List;
import org.junit.jupiter.api.Test;
import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.ActiveProfiles;
import org.springframework.test.context.jdbc.Sql;
import tp.mysbapp.entity.ProductEntity;

@SpringBootTest
@ActiveProfiles({"dev_h2","ddl_auto"})
@Slf4j
public class TestProductDaoDevH2 {

	@Autowired
	private ProductDao productDao ; //à tester

	@Test
	@Sql({"/import_products.sql"})//import_products.sql dans src/test/resources
	public void test1(){
		List<ProductEntity> products = productDao.findAll();
		assertTrue(products.size()>=4);
		log.debug("products="+products);
	}
}
```

1.4. TP "business services , transactions et test"

NB: au sein de cette formation, les aspects "Transaction" et "Service métier" ne sont pas très approfondis et on pourrait ne rien faire en Tp sur ce point là mais le minimum de code suivant permet de bien matérialiser cette couche logicielle souvent utile/indispensable sur de véritables projets d'entreprise.

`tp.mysbapp.data.Product`

```
...
//classe de données "métier" (potentiel DTO)

@Getter @Setter @ToString @NoArgsConstructor @AllArgsConstructor
public class Product {
   private Long id;
   private String label;
   private Double price;

}
```

`tp.mysbapp.util.GenericMapper`

```
package tp.mysbapp.util;

import org.mapstruct.Mapper;  import org.mapstruct.factory.Mappers;
import tp.mysbapp.data.Product;  import tp.mysbapp.entity.ProductEntity;
import java.util.List;

@Mapper(componentModel = "spring") //for @Autowired or ...
public interface MyMapStructMapper {

   MyMapStructMapper INSTANCE = Mappers.getMapper( MyMapStructMapper.class );

   //@Mapping(target="price", source="prix_or_price") //ex si noms différents
   ProductEntity productToProductEntity(Product product);

   List<ProductEntity> productListToProductEntityList(List<Product> products);
   Product productEntityToProduct(ProductEntity product);
   List<Product> productEntityListToProductList(List<ProductEntity> products);
}
```

`tp.mysbapp.service.ProductService`

```
package tp.sbapp.service;  import java.util.List;  import java.util.Optional;
import tp.mysbapp.data.Product;

public interface ProductService {
        List<Product> findAll();
        //List<Product> findByPrixMini(double prixMini);
        Optional<Product> findById(Long id);
        Product saveNew(Product p);
        void updateExisting(Product p);
        void deleteById(Long id);
        //...
}
```

`tp.mysbapp.service.ProductServiceImpl`

```
package tp.mysbapp.service;
import java.util.List;    import java.util.Optional;
import org.springframework.beans.factory.annotation.Autowired;
```

```
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import tp.mysbapp.dao.ProductDao;  import tp.mysbapp.data.Product;
import tp.mysbapp.entity.ProductEntity;   import tp.mysbapp.util.GenericMapper;
import lombok.RequiredArgsConstructor;

@Service
@Transactional
@RequiredArgsConstructor //pour injection de dépendances via constructeur (final fields)
public class ProductServiceImpl implements ProductService{
	private final MyMapStructMapper mapper;
	private final ProductDao productDao;

	public List<Product> findAll() {
		List<ProductEntity> listProdEntity = productDao.findAll();
		return mapper.productEntityListToProductList(listProdEntity);
	}

	public Optional<Product> findById(Long id) {
		Optional<ProductEntity> optProdEntity = productDao.findById(id);
		if(optProdEntity.isEmpty())
		   return Optional.empty();
		else
		   return Optional.of(mapper.productEntityToProduct(optProdEntity.get()));
	}

	public Product saveNew(Product p) {
		ProductEntity pE = mapper.productToProductEntity(p);
		ProductEntity savedPE =productDao.save(pE);
		return mapper.productEntityToProduct(savedPE);
	}

	public void updateExisting(Product p) {
		ProductEntity pE = mapper.productToProductEntity(p);
		productDao.save(pE);
	}

	public void deleteById(Long id) {productDao.deleteById(id);	}

}
```

`tp.mysbapp.service.TestProductService (dans src/test/java)`

```
package tp.mysbapp.service;

import static org.junit.jupiter.api.Assertions.assertTrue;
import java.util.List;  import org.junit.jupiter.api.Test;
import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.ActiveProfiles;
import org.springframework.test.context.jdbc.Sql;   import tp.mysbapp.data.Product;
```

```
@SpringBootTest
@ActiveProfiles({"dev_h2","ddl_auto"})
@Slf4j
public class TestProductService {

	@Autowired
	private ProductService productService ; //à tester

	@Test
	@Sql({"/import_products.sql"})//import_products.sql dans src/test/resources
	public void test1(){
		List<Product> products = productService.findAll();
		assertTrue(products.size()>=4);
		logger.debug("products="+products);
	}

}
```

NB : tout le code précédent (assez rapide à mettre en œuvre en TP) est grandement améliorable en :

- Peaufinant les remontées d'exception

1.5. TP "essentiel Api rest" et test

Programmer l'api rest t**p.mysbapp.rest.ProductRestCtrl** avec **@RestController** avec au minimum , les points d'entrée en mode GET

et tester cela avec des simples URL relatives au sein d'un navigateur :

index.html

```
…
<body>
  <h2>mysbapp (welcome)</h2>
  <h3>quelques URLs pour tester des WS REST en mode GET (sans securite)</h3>
  <a href="./rest/api-product/products/1">produit 1 au format JSON</a> <br/>
  <a href="./rest/api-product/products">tous les produits au format JSON</a> <br/>
  <hr/>
  <a href="./doc-swagger.html">documentation swagger3/openapi</a><br/>
  <hr/>
  <a href="./h2-console" target="_blank">h2-console</a><br/>
</body>
...
```

NB: en mode "ddl_auto" + "reinit" , pour ne pas avoir des tables toujours réinitialisées et vides on peut par exemple ajouter une initialisation automatique d'un jeu de données :

tp.mysbapp.init.**InitDataSet**

```
package tp.mysbapp.init;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Profile;
import org.springframework.stereotype.Component;
import jakarta.annotation.PostConstruct;  import tp.mysbapp.data.Product;
import tp.mysbapp.service.ProductService;

@Component
@Profile("reinit")
public class InitDataSet {
	@Autowired
	ProductService productService;

	@PostConstruct
	public void intialiserJeuxDeDonnees() {
		System.out.println("initialisation d'un jeux de données (en mode developpement)");
		productService.saveNew(new Product(null,"styloA",2.2));
		productService.saveNew(new Product(null,"styloB",2.3));
	}
}
```

Pour vérifier d'éventuelles parties de l'api REST en mode POST,PUT,DELETE on pourra ajouter openApiDoc et s'en servir en mode **"try it out"** après avoir ajouter ceci dans le fichier **pom.xml** :

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

et ceci dans **application.properties**

```
springdoc.swagger-ui.path=/doc-swagger.html
```

et après avoir déclencher un rebuild maven et un redémarrage de l'application.

Ecrire un début de test unitaire de l'api REST en s'appuyant sur un mock du service interne :

tp.mysbapp.rest.**TestProductRestCtrlWithServiceMock** (dans src/test/java)

avec **@WebMvcTest** et **@MockitoBean**

et

import static org.hamcrest.Matchers.hasSize;

import static org.hamcrest.Matchers.is;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;

import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.jsonPath;

import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;

1.6. TP "packaging , lancement"

Après un build maven sans erreur , on pourra écrire et lancer un script de ce genre (par exemple dans un nouveau sous répertoire "dist"du projet) pour démarrer l'application springBoot au dehors de l'IDE :

dist/lancerSpringBootApp.bat

```
set JAVA_HOME=C:\Prog\java\open-jdk\jdk-21
set PATH=%JAVA_HOME%\bin;%PATH%
echo %PATH%
REM mysbapp.jar est ici une copie de target/mysbapp-0.0.1-SNAPSHOT.jar
java  -Dspring.profiles.active=dev_h2,ddl_auto  -jar mysbapp.jar
pause
```

NB : on pourra éventuellement accompagner ce .bat avec en relatif par rapport au répertoire courant **config/application.properties** de manière à externaliser les propriétés (en prod ou pre prod).

On pourra également utiliser l'option **-Dspring.profiles.active=p1,p2** si besoin .

Attention : ***dist/****lancerSpringBootApp.bat*

dist/*mysbapp.jar*

dist/config/application.properties

ne doivent pas être mélangés au reste du code source de l'application, sinon conflits et comportements incompréhensibles en dev (depuis l'ide) !!!