I - Annexe – énoncés des Tps / Spring

1. Tp sur bases de spring-framework

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. (Tp facultatif) Très rapide aperçu sur ancienne config XML

L'ancienne configuration de Spring au format XML est aujourd'hui considérée comme obsolète. Elle est néanmoins toujours supportée par spring6 .
Ce Tp facultatif n'est à priori utile que si besoin de comprendre la structure d'une ancienne application Spring.

L'application exemple "***oldXmlSpringApp***" (au format "maven") du référentiel git
https://github.com/didier-tp/spring_2025.git
est un **exemple simple** de **configuration Spring XML** .

Il est possible de charger ce projet dans un IDE tel que eclipse ou intelliJ pour ensuite lancer l'application ou bien les tests unitaires .
La partie configuration XML se situe dans le sous répertoire **src/main/resources** .

1.2. Chargement du projet et analyses/vérifications

L'objectif de cette première série de Tps est d'appréhender les fonctionnalités essentielles de Spring via une approche très progressive en partant volontairement d'un début de projet uniquement basé sur **spring-framework** (sans spring-boot) .

Charger dans eclipse ou intelliJ le projet suivant (au format maven) :
tp/*debutAppliSpringSansSpringBoot* (*de https://github.com/didier-tp/spring_2025.git*)

NB: ce projet nécessite **java 17** comme version minimum du java .
Avec **intelliJ** , vérifier si besoin les réglages java/jdk via le menu **"file / project structure"**

- Analyser la structure de **pom.xml** (avec packaging="war")
- Repérer les principales dépendances (spring-context , …)
- Lancer l'exécution de **tp.appliSpring.exemple.ExempleApp**
- Analyser le code initial de cet exemple simple
- Beaucoup de choses seront approfondies ultérieurement

1.3. Bases de l'injection de dépendance (avec @Autowired)

- dans package ***tp.appliSpring.exemple*** , Répérer la classe ***Coordinateur*** avec le début de code suivant (à compléter)

```
package tp.appliSpring.exemple;
//...
@Component
public class Coordinateur {
//…
private MonAfficheur monAfficheur=null; //référence vers afficheur à injecter

//...
private MonCalculateur monCalculateur=null://référence vers calculateur à injecter

public void calculerEtAfficher() {
                double x=4;
                double  res =monCalculateur.calculer(x); //x*x ou bien 2*x ou bien ...
                monAfficheur.afficher("res="+res);// >> res=16 en v1 ou bien ** res=16
        }
}
```

- Au sein de la méthode main() de la classe ExempleApp , ajouter un bloc de code de ce genre :

```
Coordinateur coordinateurPrisEnChargeParSpring =
                        contextSpring.getBean(Coordinateur.class);
coordinateurPrisEnChargeParSpring.calculerEtAfficher();
```

- **Compléter le code de la classe Coordinateur (et ajuster si besoin d'autres classes) de manière à ce que cet exemple fonctionne bien.**
- On pourra coder et tester successivement plein de variantes d'injection de dépendances :
  - via **@Autowired** (ou bien @Resource ou bien @Inject)
    sans ou avec affichage des éléments injectés au sein du constructeur par défaut de la classe Coordinateur et d'une méthode initialiser() préfixée par **@PostConstruct**
  - via des ajouts de ***MonAfficheurV2*** (avec préfixe "**" plutôt que ">>")
    et ***MonCalculateurDouble*** (2*x plutôt que x*x) de manière à engendrer une ambiguité.
  - Via des ajouts de **@Qualifier** pour lever les ambiguités
  - Via une expérimentation de l'**injection par constructeur** (par exemple dans une classe "CoordinateurAvecInjectionParConstructeur")

1.4. Configurations via classes java (@Configuration, @Bean)

- Analyser la structure de la partie **tp.appliSpring.explicit**
  .*app*.SpringAppWithExplicitConf avec main()
  .beans/ avec interfaces et classes
  sans aucune annotation spring
  .conf.ExempleConfigExplicite à compléter en Tp

A faire en Tp:

phase 0: faire fonctionner l'exemple tel quel

phase 1: tenir compte des choix de prefixe et suffixe de exemples.properties via @Value("${...:..}")

avec dans src/main/resources/exemples.properties

```
preferences.prefixe=>>>
preferences.suffixe=<<<
```

tester le comportement en modifiant les valeurs des prefixe et suffixe dans le .properties

phase 2: variante ...Basic en l'absence de profile "maj"
et variante ...Maj si présence du profile "maj"
tester le comportement en activant ou pas de profile "maj" en début de main()

NB: cette partie "***explicit***" est plus complexe que la partie "***exemple***" et n'a pas beaucoup d'intérêt tel quel .
Par contre, au sein d'un projet plus complexe, la configuration explicite basée sur @Bean peut s'avérer très utile pour paramétrer des composants "spring" basés sur des classes (récupérées via maven depuis une librairie externe) dont on n'a pas le droit de changer le code source .

1.5. Mise en place d'un aspect de type "log automatique"

Mettre en place un **aspect** (via Spring AOP , paramétré via annotations de AspectJ) qui affichera des lignes de logs pour chaque appel d'une méthode d'une classe du package ***tp.appliSpring.exemple .***
On pourra par exemple préciser le temps d'exécution et les noms des méthodes invoquées.

Code initial à compléter :

```
tp.appliSpring.annotation.LogExecutionTime
tp.appliSpring.aspect.MyPerfLogAspect
```

Phase 1 du Tp :

compléter (via ajout d'annotations et de code) les classes suivantes pour que l'aspect soit déclenché sur l'exécution des méthodes des classes du package tp.appliSpring.exemple lorsque le profile "perf" est activé :

```
tp.appliSpring.aspect.MyPerfLogAspect
tp.appliSpring.exemple.ExempleConfig
tp.appliSpring.exemple.ExempleApp.main()
```

Phase 2 du Tp :

Conditionner le comportement précédent au fait que l'annotation **@LogExecutionTime** doit être placée au dessus des méthodes sur lesquelles on souhaite obtenir un temps d'exécution .

2. Tp sur transaction spring et accès aux données (jdbc,jpa) sans spring-data

NB1 : cette série de Tps peut soit être effectuée pas à pas si l'on souhaite bien comprendre la structure des anciennes applications Spring (basées uniquement sur Spring-framework) .
Si par contre, on préfère se contenter d'un très rapide aperçu pour la compréhension , on pourra directement analyser et faire fonctionner la solution (projet "appliSpringSansSpringBoot" sans "debut….") .

NB2 : le contenu initial du projet "***debutAppliSpringSansSpringBoot***" du référentiel maven https://github.com/didier-tp/spring_2025.git comporte une bonne partie des éléments des Tps ci-après → énoncés simplifiés (vérifier présence et contenu d'un fichier à améliorer plutôt que copier/coller) . En règle générale , le contenu initial des fichiers est incomplet (il faut ajouter des annotations et d'autres morceaux de code) .

2.1. Accès aux données (DataSource JDBC) , DAO

- Créer package ***tp.appliSpring.core.entity***
- Créer classe ***Compte.java***

```
package tp.appliSpring.core.entity;

public class Compte {

   private Long numero;
   private String label;
   private Double solde;

  //+get/set , constructeurs , toString()
}
```

- Créer package ***tp.appliSpring.core.dao***
- Créer interface ***DaoCompte.java***

```
package tp.appliSpring.core.dao;

import java.util.List;
import tp.appliSpring.core.entity.Compte;

public interface DaoCompte{
        Compte findById(Long numCpt);
        Compte save(Compte compte); //sauvegarde au sens saveOrUpdate
        List<Compte> findAll();
        void deleteById(Long numCpt);
    //...
}
```

- dans ***src/main/resources*** ajouter **application.properties** avec ce contenu :

```
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.url=jdbc:h2:~/mydbbank
spring.datasource.username=sa
spring.datasource.password=
```

- dans ***tp.appliSpring.core*** ajouter **MySpringApplication** avec ce contenu :

```
package tp.appliSpring.core;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

//version sans springBoot
@Configuration
@ComponentScan(basePackages = { "tp.appliSpring.core"})
//NB : Tous les sous packages de tp.appliSpring.core seront scrutés pour y découvrir
//@Component… et aussi pour y découvir d'autres classes avec @Configuration
public class MySpringApplication {

	public tatic void main(String[] args) {
		//System.setProperty("spring.profiles.active", "p1");

		AnnotationConfigApplicationContext springContext = new
				AnnotationConfigApplicationContext(MySpringApplication.class) ;
		//...
		springContext.close();
	}
}
```

- Créer package ***tp.appliSpring.core.config***
- Créer la classe ***CommonConfig.java***

```
package tp.appliSpring.core.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.PropertySource;
import org.springframework.context.support.PropertySourcesPlaceholderConfigurer;

@Configuration
@PropertySource("classpath:/application.properties")
public class CommonConfig {

	@Bean
	public static PropertySourcesPlaceholderConfigurer
	propertySourcesPlaceholderConfigurer(){
		return new PropertySourcesPlaceholderConfigurer();
		//pour pouvoir interpréter ${} in @Value()
	}
}
```

- Créer la classe ***DataSourceConfig.java***

```
package tp.appliSpring.core.config;

import javax.sql.DataSource;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate;
import org.springframework.jdbc.datasource.DriverManagerDataSource;

@Configuration
public class DataSourceConfig {

	@Value("${spring.datasource.driverClassName}")
	private String jdbcDriver;

	@Value("${spring.datasource.url}")
	private String dbUrl;

	@Value("${spring.datasource.username}")
	private String dbUsername;

	@Value("${spring.datasource.password}")
	private String dbPassword;

   @Bean(name="dataSource")
   public DataSource dataSource() {
		DriverManagerDataSource dataSource = new DriverManagerDataSource();
		dataSource.setDriverClassName(jdbcDriver);
		dataSource.setUrl(dbUrl);
		dataSource.setUsername(dbUsername);
		dataSource.setPassword(dbPassword);
		return dataSource;
   }

  //seulement utile pour le dao en version Jdbc (avec NamedParameterJdbcTemplate):
  @Bean()
   public NamedParameterJdbcTemplate namedParameterJdbcTemplate( DataSource dataSource) {
      return new NamedParameterJdbcTemplate(dataSource);
   }

}
```

- dans ***src/test/java*** et dans un package ***tp.appliSpring.dao*** à créer, ajouter cette classe de test :

```
package tp.appliSpring.dao;

import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
```

```
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit.jupiter.SpringExtension;
import tp.appliSpring.core.MySpringApplication;
import tp.appliSpring.core.dao.DaoCompte;
import tp.appliSpring.core.entity.Compte;

@ExtendWith(SpringExtension.class) //si junit5/jupiter
@ContextConfiguration(classes= {MySpringApplication.class})
public class TestCompteDao {

   private static Logger logger = LoggerFactory.getLogger(TestCompteDao.class);

        @Autowired
        @Qualifier("jdbc")
        //@Qualifier("jpa")
        private DaoCompte daoCompte; //à tester

 @Test
 public void testAjoutEtRelectureEtSuppression() {
        //hypothese : base avec tables vides et existantes au lancement du test
        Compte compte = new Compte(null,"compteA",100.0);
        Compte compteSauvegarde = this.daoCompte.save(compte); //INSERT INTO
        logger.debug("compteSauvegarde=" + compteSauvegarde);

        Compte compteRelu = this.daoCompte.findById(compteSauvegarde.getNumero());
        Assertions.assertEquals("compteA",compteRelu.getLabel());
        Assertions.assertEquals(100.0,compteRelu.getSolde());
        logger.debug("compteRelu apres insertion=" + compteRelu);

        compte.setSolde(150.0); compte.setLabel("compte_a");
        Compte compteMisAjour = this.daoCompte.save(compte); //UPDATE
        logger.debug("compteMisAjour=" + compteMisAjour);

        compteRelu = this.daoCompte.findById(compteSauvegarde.getNumero()); //SELECT
        Assertions.assertEquals("compte_a",compteRelu.getLabel());
        Assertions.assertEquals(150.0,compteRelu.getSolde());
        logger.debug("compteRelu apres miseAjour=" + compteRelu);

        //+supprimer :
        this.daoCompte.deleteById(compteSauvegarde.getNumero());

        //verifier bien supprimé (en tentant une relecture qui renvoi null)
        Compte compteReluApresSuppression =
                this.daoCompte.findById(compteSauvegarde.getNumero());
        Assertions.assertTrue(compteReluApresSuppression == null);
        }
}
```

Script de préparation de la base de données (ici en version H2) :

init_db.sql

```
DROP TABLE IF EXISTS Compte;

CREATE TABLE Compte(
        numero integer auto_increment NOT NULL,
        label VARCHAR(64),
        solde double,
        PRIMARY KEY(numero));

INSERT INTO Compte (label,solde) VALUES ('compte courant',100);
INSERT INTO Compte (label,solde) VALUES ('compte codevi',50);
INSERT INTO Compte (label,solde) VALUES ('compte 3',150);

SELECT * FROM Compte;
```

set_env.bat

```
set MVN_REPOSITORY=C:\Users\administrateur\.m2\repository
set MY_H2_DB_URL=jdbc:h2:~/mydbbank
set H2_VERSION=2.2.224
set H2_CLASSPATH=%MVN_REPOSITORY%\com\h2database\h2\%H2_VERSION%\h2-%H2_VERSION%.jar
```

create_h2_database.bat

```
cd /d %~dp0
call set_env.bat
java -classpath %H2_CLASSPATH% org.h2.tools.RunScript -url %MY_H2_DB_URL%  -user sa -script init_db.sql -showResults
pause
```

lancer_console_h2.bat

```
cd /d %~dp0
call set_env.bat
java -jar  %H2_CLASSPATH% -user "sa" -url %MY_H2_DB_URL%

REM NB: penser à se déconnecter pour éviter des futurs verrous/blocages
pause
```

NB : Toute cette structure de code et configuration sera utilisée dès le(s) TP(s) suivant(s)

2.2. Petit exemple de DAO via JDBCTemplate

NB: Ce TP pas fondamental est facultatif : à faire ou pas selon le temps disponible

coder le début de ***DaoCompteJdbc.java*** avec le code suivant (*à compléter*)

```
package tp.appliSpring.core.dao;

import java.sql.ResultSet;     import java.sql.SQLException;
import java.util.HashMap;    import java.util.List;    import java.util.Map;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
```

```
import org.springframework.jdbc.core.RowMapper;
import org.springframework.jdbc.core.namedparam.MapSqlParameterSource;
import org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate;
import org.springframework.jdbc.core.namedparam.SqlParameterSource;
import org.springframework.jdbc.support.GeneratedKeyHolder;
import org.springframework.jdbc.support.KeyHolder;
import org.springframework.stereotype.Repository;
import tp.appliSpring.core.entity.Compte;

@Repository //@Component de type DAO/Repository
@Qualifier("jdbc")
public class DaoCompteJdbc /*extends JdbcDaoSupport*/ implements DaoCompte {

private final String INSERT_SQL = "INSERT INTO compte(label, solde)  values(:label,:solde)";
private final String UPDATE_SQL = "UPDATE compte set label=:label , solde=:solde where numero=:numero";
private final String FETCH_ALL_SQL = "select * from compte";
private final String FETCH_BY_NUM_SQL = "select * from compte where numero=:numero";
private final String DELETE_BY_NUM_SQL = "delete from compte where numero=:numero";

@Autowired
private NamedParameterJdbcTemplate namedParameterJdbcTemplate;

@Override
public Compte findById(Long numCpt) {
        Compte compte = null;
        Map<String, Long> parameters = new HashMap<String, Long>();
        parameters.put("numero", numCpt);
        List<Compte> comptes = namedParameterJdbcTemplate.query(FETCH_BY_NUM_SQL,
                                                        parameters,  new CompteMapper());
        compte = comptes.isEmpty()?null:comptes.get(0);
        return compte;
}

@Override
public Compte save(Compte compte) {
        if(compte==null)
                throw new IllegalArgumentException("compte must be not null");
        return (compte.getNumero()==null)?insert(compte):update(compte);
}

public Compte insert(Compte compte) {
        KeyHolder holder = new GeneratedKeyHolder(); //to retreive auto_increment value of pk
        SqlParameterSource parameters = new MapSqlParameterSource()
                                                .addValue("label", compte.getLabel())
                                                .addValue("solde", compte.getSolde());
        namedParameterJdbcTemplate.update(INSERT_SQL, parameters, holder);
        compte.setNumero(holder.getKey().longValue());//store auto_increment pk in instance to return
        return compte;
}

public Compte update(Compte compte) {
        //A CODER/COMPLETER  EN TP
}

@Override
public List<Compte> findAll() {
       //A CODER/COMPLETER  EN TP
}
```

```
@Override
public void deleteById(Long numCpt) {
        //A CODER/COMPLETER  EN TP
}

}

//classe auxiliaire "CompteMapper" pour convertir Resultset jdbc en instance de la classe Compte :
class CompteMapper implements RowMapper<Compte> {
        @Override
        public Compte mapRow(ResultSet rs, int rowNum) throws SQLException {
                Compte compte = new Compte();
                compte.setNumero(rs.getLong("numero"));
                compte.setLabel(rs.getString("label"));
                compte.setSolde(rs.getDouble("solde"));
                return compte;
        }
}
```

- Compléter le code manquant de cette classe
- tester via le lancement de ***TestCompteDao*** (dans src/test/java)

2.3. Accès aux données via JPA/Hibernate

Ajouter dans le package ***tp.appliSpring.core.config*** la classe de configuration ***DomainAndPersistenceConfig.java*** suivante :

```
package tp.appliSpring.core.config;

import java.util.Properties;     import javax.sql.DataSource;
import jakarta.persistence.EntityManagerFactory;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.orm.jpa.JpaTransactionManager;
import org.springframework.orm.jpa.JpaVendorAdapter;
import org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean;
import org.springframework.orm.jpa.vendor.Database;
import org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter;
import org.springframework.transaction.PlatformTransactionManager;
import org.springframework.transaction.annotation.EnableTransactionManagement;

@Configuration
@EnableTransactionManagement() // "transactionManager" (not "txManager") is expected !!!
@ComponentScan(basePackages = { "tp.appliSpring.core.dao" ,
                                "tp.appliSpring.core.service" ,  "tp.appliSpring.core.init"})
public class DomainAndPersistenceConfig {

// JpaVendorAdapter (Hibernate ou OpenJPA ou ...)
@Bean
public JpaVendorAdapter jpaVendorAdapter() {
        HibernateJpaVendorAdapter hibernateJpaVendorAdapter =
                        new HibernateJpaVendorAdapter();
        hibernateJpaVendorAdapter.setShowSql(false);
        hibernateJpaVendorAdapter.setGenerateDdl(false);
```

```
        //hibernateJpaVendorAdapter.setDatabase(Database.MYSQL);
        hibernateJpaVendorAdapter.setDatabase(Database.H2);
        return hibernateJpaVendorAdapter;
}

// EntityManagerFactory
@Bean(name = { "entityManagerFactory" })
public EntityManagerFactory entityManagerFactory(JpaVendorAdapter jpaVendorAdapter,
                                                 DataSource dataSource) {
        LocalContainerEntityManagerFactoryBean factory =
                        new LocalContainerEntityManagerFactoryBean();
        factory.setJpaVendorAdapter(jpaVendorAdapter);
        factory.setPackagesToScan("tp.appliSpring.core.entity");
        factory.setDataSource(dataSource);

        Properties jpaProperties = new Properties(); // java.util
        jpaProperties.setProperty("javax.persistence.schema-generation.database.action",
                                  "drop-and-create"); //JPA>=2.1
        factory.setJpaProperties(jpaProperties);
        factory.afterPropertiesSet();
        return factory.getObject();
}

// Transaction Manager for JPA or ...
@Bean(name = "transactionManager")
public PlatformTransactionManager transactionManager(
                        EntityManagerFactory entityManagerFactory) {
        JpaTransactionManager txManager = new JpaTransactionManager();
        txManager.setEntityManagerFactory(entityManagerFactory);
        return txManager;
}
}
```

Coder au sein du package ***tp.appliSpring.core.dao*** la classe ***DaoCompteJpa***
en partant du code suivant (à compléter) :

```
package tp.appliSpring.core.dao;

import java.util.List;
import jakarta.persistence.EntityManager;
import jakarta.persistence.PersistenceContext;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Repository;
import org.springframework.transaction.annotation.Transactional;
import tp.appliSpring.core.entity.Compte;

@Repository //@Component de type DAO/Repository
@Qualifier("jpa")
public class DaoCompteJpa implements DaoCompte {

        @PersistenceContext
        private EntityManager entityManager;

        @Override
```

```
	public Compte findById(Long numCpt) {
		// A CODER/COMPLETER EN TP
	}

	/*
	public Compte save(Compte compte) {
		try {
			entityManager.getTransaction().begin();
			if(compte.getNumero()==null)
				entityManager.persist(compte);//INSERT INTO
			else
				entityManager.merge(compte);//UPDATE
			entityManager.getTransaction().commit();
		} catch (Exception e) {
			entityManager.getTransaction().rollback();
			e.printStackTrace();
		}
		return compte; //avec numero plus null (auto_incrémenté)
	}
	*/

	@Override
	@Transactional
	public Compte save(Compte compte) {
			if(compte.getNumero()==null)
				entityManager.persist(compte);//INSERT INTO
			else
				entityManager.merge(compte);//UPDATE
		return compte; //avec numero plus null (auto_incrémenté)
	}

	@Override
	public List<Compte> findAll() {
		return entityManager.createQuery("SELECT c FROM Compte c",
                                                 Compte.class)
                                    .getResultList();
	}

	@Override
	@Transactional
	public void deleteById(Long numCpt) {
		// A CODER/COMPLETER EN TP
		Compte compte = ……..
		entityManager…….(compte);
	}
}
```

- Compléter le code de la classe ci-dessus
- Ajouter toutes les **annotations** manquantes et nécessaires dans la classe ***tp.appliSpring.core.entity.Compte*** (@Entity, @Id , ….., @GeneratedValue(strategy = GenerationType.IDENTITY)
- switcher de qualificatif ~~@Qualifier("jdbc")~~ **Qualifier("jpa")** au sein de la classe ***TestCompteDao***
- Lancer le test et corriger les éventuels problèmes/erreurs .

2.4. Service Spring et gestion des transactions

Créer le nouveau package **tp.appliSpring.core.service**
Ajouter y l'interface ***ServiceCompte*** suivante :

```
package tp.appliSpring.core.service;

import java.util.List;
import tp.appliSpring.core.entity.Compte;

public interface ServiceCompte {
    Compte rechercherComptePar Numero(long numero);
    List<Compte> rechercherTousComptes();
    List<Compte> rechercherComptesDuClient(long numClient);
    Compte sauvegarderCompte(Compte compte);
    void supprimerCompte(long numCpt);
    void transferer(double montant,long numCptDeb,long numCptCred);
}
```

Ajouter la classe d'implémentation ***ServiceCompteImpl*** suivante (*à compléter*) :

```
package tp.appliSpring.core.service;

import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import tp.appliSpring.core.dao.DaoCompte;
import tp.appliSpring.core.entity.Compte;

@Service //classe de Service prise en charge par spring
public class ServiceCompteImpl implements ServiceCompte{

@Qualifier("jpa")
@Autowired
private DaoCompte daoCompte=null;

public Compte rechercherComptePar Numero(long numero) {
        return daoCompte.findById(numero);
}

public Compte sauvegarderCompte(Compte compte) {
        return daoCompte.save(compte);
}

public List<Compte> rechercherTousComptes() {
        // A CODER/COMPLETER EN TP
}

public List<Compte> rechercherComptesDuClient(long numClient) {
        //return null; //version zero
```

```
        return this.rechercherTousComptes(); //V1 (provisoire)
        //future version V2 (via un nouvel appel sur DAO exploitant @ManyToOne ou bien ...)
}

public void supprimerCompte(long numCpt) {
        // A CODER/COMPLETER EN TP
}

@Transactional(/*propagation = Propagation.REQUIRED*/) //REQUIRED par defaut
public void transferer(double montant, long numCptDeb, long numCptCred) {
  try {
        // transaction globale initialisée dès le début de l'exécution de transferer
        Compte cptDeb = this.daoCompte.findById(numCptDeb);
              //le dao exécute son code dans la grande transaction
              //commencée par le service sans la fermer et l'objet cptDeb remonte à l'état persistant
        cptDeb.setSolde(cptDeb.getSolde() - montant);
        //this.daoCompte.save(cptDeb); //facultatif si @Transactional


        //idem pour compte à créditer
        Compte cptCred= this.daoCompte.findById(numCptCred);
        cptCred.setSolde(cptCred.getSolde() + montant);
        //this.daoCompte.save(cptCred) //facultatif si @Transactional

        //en fin de transaction réussie (sans exception) , toutes les modification effectuées
        //sur les objets à l'état persistant seront répercutées en base (.save() automatiques)
        } catch (Exception e) {
                throw new RuntimeException("echec virement " + e.getMessage() , e);
                 //rollback se fait de façon fiable
                 //ou bien throw new
                //ClasseExceptionPersonnaliseeHeritantDeRuntimeException("echec virement" , e);
        }
}
}
```

Au sein de ***src/test/java*** et du package ***tp.appliSpring.core.service*** (à créer) , ajouter la classe de test ***TestServiceCompte*** suivante :

```
package tp.appliSpring.service;

import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ActiveProfiles;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit.jupiter.SpringExtension;
import tp.appliSpring.core.MySpringApplication;
import tp.appliSpring.core.entity.Compte;
import tp.appliSpring.core.service.ServiceCompte;

@ExtendWith(SpringExtension.class)
```

```
@ContextConfiguration(classes= {MySpringApplication.class})
//@ActiveProfiles({ "embeddedDB" , "dev" , "perf" })
public class TestServiceCompte {

private static Logger logger = LoggerFactory.getLogger(TestServiceCompte.class);

@Autowired
private ServiceCompte serviceCompte; //à tester


@Test
public void testVirement() {
        Compte compteASauvegarde = this.serviceCompte.sauvegarderCompte(
                new Compte(null,"compteA",300.0));
        Compte compteBSauvegarde = this.serviceCompte.sauvegarderCompte(
                 new Compte(null,"compteB",100.0));
        long numCptA = compteASauvegarde.getNumero();
        long numCptB = compteBSauvegarde.getNumero();
        //remonter en memoire les anciens soldes des compte A et B avant virement
        //(+affichage console ou logger) :
        double soldeA_avant= compteASauvegarde.getSolde();
        double soldeB_avant = compteBSauvegarde.getSolde();
        logger.debug("avant bon virement, soldeA_avant="+soldeA_avant +
                        " et soldeB_avant=" + soldeB_avant);

        //effectuer un virement de 50 euros d'un compte A vers vers compte B
        this.serviceCompte.transferer(50.0, numCptA, numCptB);

        //remonter en memoire les nouveaux soldes des compte A et B apres virement
        // (+affichage console ou logger)
        Compte compteAReluApresVirement =
                        this.serviceCompte.rechercherCompteParNumero(numCptA);
        Compte compteBReluApresVirement =
                        this.serviceCompte.rechercherCompteParNumero(numCptB);
        double soldeA_apres = compteAReluApresVirement.getSolde();
        double soldeB_apres = compteBReluApresVirement.getSolde();
        logger.debug("apres bon virement, soldeA_apres="+soldeA_apres
                        + " et soldeB_apres=" + soldeB_apres);

        //verifier -50 et +50 sur les différences de soldes sur A et B :
        Assertions.assertEquals(soldeA_avant - 50, soldeA_apres,0.000001);
        Assertions.assertEquals(soldeB_avant + 50, soldeB_apres,0.000001);
}

//@Test
public void testMauvaisVirement() {
        /*  VARIANTE A CODER/COLPLETER EN TP
        COPIER/COLLER à ADPATER de  testVirement()
        AVEC
        try {
            this.serviceCompte.transferer(50.0, numCptA, -numCptB); //erreur volontaire
        } catch (Exception e) {
                logger.error("echec normal du virement " + e.getMessage());
```

```
        }
        et
        //verifier -0 et +0 sur les différences de soldes sur A et B
        Assertions.assertEquals(soldeA_avant , soldeA_apres,0.000001);
        Assertions.assertEquals(soldeB_avant , soldeB_apres,0.000001);
         */
}
}
```

Série de tests à effectuer :

1. enlever ~~@Transactional~~ au dessus de la méthode **transferer** et
   enlever les commentaires sur les lignes *this.daoCompte.save(cptDeb);*
   et this.daoCompte.save(cptCred);
2. *lancer le test* ***testVirement()*** et corriger les bugs si nécessaire
3. coder et lancer ***testMauvaisVirement***() . c'est normal si ça ne fonctionne pas bien sans l'ajout de @Transactional
4. replacer @Transactional au dessus de la méthode **transferer** et relancer le test ***testMauvaisVirement***() qui devrait normalement fonctionner .
5. replacer des commentaires sur les lignes *this.daoCompte.save(cptDeb);*
   et this.daoCompte.save(cptCred);
   Tous les tests devraient encore bien fonctionner .

3. Tp sur transactions , spring-data (avec spring-boot)

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)

3.1. (Tp facultatif) , nouveau projet via spring initializr

- Aller sur le site "spring initializr"
- Créer un nouveau projet ("demo" , "java/maven/java_17" , "jar") avec les dépendances fondamentales ("jpa" , "web" , …)
- extraire le contenu de demo.zip dans d:\tp ou ailleurs
- charger ce projet dans eclipse ou intelliJ
- ajouter un numéro de port dans application.properties
- ajouter une page index.html au sein de src/main/resources/static
- démarrer l'application (via le main()) et vérifier l'accès à la page d'accueil

NB: avec beaucoup de temps, on pourrait effectuer tous les Tps ultérieurs à partir de ce nouveau projet .

Pour gagner un peu de temps, les Tps ultérieurs seront effectués à partir d'un point de départ plus riche et bien structuré tp/debutAppliSpringWeb .

Néanmoins, pour les "débutants Spring" , enrichir petit à petit le point de départ "demo" peut être un exercice complémentaire intéressant si l'on souhaite bien comprendre toutes les configurations nécessaires au bon fonctionnement d'une application springBoot .

3.2. Chargement du projet springBoot et analyses/vérifications

L'objectif de cette seconde série de Tps est d'appréhender les fonctionnalités essentielles de Spring en mode "configuration moderne" (avec spring-boot et spring-data) .

Charger dans eclipse ou intelliJ le projet suivant (au format maven) :
tp/*debutAppliSpringWeb* (de *https://github.com/didier-tp/spring_2025.git*)

NB: ce projet nécessite **java 17** comme version minimum du java .
Avec **intelliJ** , vérifier si besoin les réglages java/jdk via le menu **"file / project structure"**

- Analyser la structure de **pom.xml** (avec packaging="jar")
- Repérer les principales dépendances (spring-boot-starter-…)
- Lancer l'exécution de **tp.appliSpring.AppliSpringApplication.main()**
- Visualiser la page d'accueil au sein d'un navigateur
  (`http://localhost:8181/appliSpring`)
  `partie fonctionnant dès le début : compte1 au format json`
- Beaucoup de choses seront approfondies ultérieurement

3.3. Analyse de application.properties et variantes (profiles)

NB :

- Le fichier **application.properties** comporte les configurations essentielles (n° port, …)
- Le fichier application**-dev**.properties (associé au profil "dev") comporte une configuration vers une base de données "h2" (embedded database sans serveur) dont l'url est `jdbc:h2:~/mydbbank`
- **Les tables de cette base de données seront re-créées (à vide) à chaque redémarrage** (du main() ou des test avec le profil "dev") du fait de la propriété `spring.jpa.hibernate.ddl-auto=create`
- D'autres profils (ex : "prod") comporte spring.jpa.hibernate.ddl-auto=**none**

3.4. Dao en mode JpaRepository

Analyser les portions de code suivantes :

tp.appliSpring.bank.persistence.entity.**CompteEntity**

tp.appliSpring.bank.persistence.repository.**CompteRepository**

Faire fonctionner le test suivant :

tp.appliSpring.core.dao.**TestCompteDao**

Phase1 du Tp :

Ajouter dans CompteRepository une méthode permettant de rechercher les comptes dont le solde est entre une valeur mini et une valeur maxi.

On proposera deux versions de cette méthode :

- une première respectant les convention de nommage des méthodes pour une requête automatique

- une seconde codée avec @Query() et JpaQL

On testera ceci en ajoutant une méthode de test au sein de TestCompteDao

Phase2 (facultative) du Tp :

- Ajouter dans OperationRepository une méthode permettant de rechercher les opérations rattachées à un numéro de compte précis et dont la date est entre une valeur mini et une valeur maxi (les paramètres dateMini , dateMaxi seront de type "Date")
- On testera ceci en ajoutant une méthode de test au sein de TestOperationDao (à coder à coté de TestCompteDao) avec par exemple SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd"); et `dateFormat.parse("2025-02-12")`

Phase3 (facultative) du Tp :

- coder une variante (ex : TestOperationDaoSql) de la classe TestOperationDao avec l'utilisation de @Sql pour injecter des données en base en début de méthode de test

Phase4 (facultative) du Tp :

- expérimentations libres (sans casser ce qui fonctionne dès le départ)

3.5. Transactions sur virement bancaire

Analyser globalement la structure de code suivante :

- tp.appliSpring.bank.core.**model**.**Compte**
- tp.appliSpring.bank.core.service.**ServiceCompte**
- tp.appliSpring.bank.core.service.direct.***ServiceCompteDirectImpl***

Code important de la classe tp.appliSpring.bank.core.service.direct.***ServiceCompteDirectImpl*** :

```
// ...
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
// ...

@Service //classe de Service prise en charge par spring
//@Transactional
@Qualifier("direct")
@Primary
public class ServiceCompteDirectImpl …. implements ServiceCompte{

// ...

@Transactional(/*propagation = Propagation.REQUIRED*/) //REQUIRED par defaut
public void transfer(double montant, long numCptDeb, long numCptCred)
    throws BankException{
  try {
        // transaction globale initialisée dès le début de l'exécution de transferer
        CompteEntity cptDeb = this.daoCompte.findById(numCptDeb).get();
              //le dao exécute son code dans la grande transaction
              //commencée par le service sans la fermer et l'objet cptDeb remonte à l'état persistant
        cptDeb.setSolde(cptDeb.getSolde() - montant);
        this.daoCompte.save(cptDeb); //facultatif si @Transactional


        //idem pour compte à créditer
        CompteEntity cptCred= this.daoCompte.findById(numCptCred).get();
        cptCred.setSolde(cptCred.getSolde() + montant);
        this.daoCompte.save(cptCred) ; //facultatif si @Transactional

        //en fin de transaction réussie (sans exception) , toutes les modification effectuées
        //sur les objets à l'état persistant seront répercutées en base (.save() automatiques)
        } catch (Exception e) {
                throw new BankException("echec virement",e);
                 //le rollback se fait de façon fiable car BankException hérite de RuntimeException
        }
  }
}
```

Au sein de ***src/test/java*** et du package ***tp.appliSpring.core.service*** , visualiser la classe de test ***TestServiceCompte*** avec la partie de code importante suivante :

```
// ...
@SpringBootTest(classes= {AppliSpringApplication.class})
@ActiveProfiles({  "dev" })
public class TestServiceCompte {

private static Logger logger = LoggerFactory.getLogger(TestServiceCompte.class);

@Autowired
private ServiceCompte serviceCompte; //à tester


@Test
public void testVirement() {
        Compte compteASauvegarde = this.serviceCompte.create(
                new Compte(null,"compteA",300.0));
        Compte compteBSauvegarde = this.serviceCompte.create(
                 new Compte(null,"compteB",100.0));
        long numCptA = compteASauvegarde.getNumero();
        long numCptB = compteBSauvegarde.getNumero();
        //remonter en memoire les anciens soldes des compte A et B avant virement
        //(+affichage console ou logger) :
        double soldeA_avant= compteASauvegarde.getSolde();
        double soldeB_avant = compteBSauvegarde.getSolde();
        logger.debug("avant bon virement, soldeA_avant="+soldeA_avant +
                        " et soldeB_avant=" + soldeB_avant);

        //effectuer un virement de 50 euros d'un compte A vers vers compte B
        this.serviceCompte.transfer(50.0, numCptA, numCptB);

        //remonter en memoire les nouveaux soldes des compte A et B apres virement
        // (+affichage console ou logger)
        Compte compteAReluApresVirement =
                        this.serviceCompte.searchById(numCptA);
        Compte compteBReluApresVirement =
                       this.serviceCompte.searchById(numCptB);
        double soldeA_apres = compteAReluApresVirement.getSolde();
        double soldeB_apres = compteBReluApresVirement.getSolde();
        logger.debug("apres bon virement, soldeA_apres="+soldeA_apres
                         + " et soldeB_apres=" + soldeB_apres);

        //verifier -50 et +50 sur les différences de soldes sur A et B :
        Assertions.assertEquals(soldeA_avant - 50, soldeA_apres,0.000001);
        Assertions.assertEquals(soldeB_avant + 50, soldeB_apres,0.000001);
}

//@Test
public void testMauvaisVirement() {
        /*   VARIANTE A CODER/COMPLETER EN TP
        COPIER/COLLER à ADPATER de  testVirement()
         AVEC
        try {
            this.serviceCompte.transfer(50.0, numCptA, -numCptB); //erreur volontaire
        } catch (Exception e) {
```

```
                logger.error("echec normal du virement " + e.getMessage());
        }
        et
        //verifier -0 et +0 sur les différences de soldes sur A et B
        Assertions.assertEquals(soldeA_avant , soldeA_apres,0.000001);
        Assertions.assertEquals(soldeB_avant , soldeB_apres,0.000001);
         */
}
}
```

Série de tests à effectuer :

1. enlever ~~@Transactional~~ au dessus de la méthode **transfer**

2. *lancer le test* ***testVirement()*** et corriger les bugs si nécessaire
3. coder et lancer ***testMauvaisVirement***() . c'est normal si ça ne fonctionne pas bien sans l'ajout de @Transactional
4. replacer @Transactional au dessus de la méthode **transfer** et relancer le test ***testMauvaisVirement***() qui devrait normalement fonctionner .
5. Autres expérimentations libres (sans casser le code de départ)

4. Tp sur Api-REST avec Spring-Mvc

Série de Tps basée sur le point de départ **tp/debutAppliSpringWeb**

4.1. Familiarisation avec la structure du projet

Analyser globalement la structure suivante :

- `tp.appliSpring.bank.core.model.`**`Compte`**
- `tp.appliSpring.bank.core.service.`**`ServiceCompte`**
- `tp.appliSpring.bank.core.init.`**`ReInitDefaultDataSet`**
  `avec profil "reInit"`
- `tp.appliSpring.generic.converter.`**`GenericMapper`**
- `tp.appliSpring.generic.dto.ApiError`
- `tp.appliSpring.generic.dto.MessageDto`
- `tp.appliSpring.generic.exception.EntityNotFoundException`
- `tp.appliSpring.bank.web.api.rest.`**`CompteRestCtrl`**

4.2. Partie "get" de l'api REST

- `Analyser le code initial de`
  `tp.appliSpring.bank.web.api.rest.`**`CompteRestCtrl`**
- **`Lancer l'application`** `AppliSpringApplication.`**`main`**`()`
  `avec les profils` **`"dev" et "reInit"`**
- `tester via la partie "compte1 au format JSON" de index.html`
  `(sur http http://localhost:8181/appliSpring )`
- `coder la partie //GET Multiple de` **`CompteRestCtrl`**
- tester via les liens hypertextes présents du haut de index.html

4.3. Gestion des statuts Http et des exceptions

- Déclencher l'url http://localhost:8181/appliSpring/rest/api-bank/v1/comptes/18888 sachant que le compte 18888 n'existe pas et visualiser un message d'erreur pas très précis (500)
- Coder et tester la version V2 de **CompteRestCtrl.**getCompteById en s'appuyant sur *ResponseEntity<?> de manière à renvoyer proprement le code 404/NOT_FOUND.*
- Revenir sur la V1 de **CompteRestCtrl.**getCompteById et tester le comportement en décommentant *@ControllerAdvice* au sein de tp.appliSpring.generic.rest.RestResponseEntityExceptionHandler
- Redémarrer et tester le nouveau comportement et le message d'erreur approprié
  NB : utiliser la console du navigateur pour visualiser le code de retour HTTP 404.

4.4. Partie "post,put,delete" de l'api REST

Coder et tester progressivement les parties "POST, DELETE , PUT" de CompteRestCtrl
et tester cela via "**Try it out**" de la partie "documentation swagger3/openapi " .

On pourra éventuellement effectuer les tests avec un outils de type "postman" ou autre .

Bien tester les cas d'erreur .

Lorsque le code java de la classe CompteRestCtrl aura été bien complété et bien testé , on pourra utiliser le mini front-end "**compteAjax**" (**html + js**) incorporé dans la partie **static** du projet de manière à invoquer l'api REST sur les comptes en mode CRUD .

4.5. Validation des entrées avec @Valid

On pourra retoucher/améliorer la méthode en mode "POST" de la classe CompteRestCtrl
en utilisant le type `tp.appliSpring.bank.web.api.dto`.CompteToCreate plutôt que Compte.

- La classe CompteToCreate comporte déjà des annotations de type @Min, @Length
- La classe tp.appliSpring.generic.rest.RestResponseEntityExceptionHandler comporte déjà la méthode `handleMethodArgumentNotValid`
- plus qu'à utiliser **@Valid** à coté de @RequestBody CompteToCreate
- bien tester le comportement avec des données valides ou pas

4.6. Test unitaire pour Api REST (tp facultatif)

Compléter la classe de test TestCompteRestCtrlWithServiceMockWithoutSecurity

permettant de tester une partie des méthodes de CompteRestCtrl .

Suggestions :

- @WebMvcTest(controllers = { CompteRestCtrl.class } , excludeAutoConfiguration = {SecurityAutoConfiguration.class}) ou autre
- @Autowired private MockMvc mvc;
- mvc.perform(get("/rest/api-bank/v1/comptes?numClient=1") ou autre

4.7. Utilisation de mapStruct (tp facultatif)

- Décommenter tous les blocs liés à mapStruct dans pom.xml
- Réactiver l'interface tp.appliSpring.bank.converter.MyBankMapper en supprimant l'extention .txt sur le fichier)
- Relancer un build maven (ex : mvn package) et visualiser le code généré dans la partie

`"target/generated-sources/annotations"`.

- Utiliser le convertisseur **MyBankMapper** plus performant que *GenericMapper* au sein des méthodes *searchWithMinimumBalance* et *searchCustomerAccounts* de la classe tp.appliSpring.bank.core.service.direct.*ServiceCompteDirectImpl* .
- Tester le bon fonctionnement après un redémarrage .

4.8. Exemple d'appel REST externe (tp facultatif)

Dans une nouvelle classe de type util.client.MyWsCall , utiliser l'ancienne api RestTemplate ou bien la nouvelle api RestClient de manière à invoquer un WS REST externe. On testera cela via une petite classe de test .

Exemple de WS-REST facile à appeler :

http://api.zippopotam.us (/fr/75001)

à appeler en mode GET

https://www.d-defrance.fr/tp/devise-api/v1/public /devises/EUR
à appeler en mode GET ou DELETE

https://www.d-defrance.fr/tp/devise-api/v1/public /devises
à appeler en mode GET ou POST

5. Tp sur DHTML via SpringMvc et Thymeleaf (ou JSP)

5.1. Analyse des exemples basiques et des templates thymeleaf

Exemples élémentaires :

- tp.appliSpring.bank.**site.controller**.`BasicController (helloWorld et calculTva)`
- src/main/resources/**templates** (displayBasicMessage.html , calculètva.html)

Point d'entrée pour tester l'exécution de ces exemples :
index.html → index-site.html → ...

Avec "layout" de Thymeleaf :

- src/main/resources/**templates/_footer.html , _header.html , _layout.html**
- src/main/resources/templates/**welcome.html , calcul_racine.html**

5.2. Tp @SessionAttribute sur calcul de racine carrée

Par défaut , chaque calcul de racine carrée est indépendant.

welcome → calculRacineCarree → welcome → calculRacineCarree (avec 0.0 par défaut).

Si l'on souhaite expérimenter une mémorisation en session du dernier calcul de racine carrée effectué on pourra ajouter **@SessionAttributes(…)** et ce qui va avec au sein de tp.appliSpring.bank.site.controller.**BasicController** .

Vérifier le nouveau comportement via une navigation de ce type :

welcome → calculRacineCarree → welcome → calculRacineCarree (autre que 0.0).

5.3. Analyse d'autres exemples et expérimentations

- Partie "inscription" de tp.appliSpring.bank.site.controller.**BasicController** et tp.appliSpring.bank.site.form.`InscriptionForm` et `src/main/resources/templates/`**`inscription.html`**

  pour exemple avec plein de syntaxes thymeleaf .

- ….

5.4. TP Virement bancaire via SpringMVC et Thymeleaf

- Navigation préalable à expérimenter :
  index → index-site → welcome → espace_client (numClient=1) → comptesDuClient
  puis comptesDuClient → virement
- Visualiser le code de tp.appliSpring.bank.site.form.**VirementForm**
- compléter `src/main/resources/templates/`**`virement.html`**
  `et tp.appliSpring.bank.site.controller.`**`BankController`**

  de manière à déclencher un virement bancaire entre des comptes du client connecté
  (avec numClient et client en session)

Suggestions :

- Au sein de virement.html on pourra saisir ou choisir les parties montant , numCptDeb et numCptCred en tant que sous parties de `th:object="${virement}"` via des syntaxes de type `th:field="*{montant}"` et envoyer le tout vers `th:action="@{/site/bank/doVirement}"` au sein d'un formulaire à déclencher en mode POST
- Au sein d'un nouveau point d'entrée de BankController (ex : "doVirement" on pourra récupérer les valeurs saisies via un paramètre d'entrée de type `@ModelAttribute("virement") VirementForm virement` puis déclencher un appel à `serviceCompte.transfer(…)`
  En cas de succès on pourra retourner `comptesDuClient(model); //pour réactualiser` et afficher nouvelle liste des comptes du client

6. Tp sur Spring-security

6.1. A savoir avec spring-boot-starter-security

En Ajoutant seulement ceci dans **pom.xml**

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

et redémarrant une application vierge sans aucune configuration.

La console affiche alors un message de ce genre :

```
Using generated security password: 656a4d96-2f13-46fd-b4b9-2e2c90b3fbb6

This generated password is for development use only. Your security configuration must be updated before running your application in production.
```

Ce mot de passe (régénéré à chaque démarrage) est pour le username "**user**"

Après avoir saisi "user" et le mot de passe attendu par copier/coller dans la boîte de dialogue , on peut accéder à la page d'accueil principale (index.html) et naviguer vers d'autres pages .

Autrement dit , la sécurité "spring" n'est réellement exploitable qu'avec un minimum de configuration et sans aucune configuration il vaut mieux de pas ajouter spring-boot-starter-security dans pom.xml .

6.2. withoutSecurity

Si en début de développement on souhaite (malgré la dépendance directe ou indirecte spring-boot-starter-security) utiliser la partie web de l'application en mode "sécurité désactivée" on peut utiliser une classe de configuration ressemblant à tp.appliSpring.**WithoutSecurityConfig** avec entre autres `auth.requestMatchers("/**").permitAll() .`

C'est dans ce mode de fonctionnement que les Tps des chapitres précédents ont été effectués.
Avec profils "dev,reInit" sélectionné au démarrage de l'application.

6.3. Sécurité sur Api REST en mode OAuth2/OIDC

- Repérer la dépendance fondamentale ***spring-boot-starter-oauth2-resource-server*** déjà placée dans pom.xml
- Repérer le paramétrage suivant au sein du fichier **application-withSecurity.properties**
  spring.security.oauth2.resourceserver.jwt.issuer-uri=
  https://www.d-defrance.fr/keycloak/realms/sandboxrealm
- Compléter la partie manquante du paramétrage de la sécurité au sein du fichier tp.appliSpring.security.**SecurityConfigForRest**
  A ce stade du Tp , la partie à compléter se situe au niveau de la sous fonction

private HttpSecurity ***restFilterChainBuilder***(HttpSecurity http) appelée par le point d'entrée @Bean @Order(1) @Profile("!withoutOAuth2")
public SecurityFilterChain restFilterChainWithOAuth2(HttpSecurity http) .

- Démarrer l'application SpringBoot avec les profils "dev,reInit,**withSecurity**"
- Tester le comportement sécurisé de la partie rest via :
  - une première tentative de compteAjax avec un ajout de compte (label="ccc" , solde=0) menant normalement au message d'erreur 401/Unauthorized
  - une seconde tentative après un login oAuth2/oidc via le compte "admin1/pwd1"

6.4. Sécurité sur Api REST sans OAuth2 avec gestion directe JWT

- Repérer la partie ***passwordEncoder*** présente dans la classe principale *AppliSpringApplication* et son utilisation au sein de la méthode .**create**() de la classe *ServiceClientDirectImpl* .
- Analyser globalement tout le code de la partie t**p.appliSpring.security.generic.standalone** et la dépendance *jjwt-impl* dans pom.xml
- Compléter le code de la méthode **loadUserByUsername** de la classe tp.appliSpring.bank.security.**MyUserDetailsService**
- Analyser la configuration de la partie **restFilterChainWithoutOAuth2()** du fichier tp.appliSpring.security.**SecurityConfigForRest**

- Démarrer l'application SpringBoot avec les profils "dev,reInit,**withSecurity,withoutOAuth2**"
- Tester le comportement sécurisé de la partie rest via :
  - un **standaloneLogin/logout** pour bien repartir de zéro .
  - une première tentative de ***compteAjax*** avec un ajout de compte (label="ccc" , solde=0) menant normalement au message d'erreur 401/Unauthorized
  - une seconde tentative après un login en mode "**standaloneLogin (sans oauth2)**" via le compte "**client_1/pwd**"

6.5. Paramétrage @PreAuthorize sur CompteRestCtrl

- Ajouter sur certaines méthodes de la classe `CompteRestCtrl` des annotations **`@PreAuthorize()`** combinant si besoin `hasRole('ADMIN') or hasRole('CUSTOMER') or hasAuthority('SCOPE_resource.write')` de manière à ce que seuls les administrateurs , "client/customer" ou bien les personnes ayants les scopes suffisants (SCOPE_resource.write ou SCOPE_resource.delete) puisse déclencher les requêtes en mode POST,PUT et DELETE .
- En mode oAuth2 , tester le comportement affiné de la sécurité en tentant les modifications au sein de **compteAjax** après s'être connecté avec le compte **"user1/pwd1" (403/Forbidden)** puis **"admin1/pwd1"**.
- En mode "standaloneLogin (sans oauth2)", tester le comportement affiné de la sécurité en tentant les modifications au sein de **compteAjax** après s'être connecté avec le compte **"user1/pwd1" (403/Forbidden)** puis **"client_1/pwd"**.

6.6. Sécurité sur partie site (thymeleaf) / Tp facultatif

- Analyser la configuration du fichier tp.appliSpring.security.**`SecurityConfigForSite`**
- On voit que le service ***UserDetailsService*** est également utilisé à ce niveau .
- Analyser les parties "**login/logout**" de la classe tp.appliSpring.bank.site.controller.**AppCtrl** ainsi que **advice/ErrorController**
- Analyser src/main/resources/**templates/login.html** et **error.html**
- Tester le comportement sécurisé de la partie "site" via :
  - "site" → "welcome" → "logout" pour bien repartir de zéro .
  - une navigation vers "espace_client" automatiquement reroutée vers "login" où il faut saisir "client_1/pwd" pour aller plus loin
  - un éventuel "logout" pour arrêter la session .

6.7. Test unitaire partiel facultatif de CompteRestCtrl en mode sécurisé

Coder (facultativement) un début de TestCompteRestCtrlWithServiceMockWithOauth2Security en s'inspirant de TestHelloRestCtrlWithMocks.java.txt

6.8. (facultatif) avec projet annexe "mysecurity-autoconfigure"

- Visualiser le contenu du projet annexe "mysecurity-autoconfigure" et la dépendance entre notre projet "debutApplispringWeb" et ce projet optionnel .
- Visualiser le déclenchement automatique de xy.MySecurityConfig via le fichier **org.springframework.boot.autoconfigure.AutoConfiguration.imports** présent dans la partie **src/main/resources** du projet **mysecurity-autoconfigure** (à construire et installer via "*mvn install*")
- Visualiser les paramétrages de type "**mysecurity.area.permit-all-list**" de **application-withSecurity.properties** qui sont spécifiquement analysés par le code interne de mysecurity-autoconfigure
- Faire évoluer le code de la classe `SecurityConfigForRest` en décommentant la partie "//optional config from mysecurity-autoconfigure … myPermissionConfigurer" et en basculant de version de restFilterChainBuilder() .
- Déclencher "mvn clean package" sur le projet "debutAppliSpringWeb"
- Redémarrer l'application en mode sécurisé et tester le bon fonctionnement
- Mémoriser (via copies en commentaires) et modifier les valeurs de "mysecurity.area.permit-all-list" et "mysecurity.area.permit-get-list" de application-withSecurity.properties
- Redémarrer l'application en mode sécurisé et visualiser un comportement altéré
- Restaurer les valeurs appropriées de "mysecurity.area.permit-all-list" et "mysecurity.area.permit-get-list"
- Redémarrer l'application en mode sécurisé et tester le bon fonctionnement

7. Tp sur packaging et supervision d'appli spring

7.1. (tp facultatif) Déploiement d'une application Spring6/SpringBoot3 au format ".war" vers tomcat10

- Faire **hériter** la classe principale AppliSpringApplicationde **SpringBootServletInitializer**
- Générer par assistant de l'IDE une redéfinition (@Override) de la méthode **configure**()
- placer ce code au sein de la méthode configure() :
  return builder.sources(AppliSpringApplication.class).profiles("dev","reInit");
- ajouter temporairement **<packaging>war</packaging>** au sein de pom.xml
- construire **target/debutAppliSpringWeb.war** en lançant **mvn package** en mode **skipTest**
- installer tomcat10 en téléchargeant le .zip et en décompactant son contenu dans un répertoire de type c:\prog ou autre .
- Ajouter si besoin ***set JAVA_HOME=chemin_menant_au_jdk17_ou21*** au sein de c:\prog\tomcat10...\bin\startup.bat
- Démarrer tomcat via startup.bat
- Vérifier son bon fonctionnement via http://localhost:8080
- recopier debutAppliSpringWeb.war au sein du répertoire c:\prog\tomcat10...**webapps**
- Vérifier le bon fonctionnement de l'application via http://localhost:8080/debutAppliSpringWeb
- arrêter tomcat10
- retirer ou mettre en commentaire **<packaging>war</packaging>** au sein de pom.xml

7.2. (tp facultatif) Déploiement d'une application Spring6/SpringBoot4 via docker

- Récupérer une copie du code de l'application sur une machine compatible docker (ex : linux ou bien windows + WSL2)
- construire une image docker en utilisant le fichier Docker file ou bien une cible maven adéquate
- lancer le conteneur docker à partir de l'image construite

7.3. (tp facultatif) actuator

- Activer certains "actuator" au sein de l'application et afficher certaines mesures via les URL REST adéquates