1. Différents types de tests (environnement spring)

- **Test purement unitaire** (*à portée très limitée*) : on ne teste qu'une toute petite partie de l'application spring (ex : quelques méthodes d'un service) en simulant (via des "mocks" le ou les DAOs en arrière plan)
- **Test d'intégration de type "end-to-end"** : on teste l'ensemble de l'application spring (c'est à dire toute la chaine "Api-REST + services_métiers + DAO + base_H2_ouAutre"
- **Test d'intégration partiel** : on teste une sous partie de l'application spring (exemples : "DAO + baseH2" ou bien "service_métier + DAO + baseH2"

  ou bien "Api-REST + mockDeServicesMetiers")
- ...

2. Test purement unitaire

Code à tester :

```
...
@Service
@Transactional
public class ServiceDeviseV2 implements ServiceDevise{
	private static Logger logger = LoggerFactory.getLogger(ServiceDeviseV2.class);

	private RepositoryDevise repositoryDevise;

	//injection de dépendance par constructeur
	public ServiceDeviseV2(RepositoryDevise repositoryDevise) {
		this.repositoryDevise=repositoryDevise;
		logger.debug("ServiceDeviseV2 instance="+this.toString()
		   + " using repositoryDevise="+repositoryDevise.getClass().getName());
	}


	@Override
	public double convertir(double montant, String codeDeviseSource, String codeDeviseCible)
	 throws NotFoundException {
		try {
			Devise deviseSource = repositoryDevise.findById(codeDeviseSource).get();
			Devise deviseCible = repositoryDevise.findById(codeDeviseCible).get();
			return montant * deviseCible.getChange() / deviseSource.getChange();
		} catch (Exception e) {
			e.printStackTrace();
			throw new NotFoundException("devise_not_found",e);//ameliorable en précision
		}
	}
....
}
```

***TestAlgorithmiePurementUnitaire**.java*

```
package com.mycompany.xyz.test;

import static org.mockito.ArgumentMatchers.anyString;
import java.util.Optional;
import org.junit.jupiter.api.Assertions;    import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;          import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.InjectMocks;
import org.mockito.Mock;                    import org.mockito.Mockito;
import org.mockito.MockitoAnnotations;
import org.mockito.junit.jupiter.MockitoExtension;
import org.slf4j.Logger;                    import org.slf4j.LoggerFactory;

import com.mycompany.xyz.entity.Devise;
import com.mycompany.xyz.repository.RepositoryDevise;
import com.mycompany.xyz.service.ServiceDeviseV2;

@ExtendWith(MockitoExtension.class) //for JUnit 5
public class TestAlgorithmiePurementUnitaire {
        //QUOI, des Mocks presque partout, mais de qui se moque-t-on ?

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

        @InjectMocks
        /* @InjectMocks pour demander à "Mockito" (sans spring) de :
            - créer une instance normale de cette classe (ici new ServiceDeviseV2(...))
             - d'injecter le ou les @Mock(s) de cette classe de test dans la classe ServiceDeviseV2()
                via un constructeur adéquat
         */
        private ServiceDeviseV2 serviceDevise;
        //à partiellement tester d'un point de vue purement algorithmique
        //et sans s'appuyer sur le contexe spring

        @Mock /* @Mock pour demander à "Mockito" (sans spring) de :
                - créer ultérieurement un Mock de l'interface
                - NB: il faudra appeler MockitoAnnotations.openMocks(this);
                     pour initialiser tous les mocks de this préfixés par @Mock
                */
        private RepositoryDevise daoDeviseMock; //mock à utiliser


        @BeforeEach
        public void reInitMock() {
                //Mockito.initMocks(this); in old Junit 4
                MockitoAnnotations.openMocks(this); //with JUnit5/Jupiter
                /* MockitoAnnotations.openMocks(this) permet de créer des instances de chaque mock
                   préfixé par @Mock au sein de this .
                   ce qui revient au même que d'écrire :
                      this.daoDeviseMock = Mockito.mock(RepositoryDevise.class);
                      this.mock2=Mockito.mock(Interface_ouClasse2.class);
                      s'il n'y avait pas d'utilisation de @Mock
                 */
        }
```

```
	@Test
	public void testConvertir() {
		double montant=100;
		String codeDeviseSource="EUR";
		String codeDeviseCible="USD";
		double montantConverti=-1;
		//1.préparation du mock en arrière plan:
		Mockito.when(daoDeviseMock.findById(codeDeviseSource))
			.thenReturn(Optional.of(new Devise("EUR","Euro",1.0)));
		Mockito.when(daoDeviseMock.findById(codeDeviseCible))
			.thenReturn(Optional.of(new Devise("USD","Dollar",1.1)));
		//2.appel de la méthode convertir sur le service et test retour
		montantConverti = serviceDevise.convertir(montant, codeDeviseSource,
							codeDeviseCible);
		logger.debug("montantConverti="+montantConverti);
		Assertions.assertEquals(montant * 1.1 , montantConverti, 0.000001);
		//3.verif service appelant 2 fois deviseDao.findById() via aspect spy de Mockito:
		Mockito.verify(daoDeviseMock, Mockito.times(2)).findById(anyString());
	}

	@Test
	public void testConvertirAvecDeviseInconnue() {
		double montant=100;
		String codeDeviseSource="EUR";
		String codeDeviseCibleInconnu="C??";
		double montantConverti=-1;
		//1.préparation du mock en arrière plan:
		Mockito.when(daoDeviseMock.findById(codeDeviseSource))
			.thenReturn(Optional.of(new Devise("EUR","Euro",1.0)));
		Mockito.when(daoDeviseMock.findById(codeDeviseCibleInconnu))
			.thenReturn(Optional.empty());
		//2.appel de la méthode convertir sur le service et test exception en retour
		try {
		  montantConverti = serviceDevise.convertir(montant, codeDeviseSource,
							codeDeviseCibleInconnu);
		  Assertions.fail("une exception aurait normalement du remonter");
		}catch(Exception ex) {
			logger.debug("exception normalement attendue="+ex.getMessage());
			Assertions.assertTrue(ex.getClass().getSimpleName()
							.equals("NotFoundException"));
		}
	}
}
```

3. Test spring avec DirtiesContext

//Basic spring component to experiment @DirtiesContext() in test class

```
package com.mycompany.xyz.ex;
import java.util.HashSet;        import java.util.Set;
import org.slf4j.Logger;         import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Component;


@Component
public class MySpringSetEx {
        private Set<String> exDataSet = new HashSet<>();

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


        public MySpringSetEx() {
                super();
                logger.debug("MySpringSetEx instance="+this.toString());
        }

        public void addDataInExDataSet(String value) {
                exDataSet.add(value);
        }

        public Set<String> getExDataSet() {
                return this.exDataSet;
        }
}
```

TestWithOrWithoutDirtyContext.java

```
package com.mycompany.xyz.test;
import java.util.Set;

import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Order;       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.boot.test.context.SpringBootTest;
import org.springframework.test.annotation.DirtiesContext;
import org.springframework.test.annotation.DirtiesContext.ClassMode;
import org.springframework.test.annotation.DirtiesContext.MethodMode;
import org.springframework.test.context.ActiveProfiles;
import org.springframework.test.context.junit.jupiter.SpringExtension;

import com.mycompany.xyz.MySpringBootApplication;
import com.mycompany.xyz.ex.MySpringSetEx;

@ExtendWith(SpringExtension.class)
@SpringBootTest(classes= {MySpringBootApplication.class})
@ActiveProfiles("embeddedDb")
public class TestWithOrWithoutDirtyContext {

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

```
	@Autowired
	private MySpringSetEx mySpringSetEx; //à tester/utiliser

	@Test
	@Order(1)
	public void addAndGetData1InMySpringContext() {
		mySpringSetEx.addDataInExDataSet("data_1a");
		mySpringSetEx.addDataInExDataSet("data_1b");
		Set<String> dataSet1 = mySpringSetEx.getExDataSet();
		logger.debug("dataSet1 =  " + dataSet1); //dataSet1 =  [data_1a, data_1b]
		Assertions.assertTrue(dataSet1.size() == 2);
	}

	/*
	@Test
	@Order(2)
	public void addAndGetData2InMySpringContextWithoutDirtiesContextBadTest() {
		mySpringSetEx.addDataInExDataSet("data_2a");
		mySpringSetEx.addDataInExDataSet("data_2b");
		Set<String> dataSet2 = mySpringSetEx.getExDataSet();
		logger.debug("dataSet2 =  " + dataSet2); //dataSet2 =  [data_1a, data_2b, data_2a, data_1b]
		Assertions.assertTrue(dataSet2.size() == 2); //failing test:  4 != 2
	}
	*/

/*
@DirtiesContext demande à réinitialiser le contextSpring (et tout son contenu : tous ses composants)
de façon à obtenir des tests aux comportements plus "unitaires"
(sans effets de bord engendrés par les tests précédents)
Ne pas en abuser car cela peut ralentir l'exécution d'une séquence de tests
Si placé sur une méthode : @DirtiesContext(methodMode = MethodMode.BEFORE_METHOD or
MethodMode.AFTER_METHOD)
Si placé sur la classe de test : @DirtiesContext(classMode = ClassMode.BEFORE_CLASS or
 ClassMode.BEFORE_EACH_TEST_METHOD or ClassMode.AFTER_EACH_TEST_METHOD
  or ClassMode.AFTER_CLASS)
*/
	@Test
	@Order(2)
	@DirtiesContext(methodMode = MethodMode.BEFORE_METHOD)
	public void addAndGetData2InMySpringContext() {
		mySpringSetEx.addDataInExDataSet("data_2a");
		mySpringSetEx.addDataInExDataSet("data_2b");
		Set<String> dataSet2 = mySpringSetEx.getExDataSet();
		logger.debug("dataSet2 =  " + dataSet2); //dataSet2 =  [data_2b, data_2a]
		Assertions.assertTrue(dataSet2.size() == 2); //with success
	}

}
```

4. Test d'intégration partiel

4.1. Version 1 sans @MockBean

WithMockDaoConfig.java

```
package com.mycompany.xyz.test;

import org.mockito.Mockito;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Primary;
import org.springframework.context.annotation.Profile;

import com.mycompany.xyz.repository.RepositoryDevise;

//NB: cette classe associée au mini-profile "mock-dao"
//n'est utile que pour la classe TestServiceDeviseWithDaoMockV1
//et n'est plus nécessaire pour la V2 qui utilise @MockBean à la place de @Autowired

@Configuration
public class WithMockDaoConfig {

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

	  @Bean()
	  @Profile("mock-dao")
	  @Primary //for overriding default spring-data-jpa dao
	  public RepositoryDevise daoDeviseMock() {
	     logger.info("Mocking: {}", RepositoryDevise.class);
	     return Mockito.mock(RepositoryDevise.class);
	  }
}
```

TestServiceDeviseWithDaoMockV1.java

```
package com.mycompany.xyz.test;

//pour assertTrue (res==5) au lieu de Assertions.assertTrue(res==5)
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertTrue;

import java.util.ArrayList;    import java.util.List;    import java.util.Optional;

import org.junit.jupiter.api.BeforeEach;       import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;      import org.mockito.Mockito;
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.junit.jupiter.SpringExtension;

import com.mycompany.xyz.MySpringBootApplication;
import com.mycompany.xyz.entity.Devise;
import com.mycompany.xyz.repository.RepositoryDevise;
import com.mycompany.xyz.service.ServiceDevise;
```

```
@ExtendWith(SpringExtension.class)
@SpringBootTest(classes= {MySpringBootApplication.class})
@ActiveProfiles({"embeddedDb","mock-dao"})
public class TestServiceDeviseWithDaoMockV1 {

	@Autowired
	private ServiceDevise serviceDevise; //à tester
	//(NB: ce test fonctionne avec l'implémentation ServiceDeviseV2)

	@Autowired
	private RepositoryDevise daoDeviseMock; //mock à utiliser

	@BeforeEach
	public void reInitMock() {
		//vérification que le dao injecté est bien un mock :
		assertTrue(Mockito.mockingDetails(daoDeviseMock).isMock());
		//reinitialisation du mock(de scope=Singleton par defaut) sur aspects stub et spy :
		Mockito.reset(daoDeviseMock);
	}

	@Test
	public void testRechercherDevises() {
		//préparation du mock (qui sera utilisé en arrière plan du service à tester):
		List<Devise> devises = new ArrayList<>();
		devises.add(new Devise("EUR","Euro",1.0));
		devises.add(new Devise("USD","Dollar",1.1));
		Mockito.when(daoDeviseMock.findAll()).thenReturn(devises);
		//vérification du résultat du service
		List<Devise> listeDevises = serviceDevise.rechercherDevises();
		System.out.println("listeDevises="+listeDevises);
		assertTrue(listeDevises.size()==2);
		//vérifier si le service a appeler 1 fois findAll() en interne sur le dao:
		Mockito.verify(daoDeviseMock, Mockito.times(1)).findAll();
	}

	@Test
	public void testRechercherDeviseParCode() {
		//préparation du mock (qui sera utilisé en arrière plan du service à tester):
		Devise d = new Devise("Ms","Monnaie de singe",1234.567);
		Mockito.when(daoDeviseMock.findById("Ms")).thenReturn(Optional.of(d));
		//vérification du résultat du service
		Devise deviseRemontee  = serviceDevise.rechercherDeviseParCode("Ms");
		System.out.println("deviseRemontee="+deviseRemontee);
		assertEquals(deviseRemontee.getNom(),"Monnaie de singe");
		//vérifier si le service a appeler 1 fois findById() en interne sur le dao:
		Mockito.verify(daoDeviseMock, Mockito.times(1)).findById(Mockito.anyString());
	}
}
```

4.2. Version 2 avec @MockBean

TestServiceDeviseWithDaoMockV2.java

```
...
import org.springframework.boot.test.mock.mockito.MockBean;
...

@ExtendWith(SpringExtension.class)
@SpringBootTest(classes= {MySpringBootApplication.class})
@ActiveProfiles({"embeddedDb"}) //plus besoin du mini profile "mock-dao"
                                //car utilisation de @MockBean dans cette V2
public class TestServiceDeviseWithDaoMockV2 {
        private static Logger logger = LoggerFactory.getLogger(TestServiceDeviseWithDaoMockV2.class);

        @Autowired
        private ServiceDevise serviceDevise; //à tester

        @MockBean
        /* @MockBean pour demander à "Spring+Mockito" de :
                - créer un Mock de l'interface
                - faire en sorte que ce Mock remplace le composant habituel
                   (un peu comme WithMockDaoConfig avec @Primary)
                - INJECTER PARTOUT (ici et dans ServiceDeviseV2) un Mock de l'interface
                   plutôt que le véritable composant spring
                */
        private RepositoryDevise daoDeviseMock; //mock à utiliser


        @BeforeEach
        public void reInitMock() {
                //vérification que le dao injecté est bien un mock
                assertTrue(Mockito.mockingDetails(daoDeviseMock).isMock());
                //reinitialisation du mock(de scope=Singleton par defaut) sur aspects stub et spy
                Mockito.reset(daoDeviseMock);
        }

        @Test
        public void testRechercherDevises() {
                //préparation du mock (qui sera utilisé en arrière plan du service à tester):
                List<Devise> devises = new ArrayList<>();
                devises.add(new Devise("EUR","Euro",1.0));
                devises.add(new Devise("USD","Dollar",1.1));
                Mockito.when(daoDeviseMock.findAll()).thenReturn(devises);
                //vérification du résultat du service
                List<Devise> listeDevises = serviceDevise.rechercherDevises();
                System.out.println("listeDevises="+listeDevises);
                assertTrue(listeDevises.size()==2);
                //vérifier si le service a appeler 1 fois findAll() en interne sur le dao:
                Mockito.verify(daoDeviseMock, Mockito.times(1)).findAll();
        }
...
}
```

5. Optimisation sur la partie à charger et tester

5.1. Via configuration adéquate

ServiceAndDaoConfig.java (à mettre dans un package externe ou bien avec un profile)

```
package com.mycompany.partial_config;

import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
import org.springframework.boot.autoconfigure.domain.EntityScan;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.jpa.repository.config.EnableJpaRepositories;

@Configuration
@EnableAutoConfiguration
@ComponentScan(basePackages = { "com.mycompany.xyz.service"  })
@EnableJpaRepositories(basePackages = { "com.mycompany.xyz.repository" })
@EntityScan(basePackages = { "com.mycompany.xyz.entity"})
public class ServiceAndDaoConfig {

}

/*
 Usage:
  @ExtendWith(SpringExtension.class)
  //@SpringBootTest
 //@SpringBootTest(classes= {MySpringBootApplication.class})
  @SpringBootTest(classes= {ServiceAndDaoConfig.class})
  in order to load "DAO + Service" components only in service test
  no need of "RestController" components for internal business service tests
  ServiceAndDaoConfig is more light than all MySpringBootApplication .
 */
```

5.2. @DataJpaTest

```
@ExtendWith(SpringExtension.class) //si junit5/jupiter
//@SpringBootTest(classes= {AppliSpringApplication.class}) //meme config que classe avec main()
@DataJpaTest //better of SpringBootTest for dao testing if use of spring-data-jpa extension
public class TestCompteDao {
…
}
```

6. Test "end-to-end" sur backend

Voir fin du chapitre "WS-REST"

….