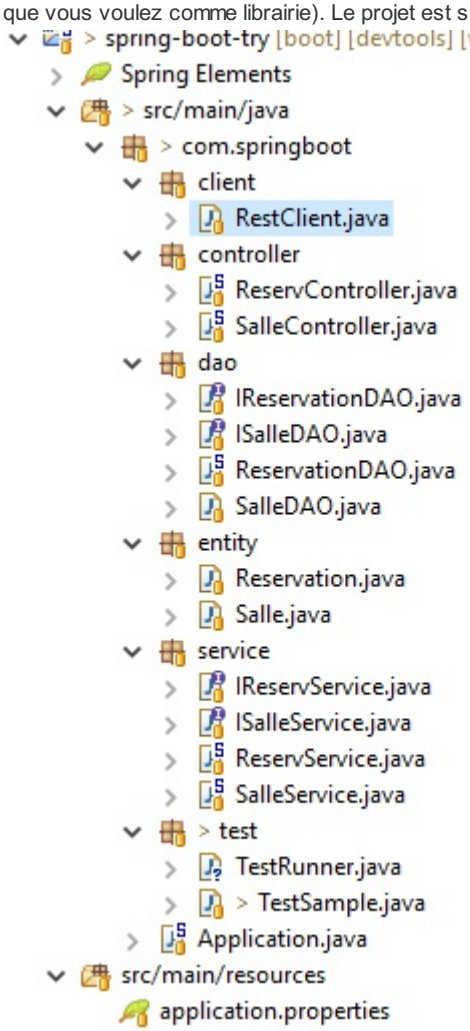
Editor: Yuxin SHI
SI 2 FISE

Compte rendue : Outils pour cycle de vie Logiciel

TP3

1. Implémenter une deuxième fonctionnalité de votre choix pour l'utilisateur enseignant, en se basant sur le principe de multi-module vue dans le cours (vous êtes libre dans le choix de l'implémentation de cette fonctionnalité, vous pouvez utiliser ce que vous voulez comme librairie). Le projet est sur github: Click github yuxin Ceci la structure du projet:



J'ai implementé une nouvelle fonctionnalité : consulter les salles, ainsi sa disponibilité. Par exemple:

- Si on veut savoir que la salle id 1 est disponible ou pas à 2017/10/24: 2017/10/24:
  http://localhost:8080/reserv/salle/1/2017-10-24
  Le résultat:

no

- Si on veut consulter tous les salles:
  http://localhost:8080/reserv/salle/
  Le résultat:

2. Centraliser les informations, factoriser les dépendances de votre projet (utiliser le principe des propriétés).
   Avec création des modules maven, j'ai créer le projet comme ça:

```
<modules>
  <module>com.springboot.controller</module>
  <module>com.springboot.service</module>
  <module>com.springboot.common</module>
  <module>com.springboot.dao</module>
  <module>com.springboot.test</module>
</modules>
```

- Le module controller c'est le coté client, il dépend de module common et le module service.
- Le module service est le serveur qui propose les services de consulter les reservations, il dépend de module common et dao.
- Le module common contient les éléments qui est utilisés pour tous les modules.
- Le module dao est aussi une partie de serveur, mais je le spécifie, il dépend du module common.
- Le module test sert à test unitaire

Un exemple de compilation:

```
NFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ com.springboot.
NFO] Nothing to compile - all classes are up to date
NFO] ------------------------------------------------------------------------
NFO] BUILD SUCCESS
NFO] ------------------------------------------------------------------------
NFO] Total time: 4.773 s
NFO] Finished at: 2017-11-14T15:48:49+01:00
NFO] Final Memory: 16M/228M
```

3. Configurer maven afin de pouvoir utiliser ce référentiel central : http://repo.maven.apache.org/maven2/.
   Dans le fichier config de maven: `setting.xml`, on change les parametres:

```
<repositories>
     <repository>
       <id>central</id>
       <url>https://repo.maven.org/maven2</url>
       <releases>
         <enabled>true</enabled>
       </releases>
     </repository>
   </repositories>
   <pluginRepositories>
     <pluginRepository>
       <id>central</id>
       <url>https://repo.maven.org/maven2</url>
       <releases>
         <enabled>true</enabled>
       </releases>
     </pluginRepository>
   </pluginRepositories>
```

4. Installer Archiva sur votre machine (idéalement sur une VM accessible en réseau depuis votre machine), configurer maven pour utiliser ce référentiel distant.
   Il y a deux moyens a utiliser Archiva, soit on l'utilise comme un repository privé principal, soit on l'utilise comme un repository dans les listes de repositories. Ici on ajoute rien dans Archiva donc on va utiliser deuxième méthode.

Après l'installation d'Archiva et la création d'admin. On ajoute un profile dans setting.xml:

```
<profile>
    <id>Archiva Repo</id>
    <activation>
        <activeByDefault>true</activeByDefault>
    </activation>
    <repositories>
        <repository>
            <id>archiva.internal</id>
            <url>http://localhost:8022/repository/internal/</url>
            <releases>
                    <enabled>true</enabled>
            </releases>
            <snapshots>
                <enabled>true</enabled>
             </snapshots>
        <repository>
    </repositories>
</profile>
```

Ces messages sont disponible dans http://localhost:8022/#repositorylist.

5. Implémenter des tests unitaires couvrant certaines méthodes proposées par l'API JUnit et Jmockit.
   Dans mon projet, j'ai ajouté un package: test, ainsi les deux classes qui permet d'effectuer les tests unitaires.
   Test de la methode salleisDispo(), je fais un mock de ISalleService pour qu'il me donne un retour boolean, ceci le code:

```
public class TestSalle {

   @Mocked
   ISalleService mockiSalleService;

   @Before
   public void setUp() throws Exception {
       // on peut factoriser l'initialisation de certaines variables ici !
   }

   /***
    * Init: Create an instance of Salle Controller and a id, a date string
    * Transation: invoke isDispo method with the valid initial parameters
    * Verification: mocked method must be successfully invoked
    ****/

   @Test
   public void testisDispo(){
       //init
       SalleController salleCtl = new SalleController();
       int id = 1;
       String date = "2017-08-22";
       System.out.println("test controller isDispo");
       new NonStrictExpectations(){
           {
               mockiSalleService.isDispo((Date) any, id);
               result = true;
           }
       };

       //Transition
       ResponseEntity<String> isIsalleCtlInvoked = salleCtl.isDispo(id, date);
       assertTrue("controller is dispo is invoked successfully", isIsalleCtlInvoked.getBody().equals("yes"));

       new Verifications() {
           {
               mockiSalleService.isDispo((Date) any, id);
               times = 1;
           }
       };
```

```
    }
}
```

6. Exécuter en mode commande les tests unitaires et vérifier que ça passe sans échec.
   Malheusement, il y a une exception que je ne peut pas resoudre meme avec les solutions proposees sur Internet:

> WARNING: JMockit w as initialized on demand, w hich may cause certain tests to fail;
> please check the documentation for better w ays to get it initialized.

Si on veut upload les artifict du projet sur le repo privé, on peut configurer le fichier pom.xml du projet en ajoutant:

```
<distributionManagement>
  <repository>
    <id>archiva.internal</id>
    <name>Archiva Repo</name>
    <url>http://localhost:8022/repository/internal/</url>
  </repository>
</distributionManagement>
```