CRR 2012/03/02 : Point projet

Agenda :

45min

- Review du CRR W-1
- Questions / Remarques
- Décisions
- Action plan W+1
- Priorités

15min

- Faire ensemble tests Selenium login OK/KO

15 min

- Code review

- Review du CRR W-1

Action	Qui	Deadline	Livrable	Status
Clean du code - update doit être propre	Tarik	23/02	Corriger les soucis imports etc dans trunk pour avoir une version utilisable	DONE (moi ça marche bien, si problemes faite moi des retours)
Ajout des role fonctionnel des tables dans le MCD.sql	Antoine	27/02	Completer TODO pour clarifier les champs des tables ou les corrigés (verifier la compatibilité syntaxe SQL pour les contrainte clé et aussi les type)	DONE 26/02: Mise au carré du script sql - standardisation de la syntaxe Testé sous MySQL fonctionnel A TESTER PAR TOUS Antoine: OK Eric: OK Tarik: OK le script s'est deroulé sans souci Rudy: OK
Refractor CSS	Antoine	29/02	Supprimer éléments inutiles Supprimer propriétés inutiles Supprimer le relatif	ABORTED Je n'avance pas et ca n'a aucune valeur ajoutée pour le moment On dédiera un sprint spécifique au style. Je préfère me focaliser sur les autres tâches
Refractor DAO	Tarik	23/02	3 DAOs réfractorés + Abstract DAO	DONE (si vous avez besoin de methode de DAO pour vos services remontez les moi en todo juste avec la signature de methode donc input/ output et en javadoc ce que vous attendez de cette methode je la ferai des que possible)
Service import	Eric Rudy	06/03 ?? 27/02	Commit les premiers tests Tests Insert base de données checkHeader	WAITING Reste DAO : Eric
Intégration Hibernate	Eric A définir	29/02 A définir par le responsable	Intégration Hibernate fonctionnelle Fichier de config 1 DAO pour le test Tâche pour intégrer les autres DAO sous Hibernate)	DONE Cf ci dessous les étapes d'intégration d'hibernate

Configuration Hibernate	Tarik	09/03	Externaliser toute la config de Hibernate au sein d'un fichier dedié : - XXX-config.xml	TODO En effet, je me suis rendu compte que l'integration de Hibernate se fait que avec des bean spring, donc on peut l'isoler bien a part pour la propreté du projet. Je rappelle que : - ioc-context.xml - mvc-context.xml - security-context.xml sont tous des fichiers spring de bean, j'ai juste fait les bon importe de xmlns et xml schema (cf PXML) donc hibernate peut aussi etre a part, surtt qu'il va grossir pour chaque table du MCD. Au cas ou Eric tu a deja essayé de faire cela et tu t rendu compte qu'il y avait un souci au niveau de la portée des bean (par exemple la dataSourceMySql) dit le moi avant ce soir pour que je perd pas de temps merci. Eric => mail
Proto tweeter	Antoine	05/03	Proto tweeter commit sur /branches/proto/tweeter permettant de récupérer des informaions tweeter qui nous intéresserait pour le projet	DONE 03/03: Commit dans proto/tweeter si vous voulez y jeter un oeil Ca fonctionne bien, une seule dépendance nécessaire, code pas très long. Ca correspond à ce qu'on veut en faire. Ca fera l'objet d'un sprint spécifique mais la faisabilité est validée La partie à clarifier sera la mise en place en mode batch etc. En tout cas trop intéressant
Déterminer l'architecture des tests de DAO (standardiser) : dans quels cas etc, voir si spring test	Antoine	29/02	Architecture définie pour écriture des tests de DAO (cf Spring par la pratique) avec Hibernate	DONE Chaque méthode de DAO doit avoir un test donc au moins : - saveOrUpdate - select * qui retourne qqchose - ROWMAPPER a tester également Cf en dessous du tableau du CRR un exemple avec Person Conclusion: chaque DAO est testé suivant le modèle du test unitaire TestUserDAO.java

Séparation base de test en intégrant hsqldb	Antoine	01/03 06/03	Clarifier fonctionnement et comment séparer les bases - est-ce qu'on utilise ça pour la base de prod Intégration hsqlb périmètre test	DONE On utilisera la base HSQLDB pour les tests, ca permettra de tester en isolation sur une base propre à chaque fois. Si ca sembel fonctionnel, facilement déployable sur serveur d'application, on se posera la question pour la production. DELAYED - Non bloquant base HSQLDB ne respecte pas standard SQL. Nous obligerait à maintenir 2 scripts. Trop lourd et pas forcément utile. On testera sur notre base locale en supprimant s'il faut les jeux de tests
Ajouter dépendance selenium projet	Antoine	02/03	Dépendance Selenium et test avant	DONE 29/02: CA FONCTIONNE !!!! faire un "mvn verify" ca exécutera tout les tests unitaires et test d'intégration
Exemple d'utilisation de OpenCsv	Tarik	1/03	- exemple d'utilisation de la librairie OpenCsv	DONE vous pouvez supprimer ce que j'ai fait mais avant essayer au moins de le comprendre.... serieusement... http://opencsv.sourceforge.net/apidocs/index.html.

- Questions / Remarques

Rappel suite modification du pom :
* plus besoin de serveur d'application en local
* pas besoin de serveur selenium en local
* maven
    - mvn jetty:run : lance le serveur d'application, recompile et déploie toutes les 10sec l'appli si changement
    - mvn selenium:start-server (ou stop-server), permet de lancer le selenium server et de lancer les tests (attention à ce que l'appli soit déployé et que le serveur appli démarré)
    - **mvn verify** : permet d'exécuter les tests unitaires et d'intégration

- Décisions
- Action plan W+1

Action	Qui	Deadline	Livrable	Status
Configuration Hibernate	Tarik	10/03	Externaliser toute la config de Hibernate au sein d'un fichier dedié : - XXX-config.xml	TODO Cf mails Tarik/Eric
Développement page de configuration des events/ eventsparameters	Tarik	14/03	Pas de mise en forme Code brut, uniquement les traitements Sur une page (ou 2 à voir), on doit pouvoir faire les opérations CRUD sur les events et les events parameters Respecter MVC Tests Selenium (penser dès le départ comment tester que ca fonctionne, exemple: mettre un message "Modifications saved" si ca a fonctionné afin de pouvoir faire un assert derrière) - au moins 8, doivent valider qu'on peut: Create, Research, Update, Delete pour Event et EventParameter)	
Integration de Log4j	Tarik	16/03	Tout les logs de l'application doivent être gérés avec Log4J	
Internationalisation du front (I18N)	Tarik	21/03	Gestion des langues dans la partie web en configurant message dans fichiers conf : Doit fonctionner sur page de login, homepage et menu Langues: Anglais et Français Possibilité de choisir la langue sur la page web (dans header à droite)	
DAOs à passer en Hibernate à terminer	Eric	10/03	DAOs User, Event et EventParameterValue en Hibernate avec leurs tests unitaires couvrant toutes les méthodes (modèle TestUserDAO.java)	

Intégration ExtJS sur une page d'exemple dans le projet	Eric	14/03	Avoir une page fonctionnelle sous ExtJs Savoir comment déployer ExtJS sur nos pages existantes Préparer présentation pour vendredi sur utilisation ExtJS pour développer	
Dev service de génération actions	Antoine	16/03	Pouvoir appeler un service qui va générer des actions correspondant à des critères définis - script SQL updated - ajout des entity nécessaires - DAOS - dev service - tests unitaires couvrant min. 80% code service	
Config machine : installer firefo 3.5 + UTF-8 eclipse	Rudy	10/03	"mvn verify" doit passer corriger les fichiers sur le svn dont les accents sont mal affichés	
Conception/Analyse d'impacts gesion des événements manuels	Rudy	13/03	Impacts sur couches : -persistance - service - présentation pour gérer des événements manuels (quels tables/champs il faut en plus de ce qui est prévu dans l'analyse fonctionnelle, quels services on a besoin, un UML si besoin de clarifier)	

- Priorités

Tarik:
- Page configuration événements
- Log4J
-Séparation Hibernate
- I18N

Eric:
- DAOs
- exemple ExtJS

Rudy:
- Config machine propre : firefo 3.5, encodage fichier configuré eclipse UTF-8, le "mvn verify" doit passer
- Conception analyse/impacts pour gestion des événements saisis manuellemment

Antoine:
- Dev service de génération d'actions

15min

- Faire ensemble tests Selenium login OK/KO : Reporté W+1

- Code review

Emplacement	Remarque	Décision prise
Hibernate	**Cf ci dessous : recapitulatif concernant les demarches a faire pour mettre en place hibernate**	**Aucune**

Prochain point : Vendredi 9 Mars

Exemple de référence d'un test DAO pour notre application (méthodes à supprimer en fonction du DAO implémenté)

```
@ContextConfiguration
public class PersonTest extends AbstractTransactionalJUnit4SpringContextTests {

    private JpaPersonDao fPersonDao;
```

```
    @Resource
    public void setPersonDao(JpaPersonDao personDao) {
        fPersonDao = personDao;
    }

    @Test
    public void testSave() {
        createAndSavePerson("David", 28);
        assertEquals(1, countRowsInTable("person"));

        Person david = getSinglePerson();
        assertEquals("Name not saved correctly", "David", david.getName());
        assertEquals("Age not saved correctly", 28, david.getAge());
    }

    @Test
    public void testGetById() {
        createAndSavePerson("David", 28);
        Person david = fPersonDao.getById(0);
        assertEquals(david.getName(), "David");
        assertEquals(david.getAge(), 28);
    }

    @Test
    public void testDelete() {
        createAndSavePerson("David", 28);
        Person david = fPersonDao.getById(0);
        fPersonDao.delete(david);
        fPersonDao.getEntityManager().flush();
        assertEquals("Deleting person failed.", 0, countRowsInTable("person"));
    }

    private Person getSinglePerson() {
        return simpleJdbcTemplate.queryForObject(
                "select * from person where id = ?", new PersonRowMapper(), 0);
    }

    private static class PersonRowMapper implements ParameterizedRowMapper<Person> {
        public Person mapRow(ResultSet rs, int rowNum) throws SQLException {
            Person result = new Person();
            result.setName(rs.getString("name"));
            result.setAge(rs.getInt("age"));
            return result;
        }
    }
}
```

Tutoriel par RAKOTOBE Eric : Comment mettre en place hibernate ?

Etape 1: Importation des dependances utilisees par hibernate dans le pom.xml

Etape 2: Configuration du iioc-context.xml

```
<!-- Declaration de la datasource JDBC -->
<bean id="mysqlDataSource" class="org.springframework.jdbc.datasource.SingleConnectionDataSource">
        <property name="driverClassName" value="${database.driver}" />
        <property name="url" value="${database.url}" />
        <property name="username" value="${database.user}" />
        <property name="password" value="${database.password}" />
 </bean>

 <!--Transaction manager Hibernate qui s'occupera de la gestion des sessions-->
 <bean id="transactionManager" class="org.springframework.orm.hibernate3.HibernateTransactionManager"
 p:sessionFactory-ref="sessionFactory"/>

 <!--La SessionFactory s'occupe de fournir les sessions à l'application quand des ordres Hibernate s'éxécutent.-->
<bean id="sessionFactory"
class="org.springframework.orm.hibernate3.annotation.AnnotationSessionFactoryBean" >
  <!--Nous definirons ici une par une les classes de domain (POJO) annotees pour etre reconnu par hibernate-->
        <property name="annotatedClasses">
                <list>
                        <value>com.miage.crm365.model.entity.User</value>
                                <value>com.miage.crm365.model.entity.Customer</value>
                        <value>com.miage.crm365.model.entity.Event</value>
                        <value>com.miage.crm365.model.entity.EventParameterValue</value>
                </list>
        </property>
        <property name="hibernateProperties">
                <props>
                                <prop key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>
                                <prop key="hibernate.show_sql">true</prop>
                </props>
        </property>
        <property name="dataSource" ref="mysqlDataSourceForHibernate"/>
   </bean>
```

Etape 3 : Les classes et interfaces

-com.miage.crm365.model.entity.User
=> POJO persisté en bdd, les attributs represententles colonnes, et l'annotation @Table(name="users") permet de specifier a hibernate que la table a persisté est la table users. Les autre annotations parlent d'elles-memes

-com.miage.crm365.model.dao.CustomHibernateDaoSupport etend HibernateDaoSupport
=> Cette classe est une illustration de l'ioc avec un setter sur sessionFactory definit dans ioc-context.xml

-com.miage.crm365.model.dao.IUserDAO
=> contrat interface permettant les opérations CRUD (Create Remove Udpate Delete) sur la table User de la bdd

-com.miage.crm365.model.dao.impl.UserDAO
=> implementation de IUserDAO

- Elle etend CustomHibernateDaoSupport pour pouvoir heriter de la methode "getHibernateTemplate", definie dans le framework hibernate et qui facilite grandement les requetes
- L'annotation **@Repository("IUserDAO")** indique a hibernate que dans le lors de la creation du bean, la classe a utilise est l'implementation UserDAO et non l'interface IUserDAO, ceci permet la flexibilite des heritages, si un jour on voulait utilisé une autre implementation, il suffirait de mettre l'annotation **@Repository("IUserDAO")** au dessus de la classe qu'on voudra utilise
- L'annotation **@Transactional** , comme son nom l'indique, permet de spécifier que les methodes annotees, ou la classe en elle-meme est transactionnelle, cad entre en interaction avec la bdd

=>**Remarque très importantes : Lors des tests unitaires, il faut aussi mettre l'annotation @Transactional au dessus du nom de la classe de test, si on voudra tester des interactions avec la bdd .**

Etape 4 : Utilisation/Injection des DAO dans les controllers

Ici, j'ai illustré l'injection dans le controller de login, pour ne pas avoir a refaire un autre controller ...

Dans le controller :

- **@Autowired**
  private IUserDAO userDAO;
  > Injection de dependance pour le DAO

- **this.userDAO.getAllUtilisateur();**
  for (User u : users){
  System.out.println("username & password = " + u.getUsername() + u.getPassword());
  }
  > Utilisation de notre DAO pour effectuer les acces a la bdd

Dans la console on obtient alors les logs de hibernate et les system.out.println de debuggage

Hibernate: select user_.username, user_.enabled as enabled0_, user_.password as password0_ from users user_ where user_.username=?

userdao = com.miage.crm365.model.dao.impl.UserDAO@1bb41d7
==========Debut Test Hibernate ==========
Hibernate: select this_.username as username0_0_, this_.enabled as enabled0_0_, this_.password as password0_0_ from users this_
users = [com.miage.crm365.model.entity.User@ced6e2, com.miage.crm365.model.entity.User@19a8c41,

com.miage.crm365.model.entity.User@16e1eff, com.miage.crm365.model.entity.User@b52f79,
com.miage.crm365.model.entity.User@13fde6e]
user = com.miage.crm365.model.entity.User@ced6e2
username === cocococo
user = com.miage.crm365.model.entity.User@19a8c41
username === erica
user = com.miage.crm365.model.entity.User@16e1eff
username === ericooopassworderico
user = com.miage.crm365.model.entity.User@b52f79
username === totototo
user = com.miage.crm365.model.entity.User@13fde6e
username === useruser
==================== Fin Test Hibernate ==================================