HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

SMARTCITY

Tests d'implémentation

Tano Iannetta

Loan Lassalle

Luana Martelli

Wojciech Myszkorowski

Camilo Pineda Serna

Jérémie Zanone

sous la direction du Professeur René Rentsch

PRO – mai 2017

Table des matières

1 Présentation

Après la mise en commun des différents composants de l'application, nous avons testé que l'application fonctionne correctement en effectuant différentes manipulations. Ces dernières correspondent à l'utilisation de l'application (par exemple, ajouter un évènement) puis la vérification que le résultat escompté est obtenu (l'évènement est présent dans la base de données, pour reprendre l'exemple précédent). Les manipulations détaillées plus bas permettent de contrôler que les différentes parties de l'application fonctionnent correctement ensemble.

Ce document présente les tests effectués.

Les tests suivants présentent brièvement le but du test et illustrent le résultat. Pour simplifier le déroulement des tests, ils ont été effectués à la suite et se basent, sans perte de généralité, sur l'état final du test précédent.

Pour tous les tests, nous considérerons comme acquis le prérequis :

> La base de données est correctement mise en place (schémas créés et données insérées) et la configuration d'accès à la base de données est correcte (fichier de configuration Hibernate.cfg.xml avec les bons *connection.username* et *connection.password*).

Pour la connexion à la base de données distante, la configuration est déjà en place. Si une connexion veut être réalisée auprès d'une base de données locale, il est alors nécessaire de modifier les valeurs des champs indiqués dans le prérequis plus haut.

2 Ouverture de l'application

But : la fenêtre de l'application est affichée et elle a accès aux évènements de la base de données.

Manipulation : exécuter l'archive.

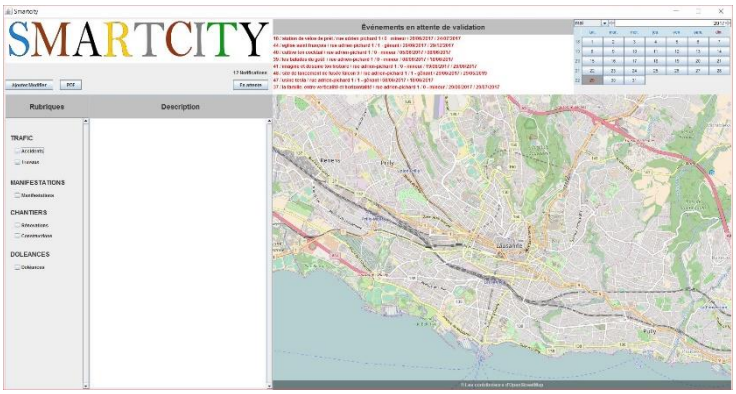


Figure 1 : Capture d'écran après l'ouverture de l'application

Résultat validé. La fenêtre est ouverte et des évènements y sont présents, comme l'atteste la liste des évènements en attente en haut au centre en rouge.

3 Validation d'un évènement en attente

Prérequis : il existe des évènements à valider (ils sont affichés en rouge dans la zone de notifications).

But : pouvoir sélectionner et valider un évènement en attente. Les champs de l'évènement sélectionné sont affichés (et ils peuvent être modifiés. cela est testé plus bas) avant la validation. L'évènement validé n'apparait plus dans la liste des évènements en attente.

Manipulations : depuis la fenêtre principale, cliquer sur [En Attente]. La fenêtre [En Attente] s'ouvre. Dans la liste déroulante, sélectionner un évènement. En bas à gauche de la fenêtre, cliquer sur [Valider]. Un pop-up de confirmation s'affiche. Cliquer sur [OK].

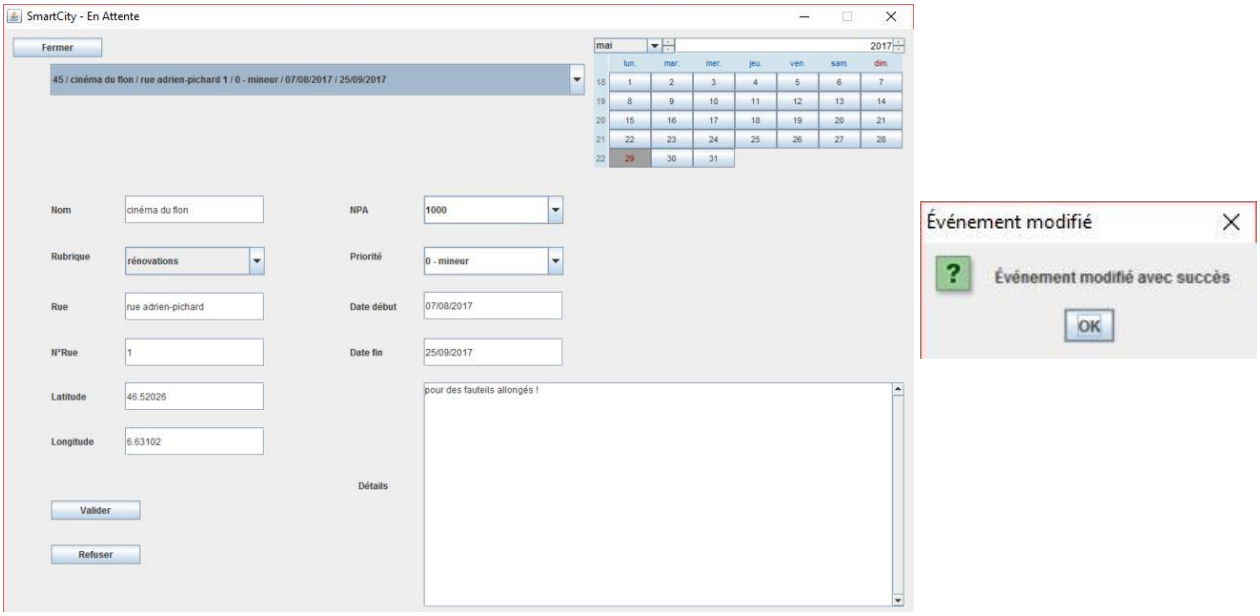


Figure 2 : À gauche, la fenêtre [En Attente] après avoir sélectionné un événement. À droite, le pop-up d'acquittement.

Résultats validés. La fenêtre et les a correctement permis de sélectionner et de valider un évènement, le tout acquitté par un pop-up. L'évènement validé n'est plus présent dans la liste (non montré).

4 Refus d'un évènement en attente

Prérequis : il existe des évènements à valider (ils sont affichés en rouge dans la zone de notifications).

But : pouvoir sélectionner et refuser un évènement en attente. Les champs de l'évènement sélectionné sont affichés avant le refus. L'évènement refusé n'apparait plus dans la liste des évènements en attente.

Manipulations : depuis fenêtre [En Attente]. Sélectionner un évènement de la liste déroulante. En bas à gauche de la fenêtre, cliquer sur [Refuser]. Un pop-up de confirmation s'affiche. Cliquer sur [OK]. La fenêtre [En attente] peut être fermée en cliquant sur le bouton [Fermer] en haut à gauche ou sur la croix en haut à droite.

Résultats validés. Aucune image n'est présentée ici, puisque similairement à « Validation d'un évènement en attente », les champs de l'évènement sont affichés, un pop-up d'acquittement s'affiche et l'évènement n'est plus présent dans la liste.

5 Ajout d'un évènement

But : ajouter un évènement dans la base de données. L'évènement doit persister jusqu'à sa suppression. Les valeurs de champs doivent être correctes avant de pouvoir créer l'évènement, autrement les champs invalides sont indiqués. Un pop-up acquitte l'insertion dans la base de données.

Manipulations : depuis la fenêtre principale, cliquer sur [Ajout/modification]. Laisser le menu déroulant sur [Ajouter un évènement]. Compléter les champs et cliquer sur [Valider] en bas à gauche jusqu'à ce que tous les champs soient corrects. Fermer le pop-up d'acquittement en clinquant sur [OK]. Pour les valeurs des champs, nous avons choisi des noms factices (comme demo-ajout-event), mais avons tout de même sélectionné des latitude et longitude vers le centre-ville de Lausanne.

Figure 3 : Une fois l'ajout de l'évènement confirmé, il apparait dans la liste déroulante de la fenêtre [Ajout/Modification] (surligné en bleu).

Résultats validés. L'évènement est correctement ajouté, comme le montre la figure **Erreur ! Source du renvoi introuvable.**. Les erreurs de saisies des champs empêchent de compléter l'ajout de l'évènement (un exemple de notification d'erreurs est présenté plus bas). Un pop-up d'acquittement s'affiche également (pas montré, mais un exemple est visible à droite de l'image **Erreur ! Source du renvoi introuvable. Erreur ! Source du renvoi introuvable.**).

6 Modification d'un évènement

Prérequis : Il existe au moins un évènement dans la base de données.

But : modifier les champs d'un évènement sélectionné. Les nouvelles valeurs sont contrôlées et la modification n'a pas lieu tant qu'il subsiste des erreurs. Un pop-up d'acquittement s'affiche.

Manipulations : depuis la fenêtre [Ajout/Modification], sélectionner l'évènement à modifier dans la liste déroulante. Modifier les valeurs des champs puis cliquer sur [Valider] en bas à gauche, tant qu'il reste des erreurs. Une fois l'évènement modifié, cliquer sur [OK] du pop-up d'acquittement. Ensuite, la fenêtre est fermée.

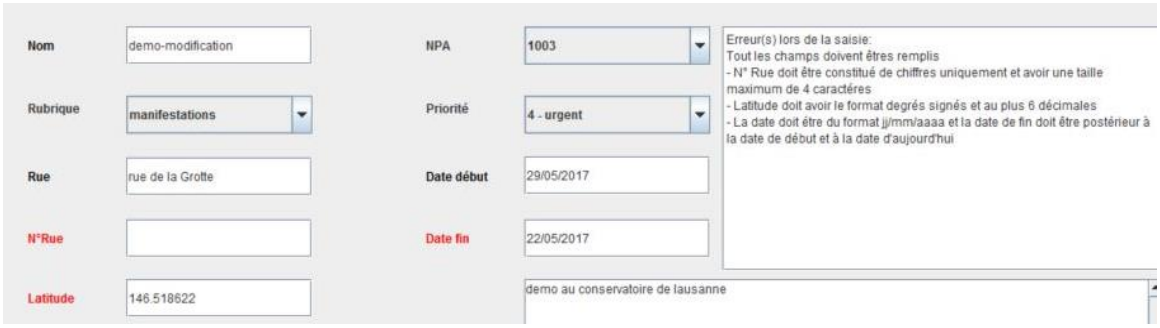


Figure 4 : Détail de la fenêtre [Ajout/Modification] signalant des erreurs dans les champs saisis. Les champs sont indiqués en rouge et une explication est donnée à droite.

Résultats validés. Lors de la modification de l'évènement, les valeurs erronées empêchent à juste titre de valider et d'enregistrer les modifications. Les champs incriminés sont signalés en rouge et une explication est fournie dans la zone droite de la fenêtre.

7 Affichage d'évènements sur la carte

Prérequis : il existe au moins un évènement dans la base de données.

But : afficher la position d'un évènement sur la carte, d'après ses coordonnées géographiques. Parcourir la carte à l'aide d'interactions avec la souris (déplacer, zoomer). Les évènements affichés dépendent de la ou les rubriques sélectionnées, de la date sélectionnée sur le calendrier.

Manipulations : sur la fenêtre principale, sélectionner des rubriques et une date correspondant à un moins un évènement. Le ou les évènements sont ainsi affichés. Cliquer et glisser sur la carte pour la déplacer. Utiliser la molette de la souris pour zoomer ou dézoomer.

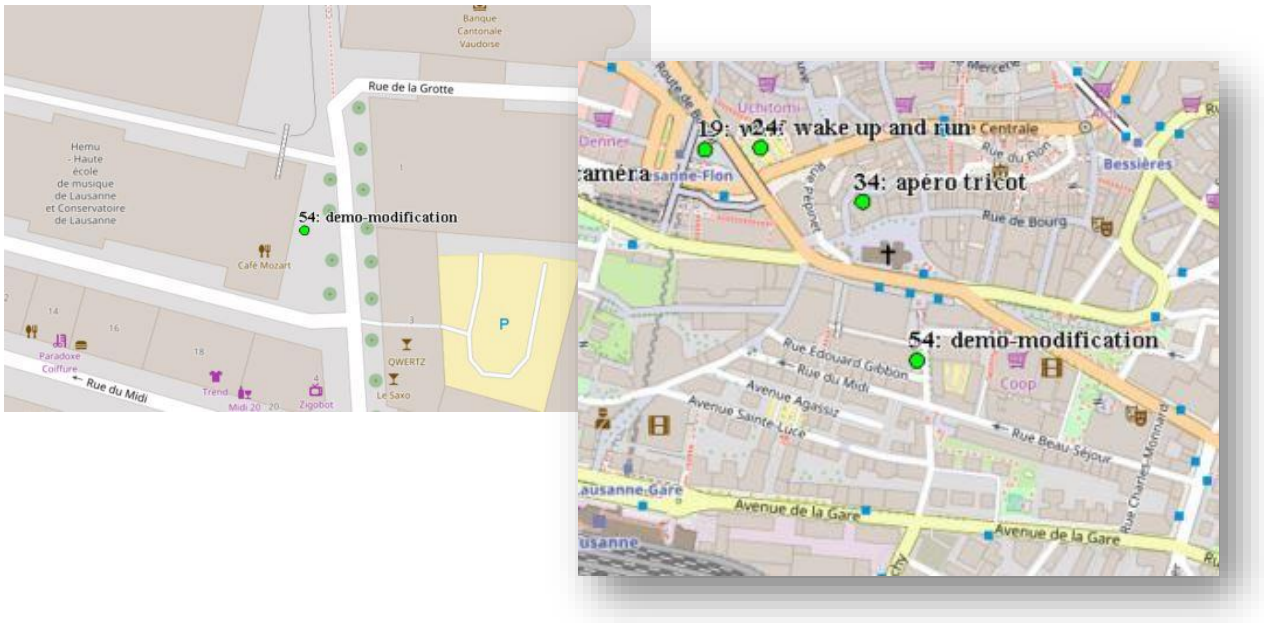


Figure 5 : Deux extraits de la carte interactive avec deux niveaux de zoom différents. Les évènements filtrés par les rubriques et la date y sont affichés.

Résultats validés : les interactions avec la carte se comportent comme attendu. Les sélections des rubriques et de la date agissent comme filtre pour les évènements à afficher. Cela se manifeste non seulement sur la carte, mais également dans la zone de description.

8 Suppression d'un évènement

Prérequis : il existe au moins un évènement dans la base de données.

But : supprimer un élément de l'interface graphique. Cet évènement ne sera plus affiché ni accessible depuis les fenêtres de l'application.

Manipulations : depuis la fenêtre principale, cliquer sur [Ajout/Modification] en haut à gauche. La fenêtre [Ajout/Modification] s'ouvre. Sélectionner l'évènement dans le menu déroulant. Cliquer sur [Supprimer] en bas à gauche. L'évènement est alors supprimé. Et la fenêtre peut être fermée.

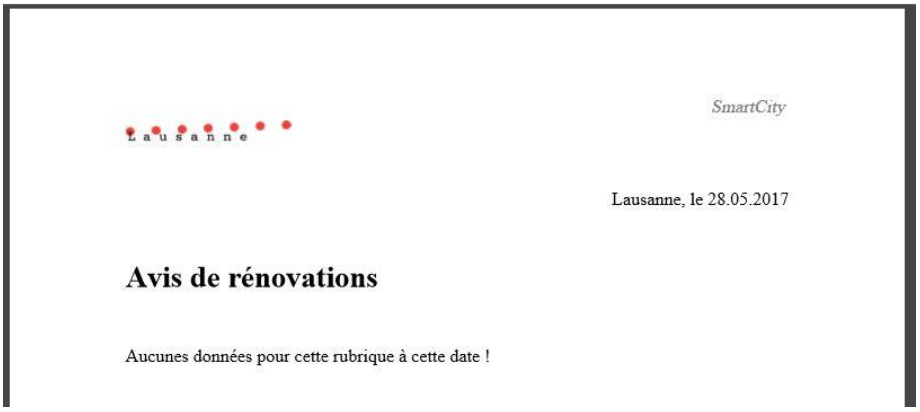
Résultats validés : lorsque l'on rouvre la fenêtre [Ajout/Modification], l'évènement n'est plus disponible dans le menu déroulant.

9 Génération du PDF

Prérequis : au moins un évènement dans la base de données, mais cette partie est mieux testée avec plus d'un évènement d'une catégorie lors d'une date fixée.

But : obtenir un PDF avec un résumé des évènements d'une catégorie. Plusieurs catégories peuvent être sélectionnées, et dans ce cas, plusieurs PDF, un par catégorie, seront créés. Les PDF sont créés dans le dossier `C:\Users\<username>\Smartcity\PDF` et leur nom comporte la date et la rubrique. Si aucun évènement n'est affecté par le filtre (rubrique et date), un PDF indiquant l'absence d'évènements est tout de même créé. Les PDF comportent un graphique avec des données statistiques. Si aucune rubrique n'est sélectionnée, aucun PDF n'est créé.

Manipulations : sur la fenêtre principale, sélectionner des rubriques et une date correspondant à un moins un évènement. Le ou les évènements sont ainsi affichés. Cliquer sur le bouton [PDF] en haut à gauche. Après le traitement,



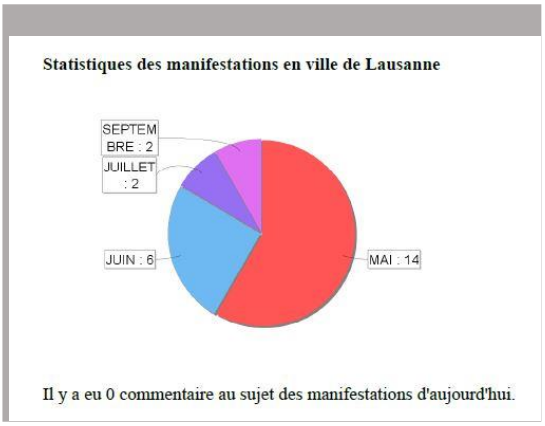


Figure 6 : En haut, début d'un PDF avec l'indication d'absence d'évènements. En bas à gauche, début d'un PDF avec au moins deux évènements. En bas à droite, la fin d'un PDF avec une statistique

Résultats validés : comme attendu, le document PDF contient toutes les informations pour une rubrique et une date données. De plus, une deuxième partie contient des statistiques plus générales sur les évènements choisis. Des documents avec l'indication « aucune donnée ».

10 Tests unitaires de la carte interactive

```java
public class TestFournisseurTuileOSM {

    final int delta = 0.00001;

    @Test
    public void testGetTuile() throws IOException {
        //récupère une tuile spécifique (HEIG-VD) en créant ensuite l'image à partir de la tuile
        FournisseurTuileOSM fournisseur = new FournisseurTuileOSM("http://a.tile.openstreetmap.org/");
        Tuile tuile = fournisseur.getTuile(17, 67960, 46219);
        ImageIO.write(tuile.image(), formatName: "png", new File( pathname: "TuileOSM.png"));
    }

    @Test
    public void testZoom() {
        for (int zoom = 12; zoom < 20; ++zoom) {
            assertEquals(zoom, new PointOSM(zoom, 0, 0).zoom());
        }
    }

    @Test
    public void testToOSM() {
        // on teste 3 positons différentes connues
        PointOSM p1 = new PointWGS84(0, 0).toOSM(0);
        assertEquals(128.0, p1.x(), delta);
        assertEquals(128.0, p1.y(), delta);

        PointOSM p2 = new PointWGS84(-180,85.0511287798)).toOSM(0);
        assertEquals(0, p2.x(), delta);
        assertEquals(0, p2.y(), delta);

        PointOSM p3 = new PointWGS84(180,-85.0511287798).toOSM(0);
        assertEquals(256, p3.x(), delta);
        assertEquals(256, p3.y(), delta);
    }

}
```

Figure 7 : Tests unitaires de la carte interactive

11 Tests de la base de données

```
class DatabaseAccessTest {

    private DatabaseAccess databaseAccess = DatabaseAccess.getInstance();

    /**
     * Test si l'on récupère l'évènement avec le même ID
     */
    @Test
    void get() {
        Evenement evenement = databaseAccess.get(Evenement.class,  id: 1);

        assertNotNull(evenement);
        assertEquals((Integer)1, evenement.getIdEvenement());
    }

    /**
     * Test si l'on récupère tous les évènements
     */
    @Test
    void get1() {

        // Récupère tous les événements
        List<Evenement> evenementList = databaseAccess.get(Evenement.class);

        assertNotNull(evenementList);
        assertNotEquals(0, evenementList.size());

        // Récupère le dernier évènement
        Evenement evenement = databaseAccess.get(Evenement.class, evenementList.size());

        assertNotNull(evenement);

        // Récupère l'évènement après le dernier évènement (non existant)
        evenement = databaseAccess.get(Evenement.class,  id: evenementList.size() + 1);
        assertNull(evenement);
    }
```

Figure 8 : Partie 1 des tests unitaires de la base de données

```java
/**
 * Test si l'enregistrement d'un évènement s'est effectué
 */
@Test
void save() {

    // Récupération d'enregistrements de la base de données
    RubriqueEnfant rubriqueEnfant = databaseAccess.get(RubriqueEnfant.class, id: 1);
    assertNotNull(rubriqueEnfant);

    Utilisateur utilisateur = databaseAccess.get(Utilisateur.class, id: 1);
    assertNotNull(utilisateur);

    Adresse adresse = databaseAccess.get(Adresse.class, id: 1);
    assertNotNull(adresse);

    Double latitude = 5.5;
    Double longitude = 4.2;
    Calendar debut = Calendar.getInstance();

    Priorite priorite = databaseAccess.get(Priorite.class, id: 1);
    assertNotNull(priorite);

    Statut statut = databaseAccess.get(Statut.class, id: 1);
    assertNotNull(statut);

    // Enregistrement du nouvel évènement
    String nomEvenement = "evenement_test";
    Evenement evenement = new Evenement(rubriqueEnfant,
            utilisateur,
            nomEvenement,
            adresse,
            latitude,
            longitude,
            debut,
            fin: null,
            details: "",
            priorite,
            statut);
    assertNotNull(evenement);
    databaseAccess.save(evenement);

    // Récupère tous les événements
    List<Evenement> evenementList = databaseAccess.get(Evenement.class);

    assertNotNull(evenementList);
    assertNotEquals(0, evenementList.size());
    assertTrue(evenementList.contains(evenement));
}
```

Figure 9 : Partie 2 des tests unitaires de la base de données

```java
/**
 * Test si la mise à jour d'un évènement s'est effectuée
 */
@Test
void update() {

    // Récupère le premier évènement avant la mise à jour
    Evenement evenementBefore = databaseAccess.get(Evenement.class,  id: 1);
    assertNotNull(evenementBefore);

    String nomEvenementBefore = evenementBefore.getNomEvenement();
    String nomEvenemenAfter = "nouveau_nom_evenement_test";

    // Mise à jour du nom de l'évènement
    evenementBefore.setNomEvenement(nomEvenemenAfter);
    databaseAccess.update(evenementBefore);

    // Récupère le premier évènement après la mise à jour
    Evenement evenementAfter = databaseAccess.get(Evenement.class,  id: 1);

    assertNotNull(evenementAfter);
    assertEquals(nomEvenemenAfter, evenementAfter.getNomEvenement());
}
```

Figure 10 : Partie 3 des tests unitaires de la base de données

```java
/**
 * Test de la classe EvenementAccess
 *
 * @author Lassalle Loan
 * @since 25.03.2017
 */
class EvenementAccessTest {

    private DatabaseAccess databaseAccess = DatabaseAccess.getInstance();
    private EvenementAccess evenementAccess = EvenementAccess.getInstance();

    /**
     * Test si les évènements récupérés possèdent tous le statut Statut_.TRAITE
     */
    @Test
    void getActif() {

        // Récupère les évènements actif
        List<Evenement> evenementsList = evenementAccess.getActif();
        assertNotNull(evenementsList);
        assertNotEquals(0, evenementsList.size());

        // Récupère le statut avec le nom Statut_.TRAITE
        List<Statut> statutList = StatutAccess.getInstance().get(Statut_.TRAITE);
        assertNotNull(statutList);
        assertEquals(1, statutList.size());

        for (Evenement evenement : evenementsList) {
            assertEquals(evenement.getStatut(), statutList.get(0));
        }
    }
```

Figure 11 : Partie 4 des tests unitaires de la base de données

```java
/**
 * Test si on obtient le même nombre d'évènements actifs
 */
@Test
void getActif1() {

    // Récupère les évènements actif
    List<Evenement> evenementsList = evenementAccess.getActif();
    assertNotNull(evenementsList);
    assertNotEquals(0, evenementsList.size());

    // Récupère les évènements actif
    List<Evenement> evenementsListExpected = evenementAccess
            .get( nomRubriqueEnfant: null,  debut: null, Calendar.getInstance(), Statut_.TRAITE);

    assertNotNull(evenementsListExpected);
    assertNotEquals(0, evenementsListExpected.size());
    assertEquals(evenementsListExpected.size(), evenementsList.size());
}

/**
 * Test si l'on récupère seulement des évènement en attente
 */
@Test
void getEnAttente() {

    // Récupère les évènements en attente
    List<Evenement> evenementsList = evenementAccess.getEnAttente();
    assertNotNull(evenementsList);
    assertNotEquals(0, evenementsList.size());

    // Récupère le statut avec le nom Statut_.EN_ATTENTE
    List<Statut> statutList = StatutAccess.getInstance().get(Statut_.EN_ATTENTE);
    assertNotNull(statutList);
    assertEquals(1, statutList.size());

    for (Evenement evenement : evenementsList) {
        assertEquals(evenement.getStatut(), statutList.get(0));
    }
}
```

Figure 12 : Partie 5 des tests unitaires de la base de données

```java
/**
 * Test si la mise à jour d'un évènement s'est effectuée
 */
@Test
void update() {

    // Récupère les évènements en attente
    List<Evenement> evenementsList = evenementAccess.getEnAttente();
    assertNotNull(evenementsList);
    assertNotEquals(0, evenementsList.size());

    // Récupère le statut Statut_.TRAITE
    List<Statut> statutList = StatutAccess.getInstance().get(Statut_.TRAITE);
    assertNotNull(statutList);
    assertEquals(1, statutList.size());

    Evenement evenementBefore = evenementsList.get(0);
    evenementBefore.setStatut(statutList.get(0));
    evenementAccess.update(evenementBefore);

    // Récupère l'évènement après la mise à jour
    Evenement evenementAfter = databaseAccess.get(Evenement.class, evenementBefore.getIdEvenement());

    assertNotNull(evenementAfter);
    assertEquals(statutList.get(0), evenementAfter.getStatut());
}
```

Figure 13 : Partie 6 des tests unitaires de la base de données