

SMARTCITY

Rapport

Tano Iannetta

Loan Lassalle

Luana Martelli

Wojciech Myszkowski

Camilo Pineda Serna

Jérémie Zanone

Sous la direction du Professeur René Rentsch

PRO – mai 2017

Table des matières

Table des matières	3
1 Introduction.....	5
1.1 But	5
1.2 Objectifs.....	5
1.3 Description	5
1.4 Organisation	6
1.5 Répartition des tâches.....	6
2 Conception – Architecture	7
2.1 Systèmes d’exploitation et outils logiciels	7
2.2 Fonctionnalités principales.....	8
2.2.1 Cas d’utilisations.....	8
2.2.2 Gestion des requêtes effectuées à la base de données.....	9
2.2.3 Modération et filtrage des requêtes	9
2.2.4 Utilisation d’un calendrier	9
2.2.5 Classification des requêtes	9
2.2.6 Ajout d'évènements	9
2.2.7 Implémentation d'une carte interactive	9
2.2.8 Génération de documents PDF	10
2.3 Technologies utilisées.....	10
2.3.1 Java	10
2.3.2 Git – GitHub	11
2.3.3 Apache Maven.....	12
2.3.4 Carte interactive	13
2.3.5 Base de données.....	16
2.3.6 Génération de documents PDF	18
2.3.7 Interface graphique utilisateur (GUI)	19
3 Implémentation.....	20
3.1 Carte interactive	20
3.1.1 Tuile	20
3.1.2 Fournisseur de tuiles	21
3.1.3 Interface graphique utilisateur.....	23
3.2 Base de données.....	24
3.2.1 Modèle de données relationnel	26
3.2.2 Diagramme des classes.....	27

3.3	Génération de documents PDF	28
3.4	Interface graphique utilisateur (GUI)	29
4	Tests de l'application.....	33
4.1	Description générale de l'environnement de test	33
4.1.1	Matériel	33
4.1.2	Systèmes d'exploitation et outils logiciels	33
4.2	Tests unitaires	34
4.2.1	Carte interactive	34
4.2.2	Base de données.....	35
4.2.3	Génération de documents PDF	35
4.3	Tests d'intégrations	36
4.4	Tests des cas d'utilisation.....	36
5	Conclusion	37
5.1	État des lieux	37
5.2	Avis général	37
5.3	Problèmes rencontrés	38
5.4	Propositions d'améliorations	38
5.4.1	Interface graphique de l'utilisateur.....	38
5.4.2	Formulaire des évènements.....	39
5.4.3	Génération de documents PDF	39
6	Annexes	40
7	Sources – Bibliographies.....	40
7.1	Ouvrages.....	40
7.2	Sites Internet	40
7.3	Vidéos en ligne	40
8	Glossaire	41

1 Introduction

1.1 But

Aujourd'hui, la plupart des citoyens ont la possibilité de communiquer très rapidement avec les services privés ou publics. Cette proximité leur permet de partager leurs avis et conseils afin d'améliorer les services qui leur sont fournis par la ville de Lausanne.

Notre projet a consisté à l'élaboration d'une application nommée « SmartCity » qui assure l'archivage et la gestion de requêtes de différents acteurs de la société actuelle. Elle permet également l'affichage du lieu, d'une description correspondant aux requêtes et la production de documents PDF des données sélectionnées.

Cette réalisation s'est faite dans le cadre du module « Projet » de la Haute École d'Ingénierie et de Gestion du Canton de Vaud (HEIG-VD).

Ce programme permet :

- de gérer les requêtes venant des citoyens, de l'administration de la ville de Lausanne ou de « personnes de confiance »
- de localiser de manière claire et précise le lieu correspondant à une requête
- d'obtenir un résumé des différentes informations archivées

Dans le cadre de ce rapport, nous exposerons les technologies qui nous ont permis de réaliser l'application. Nous décrirons les différentes fonctionnalités implémentées afin de mieux comprendre l'utilité de notre application. Les problèmes rencontrés ainsi que les changements de conception seront également abordés afin d'expliquer nos choix quant au produit final.

Il est à noter que nous réalisons la partie administrative de l'application, c'est-à-dire que notre projet sera utilisé par la ville et non par les citoyens de la ville.

1.2 Objectifs

- Consulter et organiser les requêtes venant des différents acteurs identifiés (citoyens, administration de la ville de Lausanne, « personnes de confiance »).
- Afficher les lieux correspondants aux requêtes sur une carte interactive.
- Produire des documents PDF pour fournir un résumé des informations archivées.

1.3 Description

Nous avons décidé de créer une application permettant à l'administration d'une ville (dans notre cas Lausanne) de gérer des requêtes venant de ses concitoyens, des « personnes de confiance » ou de ses propres services. Les requêtes peuvent être des demandes d'amélioration de la ville, mais également des annonces d'évènements.

Dans notre application, tous les types des requêtes sont représentés par des évènements. Ces évènements peuvent être transmis par les citoyens au travers d'un site Internet ou d'une application mobile (non réalisés dans le cadre du projet). Des entreprises ou des personnes bénéficiant de la confiance de l'administration de la ville obtiennent le statut de « personnes de confiance », par exemple le TCS pourrait soumettre des avis d'accidents. Elles peuvent, sans l'acceptation de l'administrateur, ajouter des évènements à la base de données. Ce droit est aussi accordé à l'administrateur de l'application.

L'application s'articule autour de quatre actions principales.

La première action consiste en la consultation d'évènements recueillis au sein d'une base de données.

La deuxième action est caractérisée par la gestion des évènements stockés au sein de la base de données. Chaque évènement créé par un citoyen lambda doit être validé par l'administrateur. Ces évènements en attente sont signalés dans une zone dédiée sur l'application. Une fois validé, il est possible de consulter l'évènement au travers de différentes rubriques et de visualiser, au moyen d'une carte interactive, les lieux concernés. Pour une meilleure visibilité, chaque rubrique est associée à un filtre, rendant plus aisée la lecture sur la carte.

La carte interactive permet à l'administrateur de se déplacer et d'agrandir le plan de la ville au besoin. Il est possible d'affiner les critères d'affichage selon la sélection d'une date ainsi que des rubriques auxquelles appartiennent les évènements.

La dernière fonctionnalité de l'application est de générer des documents PDF contenant des informations relatives aux évènements appartenant aux rubriques et date que l'administrateur aura précédemment choisis. Cela permettra d'obtenir une trace écrite pour un meilleur partage d'informations au sein de l'administration de la ville de Lausanne.

1.4 Organisation

Chef de projet :

- Tano Iannetta : tano.iannetta@heig-vd.ch

Chef en second :

- Wojciech Myszkowski : wojciech.myszkowski@heig-vd.ch

Membres du projet:

- Loan Lassalle : loan.lassalle@cpnv.ch
- Luana Martelli : luana.martelli@heig-vd.ch
- Camilo Pineda : camilo.pinedaserna@heig-vd.ch
- Jérémie Zalone : jeremie.zalone@heig-vd.ch

1.5 Répartition des tâches

Phases/Membres	Tano Iannetta	Wojciech Myszkowski	Loan Lassalle	Luana Martelli	Camilo Pineda	Jérémie Zalone
Dossier final	15%	20%	20%	10%	30%	15%
Analyse	10%	10%	15%	15%	10%	10%
Conception	10%	10%	10%	15%	10%	15%
Réalisation	30%	20%	30%	30%	10%	30%
Administration	10%	10%	5%	5%	10%	5%
Tests	10%	20%	10%	10%	20%	15%
Documentation de code	15%	10%	10%	15%	10%	10%

2 Conception – Architecture

2.1 Systèmes d'exploitation et outils logiciels

Lors du choix des outils de conception, il était impératif de sélectionner des logiciels multiplateformes. Cette condition devait être remplie pour la bonne raison que les membres du projet ne possèdent pas le même système d'exploitation. La plupart des outils utilisés possèdent une licence qui a été contractée personnellement ou au travers des programmes de licences fournis par l'HEIG-VD.

- Microsoft Windows 10 Familiale 64 bits
- Microsoft Windows 10 Education 64 bits
- GNU/Linux Linux Mint 64 bits
- IntelliJ IDEA Ultimate 64 bits
- Java SE Development Kit 8
- Java SE Runtime Environment 8
- Eclipse Neon 2 64 bits
- Windows Builder 4.6
- Microsoft Office Professional Plus 2016 64 bits
- MySQL Workbench 6.3
- Inkscape 0.48.5 64 bits
- StarUML 2.8.0

2.2 Fonctionnalités principales

2.2.1 Cas d'utilisations

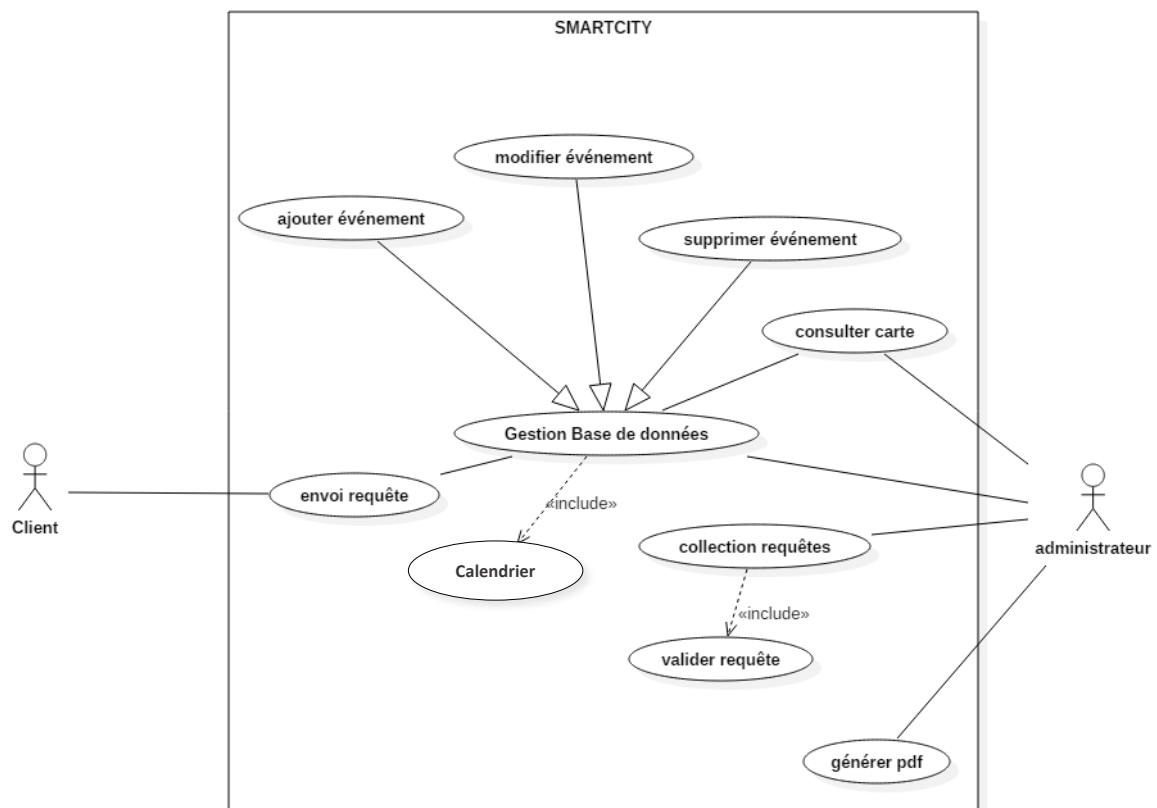


Figure 1 : Diagramme des cas d'utilisations

L'administrateur est la personne qui utilisera l'appliquatif. Cette personne a la possibilité d'effectuer plusieurs actions.

Elle peut tout d'abord valider des requêtes qui figurent dans la base de données. L'administrateur peut rajouter des événements dans la base de données et modifier ceux qui y figurent. Chaque événement a une date de début et de fin, ce qui peut servir lors du filtrage d'affichage. Une fois les requêtes validées, il est possible de les consulter sur la carte interactive.

Finalement, l'administrateur peut générer des documents PDF qui contiennent les informations selon les rubriques d'événements sélectionnées.

Dans notre cas, le client ne peut pas envoyer de requêtes, la base de données contient un certain nombre d'événements en attente de validation pour simuler ce cas d'utilisation.

2.2.2 Gestion des requêtes effectuées à la base de données

Différents types de requêtes sont exécutés par la base de données. L'ajout, la modification et la suppression d'événements sont des requêtes produites par l'administrateur. Il est le garant de l'intégrité de la base de données et le gérant des requêtes en attente de validation soumises par les citoyens.

2.2.3 Modération et filtrage des requêtes

Lorsqu'un citoyen fait une proposition d'événement, l'administrateur peut choisir de la valider ou de la refuser. Une fois validé, l'événement sera visible sur la carte interactive suivant ses dates de début et de fin. Si toutefois, l'événement est refusé, la base de données lui affecte un statut précis. Il ne sera plus visible au sein de l'application.

L'administrateur s'occupe de filtrer les requêtes faites par les utilisateurs. Pour certains comptes privilégiés, il n'y aura pas besoin de l'intervention de l'administrateur.

2.2.4 Utilisation d'un calendrier

Chaque requête comporte une date de début et une date de fin. Un calendrier sur l'application permet de filtrer l'affichage des événements qui sont actifs à cette date. Le calendrier permet aussi de simplifier les ajouts et modifications des dates spécifiques à un événement dans la fenêtre de modification.

2.2.5 Classification des requêtes

Les requêtes sont classées selon les rubriques d'événements suivantes :

- Accidents
- Travaux
- Constructions
- Renovations
- Manifestations
- Doléances

2.2.6 Ajout d'événements

Il n'y a pas que les citoyens qui peuvent proposer des requêtes ou annoncer des événements. L'administrateur peut également en ajouter directement depuis l'application.

2.2.7 Implémentation d'une carte interactive

L'application comporte une carte avec laquelle il est possible d'interagir. Cette carte référence, à l'aide de « pins », la localisation des événements dans la ville de Lausanne. Ces événements sont filtrés par leur date de début, leur date de fin et leur rubrique.

2.2.8 Génération de documents PDF

La génération de documents PDF est réalisée selon une rubrique d'évènement choisie.

Ces documents PDF sont destinés aux départements correspondant aux rubriques. Ils permettent de garder un historique détaillé des évènements au sein de la ville. On peut imaginer qu'une fois générés, ils sont envoyés aux personnes concernées.

2.3 Technologies utilisées

2.3.1 Java

Java est un langage de programmation informatique orienté objet, développé par Sun Microsystems. Nous avons décidé d'utiliser ce langage pour différentes raisons.

La première est que nous l'avons appris récemment en cours. Ce langage était encore frais dans nos esprits et son usage a l'avantage d'imposer une manière de coder claire, mais non pas laborieuse. De plus, Javadoc permet de réaliser une documentation claire et complète de codes.

Java permet aussi d'avoir un très haut niveau d'abstraction à la machine. Sa portabilité contribue à l'exécution de programmes sur tout type de machine. Ainsi, elle offre une plus grande liberté aux utilisateurs.

En ce qui concerne la sécurité, Java est considéré comme un langage fiable, stable, dû à l'utilisation d'une machine virtuelle Java. Un programme ne peut en aucun cas compromettre l'intégrité de cette dernière.

Par ailleurs, ce langage permet l'exécution « simultanée » de plusieurs tâches pour obtenir de meilleures performances. Lors de l'utilisation d'interfaces graphiques, il est indispensable d'utiliser ce mécanisme.

Pour terminer, les deux principales raisons de notre choix ont été, la disponibilité d'un grand nombre de bibliothèques tierces et les améliorations constantes apportées à Java.

2.3.2 Git – GitHub

Pour tout projet, il est primordial d'utiliser un système de versions de fichiers pour récupérer ou archiver des documents. Ceci permet de suivre l'évolution des fichiers d'un projet, d'administrer les versions successives de plusieurs documents et de maintenir à jour une copie de ces fichiers à différents endroits ou avec les différents membres du projet.

Dans ce projet, nous avons utilisé Git pour sa rapidité, sa taille extrêmement réduite, sa gestion des branches et sa décentralisation. Multiplateforme, il a permis à chacun de travailler dans son environnement préféré.

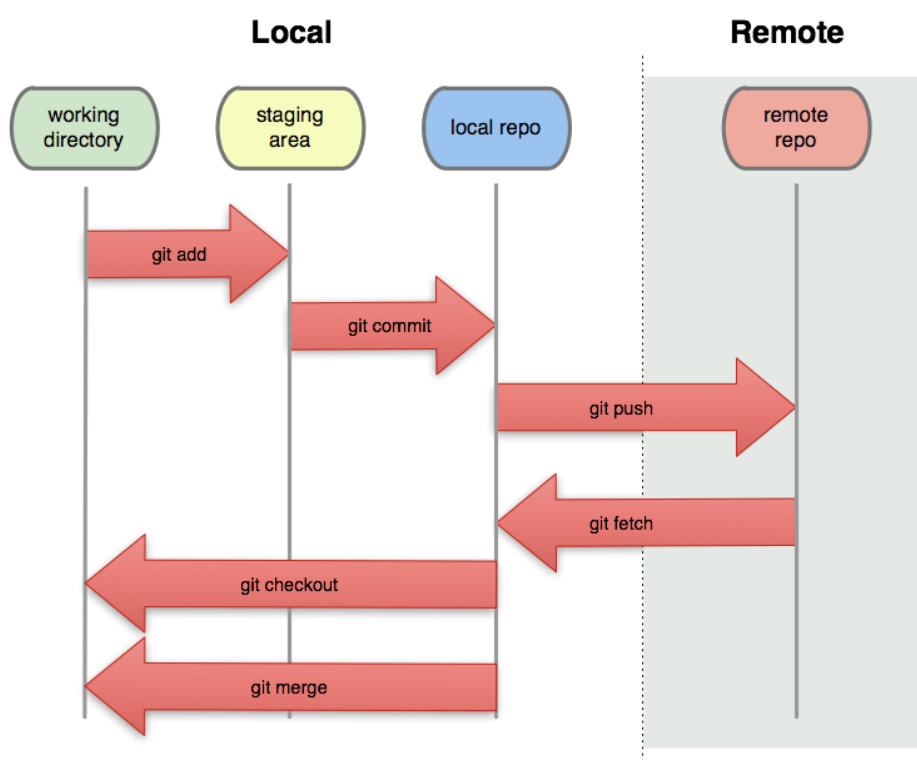


Figure 2 : Les différentes « zones » de Git et les commandes correspondantes

En complément, nous avons utilisé la plateforme GitHub pour héberger et gérer le développement de logiciels. Elle nous a permis de mettre facilement et efficacement le travail de chacun en commun, d'obtenir une vue globale de l'évolution du projet et d'avoir un échange d'informations plus rapide et à intervalle régulier.

L'atout majeur de GitHub est la possibilité d'accéder à une ressource en même temps qu'une autre personne. Lorsque deux personnes ou plus travaillent sur la même ressource en même temps et décident d'envoyer au serveur distant, les modifications effectuées vont lever un conflit lors de la fusion des fichiers du projet. À ce moment, il faudra décider quelles sont les parties à garder de celles qui ont été modifiées.

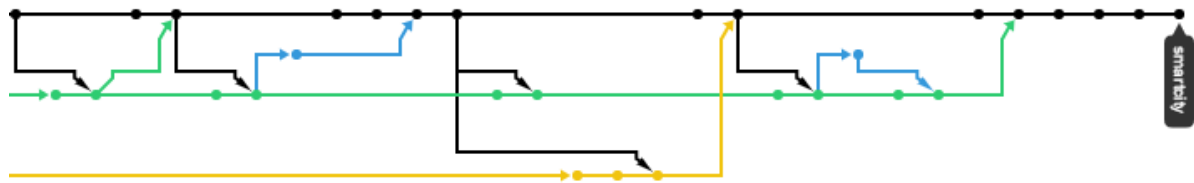


Figure 3 : Les branches du projet par rapport à une date de départ

2.3.3 Apache Maven

Nous avons utilisé Maven pour la conception architecturale de notre projet.

Maven est un outil pour la gestion et l'automatisation de production Java en général et Java EE en particulier. L'objectif est de produire un logiciel à partir des sources du programme, en optimisant les tâches réalisées et en garantissant le bon ordre de fabrication.

Maven utilise un paradigme connu sous le nom de Project Object Model (POM) qui décrit un projet logiciel, ses dépendances avec des modules externes et l'ordre à suivre pour sa production. Il est livré avec un grand nombre de tâches prédéfinies, comme la compilation de code Java ou encore la modularisation. Nous y trouvons entre autres :

- Le nom du projet
- Le numéro de version
- Les dossiers contenant le code source et les fichiers de ressources
- Les bibliothèques nécessaires à la compilation
- Les dépendances vers d'autres projets
- Les noms des contributeurs

Maven 2 se concentre sur le principe de favoriser l'utilisation des conventions plutôt que de configurer son projet. Ainsi, si l'on respecte certaines conventions définies par Maven 2, il devient inutile de préciser certaines informations dans son pom.xml. Par exemple, Maven 2 préconise l'utilisation du répertoire src/main/java pour stocker les fichiers sources du projet. En respectant ceci, il devient alors inutile de spécifier à Maven 2 où se trouvent les sources Java, ce qui allège d'autant l'écriture du pom.xml.

Avec ce simple pom.xml, il devient possible de réaliser tout le processus de construction d'un projet avec Maven 2, pour :

- Gérer les fichiers de ressources
- Compiler les sources Java
- Compiler et exécuter les tests unitaires
- Créer le fichier JAR du projet
- Déployer le JAR

2.3.4 Carte interactive

Dans notre projet, nous avons besoin d'une carte interactive qui permet de visualiser avec précision où ont lieu les événements. Cependant, cette carte n'utilise pas l'adresse des événements comme référence.

Pour représenter un point géographique, nous avons utilisé deux systèmes de coordonnées :

- Un système de coordonnées sphériques nommé WGS84, qui utilise la latitude et la longitude. Ce système est employé par défaut sur la majorité des GPS.
- Un système de coordonnées cartésiennes, utilisé par les principaux outils de géolocalisation en ligne comme OpenStreetMap.

2.3.4.1 Système WGS84

Système géodésique mondial le plus courant, il permet de géolocaliser des positions sur un plan ou une carte à partir de ses coordonnées géographiques. Nous avons décidé de l'utiliser pour localiser les différents événements sur une carte.

Chaque localisation a une représentation tridimensionnelle :

- **La longitude** et **la latitude** sont des coordonnées géographiques représentées par des valeurs angulaires, expression du positionnement est-ouest, respectivement nord-sud d'un point sur Terre.
- **L'altitude** est l'élévation verticale d'un lieu ou d'un objet par rapport à un niveau de base.

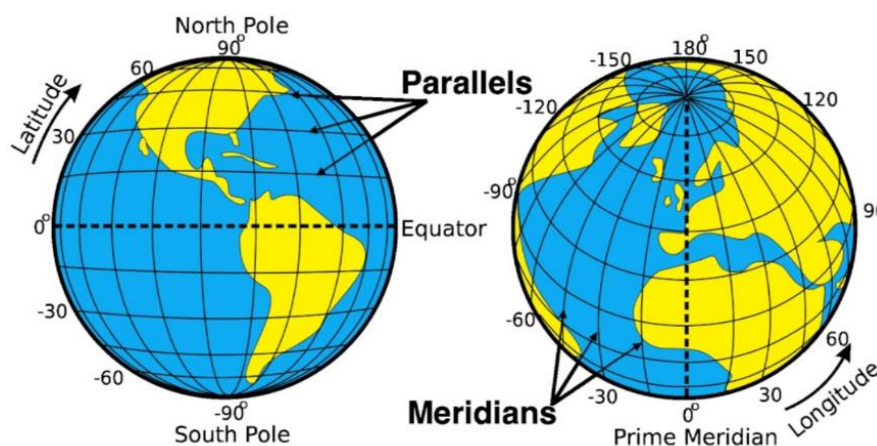


Figure 4 : Représentation du système WGS84

Le méridien de référence est celui de Greenwich. La longitude est comprise entre -180° et $+180^\circ$: les valeurs positives se trouvent à l'est et les valeurs négatives à l'ouest. La latitude se situe entre -90° et $+90^\circ$, les valeurs positives se trouvant au nord et les valeurs négatives au sud de l'équateur.

Dans ce projet, nous n'avons pas tenu compte de l'altitude puisque c'est une donnée qui ne nous était pas utile. Seules nous importaient les informations concernant la position au sol.

Ce système comporte plusieurs représentations des coordonnées. Le système le plus connu étant les degrés minutes et secondes (DMS). L'API de Google Maps et d'OpenStreetMap utilise notamment la représentation en degrés décimaux (DD). Nous l'avons choisi pour des questions de simplification.

2.3.4.2 Système cartésien

Le système cartésien dans le monde cartographique permet de localiser un endroit sur une image d'une carte du monde, sous la forme d'une projection de Mercator qui projette les pôles à l'infini. Cette image est définie par une échelle ou zoom grâce aux axes x et y dont l'unité est le pixel. Si l'échelle change alors les coordonnées également.

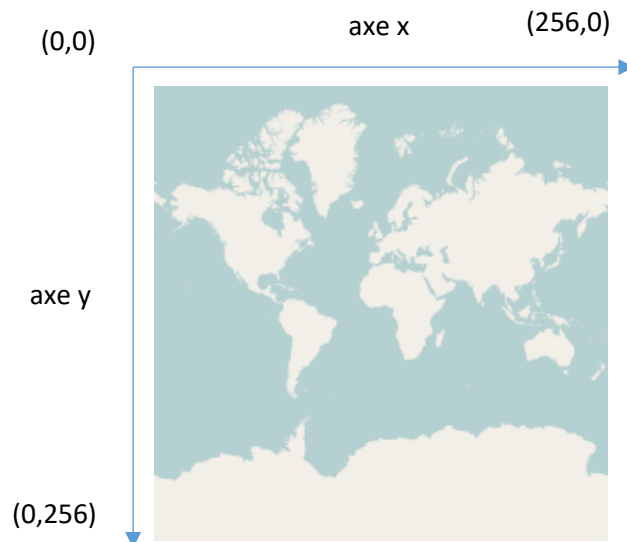


Figure 5 : Exemple d'un système cartésien au niveau de zoom 0

Le zoom 0 correspond au zoom le plus éloigné et nous permet de voir la carte en entier, dans une image carrée de 256 pixels de côté. À l'inverse, le zoom 20 permet d'obtenir une image détaillée et centrer sur une zone très précise. Cependant, les dimensions de l'image augmentent, puisqu'à chaque niveau de zoom la carte est deux fois plus grande que l'image précédente.

2.3.4.3 Conversion du système WGS84 en système cartésien

Pour passer d'un système tridimensionnel à un système bidimensionnel, nous utilisons une projection cartographique. Il existe une variété de projections qui permettent de passer de l'un à l'autre. Nous avons utilisé la projection de Mercator, utilisée par les services cartographiques tels que Google Maps et OpenStreetMap. Cette projection désigne la terre comme une sphère parfaite ce qui n'est pas le cas dans la réalité. Elle fait correspondre le point de coordonnées WGS84 ($0^{\circ}, 0^{\circ}$) avec le centre de la carte.

2.3.4.4 Carte en tuiles

Dans ce projet, nous utilisons les services cartographiques proposés par OpenStreetMap qui met à disposition des serveurs fournissant des images carrées, appelées tuiles de 256 pixels (cf. Système cartésien). Elles représentent des parties de la carte du monde à un zoom défini.

Ce petit format est très utile, car il demande peu de ressources pour son utilisation. Si le format des fichiers était plus conséquent, le transfert via le réseau Internet serait limité par la bande passante. De plus, ce format contient la quantité d'information nécessaire à nos besoins. Les tuiles de petite taille sont plus pertinentes pour notre utilisation. Mises côte à côte, elles forment l'image de la section désirée.

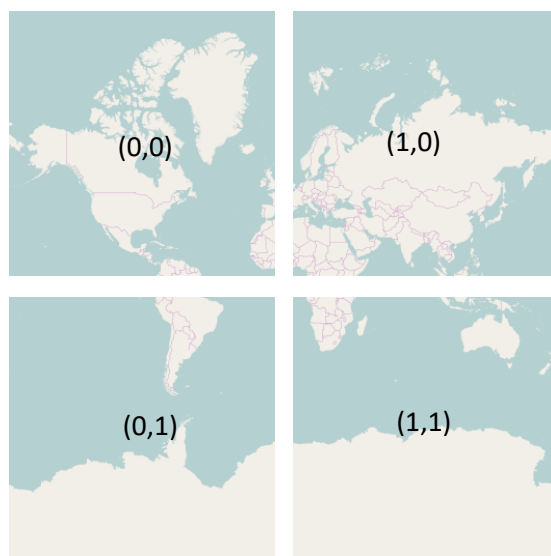


Figure 6 : Utilisation des tuiles de zoom de niveau 1

Les tuiles sont caractérisées par trois paramètres : le niveau de zoom et les coordonnées x et y. L'accès au serveur s'effectue à l'aide de l'URL : <http://a.tile.openstreetmap.org/17/67960/46219.png>. Les trois derniers numéros de l'adresse Internet représentent les paramètres cités précédemment.



Figure 7 : Tuile représentant l'HEIG-VD sur le site de Cheseaux au niveau de zoom 17

L'utilisation de ces images est gratuite selon des conditions à respecter. Dans le cadre de notre projet, qui n'est pas destiné à la commercialisation, l'utilisation des images est autorisée.

2.3.5 Base de données

L'utilisation d'une base de données est indispensable pour l'enregistrement, l'organisation et le tri d'une importante quantité de données. Bien structurée et indexée, une base de données doit répondre aux principales formes normales existantes. La normalisation des modèles de données permet ainsi de vérifier la robustesse de sa conception pour améliorer la modélisation et faciliter la mémorisation des données en évitant, par exemple, les problèmes de redondances, de mise à jour ou de cohérence.

L'accès à une base de données peut être entrepris à travers le réseau ou directement sur la machine hôte. L'application SmartCity accèdera à la base de données située en local. La création et l'utilisation d'une base de données au sein du langage Java seront facilitées par l'utilisation du framework Hibernate.

Il sera nécessaire à l'utilisateur d'installer un service tiers tel que MySQL pour fournir un gestionnaire de bases de données relationnelles. Pour ce qui est de la structure de la base de données, nous avons décidé d'utiliser la technologie de MySQL, car nous étions habitués à l'utiliser.

2.3.5.1 Modèle de données entité-association

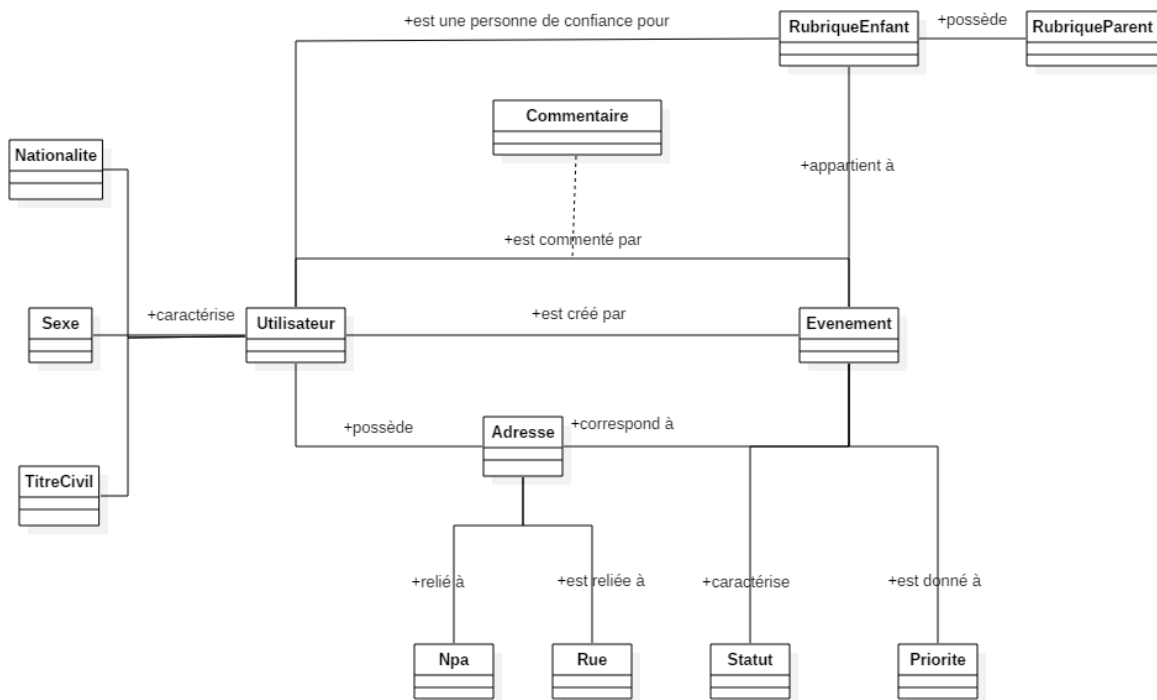


Figure 8 : Modèle de données entité-association

2.3.5.2 MySQL

MySQL est un système de gestion de bases de données relationnelles (SGBDR) qui nous a permis de créer et de mettre en place la base de données. Pour plus de simplicité, le logiciel MySQL Workbench a été utilisé.

Pour pouvoir utiliser l'application, il a fallu insérer des données. Celles-ci bien que fictives ont été insérées à travers un script. Cela permet de proposer une application prête à l'emploi.

2.3.5.3 Hibernate

Pour faciliter la gestion des données au sein de l'application, nous avons utilisé le concept d'objet persistant, permettant de stocker des objets Java en toute simplicité. Pour ceci, nous avons utilisé le framework Hibernate.

Ce dernier apporte le moyen de traduire les objets Java en données stockables dans la base. Il a fallu créer toutes les tables de la base de données en tant que classe Java et définir des fichiers de mapping pour que Hibernate fasse correspondre le code Java et la structure de la base de données.



Figure 9 : Logo du framework Hibernate

2.3.6 Génération de documents PDF

Les évènements sont stockés dans la base de données. Cependant, il est utile de pouvoir retrouver ces informations sous une autre forme. Nous avons décidé de les archiver dans des documents PDF. Ainsi, les utilisateurs de l'application retrouvent plus facilement les informations souhaitées.

De plus, chaque document contient des statistiques qui permettent d'obtenir une représentation visuelle des données.

Un PDF sera organisé en deux parties :

- La première partie sera commune à tous les types de rubriques et contiendra les informations principales (nom de la rubrique, nom de l'évènement, lieu, date, priorité et texte détaillant l'évènement)
- La deuxième partie sera présentée sous la forme de statistiques. Elles seront personnalisées selon la rubrique. Par exemple, s'il s'agit d'un évènement relatif à des travaux, on affichera le temps moyen de la durée des chantiers. S'il s'agit en revanche de doléances, on affichera plutôt le nombre de commentaires qui ont été publiés à ce sujet, et ainsi de suite pour les autres rubriques.

Si plusieurs filtres ont été sélectionnés, alors une option sera de générer automatiquement plusieurs PDF.

Pour générer le document PDF, nous avons choisi d'utiliser le kit de développement iText version 7. Ce support offre plusieurs librairies qui permettent de créer facilement des documents PDF et laisse un grand degré de liberté sur la gestion de la mise en page.



Figure 10 : Logo du kit de développement iText

Pour la partie graphique, nous avons utilisé JfreeChart. Ce kit permet de créer toutes sortes de graphiques. Dans le cadre du projet, nous avons choisi deux graphes en particulier, le diagramme en barre et le diagramme circulaire.



Figure 11 : Logo du kit de développement JFreeChart

2.3.7 Interface graphique utilisateur (GUI)

Pour l'interface graphique, nous avons décidé d'utiliser la technologie Swing.

Nous avons déjà travaillé avec cette librairie par le passé lors du semestre précédent. Comme nous avons peu travaillé avec des interfaces graphiques, nous avons préféré utiliser cette librairie connue, car notre application nécessite une interface graphique conséquente.

De plus, Swing propose tous les composants nécessaires à la réalisation de notre projet. Les composants principaux étant des panels, des boutons, des champs de saisie, des boîtes de sélection et un calendrier.

En revanche, le design de Swing n'est pas des plus épurés, mais nous avons ciblé les fonctionnalités plutôt que l'esthétique pour ce projet.

Swing impose une syntaxe de code lourde et sa relecture en est plus laborieuse qu'à l'accoutumée. Cependant, nous avons fait en sorte de séparer les éléments graphiques du reste afin d'avoir un code structuré.

En évaluant ses points positifs et négatifs, nous avons donc fait le choix d'utiliser cette librairie pour implémenter l'interface graphique de notre application.

3 Implémentation

Deux choix importants ont été faits lors de l'implémentation de l'application. Le premier concerne l'affichage de message en différentes langues dû au plurilinguisme de la Suisse.

Étant donné qu'au niveau fédéral les langues officielles sont l'allemand, le français et l'italien. Nous avons souhaité mettre en place la gestion de messages en plusieurs langues afin de faciliter l'utilisation de notre application dans d'autres villes que Lausanne. Pour ce faire, nous avons utilisé des fichiers `ResourceBundle.properties`. Ces fichiers permettent de fournir plusieurs langues pour les messages à afficher. A l'heure actuelle, nous avons implémenté uniquement le français, mais il est très simple d'ajouter de nouvelles langues en ajoutant le fichier `MessageBundle` correspondant.

Le second fut porté sur le suivi des erreurs survenues au sein de notre application. Nous avons décidé de mettre en place un système de journalisation qui permet de suivre en temps réel ou après coup les différentes erreurs survenues. Ces fichiers de log sont enregistrés dans le dossier `SmartCity/Logs`.

3.1 Carte interactive

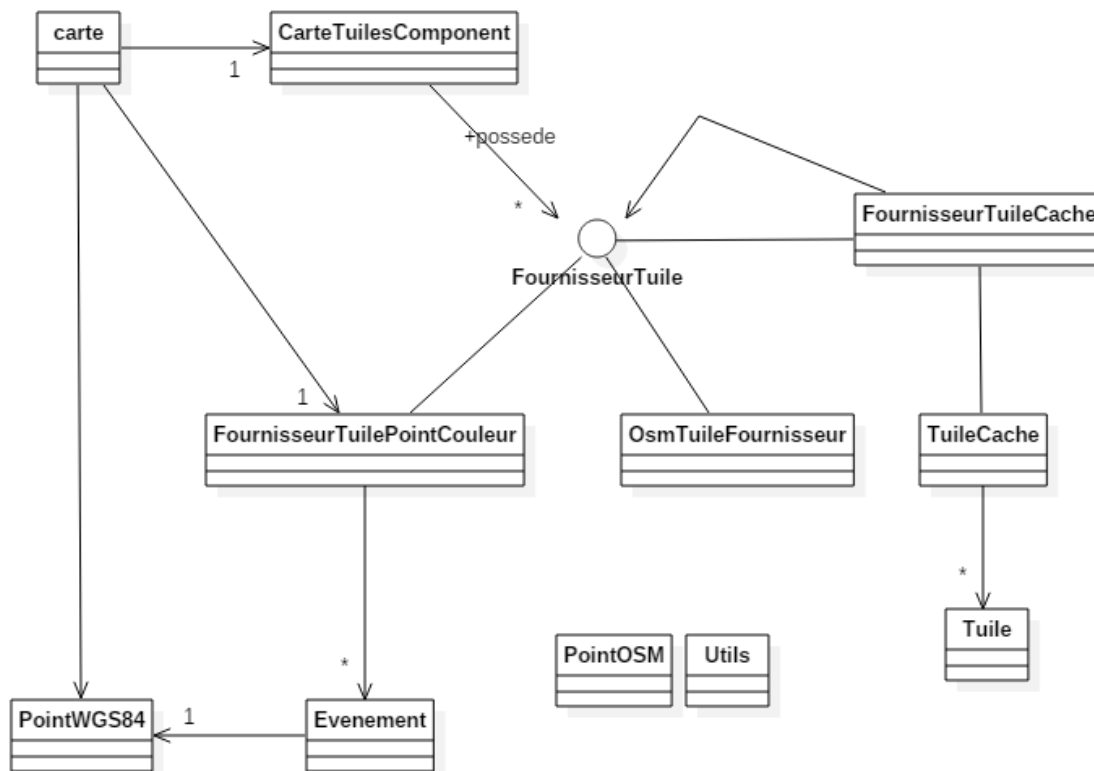


Figure 12 : Schéma de la structure des classes utilisée pour construire la carte interactive

3.1.1 Tuile

Les tuiles sont des images représentées par des instances de la classe `BufferedImage` de la bibliothèque Java. Les attributs de cette classe sont les paramètres de la tuile soit le niveau de zoom et les coordonnées x et y (cf. Carte en tuiles).

3.1.2 Fournisseur de tuiles

Un fournisseur de tuile fournit une tuile selon ses coordonnées et son niveau de zoom. Ils implémentent l'interface FournisseurTuile qui possède une seule méthode qui permet de fournir une tuile selon les paramètres donnés (zoom et coordonnées).

3.1.2.1 OSM

Ce fournisseur obtient les tuiles à partir d'un serveur distant appartenant à OpenStreetMap. Son attribut est l'URL de base du serveur (cf. Carte en tuiles). L'image est récupérée grâce à une instance de la classe URL et la méthode statique read de la classe ImageIO. Un objet Tuile est alors créé. En cas d'erreur de connexion ou de faiblesse du serveur, une tuile alternative est affichée.



Figure 13 : Tuile d'erreur

3.1.2.2 Point couleur

Le but principal de la carte est de pouvoir afficher les différents évènements en cours aux bons endroits.

Les tuiles sont des images, il est donc aisé d'en faire des modifications. Cependant, le procédé de modification est lourd, car à chaque changement du filtre d'évènements il faut reconstruire des tuiles OSM entièrement.

À ces fins, nous avons créé un autre fournisseur de tuiles (`fournisseurTuilePointCouleur`) qui possède en attribut une liste des évènements récupérés à partir de la base de données. Donc, lors d'un accès à une tuile, une nouvelle image est créée avec le nom et l'ID de l'évènement : ils apparaissent directement sur l'emplacement de celui-ci.

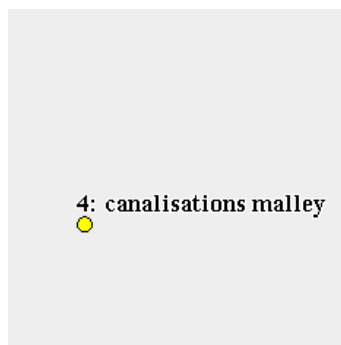


Figure 14 : Exemple d'une tuile fournie depuis le fournisseur `TuilePointCouleur`

Par la suite, les tuiles marquées de points de couleur vont se superposer aux tuiles de la carte, grâce à un fond transparent. Lors d'un changement d'affichage, seules ces tuiles sont modifiées.

3.1.2.3 Cache

Le fournisseur de tuile en cache est un transformateur qui enveloppe un fournisseur de tuile standard. Tous les nouveaux appels à une tuile sont stockés dans un objet `TuileCache` qui est une table associative de tuiles, `LinkedHashMap`.

La clé pour retrouver la tuile est un objet de type `Long` et construit à partir des coordonnées de la tuile. La classe `LinkedHashMap` offre la méthode `removeEldestEntry` qui permet de fixer une taille maximum de donnée, et une fois atteinte, s'assure que les plus anciennes sont effacées.

Si la tuile s'y trouve déjà, il n'y a pas besoin de la reconstruire. Un simple appel à la table associative est suffisant. Ce procédé n'améliore pas la rapidité lors du premier chargement, mais apporte de la fluidité une fois que les tuiles ont été chargées dans le cache.

3.1.3 Interface graphique utilisateur

La classe `CarteTuilesComponent` (sous-classe de `JComponent`) est un composant Swing capable d'afficher une carte en tuiles fournies par un ou plusieurs fournisseurs de tuiles. Elle possède deux attributs, le niveau de zoom actuel de la carte et la liste des fournisseurs de tuiles.

Le zoom et la liste des fournisseurs de tuiles sont fournis lors de la construction d'un objet `CarteTuilesComponent`.

Le niveau de zoom, compris entre 12 et 19 inclus, est modifiable par la suite lors d'une interaction avec la molette de la souris. Tout comme le zoom, la liste des fournisseurs de tuiles est modifiable grâce à la méthode `setZoom` respectivement `ajoutFournisseurTuile`.

Pour garder le même comportement d'utilisation du composant graphique `JComponent`, plusieurs méthodes de cette classe doivent être redéfinies.

Premièrement, `getPreferredSize` retourne la dimension du composant. Dans notre implémentation, cette taille est celle de la carte du monde au niveau du zoom courant.

Puis `paintComponent`, est appelée à chaque fois que le composant doit être affiché. C'est elle qui sera chargée de dessiner les tuiles des différents fournisseurs les unes par-dessus les autres, dans l'ordre des fournisseurs. Le premier fournisseur transmet le fond de la carte avec des tuiles opaques, tandis que le fournisseur de points couleur superpose des tuiles transparentes.

Le système de coordonnées de `CarteTuilesComponent` est le système de coordonnées cartésien au niveau de zoom courant, ce qui peut vite devenir gigantesque selon le zoom utilisé pixels (cf. Système cartésien). Cependant la taille du composant n'est pas un problème puisque les parties invisibles ne sont jamais dessinées et aucune mémoire ne leur est allouée.

La partie visible du composant est obtenue à l'aide la méthode `getVisibleRect`, qui retourne le rectangle visible du composant dans son propre système de coordonnées. C'est dans ce rectangle que l'on va déterminer l'ensemble des tuiles à dessiner.

Lors d'un changement de niveau de zoom ou lorsque la liste des fournisseurs de tuiles est mise à jour, il faut redessiner le composant. Un appel à la méthode `repaint` est alors effectué.

Concernant l'interaction du déplacement de la carte avec la souris, elle est gérée grâce à un objet `JViewport` positionné sur une partie de la zone dessinée par `CarteTuilesComponent`. La position de cette vue bouge et suit les mouvements de la souris.

3.2 Base de données

En premier lieu, nous avons travaillé sur l'implémentation de la base de données. Puis, nous avons implémenté les fichiers de mapping pour que Hibernate puisse connaître la structure de la base de données. Des classes représentant les tables de la base de données ont été créées pour utiliser le code Java afin d'élaborer des requêtes vers la base de données.

Avec l'utilisation du framework Hibernate, la mise en place de requête à effectuer au niveau de la base de données a été simplifiée et accélérée. Nous avons décidé de mettre en place un système de transaction permettant de ne pas altérer la base de données si une requête n'aboutissait pas. N'importe quelle requête renvoie à une liste d'objets. Lors d'une erreur, cette liste n'existe pas (valeur à null), s'il n'y a pas d'objet qui correspond à la requête effectuée la liste est vide.

Une requête est définie par des critères de sélection qui permettent à la base de données de récupérer les enregistrements correspondants.

Prenons l'exemple d'une requête de sélection d'un statut.

Pour exécuter cette requête, il est indispensable d'ouvrir ou de récupérer la session de travail correspondant à la liaison avec la base de données. Puis, pour pallier les erreurs qui surviendraient lors de l'exécution, une transaction est ouverte.

Dans le cas de notre application, un système de notification des événements existe. Dû à cela, plusieurs threads peuvent accéder simultanément à cette session de travail. Il est donc indispensable d'en protéger l'accès.

```
try {  
    Session session;  
  
    // Démarre une transaction pour la gestion d'erreur  
    synchronized (session = hibernate.getSession()) {  
        transaction = session.beginTransaction();
```

Figure 15 : Ouverture d'une session de travail et d'une transaction

Ensuite, il faut définir la table de la base de données ciblée en utilisant la classe correspondante. Pour définir des critères de sélection à la requête, il est nécessaire de créer des prédicats. Ces prédicats sont construits à partir d'un opérateur de comparaison (égal, inférieur, supérieur, etc.), de l'attribut de la table et du critère de sélection fourni par l'utilisateur.

Pour que Hibernate comprenne quel attribut de la table est utilisé lors de la comparaison, il est obligatoire de créer des Metamodels (cf. Figure 16). Ils permettent de connaître le type de l'attribut et de forcer l'utilisateur à fournir un paramètre de type semblable pour que la comparaison ait lieu.

Pour une utilisation standard des fonctions définissant les requêtes, nous avons décidé de gérer les valeurs atypiques des critères de sélection. Si la valeur de celui est null ou vide (pour une chaîne de caractères) alors la requête ne possèdera aucun critère de sélection. Elle récupérera tous les enregistrements de la table ciblée.

```
// Définit des critères de sélection pour la requête
CriteriaBuilder criteriaBuilder = hibernate.getCriteriaBuilder();
CriteriaQuery<Statut> criteriaQuery = criteriaBuilder
    .createQuery(Statut.class);
Root<Statut> statutRoot = criteriaQuery.from(Statut.class);
List<Predicate> predicateList = new ArrayList<>();

// Définit seulement les critères de sélection pour la requête des paramètres non null
// et non vide
if (nomStatut != null && !nomStatut.isEmpty()) {
    predicateList.add(criteriaBuilder.equal(statutRoot.get(
        Statut_.nomStatut),
        nomStatut.toLowerCase()));
}
}
```

Metamodel →

Figure 16 : Définition des critères de sélection de la requête

Une fois les critères de sélection définis et transmis à la requête, elle est exécutée. Si tout s'est bien passé, la transaction sera validée et la fonction retournera la liste des enregistrements correspondant aux critères. Sinon une exception sera levée.

```
criteriaQuery.where(predicateList.toArray(new Predicate[predicateList.size()]));
statutList = hibernate.createQuery(criteriaQuery).getResultList();

transaction.commit();
}
} catch (Exception e) {
    databaseAccess.rollback(e, transaction);
}
```

Figure 17 : Exécution de la requête et validation de la transaction

Il est possible d'ajouter des enregistrements à la base de données en créant de nouveaux objets Java et en les transmettant à la base de données. Il faut savoir que chaque nouvel objet créé est un nouvel enregistrement pour la base de données.

Ainsi, pour utiliser un objet contenu dans la base de données, il faut d'abord le récupérer. Pour sa suppression au sein de la base de données, il faut être vigilant aux références de cet objet, car il peut être contenu dans d'autres enregistrements. Si c'est le cas, la suppression pourra avoir lieu selon les règles définies dans la base de données.

Dans notre projet, il n'est pas possible de supprimer des événements, car le référencement de ceux-ci ne le permet pas. Ils figurent dans la table des commentaires. En fait, l'action de suppression était tout à fait possible, mais nous avons décidé de ne pas permettre la suppression de commentaires, car nous voulions garder la traçabilité de tous les événements.

Pour ce qui concerne la mise à jour des objets, Hibernate analyse tous les attributs de l'objet pour détecter la moindre différence entre l'enregistrement et l'objet.

3.2.1 Modèle de données relationnel

Nous avons pris le parti de mettre en place des triggers et des vues pour apporter une première vérification des données lors de l'enregistrement de celles-ci. Ceci a été pensé dans l'idée de réutiliser la base de données dans une autre application.

Des tables ont été créées pour stocker différents éléments tels que la nationalité, le sexe, l'état civil afin que chaque objet puisse avoir les mêmes valeurs. Dans cette même optique, l'adresse a été décomposée en 3 tables (Rue, NPA, Adresse).

Pour une meilleure homogénéité des données, nous avons décidé de mettre toutes les chaînes de caractères en minuscule. Lors d'insertion ou de modification d'enregistrements, notre application se charge de faire la conversion.

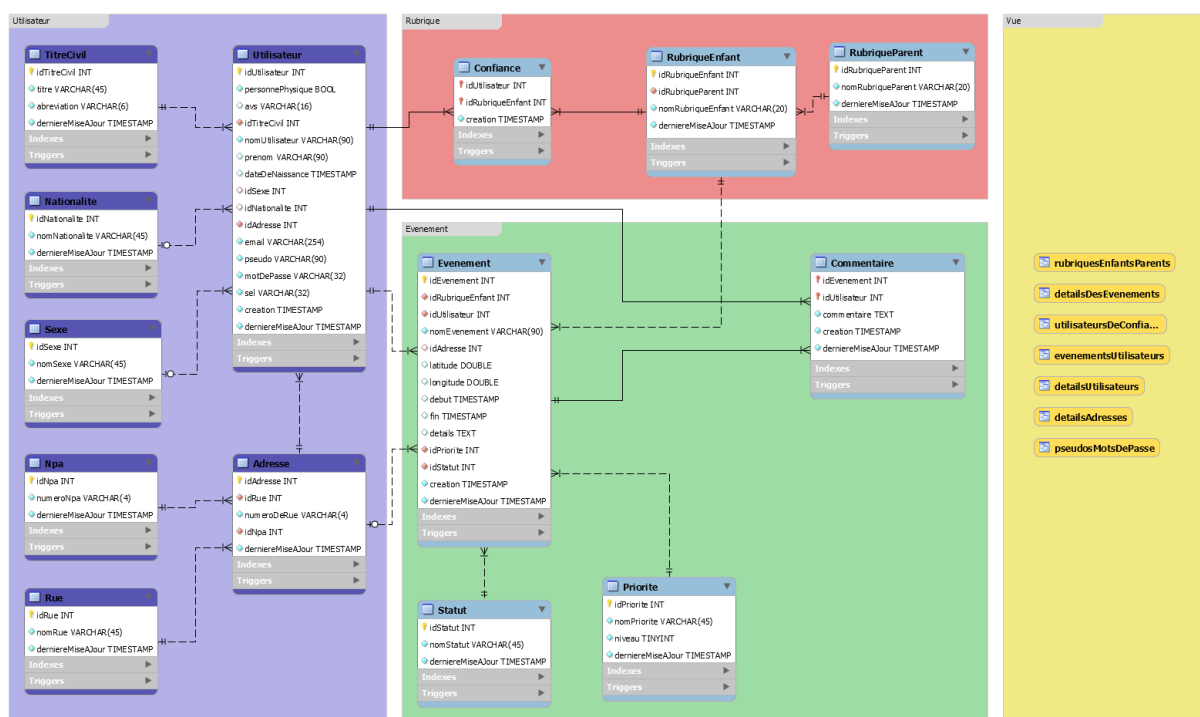


Figure 18 : Modèle de données relationnel

3.2.2 Diagramme des classes

De nombreuses classes et fonctions ont été implémentées dans le but de proposer une application complète et apte à évoluer sans difficulté.

La structure des classes permettant la création des requêtes a été factorisée au maximum. Une classe a été créée pour chacune des tables afin d'effectuer des requêtes selon un type d'objet. La plupart des fonctions de ces classes appellent des fonctions de la classe DataBaseAccess. Ceci est dû à sa généralité. La récupération d'un élément suivant son type a été implémentée différemment selon chaque classe. Cela s'avérerait indispensable pour créer des requêtes correspondant aux critères des objets.

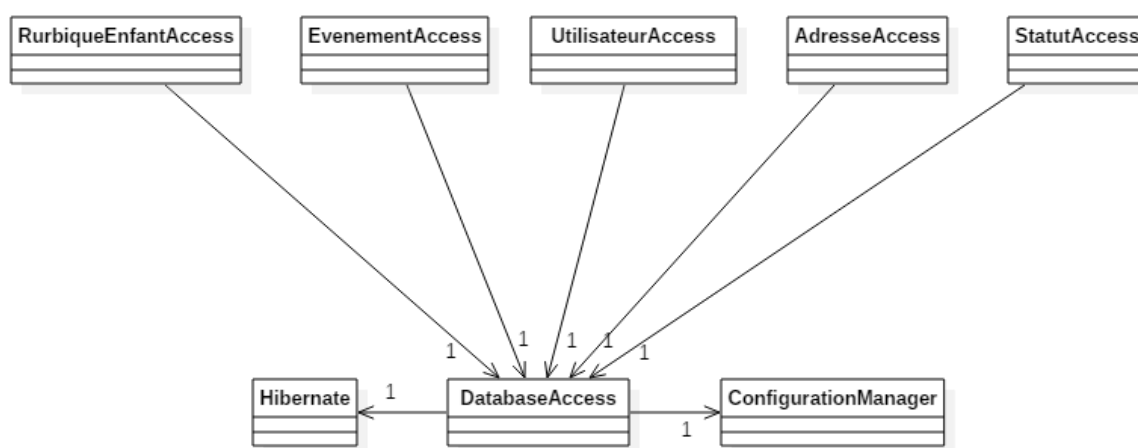


Figure 19 : Diagrammes des principales classes qui créent et exécutent les requêtes

En ce qui concerne la structure de ces classes, nous avons décidé de nous baser sur le même design pattern. Nous avons choisi le design pattern Singleton parce qu'il garantit qu'une instance unique d'une classe donnée soit créée et offre un point d'accès universel à cette instance.

3.3 Génération de documents PDF

Dans notre projet, il nous paraît important de garder une traçabilité des données. Pour ce faire, nous avons décidé de produire des documents PDF qui contiennent les informations relatives à la rubrique choisie. Ce document est composé de deux parties majeures.

La première donne des informations importantes sur les événements passés (lieu, dates). La deuxième partie est composée de statistiques concernant l'ensemble de la rubrique choisie.

Une classe principale, `GenerateurPDF`, est appelée lorsque l'utilisateur clique sur le bouton PDF de la fenêtre principale. Cette classe crée un document et le remplit avec les éléments présents dans la base de données. Les données collectées servent à réaliser un graphique.

À chaque création de documents PDF, un graphe en barres ou un graphe circulaire est généré de manière aléatoire. Ces graphiques sont créés à partir de la classe `GenerateurGraphique` qui génère une image.

Dans le cas où l'utilisateur sélectionne plusieurs rubriques, plusieurs documents PDF sont générés. Ils sont tous regroupés dans le dossier `SmartCity/PDF` du répertoire de l'utilisateur. Ils sont identifiés par la date et le nom de la rubrique sélectionnés.

Si, pour une date donnée, il n'y a pas d'événements, la première partie du document PDF sera vide. La deuxième partie qui contient les statistiques générales reste telle quelle.

Les documents doivent pouvoir interagir avec la base de données pour récupérer toutes les informations nécessaires à leur création. Pour ce faire, nous avons créé une classe qui permet de se connecter à la base de données et de lier les requêtes SQL avec des fonctions Java.

In fine, un document PDF est généré à partir du bouton PDF de la fenêtre principale. La création d'un ou plusieurs documents PDF entraîne l'ouverture du dossier contenant les nouveaux documents. À noter que, si l'utilisateur génère un document PDF et l'ouvre, puis qu'il régénère un document depuis l'application, il n'est pas régénéré. Il faut d'abord fermer le document. Ceci est dû au fait que le nom des documents est identique et que le fichier est donc en conflit avec deux applications (le lecteur de document PDF et l'application `SmartCity`). Dans ce cas, d'un point de vue visuel pour l'utilisateur, il ne se passe rien de spécial.

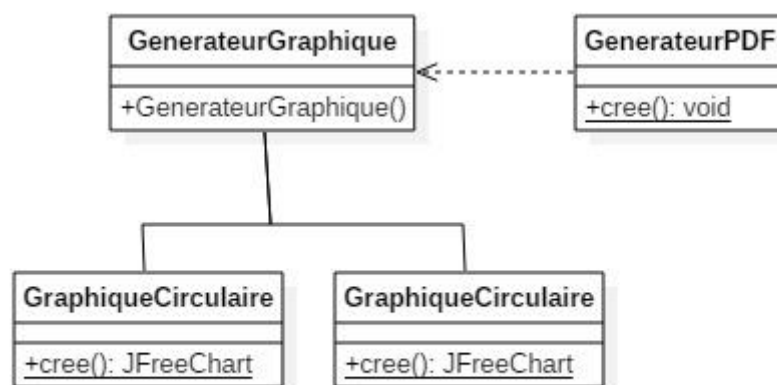


Figure 20 : Schéma de la structure des classes pour la génération d'un document PDF

3.4 Interface graphique utilisateur (GUI)

L'application est modélisée par deux fenêtres.

La première fenêtre est la fenêtre "principale" de l'application. Elle permet de voir l'affichage des événements sur la carte. Les cases de sélection (checkboxes) et le calendrier agissent comme filtres sur les événements affichés.

Cette fenêtre contient également d'autres boutons offrant les fonctionnalités suivantes : Ajout et modification des événements, validation des événements en attente et génération d'un document PDF.

La visualisation des événements est facilitée par leur positionnement sur la carte et l'affichage de leur description.

Une liste composée d'événements en attente de validation par l'utilisateur se met à jour automatiquement à intervalle régulier.

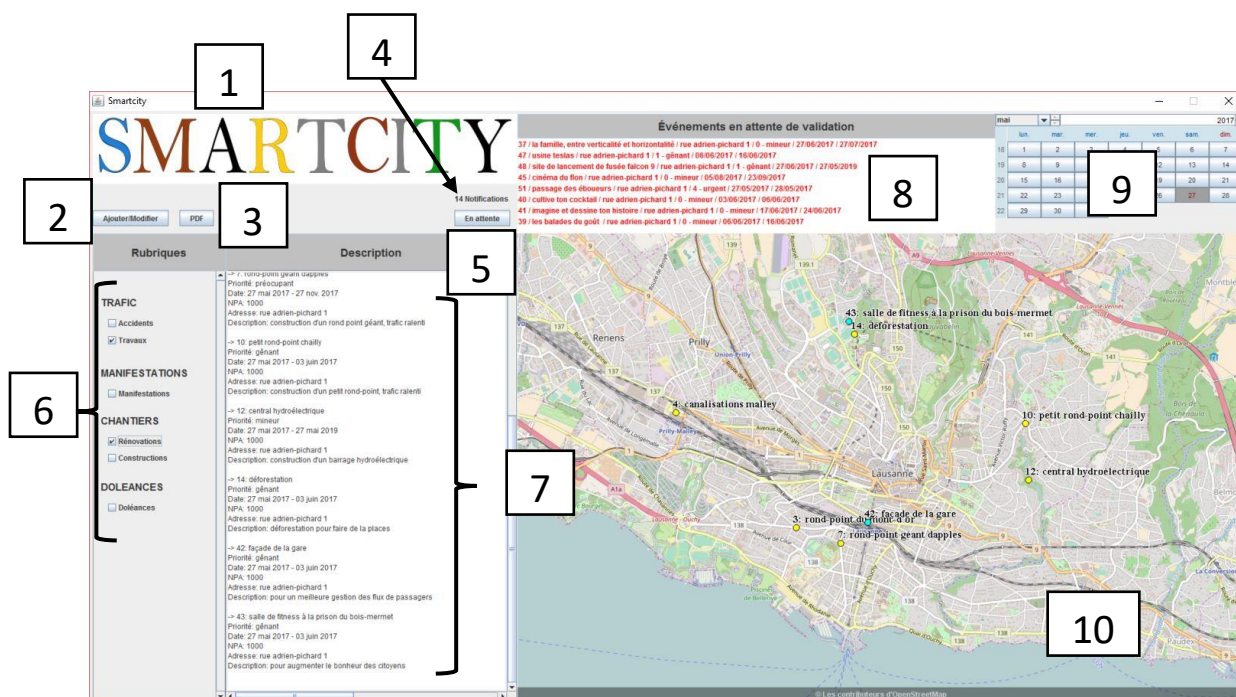


Figure 21 : Fenêtre principale de l'application

Description des composants de la fenêtre principale :

Éléments / zone	Description
1. Logo	Représente le logo de l'application
2. Bouton Ajouter/Modifier	Ce bouton ouvre sur la fenêtre d'ajout/modification d'évènements.
3. Bouton PDF	Ce bouton permet de générer un document PDF comprenant les informations relatives à la rubrique choisie (grâce aux checkboxes).
4. Notifications	Affiche le nombre d'évènements en attente de validation
5. Bouton en attente	Ouvre sur la fenêtre permettant de valider ou refuser en événement en attente.
6. Checkboxes Rubriques	Les cases cochées agissent comme filtre sur les évènements à afficher sur la carte interactive.
7. Description	Affiche les descriptions complètes des évènements filtrés
8. Liste évènements en attente	Liste déroulante affichant les évènements en attente de validation.
9. Calendrier	Agit comme filtre par la date sur les évènements à afficher
10. Carte interactive	Carte interactive affichant la position des évènements filtrés

La deuxième fenêtre de l'application apparaît dans deux contextes différents. Lorsque l'on veut ajouter ou modifier un événement de la base de données. La distinction entre ces deux actions est faite par la liste déroulante. Celle-ci propose l'option d'ajouter un nouvel événement ou d'en sélectionner un et de le modifier. Ces événements sont définis comme actifs suivant leur date de fin.

Si le choix se porte sur un événement présent dans la liste déroulante, les champs sont remplis avec les attributs de l'événement sélectionné. En modifiant la valeur de ces champs, il est possible de modifier l'événement sélectionné.

Cette fenêtre propose une validation du contenu des champs. Si une valeur n'est pas conforme, le label du champ est affiché en rouge et une note détaille le format du champ à respecter. Cette vérification est effectuée lorsque l'on clique sur le bouton valider et elle se base sur les caractères utilisés et la longueur du contenu grâce à des expressions régulières (regex).

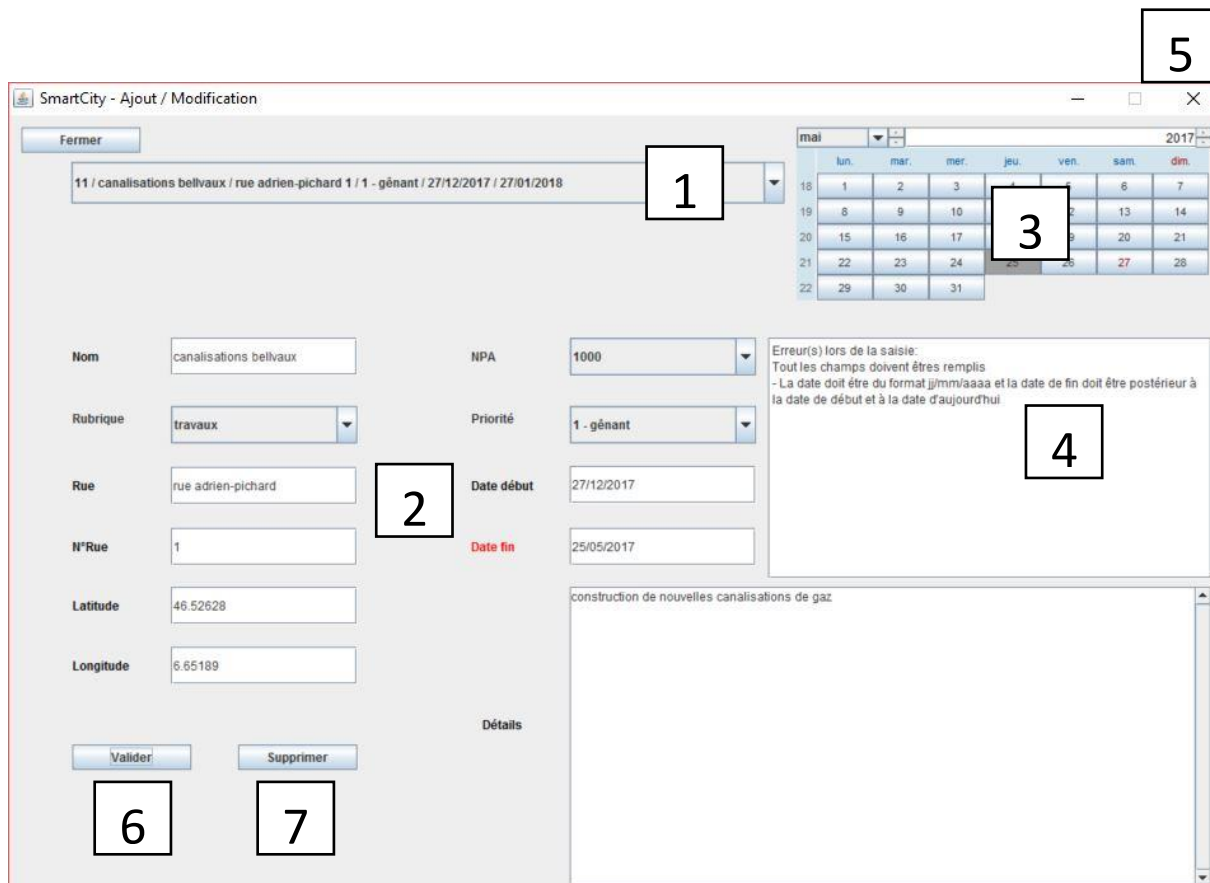


Figure 22 : Fenêtre d'ajout et de modification des évènements avec notification d'une erreur

Description des composants de la fenêtre d'ajout/modification :

Éléments / zone	Description
1. Liste déroulante	Permet de choisir si l'on veut ajouter un évènement ou modifier un élément déjà présent dans la base de données
2. Champs	Champs éditables pour définir les attributs d'un évènement
3. Calendrier	Permet de remplir les champs de date de début et de date de fin en interagissant avec les boutons du calendrier
4. Notification d'erreurs	Indique à l'utilisateur le champ incomplet et la forme attendue
5. Bouton fermer	Ferme la fenêtre courante
6. Bouton valider	Contrôle les données saisies, et si elles sont valides, ajoute l'évènement à la base de données, ou met à jour l'évènement si l'utilisateur avait choisi un évènement déjà présent dans la base de données
7. Bouton Supprimer	Supprime l'évènement sélectionné de la base de données

La même fenêtre permet la validation d'évènements en attente. Dans ce cas, l'utilisateur peut valider ou refuser un évènement.

La liste déroulante permet de sélectionner un de ces évènements. Sa sélection remplira les champs de la fenêtre. L'utilisateur pourra ensuite décider de le modifier ou non avant de le valider.

The screenshot shows a software window titled "SmartCity - En Attente". It features a "Fermer" button at the top left. A dropdown menu is open, displaying a list of events with details like location, category, and dates. A box labeled "1" highlights this list. Below the list are input fields for "Rue", "N°Rue", "Latitude", and "Longitude", each with a corresponding value. To the right, there's a "Date début" and "Date fin" field. A "Détails" section contains a text area with the description "sculpture sur argile, taille de pierre tendre, plâtre, béton cellulaire...". At the bottom, there are two buttons: "Valider" (labeled with box "2") and "Refuser" (labeled with box "3"). On the right side of the window, a calendar for May 2017 is visible, showing dates from 1 to 31.

Figure 23 : Fenêtre de validation des évènements en attente avec sa liste des évènements déroulée.

Description des composants de la fenêtre de validation :

Éléments / zone	Description
1. Liste déroulante	Permet de sélectionner un évènement en attente de validation
2. Bouton valider	Valide l'évènement
3. Bouton refuser	Refuse l'évènement

4 Tests de l'application

4.1 Description générale de l'environnement de test

Lors du développement de l'application, nous avons procédé à différents tests. Nous avons tâché d'isoler au mieux les différentes parties telles que la génération de document PDF et la carte interactive afin de mieux déceler pour chacun, ce qui ne fonctionnait pas.

L'application créée étant basée sur des fonctionnalités, il nous parut nécessaire de tester celles-ci afin de valider les objectifs en lien.

En ce qui concerne la sécurité de l'application, nous nous sommes concentrés uniquement sur les saisies de l'utilisateur. La base de données et la communication avec l'applicatif font partie des éléments critiques à protéger, mais nous avons fait le choix de ne pas y consacrer du temps.

Deux machines virtuelles étaient dédiées aux tests de validation. Elles tournaient sous Microsoft Windows 7 SP1 64 bits et Microsoft Windows 10 Familiale 64 bits. Elles ont permis de tester l'application dans des environnements très utilisés aujourd'hui.

Des tests unitaires et d'intégrations ont été effectués durant le développement de l'application sur la machine possédant Microsoft Windows 7. La base de données était stockée en local. Les données utilisées lors des tests étaient des données fictives.

Les conditions exactes des tests unitaires et d'intégrations du projet étaient :

- Microsoft Windows 7 64 bits
- Une connexion au réseau Internet et Intranet de l'HEIG-VD d'Yverdon-les-Bains
- L'archive JAR du projet située à la racine de son disque dur système

Une campagne de validation des cas d'utilisation a été effectuée sur les deux machines virtuelles par deux personnes extérieures au projet, à la fin du développement de l'applicatif. Ceci nous a permis d'obtenir des avis objectifs sur la correspondance entre l'applicatif et les cas d'utilisations annoncés. Ainsi, nous avons pu valider le bon fonctionnement de l'application.

4.1.1 Matériel

- Deux machines virtuelles de test
 - Intel® Core™ I7 4770 3.40GHz
 - 1 cœur physique, 2 cœurs logiques
 - 2 Go de RAM
 - Connexion réseau : Host-only
 - Taille du disque dur : 40.0 Gb

4.1.2 Systèmes d'exploitation et outils logiciels

- Microsoft Windows 7 Entreprise 64 bits SP1
- Microsoft Windows 10 Familiale 64 bits

4.2 Tests unitaires

4.2.1 Carte interactive

Afin de tester les méthodes telles que les constructeurs de points, nous avons utilisé des tests unitaires à l'aide du framework Junit. Il permet d'insérer des tests dans l'application et de référencer des méthodes présentes dans le corps du projet.

Cette méthodologie offre la possibilité de regrouper plusieurs tests à la suite sur un même objet comme un point par exemple. Dans le cas de la carte, nous devons tester la validité des conversions entre les points WGS84 et les points OSM.

```
@Test
public void testToOSM() {
    // on teste 3 positions 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 24 : Test unitaire pour tester la conversion des points WGS84 en OSM

Nous avons aussi effectué des tests sur le zoom qui ne peut pas dépasser certaines valeurs. Dans le cadre de notre application, nous avons décidé de borner le zoom. Celui-ci n'est pas utile au-delà de certaines valeurs et rend les informations sur la carte illisible. Grâce au test unitaire, nous pouvons tester une série de points et vérifier que le point reste valide.

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

Figure 25 : Test unitaire sur le zoom de la carte

4.2.2 Base de données

La méthode utilisée pour élaborer les classes qui génèrent les requêtes vers la base de données a facilité les tests unitaires à réaliser. La factorisation du code a permis de tester la création de requêtes rapidement. Ceux-ci ont permis de vérifier :

- La faisabilité de stocker des informations dans la base de données
- La possibilité de récupérer des enregistrements de la base de données
- La capacité à modifier des données

Il était indispensable de vérifier ces actions indépendamment des autres composants pour visualiser les erreurs le plus rapidement possible et pour pouvoir entreprendre des corrections efficaces sans parcourir tout le code de l'application.

```
/**
 * 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);
    assertEquals(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 26 : Vérification de la mise à jour d'un évènement

4.2.3 Génération de documents PDF

Nous avons essayé de générer un document PDF dans des conditions inhabituelles, afin de nous assurer que tout fonctionnait. En voici quelques-unes :

- Si la base de données est vide : le document PDF généré contient le logo d'entête, le titre et la date ainsi qu'un message indiquant qu'il n'y a pas de données.
- S'il n'existe aucun évènement à la date sélectionnée, la première partie du document PDF est vide, mais la deuxième partie contient bien les statistiques.
- Si l'utilisateur crée deux fois le même document PDF (même rubrique, même date) : le deuxième document PDF écrase le premier. Nous avons fait le choix de ne pas conserver le document précédent pour la bonne raison que le document PDF créé est plus récent.
- Si plusieurs rubriques sont sélectionnées : plusieurs documents PDF sont générés.

4.3 Tests d'intégrations

Les tests d'intégrations ont été effectués lors du regroupement des différents composants de l'application.

L'intégration de la carte dans l'application n'a pas nécessité de tests puisqu'elle n'a pas d'incidence sur les autres composants. Suite à la création de tuiles avec des points de couleur, le test de l'affichage de la localisation des événements s'est déroulé rapidement.

Une fois que la base de données a été intégrée à l'application, il a été nécessaire d'ajouter des méthodes afin de simplifier les requêtes. Initialement, l'implémentation de ces méthodes n'était pas prévue. Nous n'avons pas eu besoin de tester toutes les nouvelles méthodes créées puisque certaines utilisaient des méthodes précédemment implémentées et testées.

La génération de document PDF n'a pas été testée lors de son insertion dans l'application. Celle-ci a été testée séparément pour la mise au point de la structure de la page (titre, type de graphique, entêtes, etc.). Puis, elle a été testée lorsque la base de données contenait suffisamment de données.

4.4 Tests des cas d'utilisation

Nous avons testé séparément les différentes parties de l'application, telles que la génération de document PDF et la carte interactive afin de mieux déceler ce qui ne fonctionnait pas. Après le regroupement des parties, nous avons commencé à tester les cas d'utilisations.

Les premiers tests ont concerné la récupération d'événements de la base de données afin que ceux-ci apparaissent correctement dans la fenêtre de modifications. On peut se reporter la Figure 23, encadré 1, et voir que les événements apparaissent. Nous avons ensuite vérifié que ces événements correspondaient à ceux présents de la base de données.

Nous avons ensuite testé la modification des événements présents dans la base de données et vérifié que la modification se répercute bien sur les différentes fenêtres et dans la base de données.

- L'ajout de requête en attente se rajoute automatiquement dans la liste des événements en attente de validation. N'ayant pas fait d'application qui permette d'envoyer des événements directement dans la base de données, nous avons simulé l'ajout et celui fonctionne correctement. De plus si la liste dépasse le nombre d'affichages elle fait une mise à jour et affiche un mélange entre ceux déjà affichés et ceux qui ne le sont pas.

Un autre test a été la sélection des événements par dates.

- Si l'utilisateur sélectionne une catégorie d'événement à une date précise, alors ceux de cette catégorie s'affichent correctement sur la carte.
- Si l'utilisateur choisit de cocher plusieurs rubriques, seuls les événements actifs à cette date s'affichent sur la carte.

Il est possible de valider les événements en attente. Ceux-ci apparaissent dans la liste des événements modifiables et disparaissent de la liste des notifications.

Les tests sur la génération des documents PDF ont porté sur la sélection d'une ou plusieurs rubriques et d'une date. La vérification des documents PDF s'est basée sur la similarité de leur contenu et de celle de la base de données.

5 Conclusion

5.1 État des lieux

Au vu des délais à respecter, nous n'avons pas pu implémenter les fonctionnalités suivantes :

1. La gestion de plusieurs comptes administrateur
2. Dessins et coloriages sur la carte selon des critères de secteurs (cercles, lignes, texte)
3. Ajout d'une capture d'écran dans le document PDF de l'état de la carte
4. Calcul de chemin le plus court en évitant les zones accidentées ou en travaux

Les fonctionnalités principales décrites dans le cahier des charges au début du projet sont présentes. Lors de la réalisation du projet, quelques points ont été modifiés. Par exemple, la date de fin, qui était optionnelle au départ, est devenue obligatoire. Elle est toutefois modifiable pour pouvoir allonger ou raccourcir la durée d'un évènement.

De plus, une rubrique a dû être simplifiée. Initialement, « Doléances » devait regrouper plusieurs sous-rubriques créées dynamiquement en fonction des données soumises par les citoyens. Ainsi, nous avons décidé de transformer la rubrique « Doléances » en une rubrique simple.

Les fonctionnalités supplémentaires ne sont pas indispensables pour l'utilisation de l'application, mais elles peuvent toutefois améliorer le fonctionnement de cette dernière.

C'est pourquoi nous avons décidé de ne pas les développer, celles-ci requérant également une charge de travail trop conséquente (par exemple, calcul du chemin le plus court).

L'application ne possède pas de fonctionnalité d'authentification. Dans le cahier des charges, il s'agit d'une fonctionnalité supplémentaire qui aurait dû permettre la gestion de plusieurs comptes administrateurs. Ainsi, nous n'avons pas de moyen d'authentification et considérons l'application utilisable uniquement sur le poste d'un administrateur.

À l'heure actuelle, nous pouvons appliquer plusieurs opérations sur la carte interactive, comme déplacer la vue, zoomer et dézoomer. Mais il n'est pas possible de dessiner sur la carte.

Pour la génération de document PDF, il est possible de générer des documents en deux parties : une première partie contenant les informations des événements du jour et une seconde partie contenant des statistiques générales sur la rubrique choisie. Une fonctionnalité supplémentaire proposée était de mettre une capture d'écran de l'état de la carte dans le document PDF. Cette fonctionnalité n'a pas été réalisée.

5.2 Avis général

Grâce à ce projet, nous avons pu tester plusieurs notions apprises au cours des semestres précédents. Nous avons appliqué les notions de programmation orientée objet, de base de données et de programmation concurrente.

La séparation des tâches nous a permis de commencer le travail en parallèle et de faire avancer chaque fonctionnalité à bon rythme. Nous avons réparti ces fonctionnalités en fonction des choix et des qualités de chacun des membres du groupe, afin de nous sentir le plus impliqués possible dans la réalisation de ce projet. Il est vrai que cette répartition n'était pas toujours évidente, car le nombre de personnes dans le groupe fut conséquent au vu du nombre de fonctionnalités principales de notre application. Dans la seconde partie de l'implémentation, nous avons regroupé les différentes parties de chacun. Cette mise en commun du code s'est effectuée au prix de quelques adaptations, mais sans problème majeur.

Il est à noter que les rendez-vous étaient difficiles à mettre en place, dû aux différents emplois du temps de chaque membre. Nous avons donc dû pleinement profiter des périodes en commun mises à disposition. De plus, nous ne travaillions parfois pas en même temps ce qui retardait les réponses aux questions que nous pouvions avoir.

En fin de compte, malgré quelques obstacles nous restons satisfaits de notre application ainsi que du travail fourni.

5.3 Problèmes rencontrés

Nous avons rencontré plusieurs problèmes auxquels nous n'étions pas préparés et nous avons sous-estimé la charge de travail. Lors de l'élaboration initiale du projet, nous avions conscience de la difficulté de certaines tâches, mais nous ne savions pas précisément quelles technologies nous allions utiliser pour développer l'application et ces différents composants.

Tout d'abord, la mise en place de la base de données fut beaucoup plus longue que prévu. Ce qui a retardé l'accès et l'intégration des données au projet.

Composante phare de notre application, la carte interactive affichait des performances en deçà de nos attentes. Il a fallu trouver un moyen de pouvoir charger et afficher rapidement les tuiles de la carte. Après avoir effectué des recherches, nous avons compris qu'il était possible de mettre en cache les tuiles. Ce processus permet de les afficher plus rapidement au sein de l'application.

Une autre difficulté rencontrée fut de rassembler les principaux composants (carte, base de données et génération de document PDF). Au début de l'élaboration des composants, nous avons décidé de partir sur des sous-projets Java sans utiliser Apache Maven. Mais lors de l'intégration des composants, nous avons eu des difficultés en raison de l'utilisation de différentes bibliothèques pour chacun des sous-projets. Pour remédier à ce problème, nous avons décidé de convertir les composants en sous-projet Maven afin d'avoir plus de facilité lors de l'intégration.

5.4 Propositions d'améliorations

5.4.1 Interface graphique de l'utilisateur

Lors du choix de la bibliothèque de l'interface graphique, nous ne connaissions que JavaSwing. Elle suffisait à l'implémentation de notre application. Mais nous avons appris l'existence de JavaFX, qui à l'instar de Swing permet d'implémenter une interface graphique. Toutefois, JavaFX s'organise différemment et permet d'avoir un code beaucoup plus organisé et facile à lire. De plus, le design est plus agréable.

Une amélioration possible de notre application aurait été de passer de Swing à JavaFX. Nous n'avons pas pu effectuer cette modification dans les délais étant donné qu'une grosse partie de notre application concerne la partie graphique et qu'elle représentait une charge de travail trop importante pour être terminée à temps.

De plus, nous pourrions améliorer l'aspect général de l'application. Pour le moment, la taille de la fenêtre principale n'est pas modifiable. Pour ajouter du confort à l'utilisation, il faudrait rendre l'application « responsive design ».

Toujours dans un souci de design, nous pourrions créer et utiliser nos propres composants graphiques pour un « look » unique. Toutefois, comme nous avons mis l'accent sur les fonctionnalités, ces améliorations ne sont pas prioritaires.

5.4.2 Formulaire des évènements

Une amélioration possible du formulaire serait de changer le contrôle de la saisie du nom de la rue d'un évènement. À l'heure actuelle, nous contrôlons que les caractères soient conformes à un nom de rue, mais nous ne pouvons pas garantir son existence dans la ville de Lausanne.

Nous pourrions ajouter un contrôle de vérification. Pour ce faire, une liste proposerait les noms des rues possibles en fonction des premières lettres tapées par l'utilisateur.

5.4.3 Génération de documents PDF

Nous pourrions imaginer une amélioration au niveau des statistiques. Avec des évènements sur plusieurs années, les statistiques fournies seraient plus complètes. Il serait ainsi possible de voir une évolution des chantiers dans la ville de Lausanne ou encore de pouvoir repérer les meilleurs lieux pour des manifestations culturelles.

6 Annexes

- Cahier des charges
- Journaux de travail
- Planification initiale et son évolution
- Tests d'implémentation

7 Sources – Bibliographies

7.1 Ouvrages

- SAVITCH, Walter. Absolute Java. 5^{ième} édition, San Diego, Pearson, 2013, 1260 pages
- DELANNOY, Claude. Programmer en Java. 9^{ième} édition, Saint-Germain, Eyrolles, 2015, 916 pages

7.2 Sites Internet

Les sites Internet ci-dessous ont tous été consultés du 20.02.2017 au 30.05.2017.

- Développez : <http://www.developpez.com/>
- EPFL : <http://cs108.epfl.ch/archive/14/>
- iText : <http://itextpdf.com/>
- JBoss HIBERNATE : <https://docs.jboss.org/hibernate/stable/core.old/reference/fr/html/>
- JFreeChart : <http://www.jfree.org/jfreechart/>
- OpenClassRoom : <https://openclassrooms.com>
- OpenStreetMap : <https://www.openstreetmap.org>
- Oracle : <https://docs.oracle.com/en/java/>
- Stackoverflow : <http://stackoverflow.com/>
- TheCodersBreakfast : <http://thecodersbreakfast.net/index.php?post/2008/02/25/26-de-la-bonne-implementation-du-singleton-en-java>
- YouTube : <https://www.youtube.com/>
- Wikipedia : <http://wikipedia.org/>

7.3 Vidéos en ligne

- edureka!. Hibernate Tutorial - 1 | Hibernate Tutorial for Beginners - 1 | Edureka. [Tutoriel en ligne]. <https://www.youtube.com/watch?v=iwMW_SZPGnY>. (Consulté le 13 mars 2017).
- edureka!. Hibernate Tutorial | Hibernate Tutorial - 2 | Hibernate Tutorial for Beginners - 2 | Edureka. [Tutoriel en ligne]. <https://www.youtube.com/watch?v=VAr_likcYfE>. (Consulté le 13 mars 2017).
- edureka!. Hibernate - The Ultimate ORM Framework | Edureka. [Tutoriel en ligne]. <https://www.youtube.com/watch?v=xo_HGo812DQ>. (Consulté le 13 mars 2017).

8 Glossaire

API	Application Programming Interface
DD	Degrés Décimaux
DMS	Degré Minutes Secondes
GUI	Graphical User Interface
HEIG-VD	Haute École d'Ingénierie et de Gestion du Canton de Vaud
NPA	Numéro Postal d'Acheminement
OSM	OpenStreetMap
PDF	Portable Document Format
Regex	Regular Expression
SGBDR	Système de Gestion de Base de Données Relationnelles
TCS	Touring Club Suisse
WSG84	World Geodetic System 1984