LA2 - POO - GeoCache

Baptiste Pautrat & Gaston Deseine

GeoCache est une application java console à destinations de testeurs permettant de valider la robustesse et la souplesse de la couche d'accès aux données d'une application de gestion de GeoCaches.

Pré-requis

- Intellij Idea
- Java 15
- MySQL 5.7
- MongoDB 4.4.2

Installation

1. Récupérer le projet GitHub

   ```
   git clone https://github.com/GastonDeseineIG2I/la2-poo-geocache
   ```

2. Créer la base de données MySQL avec le fichier DDL ***geocache.sql***
3. Ouvrir le projet sur IntelliJ
4. Mettre à jour le fichier ***hibernate.cfg.xml*** en mettant les informations de connexion à la base de données MYSQL à jour
5. Définir le Java SDK à utiliser pour le projet sur IntelliJ
6. Exécuter la fonction *main()* de *Main.java*

Introduction

Sujet : Développer une application java console à destinations de testeurs permettant de valider la robustesse et la souplesse de la couche d'accès aux données d'une application de gestion de GeoCaches.

Cf. : cahier des charges.

Technologies utilisées

- Java 15.0.2
- Hibernate 5.4
- JPA 2.0
- MySQL 5.7
- MongoDB 4.4.2
- Morphia 1.3.2

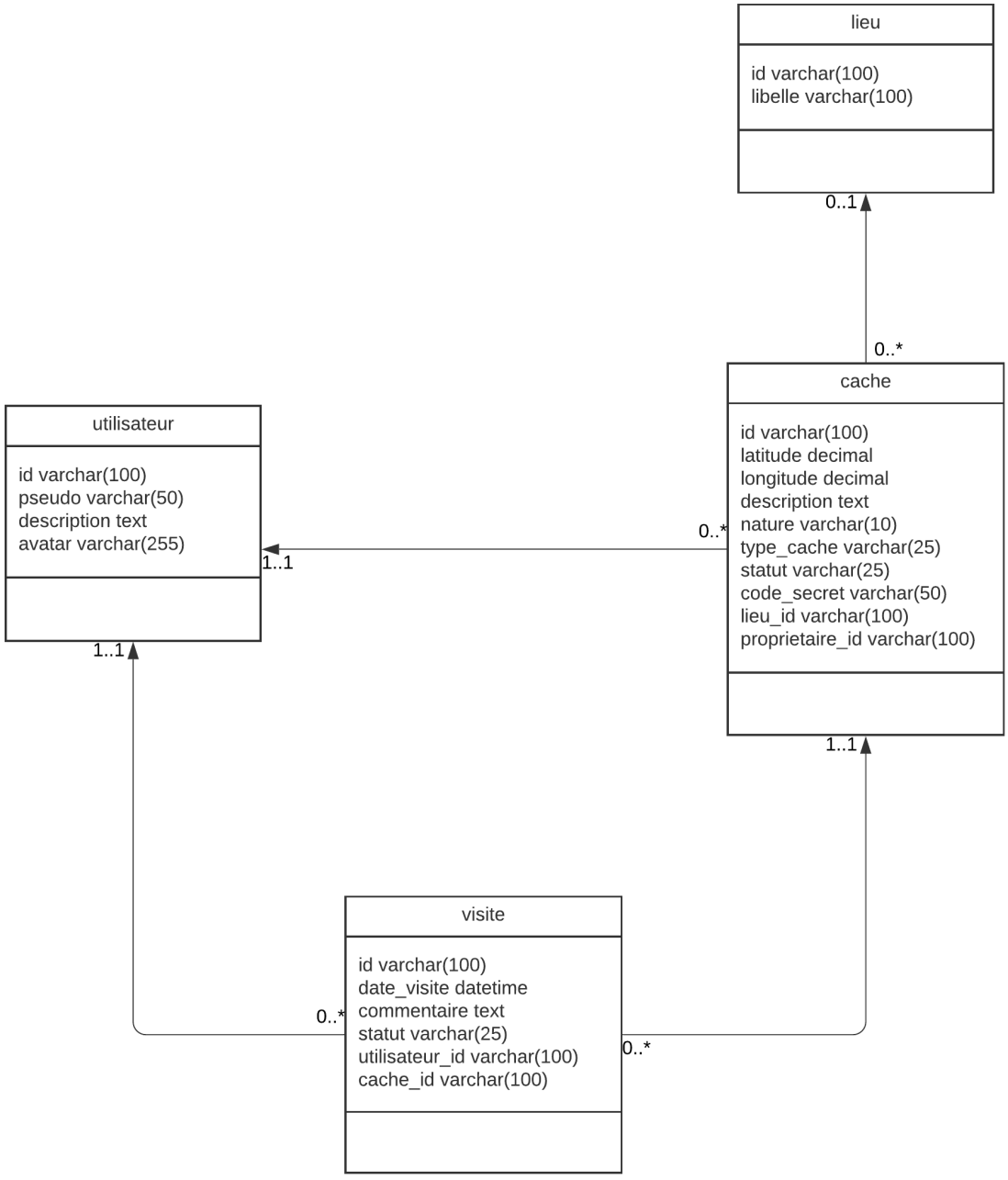
Conception

Architecture de base de données

Tables :

- Lieu : un lieu géographique définit par un libellé.
- Cache : informations permettant de situer et définir une cache.
- Utilisateur : information d'un utilisateur.
- Visite : lien entre un utilisateur et une cache.

Modèle UML :



SQL DDL

```
create schema `la2-geocache` collate latin1_swedish_ci;

create table cache
(
    id varchar(100) not null
        primary key,
    latitude decimal(12,10) null,
    longitude decimal(13,10) null,
    description text null,
    nature varchar(10) not null,
    type_cache varchar(25) default 'traditionnelle' not null,
    statut varchar(25) default 'inactive' not null,
    code_secret varchar(50) not null,
    lieu_id varchar(100) null,
    proprietaire_id varchar(100) not null
);

create index cache_lieu_id_fk
    on cache (lieu_id);

create index cache_utilisateur_id_fk
    on cache (proprietaire_id);

create table lieu
(
    libelle varchar(100) not null,
    id varchar(100) not null
        primary key
);

create table utilisateur
(
    id varchar(100) not null
        primary key,
    pseudo varchar(50) not null,
    description text null,
    avatar varchar(255) default 'default.png' not null,
    constraint utilisateur_pseudo_uindex
        unique (pseudo)
);

create table visite
(
    id varchar(100) not null
        primary key,
    date_visite datetime null,
    utilisateur_id varchar(100) not null,
    cache_id varchar(100) not null,
    commentaire text null,
    statut varchar(25) default 'En cours' not null
);

create index visite_cache_id_fk
    on visite (cache_id);
```

```
create index visite_utilisateur_id_fk
    on visite (utilisateur_id);
```

Hiérarchie des fichiers

```
├── lib
├── out
└── src
    ├── META-INF
    |   └── persistence.xml
    ├── Main.java
    ├── Menu.java
    ├── hibernate.cfg.xml
    ├── modele
    |   ├── CacheEntity.java
    |   ├── LieuEntity.java
    |   ├── UtilisateurEntity.java
    |   └── VisiteEntity.java
    └── repository
        ├── MONGODB
        |   ├── CacheRepository.java
        |   ├── LieuRepository.java
        |   ├── MONGODBRepository.java
        |   ├── UtilisateurRepository.java
        |   └── VisiteRepository.java
        ├── MYSQL
        |   ├── CacheRepository.java
        |   ├── LieuRepository.java
        |   ├── MYSQLRepository.java
        |   ├── UtilisateurRepository.java
        |   └── VisiteRepository.java
        └── RepositoryInterface.java
```

Nos choix

Voici les choix que nous avons réalisé :

- On associe un code secret à une cache qui permet de valider la visite par un utilisateur. Si le code n'est pas bon la visite n'est pas comptabilisée. Cela constitue une preuve de la visite.
- On stocke l'avatar d'un utilisateur dans un répertoire. On indique le chemin vers l'avatar de l'utilisateur dans la base de données en temps que VARCHAR(255). Le fichier de l'avatar est nommé en concaténant le pseudo de l'utilisateur (unique) avec l'extension du fichier. Une image par défaut est attribuée à l'utilisateur tant qu'il n'a pas défini la sienne.
- Nous avons fait le choix de réduire l'état d'une cache à inactif ou actif. Nous avons trouvé que en cours d'activation, fermée et suspendue étaient des sous catégories d'inactifs. Nous avons jugé que nous n'avions pas besoin de ce niveau d'information.

- Pour les visites nous avons décidé que le statut serait soit en En cours soit Terminée
- Au niveau de notre structure de code nous avons décidé de faire des repositories pour chacune de nos entités et nous avons aussi rajouté une interface *RepositoryInterface* pour y

mettre les fonctions communes à nos différents repositories. Toute la partie graphique se fait dans notre fichier Menu. Ce fichier est chargé de contacter le bon repository pour avoir les informations voulues.

Description de la solution

- Notre solution est une interface console. Sur un premier menu vous pouvez sélectionner un domaine fonctionnel (exemple les caches, les utilisateurs, ...). Puis une fois le choix réalisé on arrive un autre menu où l'on peut tester les différentes fonctionnalités associés à chaque domaine.

Difficultés

- Nous avons eu beaucoup de mal à mettre en place le switch vers MongoDB. Le problème est que nous avons développé le projet tout d'abord avec MySQL et lorsque nous avons du passer à MongoDB nous avons du changer beaucoup de choses dû aux nouvelles contraintes qui sont apparues.
- La difficulté des ID est arrivé dès l'arrivée de MongoDB. MongoDB veut des ObjectID et MySQL ne prend pas en charge ce type de donnée. Nous avons fait le choix d'utiliser 2 champs ID différents qui seraient utilisés en fonction de la BDD choisi.

```
@org.mongodb.morphia.annotations.Id
private ObjectId _id;
@Id
private String id;
```

- Nous avons eu aussi réfléchir à une façon de choisir le bon repository en fonction de la base de donnée utilisé. Notre choix s'est porté vers un système de tableau associatif qui associerait le nom du repository avec une instance du bon repository:

```
if (choixBDD.equals("MYSQL"))
{
    repository.put("cache", new repository.MYSQL.CacheRepository());
    repository.put("lieu", new repository.MYSQL.LieuRepository());
    repository.put("utilisateur", new
repository.MYSQL.UtilisateurRepository());
    repository.put("visite", new repository.MYSQL.VisiteRepository());
} else
{
    repository.put("cache", new repository.MONGODB.CacheRepository());
    repository.put("lieu", new repository.MONGODB.LieuRepository());
    repository.put("utilisateur", new
repository.MONGODB.UtilisateurRepository());
    repository.put("visite", new repository.MONGODB.VisiteRepository());
}
```

On appelle ensuite le repository de cet façon:

```
repository.get("visite")
```

Ce choix nous a permis de pouvoir factoriser nos fonctions pour ne pas avoir à dupliquer les morceaux de code faisant appel à des repositories.

Conclusion

Le projet a été très interessant car nous avons pu approfondir nos connaissances en Java / MongoDB.

Cela a été très enrichissant de découvrir Hibernate car c'est un Framework que nous ne connaissions pas du tout. Nous avions déjà utilisé des Framework de persistance des données dans d'autres langages de programmation mais jamais en Java.

Il aurait été aussi interessant de le faire avec une vraie interface graphique pour voir comment cela s'implémente en Java. Cependant il aurait fallu plus de temps pour le réaliser.

Ce projet étant une découverte pour tous les deux, il nous a fallu travailler beaucoup pour obtenir un résultat qui nous satisfait.