Projet Kot-Line

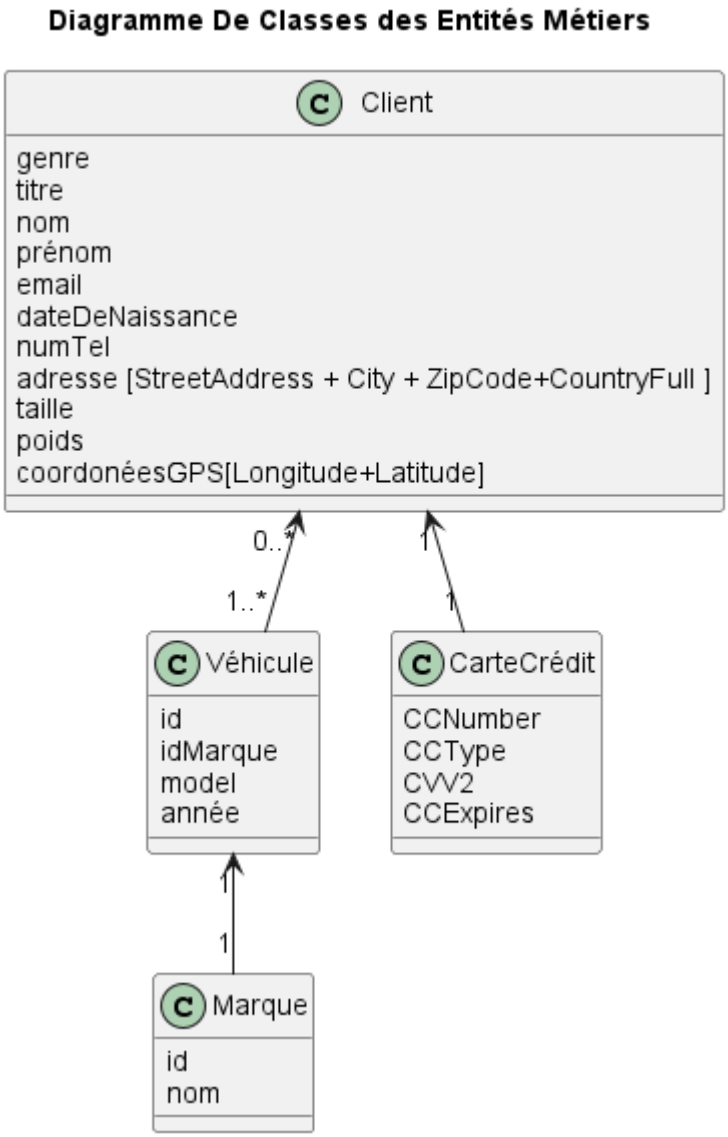
Les auteurs sont Quentin AYRAL, Sekou TRAORE et Mathieu MORGADO MARTA, lors de leur seconde année de BTS SIO. Fait en fin d'année 2022, conclu le 25/12/2022.

Contexte :

Concevoir un prototype d'application web, qui sera disponible dans l'intranet de l'entreprise `Kot-Line` permettant d'intégrer les données de fichiers dans une base de données.

Analyse :

Le diagramme de classe des entités metiers du projet :



Choix de la librairie logicielle :

Le choix de notre équipe s'est porté sur la librairie `FasterXML Jackson CSV` pour sa facilité d'analyse et d'utilisation en premier lieu mais nous avons décidé de retenir la librairie logicielle d'**`Apache`** car la lecture avec la librairie **CSV d'Apache** est vraiment simple, de plus elle ne nécessite qu'une seule dépendance et n'a pas de configuration complexe lorsque c'est le cas pour la librairie `FasterXML Jackson CSV`

Les 3 contraintes de selection :

Les contraintes de sélection à prendre en compte sont:

1. **L'âge**
    - *Si lors du transfert, une personne du fichier **CSV** n'etant pas agé entre 18 et 88 ans est rencontrée. Il faudra s'en occuper pour qu'elle n'apparaisse pas dans la table des Clients de la Base de données.*
2. **La taille**
    - *En cas de problèmes de conversions entre les tailles en InchesFeet (Pieds, pouces) et en Centimètres, il faudra faire coïncider les valeurs pour qu'elles n'aient pas de problèmes de conversions et soit bien cohérentes.*
3. **Les Doublons de cartes bancaires**
    - *Lors de la réception de nouvelles données nous risquons d'avoir des problèmes avec des personnes qui possèdent tous deux les mêmes cartes bancaires, il faudra faire en sorte de gérer ces données.*

Stratégie retenue

La stratégie retenue pour les personnes qui ne répondent pas aux conditions est :

1. **L'âge**
    - *Si une personne possède moins de 18 ans et 88 ans dans le fichier **CSV**, elle sera intégrée à la Base de Données mais sa valeur pour l'attribut "estValide" sera **false**.*
2. **La taille**
    - *En cas de problèmes de concordance entre les InchesFeet et les centimetres, la personne ayant ces erreurs sera intégree à la base de données mais l'attribut "estValide" aura la valeur **false**.*
3. **Les Doublons de cartes bancaires**
    - *Si une personne a un Numéro de carte bancaire similaire à une autre elle sera intégrée à la base de données mais sa valeur pour l'attribut "estValide" sera **false**.*

Chaque utilisateur ayant **false** à son attribut `estValide` pourra faire l'objet d'une suppression par un administrateur de l'application Web

Eviter les doublons de données

Pour éviter les doublons de données. Nous allons comparer nos données actuelles avec les donnés que l'on reçoit. Si des doublons se trouvent dans les données, on n'exploitera pas les doublons du fichier du client, mais on exploitera les nouvelles données qui ne sont pas en double.

Risques des utilisateurs malveillants

Les risques auxquels nous devons faire face sont :

1. L' `import` de fichier par de quelconques utilisateurs
2. L' `accès` à des données sensibles par de quelconques utilisateurs
3. La `suppression` de données de DataBase par de quelconques utilisateurs
4. La `surcharge` de l'application Web par l'import de fichier trop volumineux

Réalisation

Implementation des 3 contraintes de selection :

Nous avons decider d'implémenter des fonctions dans notre classe `ImportController` qui traitent les données du fichier **CSV** reçues et renvoie un booléen à la fin de chaque traitement à une variable `estValide`, cette dernière est ajoutée à chaque ligne de table de la Base de Données accueillant les données du fichier importé.

- `L□Age`

Nous avons réaliser une fonction `getAge` à laquelle nous donnons en paramètre la date de naissance présente dans le fichier **CSV** et qui la sépare/divise et calcule la période entre la date obtenue et la date actuelle

```
    fun getAge(bday : String): Int {
        val dateN = bday.split('/').toTypedArray()
        val moisN = dateN[0].toInt()
        val jourN = dateN[1].toInt()
        val anneeN = dateN[2].toInt()
        return Period.between(
            LocalDate.of(anneeN, moisN, jourN), LocalDate.now()
        ).years
    }
```

Par la suite, cette valeur constituant maintenant l'age de la personne dans le fichier **CSV** est renvoyé dans la fonction `estValide` et `estValide` renvoit true ou false en fonction de la valeur de l'age.

```
fun estValide(bday : String, tCM: String, tInch: String): Boolean{
```

```
    return if(getAge(bday)<18 || getAge(bday)>88) false else
correspondanceTaille(tCM,tInch)
}
```

- La `Taille`

Nous avons réalisé une fonction `correspondanceTaille` à laquelle nous donnons en paramètre la taille en centimetre et la taille en pieds pouces présentent dans le fichier **CSV** et qui additionnent ces dernières une fois converties en pouces avec `poucesTotaux` et les converties en centimètres et détermine si les valeurs initiales de centimètres et de feetInches sont similaires une fois la conversion faites avec toutefois une marge d'erreur de 1.5 centimètres de permise.

```
fun correspondanceTaille(tailleCM: String, tailleInch: String): Boolean {

  val feet = tailleInch.split(' ').toTypedArray()
     fun poucesTotaux(): Int{
        for(i in feet.indices){
           val pieds = feet[0].subSequence(0, feet[0].length - 1).toString().toInt()
           val pouces = feet[1].subSequence(0, feet[1].length - 1).toString().toInt()
            return pieds * 12 + pouces
        }
          return 0
        }
        val tailleCMInches = poucesTotaux() * CONVERSIONPOUCESENCM
        return !(tailleCMInches >= tailleCM.toDouble() + MARGEDERREUR ||
tailleCMInches<= tailleCM.toDouble() - MARGEDERREUR)
    }
```

Par la suite, `correspondanceTaille` est utilisé par `estValide` pour les valeurs de FeetInches et Centimetres n'étant pas concordantes.

- `Les Cartes Bancaires redondantes`

Pour eviter les doublons de cartes bancaires, nous avons réalisé une map qui permet la comparaison de valeurs (Les numéros de cartes bancaires dans le fichier csv) grâce à une clé (L'username des clients dans le fichier csv). La clé étant les usernames avec ayant pour valeur les numéros de cartes bancaires, si un client a un numéro de carte qui correspond à celui d'un autre dans la map, alors la variable 'estValide' aura la valeur false, si cependant il n'est pas dans la map, cette dernière enregistre le numéro de carte bancaire à la clé correspondant à l'username.

```
for(key in mapLoginCCNumber.keys){
   if(mapLoginCCNumber[key] == csvRecord.get("CCNumber")){
        estValide = false
        clientRepository.findByLogin(key).estValide = false
   }else{
        mapLoginCCNumber[csvRecord.get("Username")] = csvRecord.get("CCNumber")
    }
}
```

Evil User Stories

Nous avions dit plus haut que nous avons des risques vis à vis de l' `import`, de l' `accès` , de la `suppression` de données de Base de données par de quelconques utilisateurs ainsi que la `surcharge` de l'application Web par l'import de fichier trop volumineux.

Nous avons implementer des contres-mesures afin d'eviter de quelconques utilisations malveillantes de la Web App

- L'import, l'accès, la suppression

Pour minimiser les problèmes avec la base de données nous avons créer des rôles pour empêcher que quiconque puisse importer des fichiers ou traiter les données et ainsi utiliser ces fonctionnalités à des fins malveillantes, nous avons ajouté manuellement une petite liste d'utilisateurs ayant pour rôle `«ROLE_VIP»`. En effet, seuls ces derniers peuvent accéder à l'interface d'importation de fichiers et supprimer des clients de la base de données depuis l'application Web

La fonction `filterchain` definissant les droits dans la classe `WebSecurityCOnfiguration`:

```
@Bean
@Throws(java.lang.Exception::class)
    fun filterChain(http: HttpSecurity): SecurityFilterChain? {
        http.authorizeHttpRequests()
            .antMatchers("/").permitAll()
            .antMatchers("/login").permitAll()
            .antMatchers("/webjars/**").permitAll()
            .antMatchers("/error").permitAll()
            .antMatchers("/import/**").hasAnyAuthority("ROLE_VIP")
            .and()
            .formLogin()
            .defaultSuccessUrl("/")
            .usernameParameter("username")
            .passwordParameter("password")
            .and()
            .logout()
            .logoutRequestMatcher(AntPathRequestMatcher("/logout"))
            .logoutSuccessUrl("/login")
        return http.build()
    }
```

Exemple d'Admin avec role `VIP` dans la classe `DatabaseInitializer` :

```
val meyer = personRepository.save(Person("meyer", "Bertrand", "Meyer", "ROLE_VIP",
bCryptPasswordEncoder.encode("password"),"Bertrand Meyer publie rien",3002
)
```

L'accès à la page d'importation de fichier disponible uniquement si l'utilisateur **loggué** a le role de `VIP`

```
<li sec:authorize="hasAuthority('ROLE_VIP')" class="nav-item">
   <a class="nav-link" href="/import">Importer</a>
</li>
```

- La Surcharge

Pour éviter un temps de traitement trop long et de causer un bogue à l'application Web, nous avons définit une taille d'import de fichier à ne pas dépasser qui à été fixé à 3 MégaBytes, permettant de translater des fichiers assez conséquents tout en faisant attention au serveur accueillant la Web APP

Les instructions définissant la taille limite de fichier à envoyer au server dans `application.properties`

```
spring.servlet.multipart.max-file-size=3MB
spring.servlet.multipart.max-request-size=3MB
```

Projet Kot-Line par le Groupe SQM :

Notre dépôt Github

Le projet initial