Projet : Kotlin

Rapport du project d'Amir KATSIEV + Yannis MANSOUR Fait le 27/12/2022 Team : NoName

Rapport du project

Application support

Une application starter basée sur les guides suivants

- https://spring.io/guides/tutorials/spring-boot-kotlin/
- https://spring.io/guides/gs/accessing-data-mysql/

Une lecture de ces ressources est vivement recommandée.

Prérequis

- Premiers pas réussis en Kotlin
- Une machine opérationnelle (test de l'application https://github.com/ldv-melun/sbfirst)
- Avoir réalisé les exercices Exercices avec Controleur et Vue
- Avoir pris connaissance du chapitre *Introduction à JPA* du support Spring Boot
- Disposer d'un serveur Mysql à proximité (plus simplement sur votre machine de dev)

- Avoir des compétences de bases minimales en HTML/CSS et SQL

Les dépendances

- Maven
- Spring Boot Starter
- Thymeleaf
- Mysql
- Webjars
- BootStrap
- Spring security

Première Etape

Initialiser la base de données en ligne de commande

Après avoir lancé la commande `mysql` (une application qui se connecte par défaut au serveur mysql de la machine), vous créez une base de données, qui sera exploitée par l'application, ainsi qu'un utilisateur `mysql`.

Dans un second temps, vous (en tant qu'administrateur de bases de données) donnez les droits à l'utilisateur sur la base de données de l'application.

Connectez-vous à MySql en ligne de commande, via la commande `mysql`

Listing 1. puis inspirez-vous de commandes suivantes pour initialiser une base de données

```
mysql> create database db_example; -- Creates the new database
mysql> create user 'springuser'@'%' identified by 'ThePassword'; -- Creates the user
mysql> grant all on db_example.* to 'springuser'@'%'; -- Gives all privileges to the
new user on the newly created database
```

Prise en main

application.properties

Après avoir cloné l'application, et avant de lancer l'application, vérifier la conformité du contenu de `src/main/resources/application.properties` qui contient des informations utilisées par votre application pour se connecter à votre base de données créée précédemment.

Listing 2. src/main/resources/application.properties (extrait)

```
spring.datasource.url=jdbc:mysql://${MYSQL_HOST:localhost}:3306/db_example ①
spring.datasource.username=springuser ②
spring.datasource.password=ThePassword ③
spring.servlet.multipart.max-file-size=10MB ④
spring.servlet.multipart.max-request-size=10MB ④
server.error.whitelabel.enabled=false ⑤
server.error.path=/error ⑥
```

① db_example : le nom de la base de données que vous avez créée (à modifier le cas échéant)

② Idem pour le compte utilisateur MySQL utilisé par l'application pour se connecter à la base de données.

③ Mot de passe, en clair, de compte utilisateur MySQL

④ Max-file-size est défini sur 10 MB , ce qui signifie que la taille totale du fichier ne peut pas dépasser 10 MB.

⑤ Désactivation page d'erreur Spring Boot générique

⑥ Definition d'un chemin d'erreurs personnalisé

Le modèle métier

```
@Entity ①
@Table(name = "client")
class Client(

    @Column(name = "Title", unique = false, nullable = false)②
    var title: String,

    @Column(name = "Surname", unique = false, nullable = false)
    var surname: String,

    @Column(name = "GivenName", unique = false, nullable = false)
    var givenName: String,

    @Column(name = "EmailAddress", unique = false, nullable = false)
    var emailAddress: String,

    @Column(name = "Birthday", unique = false, nullable = false)
    var birthday: String,

    @Column(name = "CCType", unique = false, nullable = false)
    var ccType: String,

    @Column(name = "CCNumber", unique = false, nullable = false)
    var ccNumber: String,

    @Column(name = "CCExpires", unique = false, nullable = false)
    var ccExpires: String,

    @Column(name = "TelephoneNumber", unique = false, nullable = false)
    var telephoneNumber: String,

    @Column(name = "StreetAddress", unique = false, nullable = false)
    var streetAddress: String,

    @Column(name = "City", unique = false, nullable = false)
    var city: String,

    @Column(name = "StateFull", unique = false, nullable = false)
    var stateFull: String,

    @Column(name = "ZipCode", unique = false, nullable = false)
    var zipCode: String,

    @Column(name = "Centimeters", unique = false, nullable = false)
    var centimeters: String,

    @Column(name = "FeetInches", unique = false, nullable = false)
```

```
    var feetInches: String,

    @Column(name = "Latitude", unique = false, nullable = false)
    var latitude: String,

    @Column(name = "Longitude", unique = false, nullable = false)
    var longitude: String,

    @Column(name = "Contrainte", unique = false, nullable = true,)
    var contrainte: String?,

    @OneToMany(mappedBy = "client", cascade = [CascadeType.ALL])③
    var vehicles: List<Vehicule>,




    @Id @GeneratedValue var id: Long? = null) ④


@Entity
@Table(name = "vehicule")
class Vehicule(

    @Column(name = "year", unique = false, nullable = false)
    var year: String, ⑤

    @Column(name = "model", unique = false, nullable = false)
    var model: String,

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "marque_id")
    var marque: Marque,

    @ManyToOne(fetch = FetchType.LAZY) ⑥
    @JoinColumn(name = "client_id")
    var client: Client,

    @Id @GeneratedValue var id: Long? = null
)

@Entity
@Table(name = "marque")
class Marque(
    @Column(name = "name", unique = false, nullable = false)
    var name: String,

    @OneToMany(mappedBy = "marque", cascade = [CascadeType.ALL])
    var vehicules: List<Vehicule> = ArrayList(),
```

```
    @Id @GeneratedValue var id: Long? = null
)
```

① Désigne cette classe comme Entité, c'est à dire en relation avec une table dans le système de persistance de type SQL

② Donne des informations pour la colonne correspondant à la propriété `Titre`

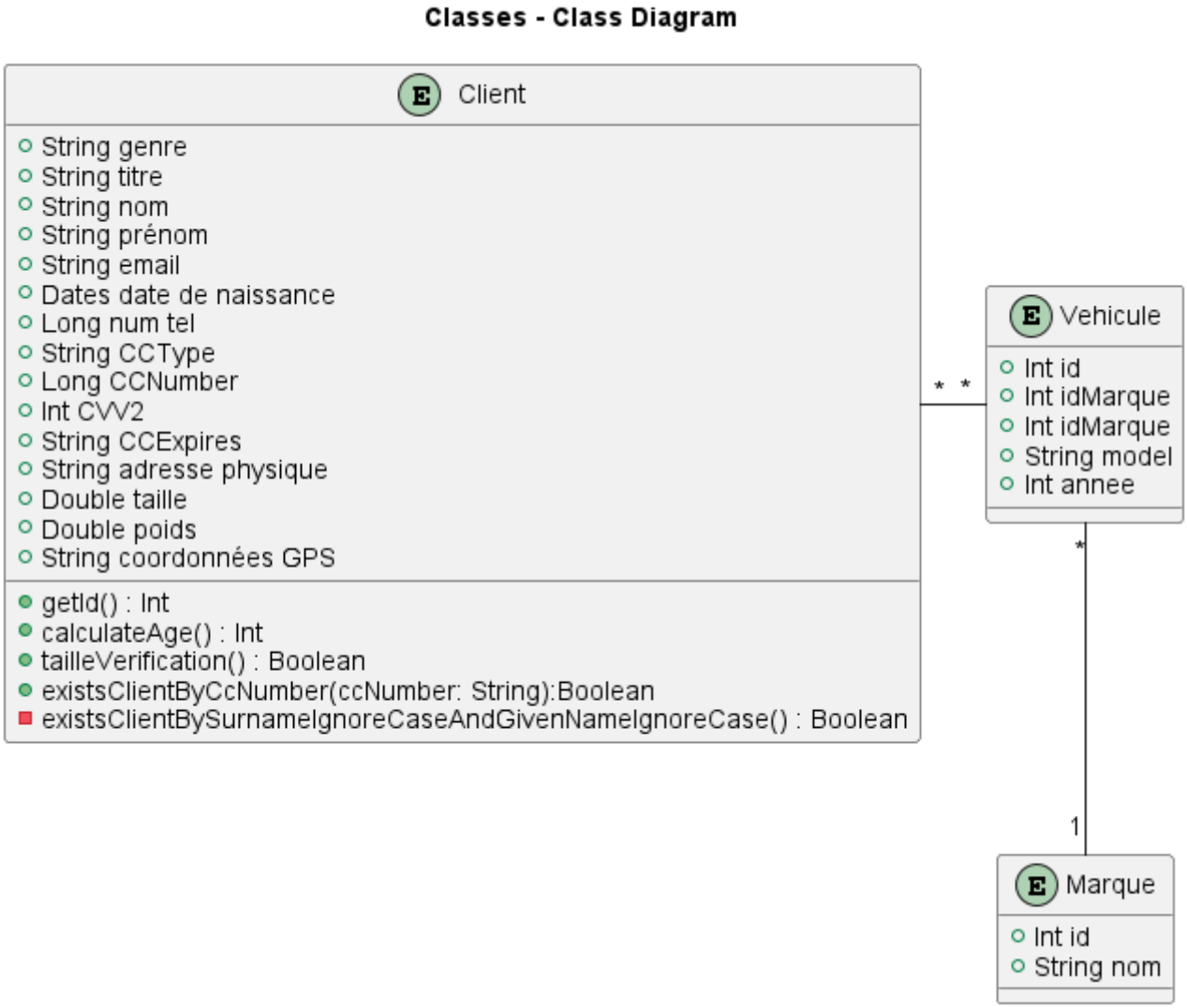
③ C'est le lien inverse d'une DF déclaré dans l'entité Vehicule.

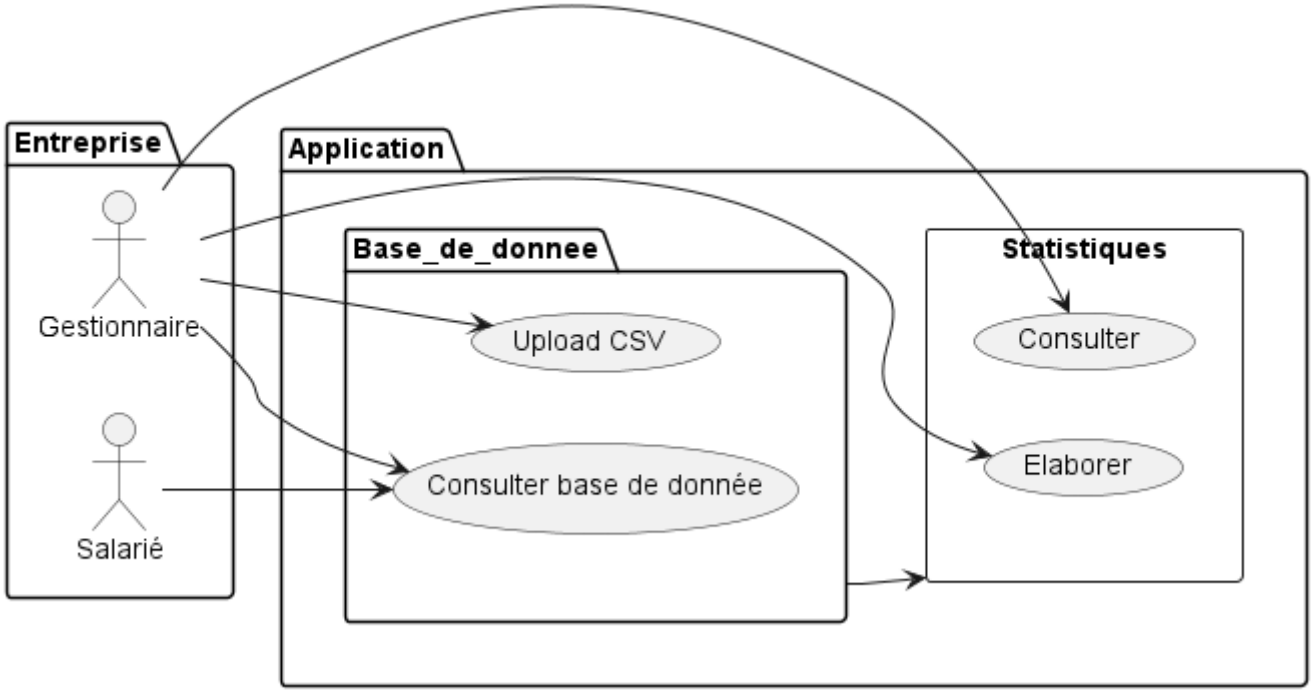
④ Clé primaire

⑤ Le nom de la colonne est directement déduit, par défaut, du nom de la propriété

⑥ Une DF, relation entre entités du modèle.

Entités du Domaine





Les repositories

Ce sont des interfaces techniques qui se chargent des opérations en liens avec la base de données, opérations d'interrogation (*query*) et d'écriture (*create, update*)

Les méthodes de ces interfaces sont soit pilotées par le schéma relationnel de la base, et donc exprimées en `SQL` , soit héritées d'interface prévues à cet effet, comme `CrudRepository` par exemple.

Listing 3. Exemple d'un repository associé à la classe entité `Client`

```
interface ClientRepository : CrudRepository<Client,Long>{

   fun existsClientBySurnameIgnoreCaseAndGivenNameIgnoreCase(surname:String,
givenName:String):Boolean

   fun existsClientByCcNumber(ccNumber: String):Boolean



}
```

Les contrôleurs

```
@Controller
 @GetMapping("/upload") ①
    fun upload(model: Model): String {
        model["title"] = "Import de personnalités"
        return "import/upload"
    }

    @PostMapping("/upload")②
    fun import(
        @RequestParam("file") file: MultipartFile,
        redirectAttributes: RedirectAttributes ③
    ): String? {
        if (file.isEmpty) {
            redirectAttributes.addFlashAttribute("message", "Please select a file to
upload")
            return "redirect:import/uploadStatus"
        }

        try {
            val inputStream: InputStreamReader = InputStreamReader(file.inputStream,
"UTF-8")
            val bufferedReader = BufferedReader(inputStream)
            val aFormat = CSVFormat.DEFAULT.builder()
                // choix des colonnes
                .setHeader("Number","Gender","NameSet","Title","GivenName",
"MiddleInitial","Surname","StreetAddress","City","State","StateFull","ZipCode","Countr
y","CountryFull","EmailAddress","Username","Password","BrowserUserAgent","TelephoneNum
ber","TelephoneCountryCode","MothersMaiden","Birthday","TropicalZodiac","CCType","CCNu
mber","CVV2","CCExpires","NationalID","UPS","WesternUnionMTCN","MoneyGramMTCN","Color"
,"Occupation","Company","Vehicle","Domain","BloodType","Pounds","Kilograms","FeetInche
s","Centimeters","Latitude","Longitude")
                .setIgnoreHeaderCase(true)
                .setSkipHeaderRecord(true)
                .setTrim(true)
                .build()
            val csvParser = CSVParser(bufferedReader, aFormat)

            var cptImportedClients: Int = 0 ④

            for (csvRecord in csvParser) { ⑤
                val p: Client = Client(
                    title = csvRecord.get("Title"),
                    surname = csvRecord.get("Surname"),
                    givenName = csvRecord.get("GivenName"),
                    emailAddress = csvRecord.get("EmailAddress"),
                    birthday = csvRecord.get("Birthday"),
                    ccType = csvRecord.get("CCType"),
```

```
                ccNumber = csvRecord.get("CCNumber"),
                ccExpires = csvRecord.get("CCExpires"),
                telephoneNumber = csvRecord.get("TelephoneNumber"),
                streetAddress = csvRecord.get("StreetAddress"),
                city = csvRecord.get("City"),
                stateFull = csvRecord.get("StateFull"),
                zipCode = csvRecord.get("ZipCode"),
                centimeters = csvRecord.get("Centimeters"),
                feetInches = csvRecord.get("FeetInches"),
                latitude = csvRecord.get("Latitude"),
                longitude = csvRecord.get("Longitude"),
                vehicle = csvRecord.get("Vehicle"),
                contrainte = ""


            )
```

① La première fonction, upload, est mappée à une requête GET au point de terminaison '/upload' et renvoie un modèle de vue appelé 'import/upload'. La deuxième fonction, import, est mappée à une requête POST vers le point de terminaison '/upload' et traite un fichier qui a été téléchargé par l'utilisateur.

② La `import` fonction a une @RequestParamannotation, qui indique à Spring de lier le 'file' paramètre de la requête au `file` paramètre de la fonction. Le `file` paramètre est une instance de MultipartFile, qui est une représentation d'un fichier téléchargé dans Spring

③ La fonction a également un `redirectAttributesparamètre`, qui est une instance de RedirectAttributes. Ceci est utilisé pour stocker les attributs qui seront disponibles dans le modèle de la prochaine requête, permettant aux données d'être transmises entre les requêtes lors de la redirection de l'utilisateur.

④ La fonction garde une trace du nombre de clients importés avec succès lors du traitement du fichier.

⑤ Le code utilise une boucle for pour parcourir chaque enregistrement du fichier CSV, qui est stocké dans l'objet csvParser. Pour chaque enregistrement, il crée un objet Client en utilisant l'opérateur de création d'objet Client (Client(...)) et en passant les valeurs de chaque champ sous forme de chaîne de caractères à partir de l'objet csvRecord.

Les 3 contraintes

Nous avons 3 fonctions qui vérifient si les données rentrent dans ces contraintes

Seules les personnes majeures et n'ayant pas atteint l'âge de 88 ans à la date de l'importation du fichier devront être sélectionnées.

```
fun calculateAge(birthday: String?): Int {
       val sdf = SimpleDateFormat("dd/MM/yyyy") ①
       val dateOfBirth: Date = sdf.parse(birthday)
       val currentCalendar = Calendar.getInstance() ②
       val currentDate: Date = currentCalendar.time ③
       val ageInMillis: Long = currentDate.getTime() - dateOfBirth.getTime()
       val ageInYears = ageInMillis / (365L * 24 * 60 * 60 * 1000) ④
       return ageInYears.toInt() ⑤
   }
```

① La fonction définit un objet SimpleDateFormat avec le format de date "dd/MM/yyyy", qui indique comment la chaîne de caractères de la date de naissance doit être interprétée.

② Elle crée un objet Calendar qui représente la date courante.

③ Elle récupère la date courante sous forme d'objet Date.

④ Elle calcule l'âge en années en divisant la différence en milisecondes par la durée d'une année en milisecondes.

⑤ Elle retourne l'âge en années sous forme d'entier.

Prise en compte de la Contrainte-de-taille.

```
    fun tailleVerification(centimeters: String, feetInches: String): Boolean {
       val words: List<String> = feetInches.split(" ") ①
       val pied = words[0].substring(0, words[0].length - 1).toInt() ②
       val pouce = words[1].substring(0, words[1].length - 1).toInt()
       val feetInchesToCm = ((pied * 30.48) + (pouce * 2.54)) ③


       if (abs(feetInchesToCm-centimeters.toDouble()) > 1.15) {
           return true
       }
       return false

   }
```

① Elle découpe la chaîne de caractères représentant la taille en pieds et pouces en une liste de mots en utilisant la méthode split avec l'espace comme délimiteur.

② Elle récupère le nombre de pieds et de pouces sous forme d'entiers en utilisant la méthode substring et la fonction toInt.

③ Elle calcule la taille en centimètres en convertissant le nombre de pieds et de pouces en centimètres.

④ Elle calcule la différence entre la taille en centimètres et la taille en pieds et pouces en utilisant la fonction abs, si la différence est supérieure à 1.15 cm et retourne le résultat de cette vérification sous forme de booléen.

Prise en compte de la Contrainte-de-ccn.

```
    fun existsClientByCcNumber(ccNumber: String):Boolean ①
```

① La méthode définie dans cette interface permet de vérifier si un client existe dans la base de données en fonction de son numéro de carte de crédit. Elle retourne un booléen indiquant si le client existe ou non.

Ces fonctions nous les vérifions après la lecture de chaque enregistrement du fichier CSV et de récupérer les valeurs de chaque champ sous forme de chaîne de caractères.

```
            for (csvRecord in csvParser) {
                val p: Client = Client(
                    title = csvRecord.get("Title"),
                    surname = csvRecord.get("Surname"),
                    givenName = csvRecord.get("GivenName"),
                    emailAddress = csvRecord.get("EmailAddress"),
                    birthday = csvRecord.get("Birthday"),
                    ccType = csvRecord.get("CCType"),
                    ccNumber = csvRecord.get("CCNumber"),
                    ccExpires = csvRecord.get("CCExpires"),
                    telephoneNumber = csvRecord.get("TelephoneNumber"),
                    streetAddress = csvRecord.get("StreetAddress"),
                    city = csvRecord.get("City"),
                    stateFull = csvRecord.get("StateFull"),
                    zipCode = csvRecord.get("ZipCode"),
                    centimeters = csvRecord.get("Centimeters"),
                    feetInches = csvRecord.get("FeetInches"),
                    latitude = csvRecord.get("Latitude"),
                    longitude = csvRecord.get("Longitude"),
                    vehicle = csvRecord.get("Vehicle"),
                    contrainte = "" ①


                )

                if (!clientRepository
.existsClientBySurnameIgnoreCaseAndGivenNameIgnoreCase(p.surname, p.givenName)){ ②
                    val values = mutableListOf<String>() ③
                    if  (service.calculateAge(p.birthday) < 18 || service.
calculateAge(p.birthday) > 80) { ④
                        values.add("Age")
                    }
```

```
                if (service.tailleVerification(p.centimeters,p.feetInches)){ ⑤
                    values.add("Taille")
                }
                if (clientRepository.existsClientByCcNumber(p.ccNumber)){ ⑥
                    values.add("Doublons")
                }
                p.contrainte = values.joinToString(", ") ⑦
                cptImportedClients++
                clientRepository.save(p)
            }
```

① `contrainte` est initialisé à une chaîne vide, ce qui signifie qu'il n'y a pas de contrainte pour l'enregistrement en cours.

② Avant d'enregister les donnes, nous vérifions si les clients existant pas

③ `mutableListOf`, est utilisée pour stocker les contraintes qui s'appliquent à l'enregistrement en cours. Des chaînes de caractères sont ajoutées à la liste si certaines conditions sont remplies, et la liste est convertie en une chaîne de caractères unique en utilisant la méthode joinToString avant d'être affectée à la propriété contrainte de l'objet Client.

④ Le code vérifie d'abord si l'âge du client, calculé à l'aide de la fonction calculateAge, est inférieur à 18 ans ou supérieur à 80 ans. Si c'est le cas, il ajoute la chaîne "Age" à la liste values.

⑤ Il vérifie ensuite si la différence entre la taille du client exprimée en centimètres et en pieds et pouces est supérieure à 1.15 cm, en utilisant la fonction tailleVerification. Si c'est le cas, il ajoute la chaîne "Taille" à la liste values.

⑥ Enfin, il vérifie si un client avec le même numéro de carte de crédit existe déjà dans la base de données en appelant la méthode existsClientByCcNumber de l'interface ClientRepository. Si un tel client existe, il ajoute la chaîne "Doublons" à la liste values.

⑦ La méthode joinToString est une méthode de la classe Iterable qui prend en paramètres un séparateur (ici une chaîne de caractères contenant une virgule et un espace) et qui concatène tous les éléments de la liste en une seule chaîne de caractères en utilisant ce séparateur.

Extrait de la vue de la liste des Clients

Listing 4. src/main/resources/templates/client/index.html

```
<div class="table-responsive">
    <table class="table w-auto small">
        <thead>
        <tr>
            <th>Titre</th>
            <th>Nom</th>
            <th>Prenom</th>
            <th>Adresse</th>
            <th>Zip Code</th>
            <th>Mail</th>
            <th>Telephone</th>
            <th>Data de Naissance</th>
            <th>CC Type</th>
            <th>CC Number</th>
            <th>CC Expires</th>
            <th>CC Vehicule</th>
            <th>CC FeetInches</th>
            <th>Centimetre</th>
            <th>Latitude</th>
            <th>Longitude</th>
            <th>Contrainte</th>

        </tr>
        </thead>
        <tbody>
        <div class="container">
            <div th:if="${message}" class="alert alert-danger" role="alert">
                <span th:text="${message}"></span>
            </div>
        </div>
        <th:block th:each="client: ${listClients}"> ①
            <tr th:if="${client.contrainte.length() != 0}" ②
                th:class="''table-danger''">
                <td th:text="${client.getTitle()}">Titre</td>
                <td th:text="${client.getGivenName()}">Nom</td>
                <td th:text="${client.getSurname()}">Prenom</td>
                <td th:text="${client.getStreetAddress()}">Adresse</td>
                <td th:text="${client.getZipCode()}">Zip Code</td>
                <td th:text="${client.getEmailAddress()}">Mail</td>
                <td th:text="${client.getTelephoneNumber()}">TelephoneNumber</td>
                <td th:text="${client.getBirthday()}">Data de Naissance</td>
                <td th:text="${client.getCcType()}">CC Type</td>
                <td th:text="${client.getCcNumber()}">CC Number</td>
                <td th:text="${client.getCcExpires()}">CC Expires</td>
                <td th:text="${client.getVehicle()}">Vehicle</td>
                <td th:text="${client.getFeetInches()}">FeetInches</td>
```

```
                <td th:text="${client.getCentimeters()}">Centimeters</td>
                <td th:text="${client.getLatitude()}">Latitude</td>
                <td th:text="${client.getLongitude()}">Longitude</td>
                <td th:text="${client.getContrainte()}">Contrainte</td>
                <td><a sec:authorize="hasAuthority('ROLE_VIP')" th:href=
"@{/client/{id}(id=${client.id})}"④
                       class="btn btn-danger"> <i
                        class="fas fa-user-times ml-2">Supprimer</i>
                </a></td>

            </tr>
            <tr th:unless="${client.contrainte.length() != 0}"class="table-light" > ③
                <td th:text="${client.getTitle()}">Titre</td>
                <td th:text="${client.getGivenName()}">Nom</td>
                <td th:text="${client.getSurname()}">Prenom</td>
                <td th:text="${client.getStreetAddress()}">Adresse</td>
                <td th:text="${client.getZipCode()}">Zip Code</td>
                <td th:text="${client.getEmailAddress()}">Mail</td>
                <td th:text="${client.getTelephoneNumber()}">TelephoneNumber</td>
                <td th:text="${client.getBirthday()}">Data de Naissance</td>
                <td th:text="${client.getCcType()}">CC Type</td>
                <td th:text="${client.getCcNumber()}">CC Number</td>
                <td th:text="${client.getCcExpires()}">CC Expires</td>
                <td th:text="${client.getVehicle()}">Vehicle</td>
                <td th:text="${client.getFeetInches()}">FeetInches</td>
                <td th:text="${client.getCentimeters()}">Centimeters</td>
                <td th:text="${client.getLatitude()}">Latitude</td>
                <td th:text="${client.getLongitude()}">Longitude</td>
                <td th:text="${client.getContrainte()}">Contrainte</td>
                <td><a sec:authorize="hasAuthority('ROLE_VIP')" th:href=
"@{/client/{id}(id=${client.id})}"④
                       class="btn btn-danger"> <i
                        class="fas fa-user-times ml-2">Supprimer</i>
                </a></td>
            </tr>
            </th:block>
        </tbody>
    </table>
```

① Un *foreach* en Thymeleaf. L'instruction `th:each="client : ${listClient}` déclare une variable de boucle nommée `client`.

② La directive `th:if` indique qu'une condition doit être évaluée et que la balise `<tr>` ne doit être incluse dans le document HTML que si la condition est vraie, la condition est une expression booléenne qui vérifie si la longueur de la chaîne de caractères contenue dans la propriété contrainte de l'objet client est différente de zéro. Si c'est le cas, la ligne du tableau sera affichée en rouge, sinon elle sera ignorée.

③ La directive `th:unless` est similaire à `th:if`, mais elle inverse le résultat de l'expression booléenne évaluée. Si la condition est vraie, la colonne va être affichée en blanc

④ La directive `sec:authorize` indique que l'accès à l'élément HTML qu'elle entoure est contrôlé par

Spring Security. Dans ce cas en particulier, l'élément HTML est un lien (<a>) qui ne sera accessible que si l'utilisateur connecté a le rôle ROLE_VIP. Si l'utilisateur n'a pas ce rôle, le lien ne sera pas affiché, ce lien permettra à l'utilisateur qui a un rôle "VIP" de gérer les données qui contiennent des contraintes

Consultation des donnes clients

- Les donnes clients pourront être consultées par un simple utilisateur mais ne pourront pas être modifiées ou supprimées, pour cela il nous faut un utilisateur VIP

Nous avons créer un gestionnaire d'authentification pour une application web a l'aide du fichier `WebSecurityConfiguration`

```
class WebSecurityConfiguration {

    @Bean
    @Throws(java.lang.Exception::class)
    fun authManager( ①
        http: HttpSecurity,
        bCryptPasswordEncoder: BCryptPasswordEncoder,
        userDetailsService: CustomUserServiceDetails
    ): AuthenticationManager? {
        return http.getSharedObject(AuthenticationManagerBuilder::class.java)
            .userDetailsService(userDetailsService)
            .passwordEncoder(bCryptPasswordEncoder) ②
            .and()
            .build()
    }

    @Bean
    @Throws(java.lang.Exception::class)
    fun filterChain(http: HttpSecurity): SecurityFilterChain? { ③
        http.authorizeHttpRequests() ④
            .antMatchers("/").permitAll()
            .antMatchers("/login").permitAll()
            .antMatchers("/access-denied").permitAll()
            .antMatchers("/webjars/**").permitAll()
            .antMatchers("/error").permitAll() /.antMatchers("/admin/**").permitAll()
            .antMatchers("/upload/**").hasAnyAuthority("ROLE_VIP")
            .antMatchers("/addPerson/**").hasAnyAuthority("ROLE_VIP")
            .antMatchers("/client/**").hasAnyAuthority("ROLE_VIP") ⑤
            .antMatchers("/clients").hasAnyAuthority("ROLE_USER","ROLE_VIP") ⑥
            .and() //.csrf().disable()
            .formLogin() ⑦
//            .loginPage("/login").failureUrl("/login?error=true")
            .defaultSuccessUrl("/")
            .usernameParameter("username")
            .passwordParameter("password")
            .and()
            .logout()
            .logoutRequestMatcher(AntPathRequestMatcher("/logout"))
            .logoutSuccessUrl("/")
            .and()
            .exceptionHandling().accessDeniedHandler(accessDeniedHandler())
        return http.build()
```

```
    }
```

① La fonction `authManager` prend en paramètre un objet `HttpSecurity`, un objet `BCryptPasswordEncoder` et un objet `CustomUserServiceDetails` et retourne un objet `AuthenticationManager`. Elle est appelée lors de la configuration de la sécurité de l'application pour créer le gestionnaire d'authentification qui sera utilisé pour authentifier les utilisateurs.

② On utilise également la méthode `passwordEncoder` pour définir l'objet `BCryptPasswordEncoder` qui sera utilisé pour encoder les mots de passe avant de les stocker dans la base de données.

③ La fonction `filterChain` prend en paramètre un objet `HttpSecurity` et retourne un objet `SecurityFilterChain`. Elle est appelée lors de la configuration de la sécurité de l'application pour définir les règles de sécurité qui s'appliquent aux requêtes HTTP.

④ Nous avons également la méthode `authorizeHttpRequests` de l'objet http pour définir les règles de sécurité pour différentes URL de l'application. Il utilise la méthode `antMatchers` pour spécifier les URL à protéger et la méthode `permitAll` pour indiquer que tous les utilisateurs peuvent accéder à ces URL. Il utilise également la méthode `hasAnyAuthority` pour indiquer que seuls les utilisateurs ayant au moins une des autorisations spécifiées peuvent accéder à certaines URL.

⑤ Seuls les utilisateurs qui ont un rôle "ROLE_VIP" pourront supprimer des clients qui contiennent des contraintes

⑥ Cette ligne indique que les utilisateurs ayant au moins l'un des rôles "ROLE_USER" ou "ROLE_VIP" sont autorisés à accéder à l'URL "/clients".

⑦ On utilise également la méthode `formLogin` pour configurer la connexion de l'utilisateur via le formulaire de connexion et la méthode de déconnexion pour configurer la déconnexion de l'utilisateur.

Exemples d’Evil User Stories et leurs contre-mesures potentielles :

1. Un utilisateur malveillant essaie de découvrir les mots de passe d’autres utilisateurs en utilisant une attaque par dictionnaire ou une attaque de force brute.
    - Contre-mesure : utilisez des mots de passe forts et uniques pour chaque compte d’utilisateur et imposez des limites à la longueur et à la complexité des mots de passe. En outre, mettez en place des contrôles pour détecter et bloquer les tentatives de connexion répétées à l’aide de mots de passe incorrects.
2. Un utilisateur malveillant essaie de dégrader les performances de l’application en utilisant des requêtes malveillantes ou en envoyant du trafic de spam.
    - Contre-mesure : Mettre en place des contrôles de sécurité pour détecter et bloquer les requêtes malveillantes ou le trafic de spam, et utiliser des techniques de mise en cache et de gestion de la charge pour améliorer les performances de l’application.

Conclusion

En résumé, Apache Commons CSV est une bibliothèque open source utile pour travailler avec des fichiers CSV en Java/Kotlin. Il est facile à utiliser et à apprendre, prend en charge plusieurs formats de fichiers CSV, fait un excellent travail avec les fichiers CSV volumineux et peut être utilisé gratuitement dans de nombreux projets. Spring Security est également facile à configurer et à personnaliser, grâce à son modèle de configuration basé sur des annotations et à sa grande flexibilité. Il est également facile d’intégrer Spring Security à d’autres bibliothèques Java pour étendre ses capacités de sécurité.