

SafetyNet-Alert

Présentation & Besoins du Projet

L'application SafetyNet Alert permet de fournir des informations sur des personnes se trouvant dans une zone prédéfinie lors d'épisodes catastrophes tels que les inondations, incendies, accidents, etc...

Pour se faire, à partir d'un fichier de données JSON existant, l'application doit, une fois ce dernier lu, établir deux catégories d'endpoints permettant:

1. De fournir une réponse sous forme de fichier JSON à partir des requêtes reçues sur les URL pré-définies.
2. Pour d'autres systèmes, d'interagir avec l'application sur des endpoints bien précis afin d'effectuer des modifications et mises à jour de données.

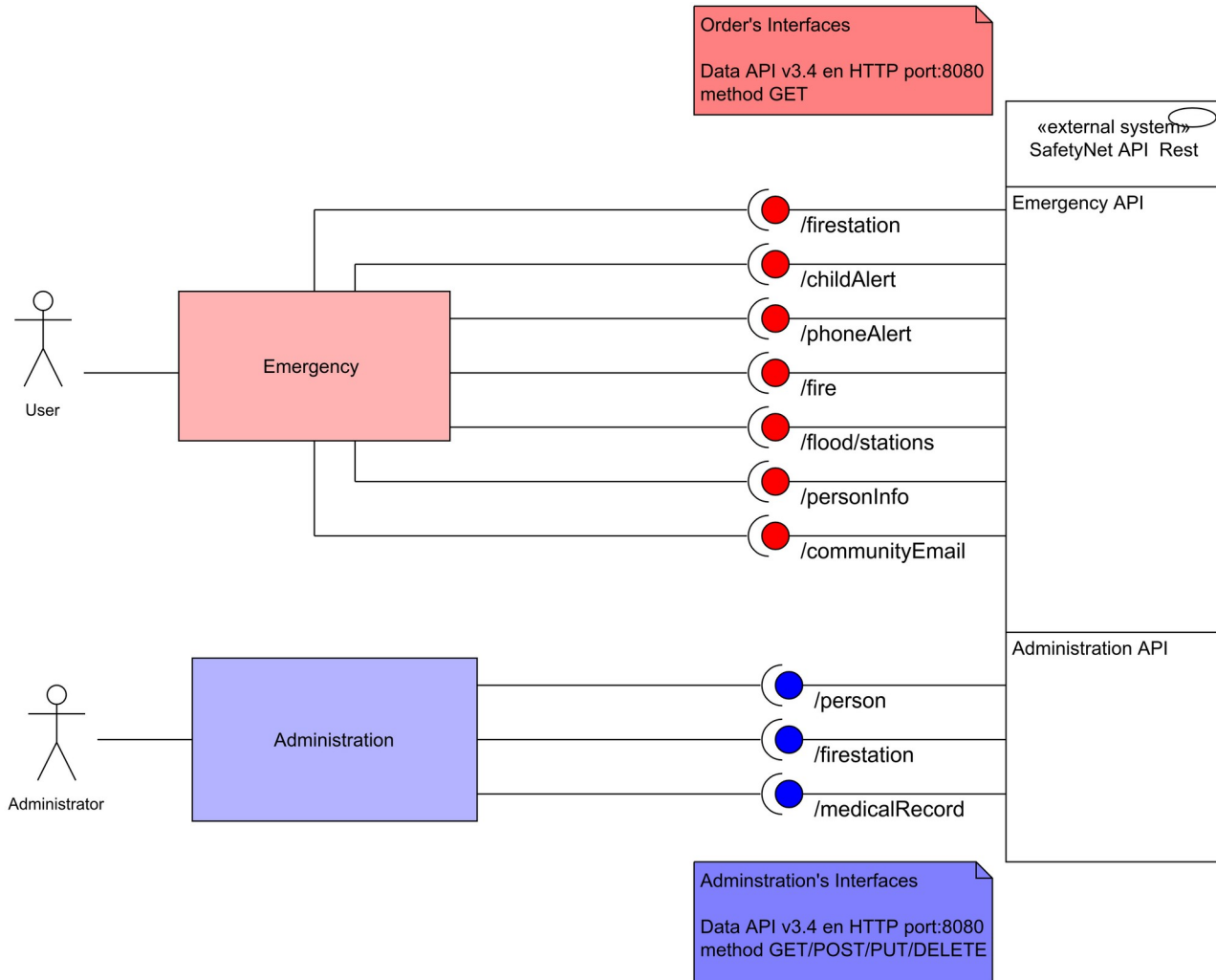
La réussite de l'application se portera donc essentiellement sur la mise en place d'un REST API permettant d'exposer les différents endpoints ainsi que la vérification de ses derniers aux moyens de tests unitaires...

Modèle de Domaine

Pour répondre à ces besoins et avoir une compréhension globale des objectifs de l'application, nous avons défini le modèle de domaine suivant constitué de :

Une **Emergency API** : qui s'occupera d'exposer les endpoints relatifs aux requêtes d'URLS pré-définies.

Une **Administration API** qui s'occupera d'exposer les endpoints relatifs aux besoins de mise à jour des données.

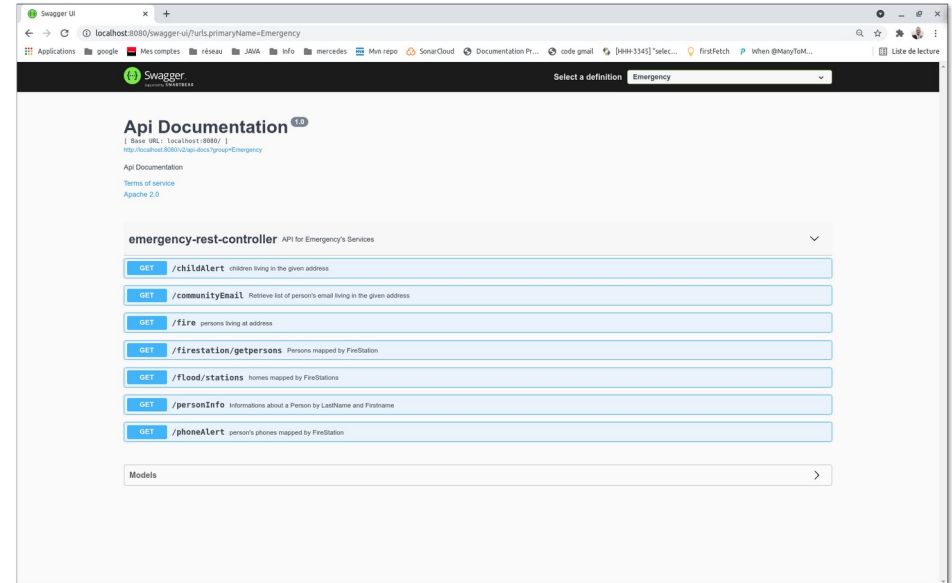
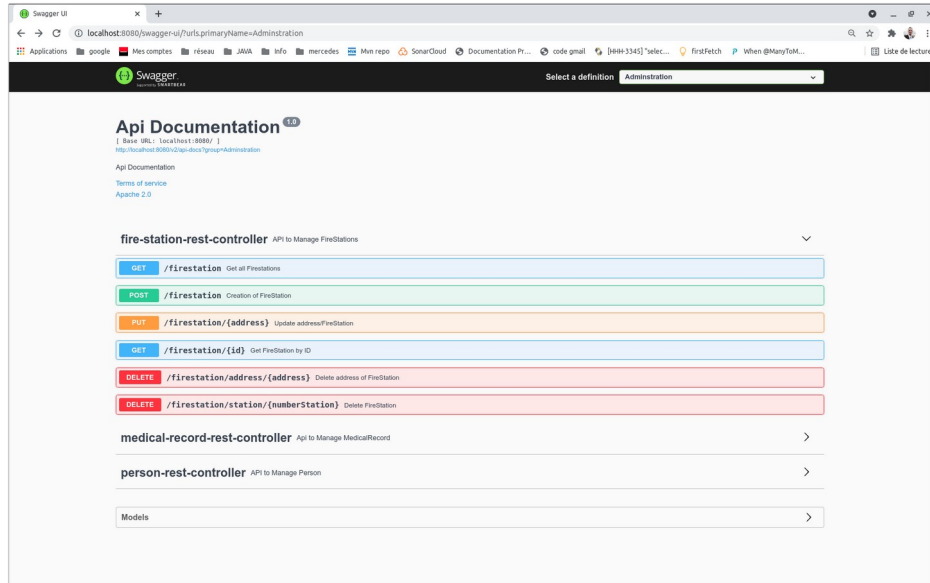


User Stories & critères d'acceptation

Suite au Modèle de domaine précédent, nous avons définis les user stories pour chaque requêtes sur les endpoints demandés.

- Ces derniers ainsi que leurs critères d'acceptation sont détaillés dans la partie annexe de ce document [Page 25](#)
- Il est à noter également qu'une documentation des API a été produites avec l'outil Swagger pour comprendre les types de requêtes possibles ainsi que les réponses obtenues pour chaque endpoint à l'adresse suivante :

<http://localhost:8080/swagger-ui/>



Solution Techniques

A partir des besoins, du modèle de domaine et des user stories précédemment détaillés, il a été défini:

- Un choix de structure de projet en fonction des spécifications et stack technique de l'application
- Un diagramme de classes permettant de définir les classes et interfaces utilisées, leurs relations et adopter un langage omniprésent pour ce projet
- Un Modèle Physique des Données afin de définir les relations entre les différentes entités et concevoir l'implémentation de la base de donnée.

Structure du Projet

Pour répondre aux besoins et spécifications techniques demandés, le projet a été structuré en plusieurs packages afin de répondre au motif Modèle-Vue-Contrôleur :

- **Configuration:** dédié a la configuration de la documentation swagger des Rest API.
- **Controller:** qui comprends deux packages admin et emergency regroupant les contrôleurs permettant d'exposer les endpoints
- **Database:** qui permet de charger des le démarrage de l'application le fichier de données nommé data.json.
- **Dto:** qui comprend un modèle DTO spécifiquement créer pour répondre uniquement aux requêtes sur les endpoints d' emergency API.
- **Exceptions:** qui comprends toutes les exceptions recensées pour l'application et un GlobalExceptionHandler pour les gérer.
- **Model:** qui regroupe les entités représentant les données de l'application
- **Repository:** qui regroupe les interfaces nécessaires à la persistance des données
- **Service:** qui regroupe les services dédiés à la logique métier.

Enfin, on trouvera la méthode Main de l'application et une classe implémentant CommandLineRunner pour le démarrage.

A noter, que l'on trouvera dans 'src/test/' les mêmes packages comprenant les tests d'intégration et unitaires.

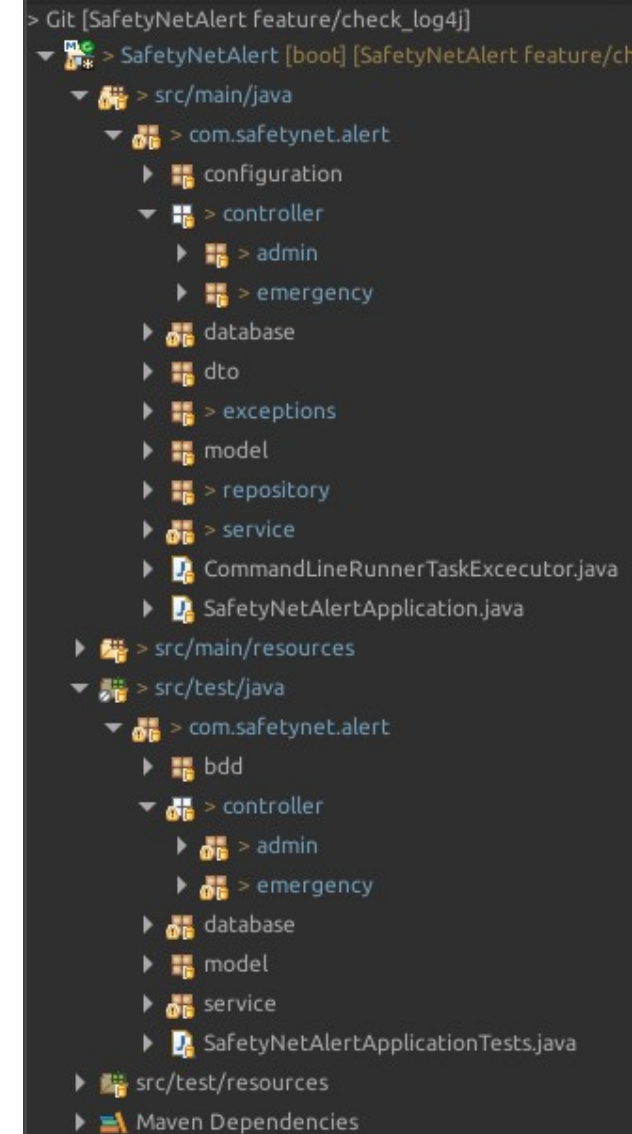


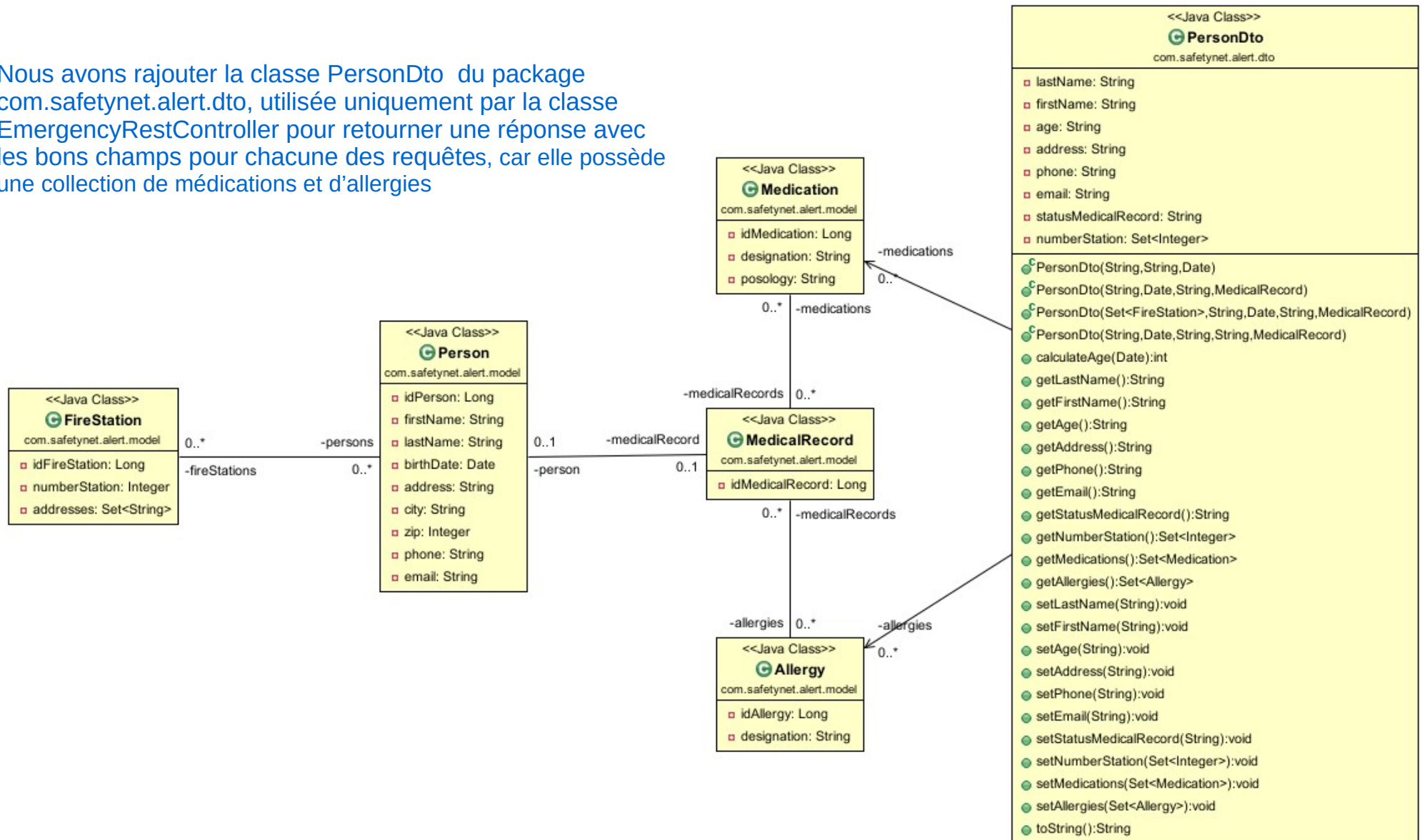
Diagramme de classes

Pour le diagramme de classes, étant donné le nombre important de classes pour l'application, nous l'avons scindé en fonction des différents packages :

- `com.safetynet.alert.model` + la classe `PersonDto` : pour avoir une représentation des différents modèles
- `com.safetynet.alert.controller` + `service` + `repository` : pour avoir une vue sur la conception et les relations entre les différentes couches
- `com.safetynet.alert` + `configuration` + `database` : pour avoir une compréhension globale du démarrage de l'application
- `Com.safetynet.alert.exceptions` : pour comprendre le principe de capture des exceptions dans l'application

com.safetynet.alert.model

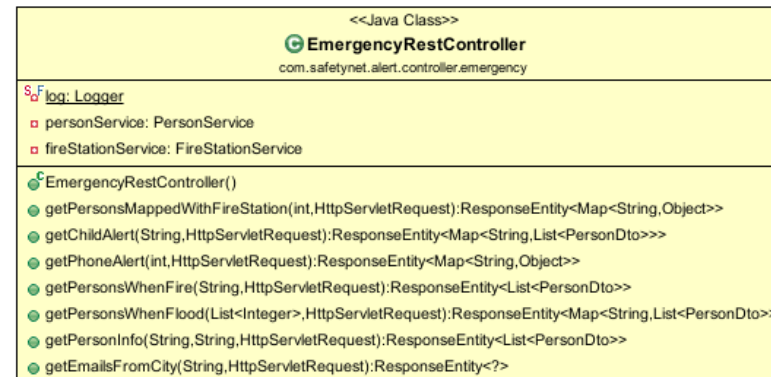
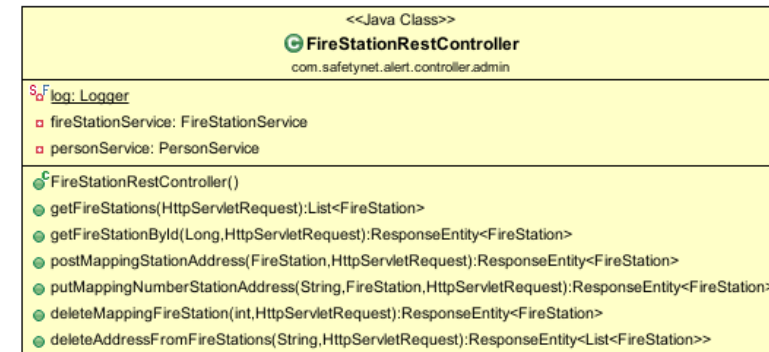
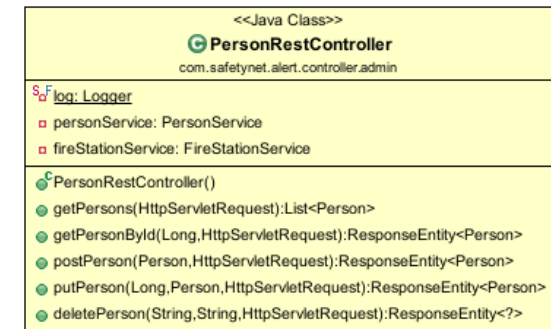
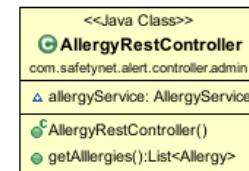
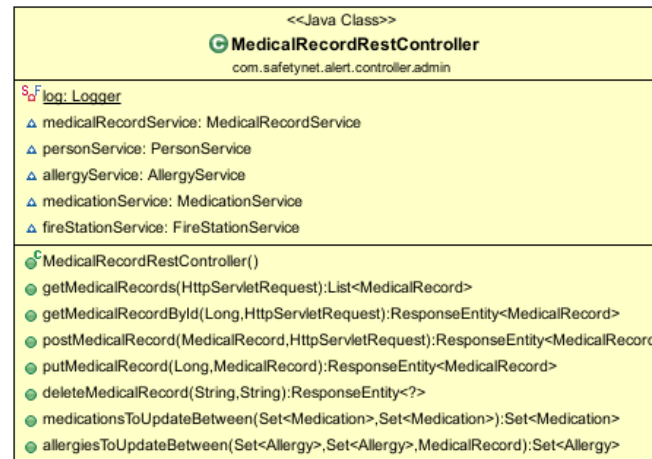
Nous avons rajouter la classe PersonDto du package com.safetynet.alert.dto, utilisée uniquement par la classe EmergencyRestController pour retourner une réponse avec les bons champs pour chacune des requêtes, car elle possède une collection de médicaments et d'allergies



Com.safetynet.alert.controller+service+repository

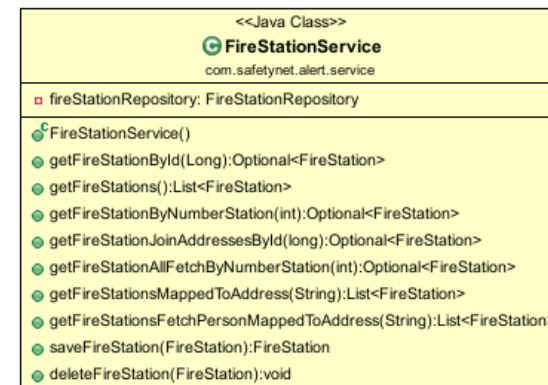
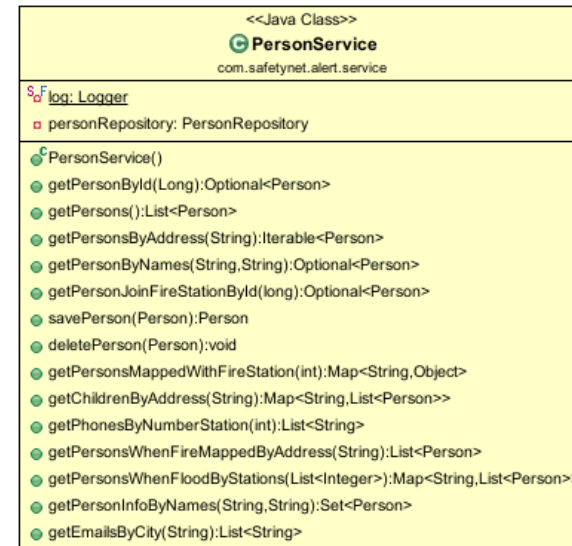
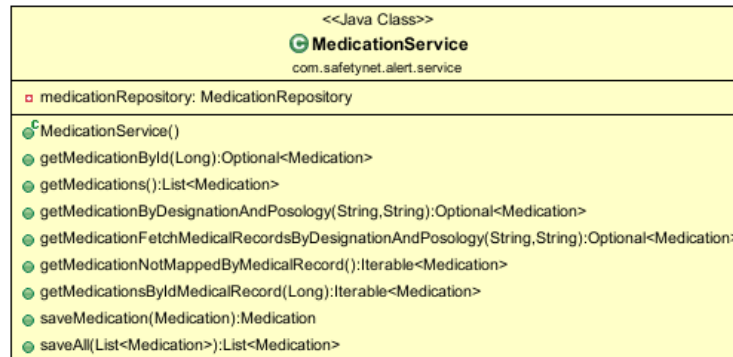
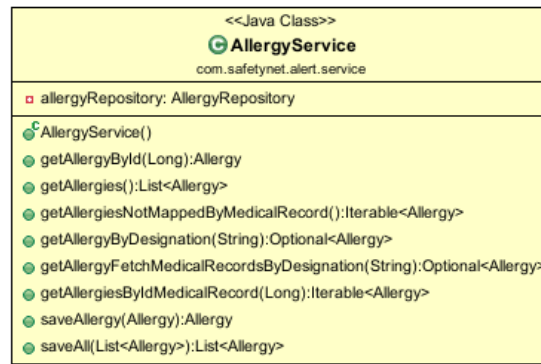
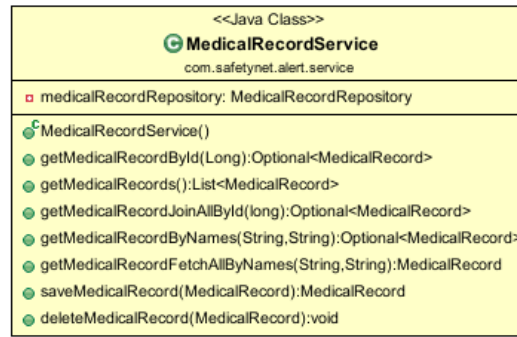
Tout d'abord, les classes contrôleurs permettant de gérer les requêtes et réponses pour chaque endpoint des deux API :

- administration API (package admin)
- emergency API (package emergency)



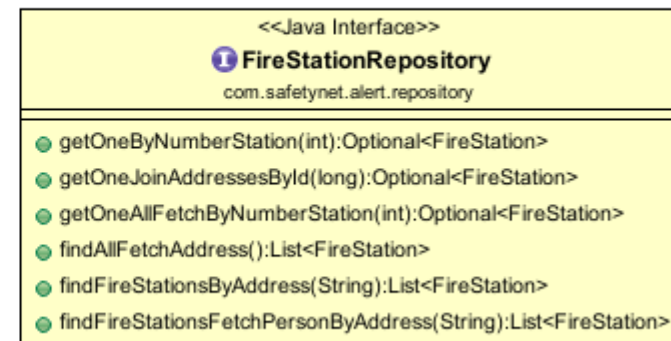
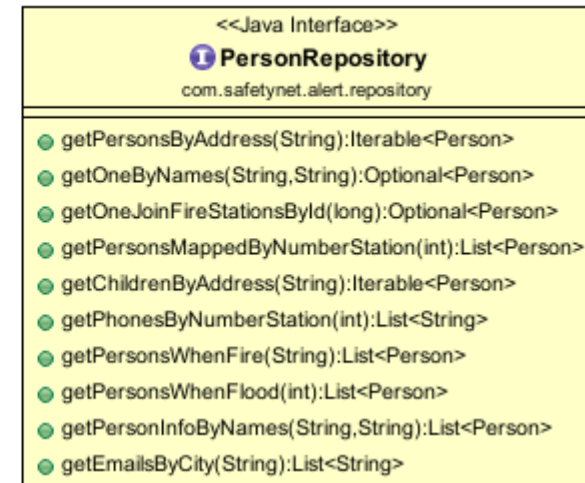
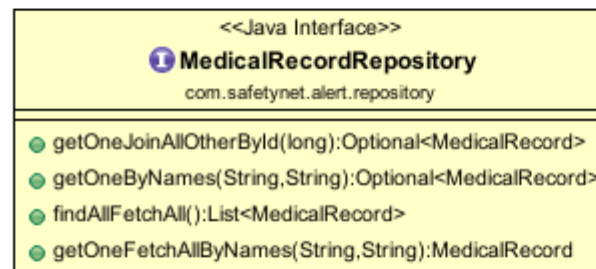
Com.safetynet.alert.controller+service+repository

Les classes services permettant de traiter la logique métier de l'application

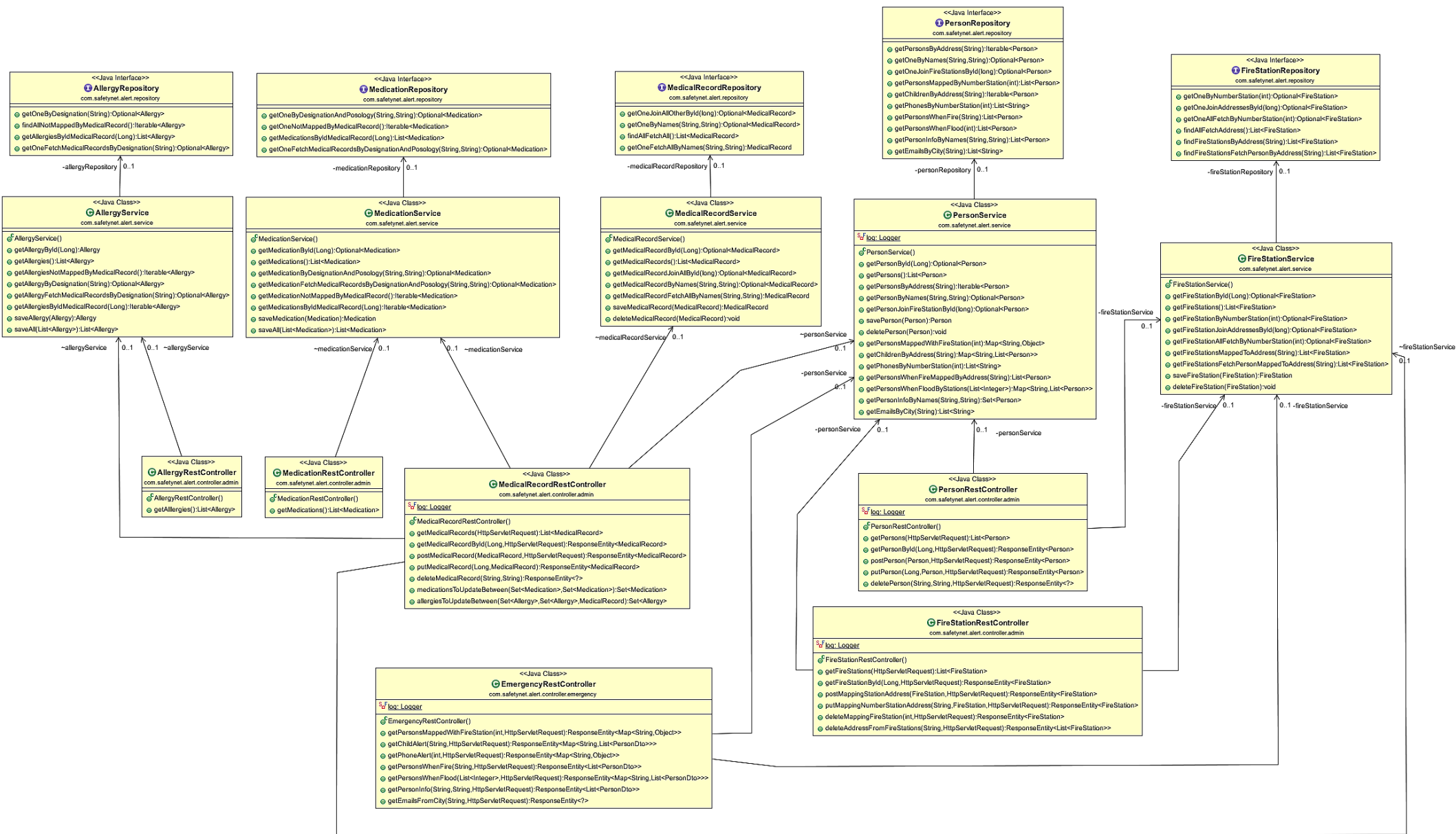


Com.safetynet.alert.controller+service+repository

Les interfaces Repository héritant du JpaRepository permettant de gérer la pertinence des données



Com.safetynet.alert.controller+service+repository



com.safetynet.alert+configuration+database

La classe Main de l'application n'est autre que **SafetyNetApplication**.

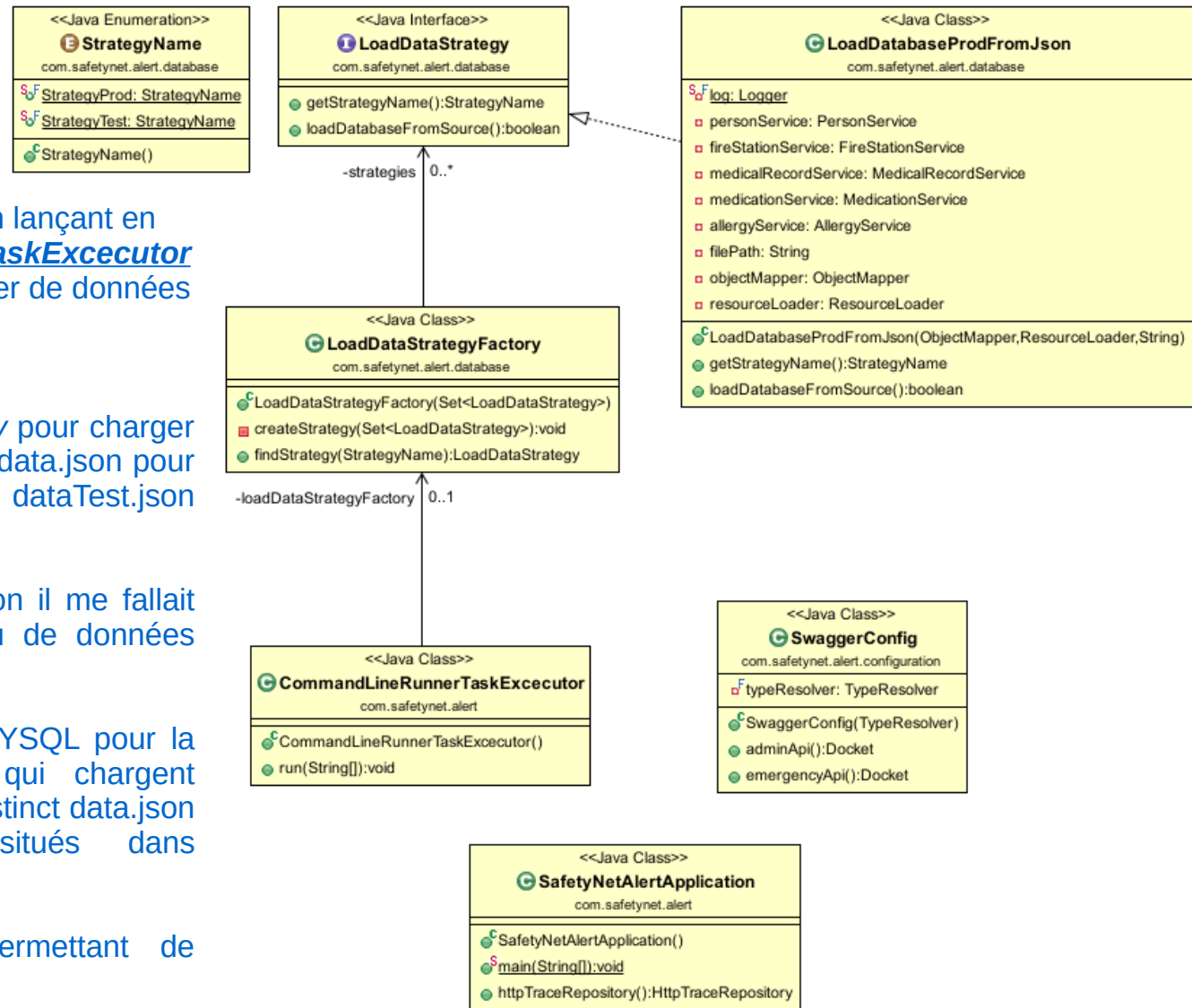
C'est elle qui démarrera l'application mais en lançant en premier la classe **CommandLineRunnerTaskExcecutor** qui commencera d'abord par charger le fichier de données essentiel à l'application en utilisant la classe **LoadDataStrategyFactory**

Nous avons utilisé ce design pattern *Factory* pour charger au démarrage de l'application soit le fichier data.json pour un environnement production soit le fichier dataTest.json pour un environnement test.

En effet , ayant créer des tests d'intégration il me fallait utiliser une base de donnée avec un jeu de données particulier.

Nous avons donc une base de donnée MYSQL pour la production et une H2 pour les test qui chargent respectivement deux fichiers de données distinct data.json et dataTest.json tous les deux situés dans « src/resources/json »

A noter, la classe **SwaggerConfig** permettant de configurer la documentation des API

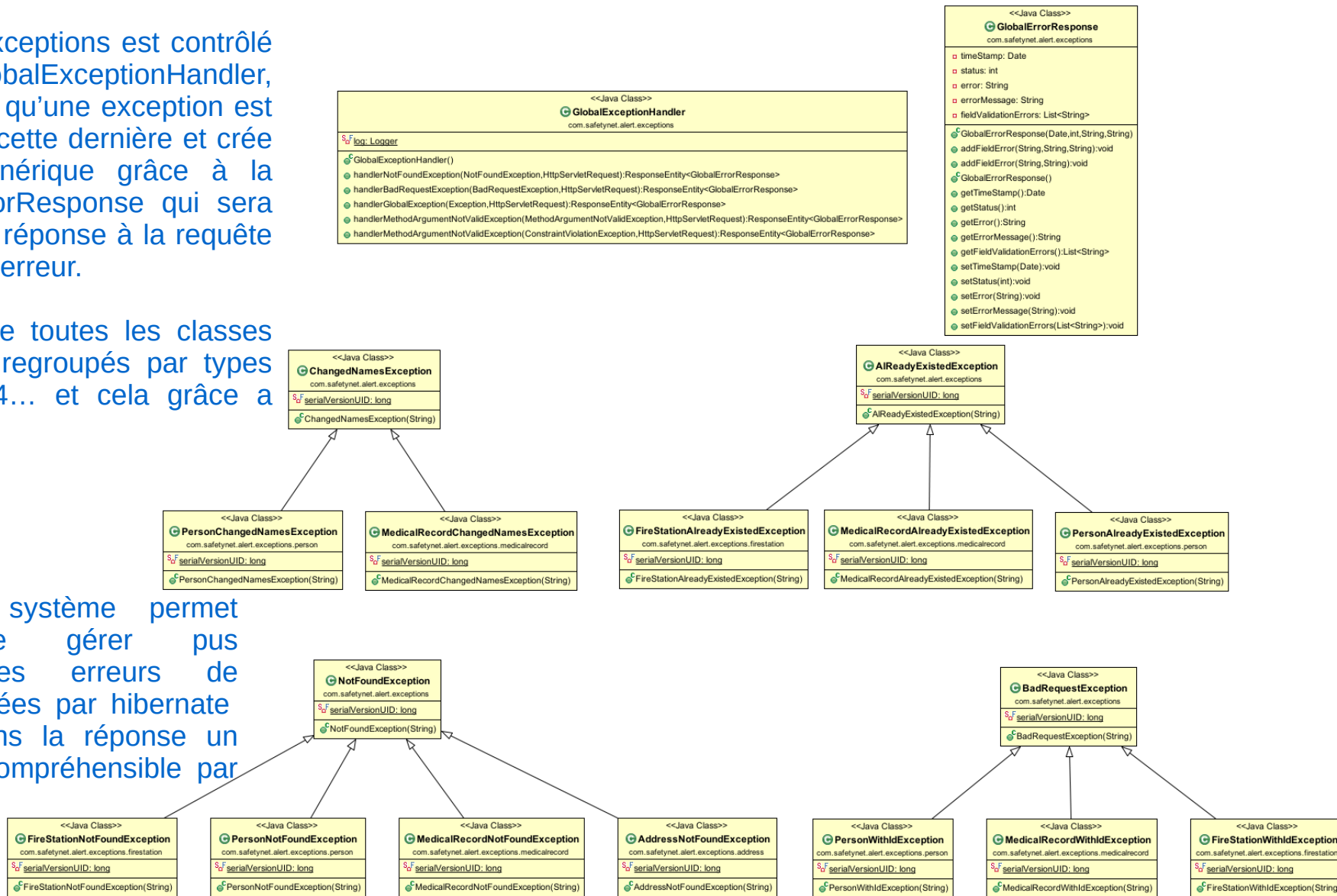


com.safetynet.exceptions

La gestion des exceptions est contrôlé par la classe `GlobalExceptionHandler`, qui a chaque fois qu'une exception est lancée, récupère cette dernière et crée une réponse générique grâce à la classe `GlobalErrorResponse` qui sera renvoyée dans la réponse à la requête ayant produit une erreur.

Il est à noter que toutes les classes d'exception sont regroupées par types d'erreur 400, 404... et cela grâce à l'héritage de java

De plus, ce système permet également de gérer plus efficacement les erreurs de validation détectées par hibernate et d'afficher dans la réponse un message plus compréhensible par l'humain



Modèle Physique de Données

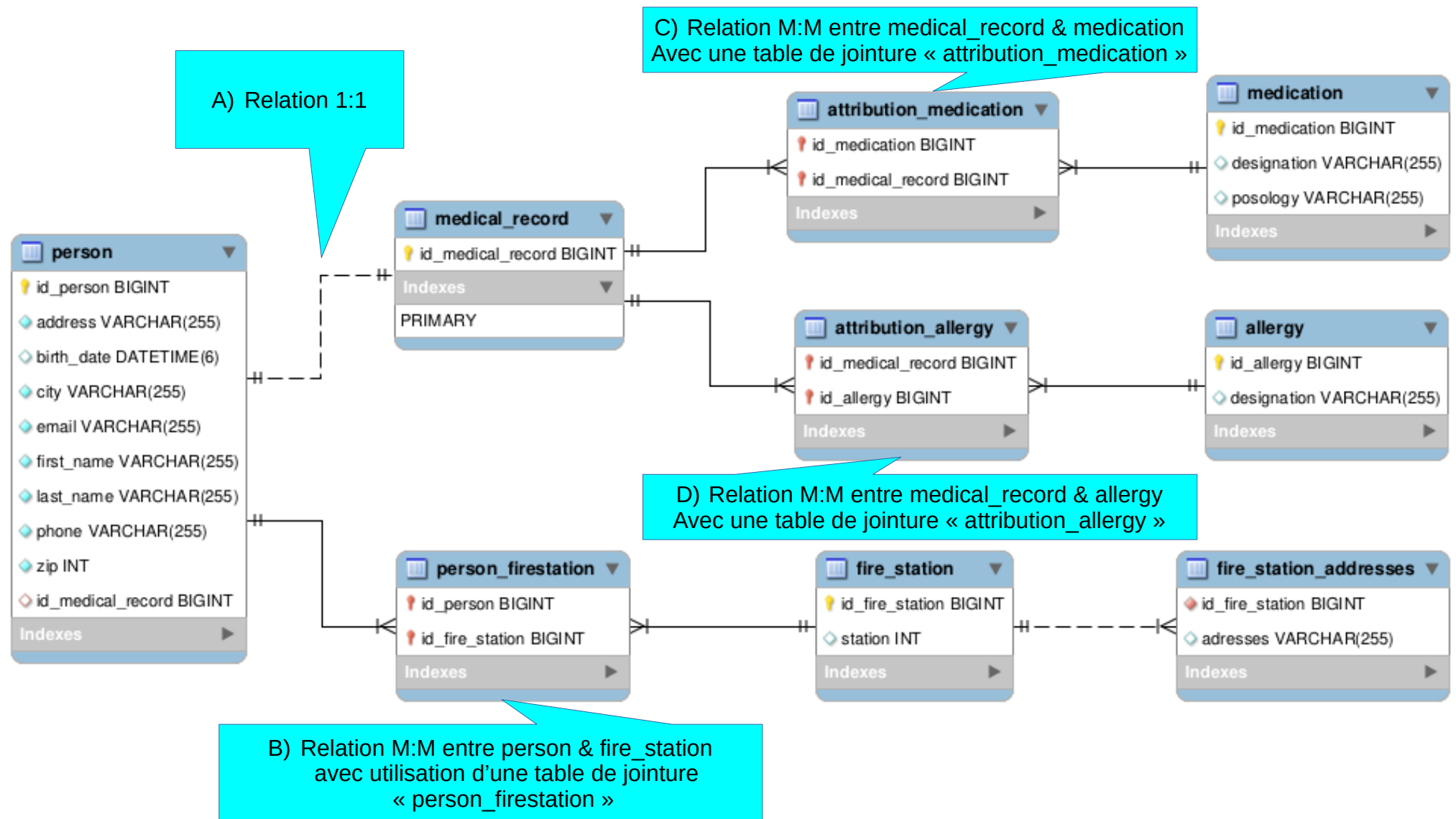
J'ai voulu définir le MPD pour mettre en avant et expliquer les choix que j'ai fait pour déterminer les relations entre chaque entité.

Cela nous permettra également de voir le tables de jointures

A partir de ce MPD, nous verrons l'implémentation du code et notamment les cas particuliers auxquels j'ai pu être confronté:

- Lazy initialization error to fetch collections
- Detached entities when persisting data

Modèle Physique de Données



Relation One To One Person/MedicalRecord

A) Relation bidirectionnelle One to One entre Person et MedicalRecord

En effet une personne ne peut avoir qu'un dossier médical et un dossier est relié à une et une seule personne

Implémentation en java :

```
39 @Entity
40 @JsonPropertyOrder({"idMedicalRecord",
41                     "person",
42                     "medications",
43                     "allergies"})
44 public class MedicalRecord {
45
46
47     @OneToOne(mappedBy = "medicalRecord",
48               cascade = {CascadeType.DETACH,
49                           CascadeType.MERGE,
50                           CascadeType.REFRESH,
51                           CascadeType.PERSIST})
52     @JsonManagedReference(value = "person_medicalRecord")
53     @ApiModelProperty(notes = "Person owner of Medicalrecord")
54     private Person person;
```

```
54 @Entity
55 @ApiModel
56 public class Person {
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142 @OneToOne(cascade = CascadeType.ALL)
143 @JoinColumn(name = "idMedicalRecord",
144             referencedColumnName = "idMedicalRecord")
145 @JsonBackReference(value = "person_medicalRecord")
146 @ApiModelProperty(notes = "MedicalRecord of Person")
147
148 private MedicalRecord medicalRecord;
```

On utilise un CascadeType.ALL dans la relation Person → MedicalRecord pour pouvoir supprimer automatiquement un MedicalRecord affecté à une Personne lorsqu'on supprime cette dernière.

Par contre dans MedicalRecord CascadeType.REMOVE n'est pas présent car si on supprime un dossier médical, on ne supprime pas la personne anciennement associé à ce dernier.

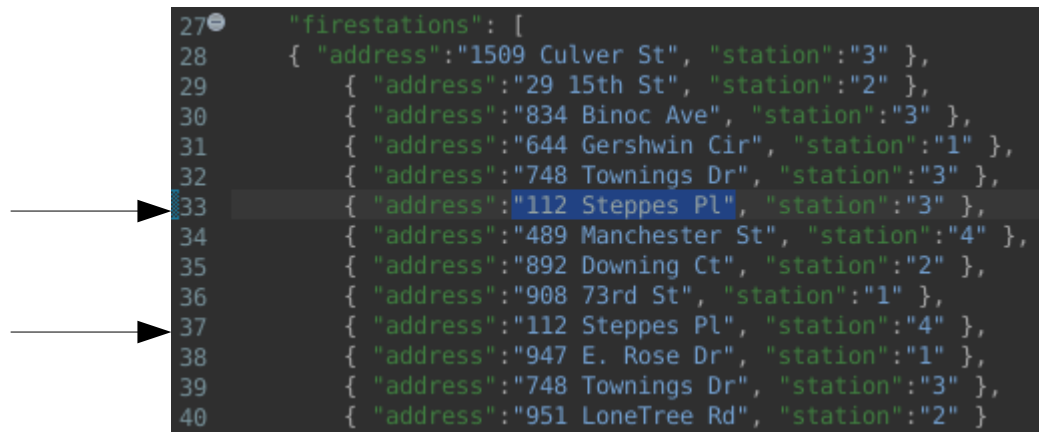
Par défaut , il est à noter aussi que pour la relation Person/MedicalRecord ,hibernate utilise un lazy fetching strategy.

Relation Many To Many Person/FireStation

B) Relation bidirectionnelle Many To Many entre Person et FireStation

En effet, par définition je pensais qu'une personne était affectée a une seule caserne de pompier et qu'une caserne de pompier s'occupait de une ou plusieurs personnes donc une relation one to many...

Cependant en regardant bien le fichier de donnée , on remarque que l' adresse "112 Steppes Pl" est affecté a deux casernes de pompiers numéro 3 et 4



```
27  "firestations": [  
28    { "address": "1509 Culver St", "station": "3" },  
29    { "address": "29 15th St", "station": "2" },  
30    { "address": "834 Binoc Ave", "station": "3" },  
31    { "address": "644 Gershwin Cir", "station": "1" },  
32    { "address": "748 Townings Dr", "station": "3" },  
33    { "address": "112 Steppes Pl", "station": "3" },  
34    { "address": "489 Manchester St", "station": "4" },  
35    { "address": "892 Downing Ct", "station": "2" },  
36    { "address": "908 73rd St", "station": "1" },  
37    { "address": "112 Steppes Pl", "station": "4" },  
38    { "address": "947 E. Rose Dr", "station": "1" },  
39    { "address": "748 Townings Dr", "station": "3" },  
40    { "address": "951 LoneTree Rd", "station": "2" }
```

Une personne peut donc être associée a plusieurs casernes : Par conséquence, la relation one to many devient une relation Many to Many

Relation Many To Many Person/FireStation

B) Relation bidirectionnelle Many to Many entre Person et MedicalRecord

Implémentation en java :

```
54 @Entity
55 @ApiModel
56 public class Person {
57
```

```
151 @ManyToMany(fetch = FetchType.LAZY,
152             cascade = {CascadeType.DETACH,
153                       CascadeType.MERGE,
154                       CascadeType.PERSIST,
155                       CascadeType.REFRESH})
156 @JoinTable(
157     name = "person_firestation",
158     joinColumns = {@JoinColumn(name = "idPerson")},
159     inverseJoinColumns = {@JoinColumn(name = "idFireStation")})
160 @JsonIgnore
161 @ApiModelProperty(notes = "List of FireStations mapped with address of Person")
162 private Set<FireStation> fireStations = new HashSet<>();
```

```
46 @Table(name = "FireStation",
47         uniqueConstraints = @UniqueConstraint(columnNames = {"idFireStation",
48                                                         "station"}))
49 public class FireStation {
```

```
73 @ManyToMany(fetch = FetchType.LAZY,
74             cascade = {CascadeType.DETACH,
75                       CascadeType.MERGE,
76                       CascadeType.REFRESH})
77 @JoinTable(
78     name = "person_firestation",
79     joinColumns = @JoinColumn(name = "idFireStation"),
80     inverseJoinColumns = @JoinColumn(name = "idPerson"))
81 @JsonIgnore
82 @ApiModelProperty(notes = "List of Persons managed by FireStation")
83 private Set<Person> persons = new HashSet<>();
```

CascadeType.PERSIST dans la relation FireStation → Person , a été enlevé pour pouvoir persister une caserne de pompier sans persister la (les) personne(s) associée(s)

Dans Person et FireStation, **CascadeType.REMOVE** n'est pas présent ainsi si on supprime une personne, on ne supprime pas la caserne de pompier anciennement associé à ce dernier et vice versa.

Il est à noter aussi que pour la relation Person/FireStation ,on utilise un **lazy fetching strategy** pour éviter d'aller chercher toutes les collections de Firestation inutilement par exemple lorsque l'on veut récupérer une personne et vice versa.

De plus, on remarquera que pour cette relation, on dispose d'une **nouvelle table de jointure** : «**person_firestation** »

Relation Many To Many MedicalRecord/Medication

C) Relation bidirectionnelle Many to Many entre MedicalRecord et Medication

Un dossier médical peut contenir plusieurs médicaments et une médication peut appartenir à plusieurs dossiers médical

Implémentation en java :

```
39 @Entity
40 @JsonPropertyOrder({"idMedicalRecord",
41                     "person",
42                     "medications",
43                     "allergies"})
44 public class MedicalRecord {
45
```

```
67 @ManyToMany(
68     fetch = FetchType.LAZY,
69     cascade = {CascadeType.DETACH,
70               CascadeType.MERGE,
71               CascadeType.PERSIST,
72               CascadeType.REFRESH})
73 @JoinTable(
74     name = "attribution_medication",
75     joinColumns = {@JoinColumn(name = "idMedicalRecord")},
76     inverseJoinColumns = {@JoinColumn(name = "idMedication")}
77 // @OrderBy("idMedication") // to impose jsonPath to be ordered by id
78 @ApiModelProperty(notes = "list of medications in MedicalRecord")
79
80 private Set<Medication> medications = new HashSet<>();
```

```
37 @Entity
38 @JsonPropertyOrder({"idMedication",
39                     "designation",
40                     "posology"})
41 public class Medication {
```

```
60 @ManyToMany(
61     fetch = FetchType.LAZY,
62     cascade = {CascadeType.DETACH,
63               CascadeType.MERGE,
64               CascadeType.REFRESH})
65 @JoinTable(
66     name = "attribution_medication",
67     joinColumns = @JoinColumn(name = "idMedication"),
68     inverseJoinColumns = @JoinColumn(name = "idMedicalRecord"))
69 @JsonIgnore
70
71 private Set<MedicalRecord> medicalRecords = new HashSet<>();
```

CascadeType.PERSIST dans la relation Medication → MedicalRecord , a été enlevé pour pouvoir persister une médication sans persister le (les) dossier(s) médical(aux)

Dans Medication et MedicalRecord, **CascadeType.REMOVE** n'est pas présent ainsi si on supprime un dossier médical, on ne supprime pas le(les) médicaments anciennement associé à ce dernier et vice versa.

Il est à noter aussi que pour la relation MedicalRecord/Medication ,on utilise aussi un lazy fetching strategy pour éviter d'aller chercher toutes les collections inutilement.

De plus, pour créer cette relation, nous avons **une table de jointure : « attribution_medication »**

Relation Many To Many MedicalRecord/Allergy

D) Relation bidirectionnelle Many to Many entre MedicalRecord et Allergy

Un dossier médical peut contenir plusieurs allergies et une allergie peut appartenir à plusieurs dossiers médicaux

Implémentation en java :

```
39 @Entity
40 @JsonPropertyOrder({"idMedicalRecord",
41                     "person",
42                     "medications",
43                     "allergies"})
44 public class MedicalRecord {
45
```

```
83 @ManyToMany(
84     fetch = FetchType.LAZY,
85     cascade = {CascadeType.DETACH,
86               CascadeType.MERGE,
87               CascadeType.PERSIST,
88               CascadeType.REFRESH})
89 @JoinTable(
90     name = "attribution_allergy",
91     joinColumns = {@JoinColumn(name = "idMedicalRecord")},
92     inverseJoinColumns = {@JoinColumn(name = "idAllergy")})
93 // @OrderBy("idAllergy") // to impose jsonPath to be ordered by id
94 @ApiModelProperty(notes = "list of allergies in Medicalrecord")
95
96 private Set<Allergy> allergies = new HashSet<>();
```

```
40 @Entity
41 @JsonPropertyOrder({"idAllergy",
42                     "designation"})
43 public class Allergy {
```

```
57 @ManyToMany(
58     fetch = FetchType.LAZY,
59     cascade = {CascadeType.DETACH,
60               CascadeType.MERGE,
61               CascadeType.REFRESH})
62 @JoinTable(
63     name = "attribution_allergy",
64     joinColumns = {@JoinColumn(name = "idAllergy")},
65     inverseJoinColumns = {@JoinColumn(name = "idMedicalRecord")})
66 @JsonIgnore
67
68 private Set<MedicalRecord> medicalRecords = new HashSet<>();
```

CascadeType.PERSIST dans la relation Allergy → MedicalRecord, a été enlevé pour pouvoir persister une allergie sans persister le (les) dossier(s) médical(aux)

Dans Allergy et MedicalRecord, **CascadeType.REMOVE** n'est pas présent ainsi si on supprime un dossier médical, on ne supprime pas le(les) allergie(s) anciennement associée(s) à ce dernier et vice versa.

Il est à noter aussi que pour la relation MedicalRecord/Allergy, on utilise aussi un lazy fetching strategy pour éviter d'aller chercher toutes les collections inutilement.

De plus, pour créer cette relation, nous avons une table de jointure : « attribution_allergy »

Lazy initialization errors

Dans le projet, toutes les relations entre les entités sont en lazy fetching strategy pour éviter d'aller chercher des collections inutilement lors de requêtes et gagner en performance.

Par ailleurs, j'ai mis l'option `spring.jpa.open-in-view` à `false` pour ne pas avoir de session ouverte lors de requêtes et gérer correctement tous les lazy initialization exceptions

Par conséquence, lorsque cela est nécessaire d'aller chercher une collection bien précise, j'utilise alors des queries spécifiques qui sont implémentées dans chaque repository lorsque j'en ai besoin.

Exemples :

```
44 @Query("select mr from MedicalRecord as mr "  
45       + " left join fetch mr.person as p "  
46       + " left join fetch p.fireStations "  
47       + " left join fetch mr.medications m "  
48       + " left join fetch mr.allergies a "  
49       + " where p.lastName=?1 and p.firstName=?2")  
50 MedicalRecord getOneFetchAllByNames(String lastName, String firstName);
```

Ici pour récupérer un dossier médical avec comme identifiant le nom et prénom de la personne, comme j'ai besoin par la suite de toutes les casernes de pompiers associés à la personne et de toutes les médications et allergies du dossier médical.

J'utilise dans mes queries JPQL, le mot clef `fetch` pour aller chercher ces collections en entier.

Remarque : le premier fetch utilisé est obligatoire pour pouvoir récupérer les casernes de pompiers de la personne...

Detached entity passed to persist

Dans le projet, pour les méthodes POST, notamment pour modifier un dossier médical, j'ai été plusieurs fois confronté à cette erreur. De part les relations définies entre les entités, lorsque je persistais un nouveau dossier médical pour une personne déjà existante dans la base de donnée.

Pour remédier à cela :

Je n'ai pas enlever le CascadeType.PERSIST dans MedicalRecord car j'en ai besoin lorsque la personne n'existe pas et qu'il faut également la persister...

J'ai opté pour la persistance de la personne existante en premier pour qu'ensuite, grâce au CascadeType.PERSIST dans Person, hibernate persiste ainsi automatiquement le nouveau dossier médical

```
264 //save Person and medicalRecord
265 if (alreadyExistedPerson) {
266
267     // use of PersonService to save new MedicalRecord,allergies and medications
268     // with CascadeType.PERSIST
269     // because if we use MedicalRecord.save( new medicalRecord) ,
270     // we have a detached entity with person because it's already in DB
271     personService.savePerson(currentPerson);
272     log.debug("\nUpdate the already existed Person with new MedicalRecord\n");
273
274 } else {
275
276     // check when it's a new person if there is a fireStation
277     // to map with address of Person
278
279     List<FireStation> fireStationsMappedToAddress =
280         fireStationService.getFireStationsFetchPersonMappedToAddress(
281             currentPerson.getAddress());
282
283     if (!fireStationsMappedToAddress.isEmpty()) {
284
285         for (FireStation fireStation : fireStationsMappedToAddress) {
286
287             fireStation.addPerson(currentPerson);
288             // saving fireStation -> person is only updated but
289             // medicalRecord is saved too because of Cascade Persist of Person
290             fireStationService.saveFireStation(fireStation);
291             log.debug("\nAdd Mapped FireStation to new Person\n");
292             log.debug("\nSave new MedicalRecord and new Person\n");
293         }
294
295     } else {
296
297         // use of MedicalRecordService to save new Person, allergies,Medications
298         // with CascadeType.PERSIST
299         medicalRecordService.saveMedicalRecord(savedMedicalRecord);
300         log.debug("\nSave new MedicalRecord and new Person\n");
301     }
```

En Résumé

L'application SafetyNetAlert réponds aux exigences suivantes

- Le serveur démarre et lis le fichier de donnée
- Tous les endpoints sont fonctionnels et regroupés en deux catégories :
 - *Administration*
 - *emergency*
- Les actuators health,info,trace et metrics sont disponibles à l'adresse :
<http://localhost/actuator/>
- Les log permettent de tracer les requêtes et réponses ainsi que les étapes de calcul
- Maven permet d'exécuter les tests d'intégration et unitaire mais aussi de déployer un site dans lequel vous trouverez :
 - *Le rapport de test SureFire*
 - *le rapport de Jacoco pour la couverture de code*
 - *La Javadoc*
- Une documentation des API se trouvent a l'adresse suivante :
<http://localhost:8080/swagger-ui/>

Annexe A

User Stories et leurs critères d'acceptation

Emergency API

Elle expose 7 endpoints permettant de conduire à des informations bien précises sur le statuts des données de l'application et pour chacun d'eux, elle retournera un résultat sous forme de fichier JSON.

emergency-rest-controller API for Emergency's Services		▼
GET	/childAlert	children living in the given address
GET	/communityEmail	Retrieve list of person's email living in the given address
GET	/fire	persons living at address
GET	/firestation/getpersons	Persons mapped by FireStation
GET	/flood/stations	homes mapped by FireStations
GET	/personInfo	Informations about a Person by LastName and Firstname
GET	/phoneAlert	person's phones mapped by FireStation

Nous allons voir pour chaque endpoint, leur cas d'utilisations avec leurs critères d'acceptation dans les slides suivants...

En tant que système de service d'urgence.

Je veux pouvoir récupérer la liste d'enfants (tout individu âgé de 18 ans ou moins) habitant à adresse précise.

Dans le but de connaître le nombre d'enfants et d'adultes habitant à cette adresse

Scénario :

- 1) L'adresse n'existe pas
- 2) Il existe des enfants à l'adresse demandée.
- 3) Il n'y a pas d'enfants à l'adresse demandée.

Étant donné : que l'application SafetyNet Alert a son Emergency API activée.

Lorsque : le système du service d'urgence envoie une requête sur l'URL suivante :

<http://localhost:8080/childAlert?address=<adressechoisie>>

Alors : l'API renvoie au système de service d'urgence une réponse sous forme de fichier JSON

1) Adresse inexistante : L'API retourne une erreur 404 avec le message suivant :

```
1 {
2   "timestamp": "2021-07-27T09:12:04.642+00:00",
3   "status": 404,
4   "error": "Not Found",
5   "errorMessage": "this address : 26 avenue marechal Foch was not found. Please choose a existed address",
6   "fieldValidationErrors": []
7 }
```

2) un liste d'enfants comprenant le prénom et le nom de famille de chaque enfant, son âge et une liste des autres membres du foyer.

```
1 {
2   "children": [
3     {
4       "lastName": "Boyd",
5       "firstName": "Roger",
6       "age": "3"
7     },
8     {
9       "lastName": "Boyd",
10      "firstName": "Tenley",
11      "age": "9"
12    }
13  ],
14  "otherMembers": [
15    {
16      "lastName": "Boyd",
17      "firstName": "Jacob",
18      "age": "32"
19    },
20  ]
21 }
```

3) S'il n'y a pas d'enfant, la liste est vide.

GET

/communityEmail Retrieve list of person's email living in the given address

En tant que système de service d'urgence.

Je veux pouvoir récupérer la liste des adresses mail de tous les habitants d'une ville.

Scénario :

1) La ville n'existe pas ou il n'y a pas d'habitants.

2) La ville existe et des personnes y résident.

Dans le but de leurs envoyer un mail d'alerte.

Étant donné : que l'application SafetyNet Alert a son Emergency API activée

Lorsque : le système du service d'urgence envoie une requête sur l'URL suivante :

<http://localhost:8080/communityEmail?city=<villeChoisie>>

GET

/communityEmail Retrieve list of person's email living in the given address

Alors : l'API renvoie au système de service d'urgence une réponse sous forme de fichier JSON.

1) L'API retourne le message suivant : 1 there is nobody in this city or the city is not indexed in database

2) L'API retourne la liste des mails des habitants de la ville choisie sans doublons et rangés par ordre alphabétique

```
1  [
2    "aly@imail.com",
3    "bstel@email.com",
4    "clivfd@ymail.com",
5    "drk@email.com",
6    "gramps@email.com",
7    "jaboyd@email.com",
8    "jpeter@email.com",
9    "lily@email.com",
10   "reg@email.com",
11   "soph@email.com",
12   "ssanw@email.com",
13   "tcoop@ymail.com",
14   "tenz@email.com",
15   "ward@email.com",
16   "zarc@email.com"
17 ]
```

GET

/fire persons living at address

En tant que système de service d'urgence.

Je veux pouvoir récupérer la liste des habitants vivant à l'adresse donnée ainsi que les numéros de caserne de pompiers la desservant.

Dans le but de les prévenir lors d'un incendie.

Scénario :

- 1) L'adresse n'existe pas
- 2) Il existe des habitants à l'adresse demandée.
- 3) Il n'y a pas d'habitants à l'adresse demandée.

Étant donné : que l'application SafetyNet Alert a son Emergency API activée.

Lorsque : le système du service d'urgence envoie une requête sur l'URL suivante :

<http://localhost:8080/fire?address=<adresseChoisie>>

Alors : l'API renvoie au système de service d'urgence une réponse sous forme de fichier JSON

1) Adresse inexistante : L'API retourne une erreur 404 avec le message suivant :

```

1 {
2   "timestamp": "2021-07-27T09:12:04.642+00:00",
3   "status": 404,
4   "error": "Not Found",
5   "errorMessage": "this address : 26 avenue marechal Foch was not found. Please choose a existed address",
6   "fieldValidationErrors": []
7 }
```

2) L'API retourne la liste des habitants vivant a l'adresse choisie. Elle inclut le nom, le numéro de téléphone, l'âge et les antécédents médicaux (médicaments, posologie et allergies) pour chaque personne et le(s) numéro(s) de caserne de pompiers dont elle dépends.

```

1 {
2   {
3     "lastName": "Boyd",
4     "age": "56",
5     "phone": "841-874-9888",
6     "numberStation": [
7       3,
8       4
9     ],
10    "medications": [
11      {
12        "idMedication": 13,
13        "designation": "aznol",
14        "posology": "200mg"
15      }
16    ],
17    "allergies": [
18      {
19        "idAllergy": 1,
20        "designation": "nillacilan"
21      }
22    ]
23  },
24 }
```

3) L'API renvoie une liste vide

En tant que système de service d'urgence.

Je veux pouvoir récupérer la liste des personnes couvertes par la caserne de pompiers correspondante, ainsi qu'un décompte du nombre d'adultes et d'enfants.

Dans le but de les contacter.

Scénario :

1) La caserne de pompier n'existe pas.

2) La caserne de pompier existe et couvre des personnes

Étant donné : que l'application SafetyNet Alert a son Emergency API activée.

Lorsque : le système du service d'urgence envoie une requête sur l'URL suivante :

<http://localhost:8080/firestation/getpersons?stationNumber=<numéroDeCaserneChoisie>>

NB : modification de l'URL pour dissocier de l'URL: <http://localhost:8080/firestation/{id}>

Alors : l'API renvoie au système de service d'urgence une réponse sous forme de fichier JSON

1) La caserne n'existe pas : l'API retourne un message d'erreur 404 suivant :

```
1 {
2   "timestamp": "2021-07-27T10:52:39.591+00:00",
3   "status": 404,
4   "error": "Not Found",
5   "errorMessage": "FireStation with number_station: 6 was not found! please choose another existed one!",
6   "fieldValidationErrors": []
7 }
```

2) L'API retourne la liste des habitants couvert par la caserne de pompier au numéro demandé. Elle inclut le prénom, nom, adresse, numéro de téléphone de chaque personne et un décompte d'adultes et d'enfants.

```
1 {
2   "AdultCount": 8,
3   "ChildrenCount": 3,
4   "persons": [
5     {
6       "firstName": "John",
7       "lastName": "Boyd",
8       "address": "1509 Culver St",
9       "phone": "841-874-6512"
10    },
11    {
12      "firstName": "Jacob",
13      "lastName": "Boyd",
14      "address": "1509 Culver St",
15      "phone": "841-874-6513"
16    },
17    {
18      "firstName": "Tenley",
19      "lastName": "Boyd",
20      "address": "1509 Culver St",
21      "phone": "841-874-6512"
22    },
23  ]
24 }
```

En tant que système de service d'urgence.

Je veux pouvoir récupérer la liste de tous les foyers couverts par les casernes de pompiers correspondantes.

Dans le but de les contacter.

Scénario :

1) Une des casernes de pompiers dans la liste n'existe pas.

2) Les casernes de pompiers existent et couvrent des personnes

Étant donné : que l'application SafetyNet Alert a son Emergency API activée.

Lorsque : le système du service d'urgence envoie une requête sur l'URL suivante :

<http://localhost:8080/flood/stations?stations=<listeNumérosDeCaserneChoisis>>

Alors : l'API renvoie au système de service d'urgence une réponse sous forme de fichier JSON

1) Une des casernes n'existe pas : l'API retourne un message d'erreur 404 suivant :

```
1 {
2   "timestamp": "2021-07-27T13:51:14.437+00:00",
3   "status": 404,
4   "error": "Not Found",
5   "errorMessage": "the Firestation with numberStation: 6 was not found.Please replace it by existed FireStation",
6   "fieldValidationErrors": []
7 }
```

2) L'API retourne la liste des habitants couverts par les casernes de pompier aux numéros demandés en les regroupant par adresse. Elle inclut le nom, numéro de téléphone, âge et antécédents médicaux(médicaments,posologie et allergies) de chaque personne.

```
1 {
2   "644 Gershwin Cir": [
3     {
4       "lastName": "Duncan",
5       "age": "20",
6       "phone": "841-874-6512",
7       "medications": [],
8       "allergies": [
9         {
10           "idAllergy": 4,
11           "designation": "shellfish"
12         }
13       ]
14     }
15   ],
16   "908 73rd St": [
17     {
18       "lastName": "Walker",
19       "age": "41",
20       "phone": "841-874-8547",
21       "medications": [
22         {
23           "idMedication": 12,
24           "designation": "thradox",
25           "posology": "700mg"
26         }
27       ],
28       "allergies": [
29         {
30           "idAllergy": 6,
31           "designation": "illisoxian"
32         }
33       ]
34     }
35   ],
36 }
```

En tant que système de service d'urgence.

Je veux pouvoir récupérer les informations sur la personne ayant comme nom et prénom, le nom et prénom demandé. Si d'autres personnes portent le même nom , elles doivent aussi apparaître. Les informations doivent comprendre le nom, l'âge, l'adresse e-mail, et les antécédents médicaux.

Dans le but de le(s) contacter.

Scénario :

- 1) La personne ayant le nom et prénom demandé n'existe pas.
- 2) La personne ayant le nom et prénom demandé n'existe pas mais il existe des personnes portant le nom demandé.
- 3) La personne ayant le nom et prénom demandé existe.

Étant donné : que l'application SafetyNet Alert a son Emergency API activée.

Lorsque : le système du service d'urgence envoie une requête sur l'URL suivante :

<http://localhost:8080/personinfo?firstName=<prénom>&lastName=<nom>>

Alors : l'API renvoie au système de service d'urgence une réponse sous forme de fichier JSON

1) l'API retourne un message d'erreur 404 suivant :

```
1 {  
2   "timestamp": "2021-07-27T15:06:37.098+00:00",  
3   "status": 404,  
4   "error": "Not Found",  
5   "errorMessage": "Person with firstName: Dorian and lastName: Delaval was not found.Please choose another names!",  
6   "fieldValidationErrors": []  
7 }
```

2) L'API retourne un message d'erreur 404 suivant :

```
1 {  
2   "timestamp": "2021-07-27T15:10:21.034+00:00",  
3   "status": 404,  
4   "error": "Not Found",  
5   "errorMessage": "Person with firstName: Dorian and lastName: Boyd was not found. But  
6     some persons with this LastName Boyd exist! Please choose another firstName !",  
7   "fieldValidationErrors": []  
8 }
```

- 3) L'API retourne la liste des personnes portant le nom demandé avec en première position la personne dont le prénom correspond au prénom demandé. Cette liste comporte le nom, l'adresse, l'âge, l'e-mail et leur antécédents médicaux(médicaments, posologies et allergies)

```
1  {
2    {
3      "lastName": "Boyd",
4      "age": "37",
5      "address": "1509 Culver St",
6      "email": "jaboyd@email.com",
7      "medications": [
8        {
9          "idMedication": 1,
10         "designation": "aznol",
11         "posology": "350mg"
12       },
13       {
14         "idMedication": 2,
15         "designation": "hydrapermazol",
16         "posology": "100mg"
17       }
18     ],
19     "allergies": [
20       {
21         "idAllergy": 1,
22         "designation": "nillacilan"
23       }
24     ]
25   },
26   {
27     "lastName": "Boyd",
28     "age": "32",
29     "address": "1509 Culver St",
30     "email": "drk@email.com",
31     "medications": [
32       {
33         "idMedication": 3,
34         "designation": "aznol",
35         "posology": "350mg"
36       },
37       {
38         "idMedication": 4,
39         "designation": "hydrapermazol",
40         "posology": "100mg"
41       }
42     ],
43     "allergies": [
44       {
45         "idAllergy": 2,
46         "designation": "nillacilan"
47       }
48     ]
49   }
50 ]
```

En tant que système de service d'urgence.

Je veux pouvoir récupérer la liste des numéros de téléphone des résidents desservis par la caserne de pompiers en indiquant son numéro.

Dans le but d'envoyer des textos d'alerte.

Scénario :

- 1) La caserne de pompier avec le numéro demandé n'existe pas.
- 2) La caserne existe.

Étant donné : que l'application SafetyNet Alert a son Emergency API activée.

Lorsque : le système du service d'urgence envoie une requête sur l'URL suivante :

<http://localhost:8080/phoneAlert?firestation=<numeroCasernePompier>>

Alors : l'API renvoie au système de service d'urgence une réponse sous forme de fichier JSON

1) Caserne inexistante : L'API retourne une erreur 404 avec le message suivant :

```
1 {
2   "timeStamp": "2021-07-27T10:52:39.591+00:00",
3   "status": 404,
4   "error": "Not Found",
5   "errorMessage": "FireStation with number_station: 6 was not found! please choose another existed one!",
6   "fieldValidationErrors": []
7 }
```

2) L'API retourne la liste des numéro de téléphone des habitants desservis par la caserne de pompiers. Cette liste ne contient pas de doublons et est triée

```
1 {
2   "phones": [
3     "841-874-6512",
4     "841-874-6513",
5     "841-874-6544",
6     "841-874-6741",
7     "841-874-6874",
8     "841-874-8888",
9     "841-874-9888"
10  ]
11 }
```

Administration API

Elle expose au total 16 endpoints permettant d'effectuer des mises à jours de données sur chaque entité.

Nous allons voir pour chaque endpoint, leur cas d'utilisations avec leurs critères d'acceptation dans les slides suivants...

fire-station-rest-controller API to Manage FireStations		▼
GET	/firestation	Get all Firestations
POST	/firestation	Creation of FireStation
PUT	/firestation/{address}	Update address/FireStation
GET	/firestation/{id}	Get FireStation by ID
DELETE	/firestation/address/{address}	Delete address of FireStation
DELETE	/firestation/station/{numberStation}	Delete FireStation
medical-record-rest-controller API to Manage MedicalRecord		▼
GET	/medicalRecord	MedicalRecords
POST	/medicalRecord	Create a MedicalRecord
GET	/medicalRecord/{id}	MedicalRecord by Id
PUT	/medicalRecord/{id}	Update MedicalRecord
DELETE	/medicalRecord/{lastName}/{firstName}	Delete MedicalRecord
person-rest-controller API to Manage Person		▼
GET	/person	Persons
POST	/person	Create a new Person
GET	/person/{id}	Person with ID
PUT	/person/{id}	Update an existed Person by giving it's ID
DELETE	/person/{lastName}/{firstName}	Delete an existed Person by giving it's LastName and FirstName

FireStation API

fire-station-rest-controller API to Manage FireStations		▼
GET	/firestation	Get all Firestations
POST	/firestation	Creation of FireStation
PUT	/firestation/{address}	Update address/FireStation
GET	/firestation/{id}	Get FireStation by ID
DELETE	/firestation/address/{address}	Delete address of FireStation
DELETE	/firestation/station/{numberStation}	Delete FireStation

En tant que système d'administration.

Je veux pouvoir récupérer la liste des casernes de pompiers existantes.

Dans le but d' avoir des informations sur ces casernes.

Étant donné : que l'application SafetyNet Alert a son Administration API activée.

Lorsque : le système du service d'urgence envoie une requête sur l'URL suivante :

<http://localhost:8080/firestation>

Alors : L'API retourne sous forme de fichier JSON la liste des casernes de pompiers existantes avec leur numéro, les adresses qu'elle desservent.

```
1  [
2    {
3      "idFireStation": 1,
4      "numberStation": 3,
5      "addresses": [
6        "748 Townings Dr",
7        "1509 Culver St",
8        "834 Binoc Ave",
9        "112 Steppes Pl"
10     ]
11   },
12   {
13     "idFireStation": 2,
14     "numberStation": 2,
15     "addresses": [
16       "951 LoneTree Rd",
17       "892 Downing Ct",
18       "29 15th St"
19     ]
20   },
21   {
22     "idFireStation": 3,
23     "numberStation": 1,
24     "addresses": [
25       "908 73rd St",
26       "947 E. Rose Dr",
27       "644 Gershwin Cir"
28     ]
29   },
30   {
31     "idFireStation": 4,
32     "numberStation": 4,
33     "addresses": [
34       "489 Manchester St",
35       "112 Steppes Pl"
36     ]
37   }
38 ]
```

GET

/firestation/{id} Get FireStation by ID

En tant que système de service d'administration.

Je veux pouvoir récupérer les informations d'une caserne de pompier en indiquant son ID.

Dans le but d'étudier ces informations

Scénario :

1) La caserne de pompier avec l'identifiant demandé n'existe pas.

2) La caserne existe.

Étant donné : que l'application SafetyNet Alert a son Administration API activée.

Lorsque : le système du service d'urgence envoie une requête sur l'URL suivante :

<http://localhost:8080/firestation/{id}>

Alors : l'API renvoie au système d'administration une réponse sous forme de fichier JSON

1) Caserne inexistante : L'API retourne une erreur 404 avec le message suivant :

```
1 {  
2   "timestamp": "2021-07-27T16:08:57.975+00:00",  
3   "status": 404,  
4   "error": "Not Found",  
5   "errorMessage": "FireStation with Id:6 was not found",  
6   "fieldValidationErrors": []  
7 }
```

2) L'API retourne les informations sur la caserne de pompier comme ci dessous :

```
1 {  
2   "idFireStation": 3,  
3   "numberStation": 1,  
4   "addresses": [  
5     "908 73rd St",  
6     "947 E. Rose Dr",  
7     "644 Gershwin Cir"  
8   ]  
9 }
```

En tant que système de service d'administration.

Je veux pouvoir ajouter une nouvelle caserne de pompier avec une(des) adresse(s) qu'elle desservira

Dans le but d'ajouter un mapping caserne/adresse

Scénario :

1) La nouvelle caserne de pompier existe déjà.

2) La nouvelle caserne n'existe pas.

Étant donné : que l'application SafetyNet Alert a son Administration API activée.

Lorsque : le système du service d'urgence envoie une requête sur l'URL suivante :

<http://localhost:8080/firestation>

POST**/firestation** Creation of FireStation

Alors : l'API renvoie au système d'administration une réponse sous forme de fichier JSON

1) Caserne existante : L'API retourne une erreur 400 avec le message suivant :

```
1 {
2   "timeStamp": "2021-07-27T16:30:50.967+00:00",
3   "status": 400,
4   "error": "Bad Request",
5   "errorMessage": "this FireStation with NumberStation: 3 already Existed",
6   "fieldValidationErrors": []
7 }
```

2) L'API retourne les informations sur la caserne de pompier nouvellement créée comme ci dessous :

```
1 {
2   "idFireStation": 5,
3   "numberStation": 10,
4   "addresses": [
5     "26 av maréchal Foch"
6   ]
7 }
```

NB : les personnes ayant comme adresse celles ajoutées a cette nouvelle caserne de pompiers, se verront automatiquement associés à cette dernière de part la relation M:M entre FireStation et Person.

En tant que système de service d'administration.

Je veux pouvoir mettre à jour le numéro de la caserne de pompiers d'une adresse

Dans le but de modifier le mapping caserne/adresse

Scénario :

- 1) Le numéro de la caserne de pompier n'existe pas.
- 2) La caserne a déjà l'adresse donnée d'enregistrée.
- 3) Le numéro de caserne existe et l'adresse n'est pas enregistrée.

Étant donné : que l'application SafetyNet Alert a son Administration API activée.

Lorsque : le système du service d'urgence envoie une requête sur l'URL suivante :

<http://localhost:8080/firestation/{adresse}>

Alors : l'API renvoie au système d'administration une réponse sous forme de fichier JSON

1) Le numéro de la caserne n'existe pas : L'API retourne une erreur 404 avec le message suivant :

```

1  {
2    "timestamp": "2021-07-28T12:00:32.246+00:00",
3    "status": 404,
4    "error": "Not Found",
5    "errorMessage": "fireStation given in body request with numberStation:10 doesn't exist !",
6    "fieldValidationErrors": []
7  }

```

2) Adresse déjà enregistrée : L'API retourne un message d'erreur 400 suivant :

```

1  {
2    "timestamp": "2021-07-28T12:21:20.072+00:00",
3    "status": 400,
4    "error": "Bad Request",
5    "errorMessage": "This FireStation already mapped with given address.Please give a another fireStation to map !",
6    "fieldValidationErrors": []
7  }

```

3) L'API retourne la caserne de pompier nouvellement modifiée avec sa nouvelle adresse :

```

1  {
2    "idFireStation": 2,
3    "numberStation": 2,
4    "addresses": [
5      "951 LoneTree Rd",
6      "26 av marechal foch",
7      "892 Downing Ct",
8      "29 15th St"
9    ]
10 }

```

NB: les personnes ayant comme adresse celle ajoutée a cette caserne de pompiers, se verront automatiquement associés à cette dernière de part la relation M:M entre FireStation et Person.

En tant que système de service d'administration.

Je veux pouvoir supprimer une adresse associée a une(des) caserne(s) de pompiers.

Dans le but d' enlever le mapping caserne(s)/adresse donnée

Scénario :

- 1)L'adresse n'est associée a aucune caserne
- 2)L'adresse est associé a une (des) caserne(s) de pompier.

Étant donné : que l'application SafetyNet Alert a son Administration API activée.

Lorsque : le système du service d'urgence envoie un requête sur l'URL suivante :

<http://localhost:8080/firestation/address/{adresseASupprimer}>

Alors : l'API renvoie au système d'administration une réponse sous forme de fichier JSON

1) L'adresse n'est pas associée a une caserne : L'API retourne une erreur 404 avec le message suivant :

```
1 {
2   "timestamp": "2021-07-28T15:29:53.775+00:00",
3   "status": 404,
4   "error": "Not Found",
5   "errorMessage": "There is no FireStation mapped with this address:26 av marechal foch",
6   "fieldValidationErrors": []
7 }
```

2) Adresse associée à une (des) caserne(s) : L'API retourne la liste des casernes modifiées par la suppression de l'adresse :

```
1 [
2   {
3     "idFireStation": 1,
4     "numberStation": 3,
5     "addresses": [
6       "748 Townings Dr",
7       "1509 Culver St",
8       "834 Binoc Ave"
9     ]
10  },
11  {
12    "idFireStation": 4,
13    "numberStation": 4,
14    "addresses": [
15      "489 Manchester St"
16    ]
17  }
18 ]
```

NB : La suppression de l'adresse dans la (les) caserne(s), implique de manière automatique, la suppression de l'association Person/FireStation pour les personnes habitant cette dite adresse

En tant que système de service d'administration.

Je veux pouvoir supprimer la(les) adresse(s) associée(s) a une caserne de pompiers spécifiée par son numéro.

Dans le but d' enlever le mapping caserne/adresse(s)

Scénario :

1) La caserne de pompier n'existe pas

2) La caserne existe.

Étant donné : que l'application SafetyNet Alert a son Administration API activée.

Lorsque : le système du service d'urgence envoie une requête sur l'URL suivante :

<http://localhost:8080/firestation/station/{numéroCaserne}>

Alors : l'API renvoie au système d'administration une réponse sous forme de fichier JSON

1) La caserne n'existe pas: L'API retourne une erreur 404 avec le message suivant :

```
1  {
2    "timeStamp": "2021-07-28T15:53:12.260+00:00",
3    "status": 404,
4    "error": "Not Found",
5    "errorMessage": "FireStation with NumberStation:6 was not found",
6    "fieldValidationErrors": []
7  }
```

2) L'API retourne la caserne modifiée par la suppression de toutes ses adresses associées :

```
1  {
2    "idFireStation": 4,
3    "numberStation": 4,
4    "addresses": []
5  }
```

NB : La suppression des adresses de la caserne implique de manière automatique, la suppression de l'association Person/FireStation pour les personnes qui étaient associées a cette caserne de pompiers

MedicalRecord API

medical-record-rest-controller Api to Manage MedicalRecord



GET `/medicalRecord` MedicalRecords

POST `/medicalRecord` Create a MedicalRecord

GET `/medicalRecord/{id}` MedicalRecord by Id

PUT `/medicalRecord/{id}` Update MedicalRecord

DELETE `/medicalRecord/{lastName}/{firstName}` Delete MedicalRecord

En tant que système d'administration.

Je veux pouvoir récupérer la liste des carnets de santé de chaque personne

Dans le but d' avoir des informations sur ces carnets.

Étant donné : que l'application SafetyNet Alert a son Administration API activée.

Lorsque : le système du service d'urgence envoie une requête sur l'URL suivante :

<http://localhost:8080/medicalRecord>

Alors : L'API retourne sous forme de fichier JSON la liste des carnets de santé avec les antécédents médicaux(médicaments, posologies et allergies)

```
1  [
2  {
3      "idMedicalRecord": 1,
4      "person": {
5          "idPerson": 1,
6          "firstName": "John",
7          "lastName": "Boyd",
8          "birthDate": "03/05/1984",
9          "address": "1509 Culver St",
10         "city": "Culver",
11         "zip": 97451,
12         "phone": "841-874-6512",
13         "email": "jaboyd@email.com"
14     },
15     "medications": [
16         {
17             "idMedication": 1,
18             "designation": "aznol",
19             "posology": "350mg"
20         },
21         {
22             "idMedication": 2,
23             "designation": "hydrapermazol",
24             "posology": "100mg"
25         }
26     ],
27     "allergies": [
28         {
29             "idAllergy": 1,
30             "designation": "nillacilan"
31         }
32     ]
33 },
34 {
35     "idMedicalRecord": 2,
```


GET

/medicalRecord/{id} MedicalRecord by Id

En tant que système de service d'administration.

Je veux pouvoir récupérer les informations d'un carnet de santé en indiquant son ID.

Dans le but d'étudier ces informations

Scénario :

1)Le carnet de santé avec l'identifiant demandé n'existe pas.

2)Le carnet de santé existe.

Étant donné : que l'application SafetyNet Alert a son Administration API activée.

Lorsque : le système du service d'urgence envoie une requête sur l'URL suivante :

<http://localhost:8080/medicalRecord/{id}>

Alors : l'API renvoie au système d'administration une réponse sous forme de fichier JSON

1) Carnet inexistante : L'API retourne une erreur 404 avec le message suivant :

```
1 {
2   "timestamp": "2021-07-28T17:27:53.995+00:00",
3   "status": 404,
4   "error": "Not Found",
5   "errorMessage": "MedicalRecord with id:56 was not found!",
6   "fieldValidationErrors": []
7 }
```

2) L'API retourne les informations sur le carnet de santé comme ci dessous :

```
1 {
2   "idMedicalRecord": 10,
3   "person": {
4     "idPerson": 10,
5     "firstName": "Tony",
6     "lastName": "Cooper",
7     "birthDate": "03/05/1994",
8     "address": "112 Steppes Pl",
9     "city": "Culver",
10    "zip": 97451,
11    "phone": "841-874-6874",
12    "email": "tcoop@ymail.com"
13  },
14  "medications": [
15    {
16      "idMedication": 8,
17      "designation": "dodoxadin",
18      "posology": "30mg"
19    },
20    {
21      "idMedication": 7,
22      "designation": "hydrapermazol",
23      "posology": "300mg"
24    }
25  ],
26  "allergies": [
27    ,
28  ]
29 }
```

En tant que système de service d'administration.

Je veux pouvoir ajouter un dossier médical

Dans le but d' enregistrer les informations de santé d'une personne.

Scénario :

- 1) La personne du nouveau dossier médical existe et a déjà un dossier existant.
- 2) La personne du nouveau dossier médical existe mais n'a pas de dossier existant.
- 3) La personne du nouveau dossier médical n'existe pas.

Étant donné : que l'application SafetyNet Alert a son Administration API activée.

Lorsque : le système du service d'urgence envoie une requête sur l'URL suivante :

<http://localhost:8080/medicalRecord>

Alors : l'API renvoie au système d'administration une réponse sous forme de fichier JSON

1) La personne existe et a déjà un dossier médical. L'API retourne une erreur 400 avec le message suivant :

```

1 {
2   "timestamp": "2021-07-29T08:36:02.225+00:00",
3   "status": 400,
4   "error": "Bad Request",
5   "errorMessage": "MedicalRecord for this person already exist! Please chose another Person to map with",
6   "fieldValidationErrors": []
7 }
```

2) La personne existe mais n'a pas de dossier médical : L'API retourne une réponse 201 avec le message suivant :

```

1 {
2   "idMedicalRecord": 24,
3   "person": {
4     "idPerson": 24,
5     "firstName": "Boris",
6     "lastName": "Delaval",
7     "address": "1509 Culver St",
8     "city": "Cassis",
9     "zip": 13260,
10    "phone": "061-846-0160",
11    "email": "delaval.https@email.com"
12  },
13  "medications": [
14    {
15      "idMedication": 19,
16      "designation": "medication1",
17      "posology": "350mg"
18    },
19    {
20      "idMedication": 20,
```

NB : On part du principe que même si les champs, autre que lastname et firstname, ont été modifié dans le body de la requete, ils ne seront pas enregistrés car par définition, un post est la création d'un nouveau MedicalRecord et non sa modification.

3) La personne du dossier médical n'existe pas : L'API retourne les informations sur le carnet de santé comme ci dessous :

NB :

La personne n'existant pas mais ses données sont renseignées, elle sera donc créée et enregistrée. Le mapping FireStation/address sera mis à jour.

Concernant les médicaments et allergies, si ceux sont de nouvelles données, elles seront également créées et enregistrées, sinon le mapping médicament,allergie/MedicalRecord sera simplement mis à jour

```
1  {
2    "idMedicalRecord": 25,
3    "person": {
4      "idPerson": 25,
5      "firstName": "Dorian",
6      "lastName": "Delaval",
7      "address": "1509 Culver St",
8      "city": "Cassis",
9      "zip": 13260,
10     "phone": "061-846-0160",
11     "email": "delaval.https@email.com"
12   },
13   "medications": [
14     {
15       "idMedication": 21,
16       "designation": "medication2",
17       "posology": "100mg"
18     },
19     {
20       "idMedication": 19,
21       "designation": "medication1",
22       "posology": "350mg"
23     }
24   ],
25   "allergies": [
26     {
27       "idAllergy": 1,
28       "designation": "nillacilan"
29     }
30   ]
31 }
```

En tant que système de service d'administration.

Je veux pouvoir mettre à jour un dossier médical existant

Dans le but de mettre à jour les informations de santé d'une personne.

Scénario :

- 1) L'id du dossier médical n'existe pas.
- 2) Le nom et le prénom du dossier médical ne sont pas les mêmes.
- 3) Le nom et prénom du dossier médical sont les mêmes.

Étant donné : que l'application SafetyNet Alert a son Administration API activée.

Lorsque : le système du service d'urgence envoie une requête sur l'URL suivante :

<http://localhost:8080/medicalRecord/{id}>

Alors : l'API renvoie au système d'administration une réponse sous forme de fichier JSON

1) L'Id dossier médical n'existe pas. L'API retourne une erreur 404 avec le message suivant :

```
1  {
2    "timestamp": "2021-07-28T17:27:53.995+00:00",
3    "status": 404,
4    "error": "Not Found",
5    "errorMessage": "MedicalRecord with id:56 was not found!",
6    "fieldValidationErrors": []
7  }
```

2) Le nom et prénom du dossier médical sont différents ou non renseignés : L'API retourne une réponse 400 avec le message suivant :

```
1  {
2    "timestamp": "2021-08-02T13:46:19.687+00:00",
3    "status": 400,
4    "error": "Bad Request",
5    "errorMessage": "Can't change names of person in a MedicalRecord! Please don't modify firstName and LastName of the Person",
6    "fieldValidationErrors": []
7  }
```

3) Le nom et prénom du dossier médical correspondent : L'API retourne une réponse 200 avec les informations sur le carnet de santé comme ci dessous :

NB :

Concernant les médications et allergies, si ceux sont de nouvelles données, elles seront également créées et enregistrées, sinon le mapping médication,allergie/MedicalRecord sera simplement mis à jour

```

1  {
2    "idMedicalRecord": 25,
3    "person": {
4      "idPerson": 25,
5      "firstName": "Dorian",
6      "lastName": "Delaval",
7      "address": "1509 Culver St",
8      "city": "Cassis",
9      "zip": 13260,
10     "phone": "061-846-0160",
11     "email": "delaval.https@email.com"
12   },
13   "medications": [
14     {
15       "idMedication": 21,
16       "designation": "medication2",
17       "posology": "100mg"
18     },
19     {
20       "idMedication": 19,
21       "designation": "medication1",
22       "posology": "350mg"
23     }
24   ],
25   "allergies": [
26     {
27       "idAllergy": 1,
28       "designation": "nillacilan"
29     }
30   ]
31 }
```


DELETE

/medicalRecord/{lastName}/{firstName} Delete MedicalRecord

En tant que système de service d'administration.

Je veux pouvoir supprimer un dossier médical existant avec comme identifiant le nom/prénom de la personne.

Dans le but de mettre à jour les informations de santé d'une personne.

Scénario :

1) La personne n'existe pas.

2) La personne du dossier médical existe.

Étant donné : que l'application SafetyNet Alert a son Administration API activée.

Lorsque : le système du service d'urgence envoie une requête sur l'URL suivante :

<http://localhost:8080/medicalRecord/{lastname}/{firstname}>

Alors : l'API renvoie au système d'administration une réponse sous forme de fichier JSON

1) La personne du dossier médical n'existe pas. L'API retourne une erreur 404 avec le message suivant :

```
1  {
2    "timestamp": "2021-08-02T13:52:13.994+00:00",
3    "status": 404,
4    "error": "Not Found",
5    "errorMessage": "MedicalRecord was not found because lastname and firstname didn't match
6                      with anybody: Please chose valid couple firstName/LastName",
7    "fieldValidationErrors": []
8  }
```

2) La personne du dossier médical existe: L'API retourne une réponse 200 avec le message suivant :

```
1  MedicalRecord of Person Boyd Jacob was deleted
```

Person API

person-rest-controller API to Manage Person		▼
GET	/person	Persons
POST	/person	Create a new Person
GET	/person/{id}	Person with ID
PUT	/person/{id}	Update an existed Person by giving it's ID
DELETE	/person/{lastName}/{firstName}	Delete an existed Person by giving it's LastName and FirstName

En tant que système d'administration.

Je veux pouvoir récupérer la liste des personnes existantes.

Dans le but d'avoir des informations sur ces personnes.

Étant donné : que l'application SafetyNet Alert a son Administration API activée.

Lorsque : le système du service d'urgence envoie un requête sur l'URL suivante :

<http://localhost/person>

Alors : L'API retourne sous forme de fichier JSON la liste des personnes existantes.

```

1  [
2  {
3      "idPerson": 1,
4      "firstName": "John",
5      "lastName": "Boyd",
6      "birthDate": "03/05/1984",
7      "address": "1509 Culver St",
8      "city": "Culver",
9      "zip": 97451,
10     "phone": "841-874-6512",
11     "email": "jaboyd@email.com"
12 },
13 {
14     "idPerson": 2,
15     "firstName": "Jacob",
16     "lastName": "Boyd",
17     "birthDate": "03/05/1989",
18     "address": "1509 Culver St",
19     "city": "Culver",
20     "zip": 97451,
21     "phone": "841-874-6513",
22     "email": "drk@email.com"
23 },
24 {
25     "idPerson": 3,
26     "firstName": "Tenley",
27     "lastName": "Boyd",
28     "birthDate": "02/17/2012",
29     "address": "1509 Culver St",
30     "city": "Culver",
31     "zip": 97451,
32     "phone": "841-874-6512",
33     "email": "tenz@email.com"
34 },

```

GET

/person/{id} Person with ID

En tant que système de service d'administration.

Je veux pouvoir récupérer les informations d'une personne en indiquant son ID.

Dans le but d'étudier ces informations

Scénario :

1) La personne avec l'identifiant demandé n'existe pas.

2) La personne existe.

Étant donné : que l'application SafetyNet Alert a son Administration API activée.

Lorsque : le système du service d'urgence envoie une requête sur l'URL suivante :

<http://localhost:8080/person/{id}>

Alors : l'API renvoie au système d'administration une réponse sous forme de fichier JSON

1) personne inexistante : L'API retourne une erreur 404 avec le message suivant :

```
1 {  
2   "timestamp": "2021-08-02T14:07:47.630+00:00",  
3   "status": 404,  
4   "error": "Not Found",  
5   "errorMessage": "Unable to found a person with id:26",  
6   "fieldValidationErrors": []  
7 }
```

2) L'API retourne un message 200 avec les informations sur la personne comme ci dessous :

```
1 {  
2   "idPerson": 1,  
3   "firstName": "John",  
4   "lastName": "Boyd",  
5   "birthDate": "03/05/1984",  
6   "address": "1509 Culver St",  
7   "city": "Culver",  
8   "zip": 97451,  
9   "phone": "841-874-6512",  
10  "email": "jaboyd@email.com"  
11 }
```

En tant que système de service d'administration.

Je veux pouvoir ajouter une nouvelle personne

Dans le but de compléter la base de donnée

Scénario :

1) La nouvelle personne existe déjà.

2) La personne n'existe pas.

Étant donné : que l'application SafetyNet Alert a son Administration API activée.

Lorsque : le système du service d'urgence envoie une requête sur l'URL suivante :

<http://localhost:8080/person>

Alors : l'API renvoie au système d'administration une réponse sous forme de fichier JSON

1) Personne existante : L'API retourne une erreur 400 avec le message suivant :

```
1 {  
2   "timeStamp": "2021-08-02T14:13:25.497+00:00",  
3   "status": 400,  
4   "error": "Bad Request",  
5   "errorMessage": "this Person with firstname:John and lastname:Boyd already exist ! Can't add an already existed Person!",  
6   "fieldValidationErrors": []  
7 }
```

2) Personne n'existe pas : L'API retourne un message 201 avec les informations nouvellement créées comme ci dessous :

NB :
L'adresse de la nouvelle
Personne est pris en
compte et on mets a jour
automatiquement le
mapping fireStation/address

```
1 {  
2   "idPerson": 26,  
3   "firstName": "Dorian",  
4   "lastName": "Delaval",  
5   "address": "1509 Culver St",  
6   "city": "Cassis",  
7   "zip": 13260,  
8   "phone": "061-846-0160",  
9   "email": "delaval.https@email.com"  
10 }
```


En tant que système de service d'administration.

Je veux pouvoir mettre a jour les informations d'une personne(hormis nom et prénom)avec comme identifiant son id.

Dans le but de modifier les informations sur une personne

Scénario :

- 1)Le nom et prénom de la personne avec l'Id demandé ne sont pas corrects.
- 2)Le nom et prénom correspondent.

Étant donné : que l'application SafetyNet Alert a son Administration API activée.

Lorsque : le système du service d'urgence envoie un requête sur l'URL suivante :

<http://localhost:8080/person/{id}>

Alors : l'API renvoie au système d'administration une réponse sous forme de fichier JSON

1) Le nom et prénom ne correspondent pas : L'API retourne une erreur 400 avec le message suivant :

```
1 {
2   "timestamp": "2021-08-02T14:21:29.934+00:00",
3   "status": 400,
4   "error": "Bad Request",
5   "errorMessage": "When updating a person with id:2 you can't change names",
6   "fieldValidationErrors": []
7 }
```

2) Le nom et prénom correspondent : L'API retourne une réponse 200 avec les informations suivantes :

NB :

Lorsque l'adresse est modifiée ,
automatiquement on change le
mapping FireStation/Person.

```
1 {
2   "idPerson": 2,
3   "firstName": "Jacob",
4   "lastName": "Boyd",
5   "birthDate": "03/05/1984",
6   "address": "29 15th St",
7   "city": "Culver",
8   "zip": 97451,
9   "phone": "841-874-6512",
10  "email": "jaboyd@email.com"
11 }
```

DELETE

/person/{lastName}/{firstName} Delete an existed Person by giving it's LastName and FirstName

En tant que système de service d'administration.

Je veux pouvoir supprimer une personne.

Dans le but de supprimer une personne

Scénario :

- 1) Le nom/prénom ne correspond à aucune personne
- 2) La personne avec le nom et prénom existe.

Étant donné : que l'application SafetyNet Alert a son Administration API activée.

Lorsque : le système du service d'urgence envoie une requête sur l'URL suivante :

<http://localhost/person/{lastName}/{firstName}>

Alors : l'API renvoie au système d'administration une réponse sous forme de fichier JSON

1) Le nom/prénom ne correspond pas : L'API retourne une erreur 404 avec le message suivant :

```
1 {  
2   "timestamp": "2021-08-02T14:35:02.107+00:00",  
3   "status": 404,  
4   "error": "Not Found",  
5   "errorMessage": "Deleting Person with lastName: Delaval and FirstName: Boris was not Found",  
6   "fieldValidationErrors": []  
7 }
```

2) Personne existante : L'API retourne une réponse 200 avec les informations suivantes :

NB :
Automatiquement, le dossier médical
de la personne est supprimé et le
mapping Firestation/person modifié.