

# Dossier d'expertise Back-End

Réalisé par : Emilie Tombuyse

## Sommaire

### 1. Modélisation de la base de données

1.1 MCD

1.2 MLD

1.3 MPD

1.4 Relation many to many

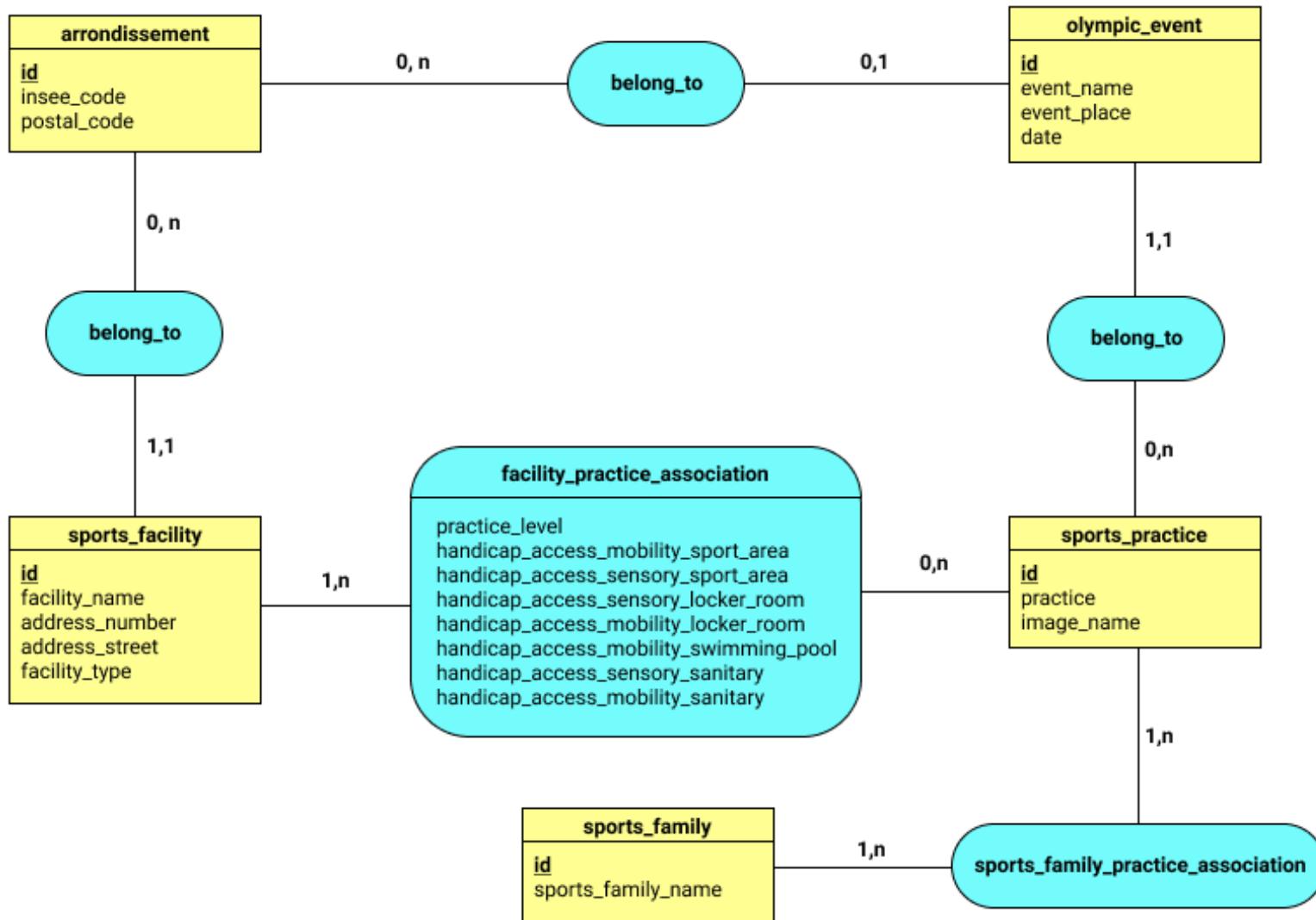
### 2. Symfony & API

2.1 Approche technique de création de l'API

2.2 Documentation de l'API

# 1. Modélisation de la base de données

## 1.1 MCD



## Explications du MCD :

Afin de créer notre base de données, nous avons utilisé **4 sources d'information différentes** :

- **un article** pour les **lieux des épreuves olympiques** à Paris en 2024 :

<https://sport.francetvinfo.fr/les-jeux-olympiques/jeux-olympiques-retrouvez-tous-les-sites-de-paris-2024>

- **un article** pour le **planning des Jeux Olympiques de 2020** à Tokyo :

<https://www.kanpai.fr/societe-japonaise/calendrier-jo-2020-tokyo>

- **une base de données** pour les **établissements sportifs** de Paris :

<https://www.data.gouv.fr/fr/datasets/recensement-des-equipements-sportifs-a-paris-idf/>

- **une base de données** pour les **arrondissements** de Paris :

<https://opendata.paris.fr/explore/dataset/arrondissements/information/>

Nous n'avions pas besoin de toutes les données des bases de données récupérées. Voici donc ci-après quelques explications sur les différentes entités construites sur base de ces sources d'information.

Afin de récupérer les données sélectionnées dans le MCD, nous avons élaboré un script PHP disponible dans le dossier [database/generate\\_database\\_data.php](#) du repository.

## Entité arrondissement :

arrondissement
<u>id</u>
insee_code
postal_code

Pour les arrondissements, nous n'avions besoin que de leur **code insee** et de leur **code postal**, l'**id** étant le même que celui du numéro de l'arrondissement de Paris (de 1 à 20). Le code postal est utilisé en front dans l'affichage de l'adresse d'un établissement, tandis que le code insee se retrouve dans la base de données des établissements sportifs de Paris. Grâce à cela, il est possible de savoir à quel arrondissement appartient un établissement sportif.

## Entité sports\_facility :

sports_facility
<u>id</u>
facility_name
address_number
address_street
facility_type

Cette entité représente un **établissement sportif**. Chaque établissement possède un **nom**, un **numéro d'adresse**, un **nom de rue**, et un **type d'établissement** (ex: piscine, centre sportif, salle de fitness...). Après vérification avec un table de jointure, un établissement n'a qu'un seul et unique type, donc pas besoin d'entité supplémentaire. Ces informations proviennent de la base de données des établissements sportifs de Paris. Un établissement appartient à un et un seul arrondissement et un arrondissement peut avoir entre zéro et N établissements sportifs, d'où son lien avec l'entité arrondissement. Ces champs sont utilisés pour l'affichage des adresses des établissements.

## Entité sports\_practice :

sports_practice
<u>id</u>
practice
image_name

Cette entité désigne une **pratique sportive** comme la gymnastique rythmique, gymnastique artistique, le basket... Elle possède un **id**, un **nom de pratique** et un **nom d'image** (image ayant le même nom et stockée en front dans les assets). Elle est reliée à la table sports\_facility car on peut pratiquer une ou plusieurs pratiques sportives dans un établissement sportif. Une pratique peut, elle, être exercée dans plusieurs établissements différents ou aucun (il s'agit alors d'une épreuve olympique non praticable dans un établissement sportif de Paris). Ces données sont utilisées pour afficher les icônes et le nom d'une pratique sportive en front.

## Entité sports\_family :

sports_family
<u>id</u>
sports_family_name

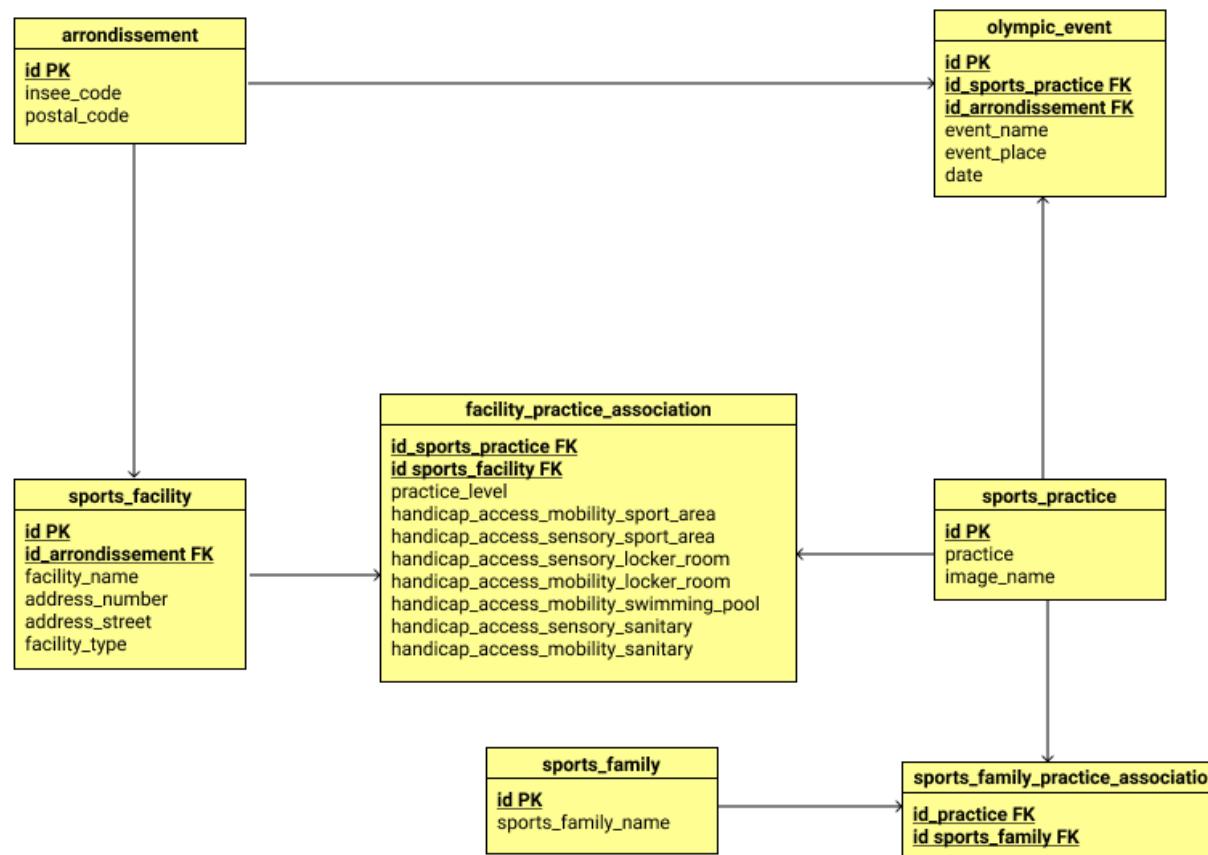
Cette entité représente une **famille de sport** à laquelle appartiennent des pratiques sportives. Par exemple, la famille des sports de ballons regroupe les sports comme le football, le basket, le rugby... Cette information sert à déterminer les **sports similaires** d'une pratique sportive pour notre datavisualisation. L'entité est composée du **nom de la famille** et d'un **id**. Elle est reliée à l'entité sports\_practice par une association many to many car une pratique sportive peut appartenir à une ou plusieurs familles (le waterpolo est un sport de nage mais aussi un sport de ballon) et une famille de sport contient une à N pratiques sportives.

## Entité olympic\_event :

olympic_event
<b>id</b>
event_name
event_place
date

Une épreuve olympique est définie par un **id**, un **nom d'épreuve officiel**, un **lieu** et une **date**. Cette entité est reliée à l'entité arrondissement car une épreuve peut se dérouler ou non dans un arrondissement de Paris et il peut y avoir plusieurs épreuves différentes dans un même arrondissement. Elle est également liée à l'entité **sports\_practice** car une et une seule pratique sportive est jouée lors d'une épreuve olympique, et que toutes les pratiques sportives ne sont font pas forcément l'objet d'une épreuve olympique. Elles peuvent cependant être jouées sur plusieurs jours ou à plusieurs endroits un même jour (football). Ces données sont utilisées pour afficher les épreuves en cours dans un arrondissement à une date précise.

## 1.2 MLD



En passant du MCD au MLD, les associations disparaissent pour laisser place à des tables de jointures dans le cas d'une association many to many, et à un référencement de clés étrangères dans les tables n'ayant pas N comme maximal de cardinalité d'association pour les autres associations (one to many et many to one). On peut voir grâce aux flèches que les clés étrangères proviennent de leur table d'origine qu'elles réfèrent dans une autre table.

#### Exemple clé étrangère :

La clé étrangère **id\_arrondissement** réfère la table arrondissement dans la table **sports\_facility**, car chaque établissement sportif est lié à un arrondissement.

sports_facility	
<b>id PK</b>	
<b>id_arrondissement FK</b>	
facility_name	
address_number	
address_street	
facility_type	

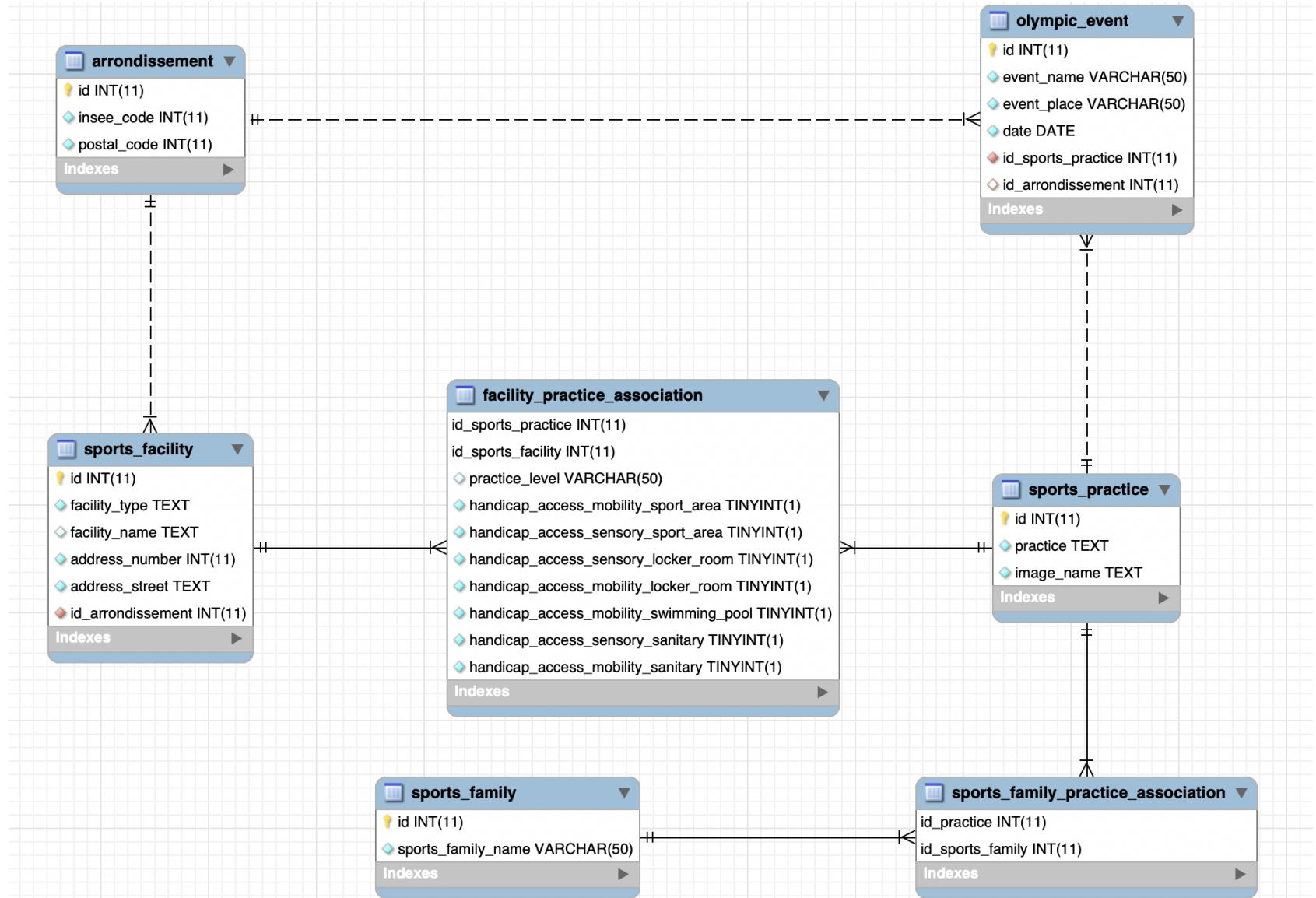
#### Exemple table de jointure :

Vu qu'une famille de sport peut avoir plusieurs pratiques et qu'une pratique peut avoir plusieurs familles de sport, on ne peut pas mettre de clé étrangère dans une des deux tables, on a donc créé une table de jointure avec les deux clés étrangères qui réfèrent leurs tables respectives. C'est la combinaison de ces deux clés étrangères qui forment l'identifiant de la table.

sports_family_practice_association	
<b>id_practice FK</b>	
<b>id_sports_family FK</b>	

Pratique	Famille	Pratique	Famille
Waterpolo	Sport de ballon	Basket	Sport de ballon
Waterpolo	Sport de nage	Football	Sport de ballon

## 1.3 MPD



Ce MPD (modèle physique des données) a été réalisé avec **MySQL Workbench**.

On peut y voir les clés primaires de chaque table, les clés étrangères et surtout la précision du type de donnée pour chaque champ. Ces derniers sont de type texte, entier, varchar, booléen et date.

Le type **texte** a été utilisé lorsque la longueur du texte n'était pas connue ou susceptible d'être assez longue.

Le type **varchar** a été utilisé pour les textes dont la longueur était relativement courte.

Le type **booléen** (tinyint) a été utilisé lorsque l'information stockée pouvait être vraie ou fausse.

Le type **date** a été utilisé pour les dates d'évènements olympiques (format YYYY-MM-DD).

## 1.4 Relation many to many

Cette relation many to many étant un peu particulière, quelques mots d'explication sont nécessaires.



Tout d'abord, cette relation lie l'**établissement sportif** à l'**entité pratique sportive**. Leurs cardinalités nous indiquent que dans un établissement sportif, on peut pratiquer au minimum une pratique sportive, au maximum N (dans une piscine on peut faire du plongeon et de l'aquagym). Elles nous indiquent également qu'une pratique sportive peut n'être disponible dans aucun établissement sportif (cas où il s'agit d'une épreuve olympique non praticable dans un établissement sportif à Paris comme l'aviron) ou au contraire être praticable dans plusieurs établissements sportifs (on peut nager dans plusieurs piscines différentes, pas une seule). **La spécificité de cette association** est qu'elle comporte des informations supplémentaires, à savoir, un **niveau de pratique sportive** et des **accès handicap** de différentes sortes. Elle sont reprises dans l'association car il faut un établissement ET une pratique sportive pour déterminer ces informations. En effet, un niveau et des accès sont définis pour une pratique en particulier, dans un établissement particulier et n'est pas propre à un établissement seul ou à une pratique seule. Cette relation donnera lieu comme vu plus haut dans le MLD à une table de jointure contenant les deux clés étrangères des deux tables, plus le reste des champs de l'association.

## 2. Symfony & API

### 2.1 Approche technique de création de l'API

Afin de créer notre API, nous avons commencé par créer un projet symfony grâce à la commande :

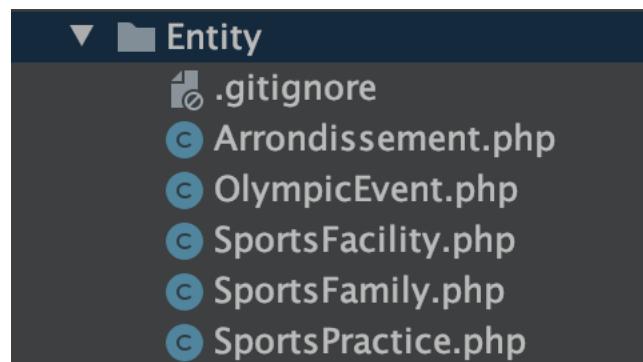
```
composer create-project symfony/skeleton:"4.4.*" nomdenotreprojet
```

#### Création des entités

Ensuite, après avoir configuré le fichier **.env** afin d'établir une connexion avec la base de données et d'avoir installé Doctrine ORM, nous avons **généré automatiquement les entités** sur base de celle-ci en utilisant la commande suivante :

```
php bin/console doctrine:mapping:import "App\Entity" annotation --path=src/Entity
```

Nous avions à présent les entités suivantes :



## Création des repositories

L'annotation `@ORM\Entity` de chaque entité a ensuite été complétée avec le nom du futur repository correspondant.

```
/**  
 * Arrondissement  
 *  
 * @ORM\Table(name="arrondissement")  
 * @ORM\Entity  
 */
```

```
/**  
 * Arrondissement  
 *  
 * @ORM\Table(name="arrondissement")  
 * @ORM\Entity(repositoryClass="App\Repository\ArrondissementRepository")  
 */
```

Ces repositories ont par la suite été générés grâce à la commande suivante :

```
php bin/console make:entity --regenerate
```



## Création des controllers

Afin de déterminer les différents controllers dont nous avions besoin, nous avons commencé par **schématiser chaque réponse JSON** dont le front aurait besoin pour faire fonctionner chaque partie de notre site. Voici les 8 réponses auxquelles nous sommes parvenus (voir page suivante) :

Adresses des établissements et leurs accès:

```
[  
  {  
    facilityType: 'Salle de sport',  
    facilityName: 'Club i sport',  
    addressNumber: 29,  
    addressStreet: 'Rue saint vincent',  
    arrondissement: '75001',  
    handicapAccessMobilitySportArea: true,  
    handicapAccessSensorySportArea: false,  
    handicapAccessSensoryLockerRoom: true,  
    handicapAccessMobilityLockerRoom: false,  
    handicapAccessMobilitySwimmingPool: false,  
    handicapAccessSensorySanitary: true,  
    handicapAccessMobilitySanitary: false  
  },  
  { ...  
  },  
  { ...  
  }  
]
```

Liste des filtres de niveau disponibles :

```
['Compétition', 'Loisir', 'Scolaire', 'Entrainement']
```

Liste de toutes les épreuves olympiques

```
[  
  {  
    id: 505,  
    practice: "Course et marche sur route",  
    image: "course-et-marche-sur-route",  
    facilitiesAmount: 3  
  },  
  { ...  
  },  
  { ...  
  }  
]
```

Liste des épreuves olympiques d'une date + leurs sports similaires :

```
let response = {  
  olympicSports: [  
    {  
      id: 8801,  
      practice: "Volley-ball",  
      image: "volley-ball",  
      facilityAmount: 217  
    },  
    { ...  
    }  
  ],  
  relatedSports: [  
    {  
      id: 901,  
      practice: "Ballon au poing",  
      image: "ballon-au-poing",  
      facilityAmount: 1  
    },  
    { ...  
    }  
  ]  
}
```

Un sport sélectionné + ses sports similaires :

```
let response = {  
  selectedSportData:  
  {  
    id: 5201,  
    practice: "Natation course sportive",  
    image: "natation-course-sportive",  
    facilityAmount: 50  
  },  
  otherFamilies: [  
    {  
      id: 2801,  
      practice: "Hockey subaquatique",  
      image: "hockey-subaquatique",  
      facilityAmount: 1  
    },  
    { ...  
    },  
    { ...  
    }  
  ]  
}
```

Nombre d'établissements sportifs pour chaque arrondissement :

```
[  
  {  
    arrondissement: 1,  
    amountFacilities: 1  
  },  
  {  
    arrondissement: 2,  
    amountFacilities: 1  
  },  
  {  
    arrondissement: 3,  
    amountFacilities: 1  
  },  
  {  
    arrondissement: 4,  
    amountFacilities: 0  
  },  
  {  
    arrondissement: 5,  
    amountFacilities: 2  
  },  
  {  
    arrondissement: 6,  
    amountFacilities: 2  
  },  
  {  
    arrondissement: 7,  
    amountFacilities: 2  
  },  
  { ...  
  }  
]
```

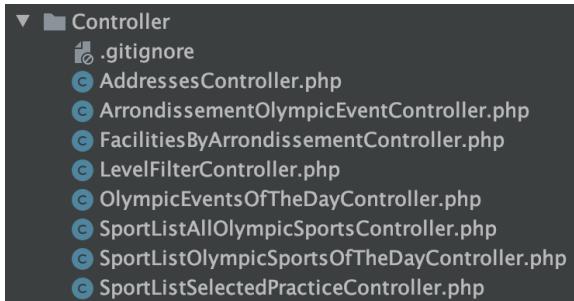
Liste des épreuves olympiques d'une date précise :

```
[  
  {  
    id: 2901,  
    practice: "Football",  
    image: "football"  
  },  
  { ...  
  }  
]
```

Épreuves dans un arrondissement :

```
[  
  {  
    practice: "Volley-ball",  
    imageName: "volley-ball",  
    place: "Paris Arena"  
  },  
  { ...  
  },  
  { ...  
  }  
]
```

Nous avons donc créé **un controller pour chaque réponse** différente attendue, ce qui veut dire un total de 8 controllers.



Nous avons ensuite listé les différents paramètres que prendrait chaque route. Ils sont au nombre de 5. Voici une présentation de chacun :

- **{id\_practice}** : l'id de la pratique sportive sélectionnée (football, basket, nage synchronisée...). Ex: 5201
- **{handicap\_mobility}** : booléen indiquant si le filtre "handicap mobilité" a été activé ou non.
- **{handicap\_sensory}** : booléen indiquant si le filtre "handicap sensoriel" a été activé ou non.
- **{level}** : niveau de pratique sélectionné (string "false" si aucun n'a été sélectionné). Ex: Compétition
- **{arrondissement}** : numéro de l'arrondissement sélectionné, ce paramètre est parfois optionnel. Ex: 3

## Requêtes DQL et SQL

Nous avons majoritairement utilisé **QueryBuilder** afin de construire des **requêtes DQL** pour la récupération de nos données dans les repositories du projet. Seules les requêtes utilisant la table de jointure **facility\_practice\_association** ont été écrites en **requêtes SQL** car nous avions jugé plus fastidieux de créer une nouvelle entité afin d'avoir accès aux différents champs de la table, plutôt que d'avoir directement recours aux requêtes SQL.

```
$qb = $this->createQueryBuilder('f');
$qb->select('f.id')
    ->where('f.id_practice MEMBER OF f.idPractice')
    ->setParameter('id_practice', $id_practice);

$query = $qb->getQuery();
$result = $query->getResult();
```

```
$conn = $this->getEntityManager()
    ->getConnection();

$sql = "SELECT f.id_arondissement, COUNT(f.id) as amount_facilities FROM sports_facility f
    INNER JOIN facility_practice_association a ON f.id = a.id_sports_facility
    INNER JOIN sports_practice p ON p.id = a.id_sports_practice
    WHERE p.id = :id
    GROUP BY f.id_arondissement";

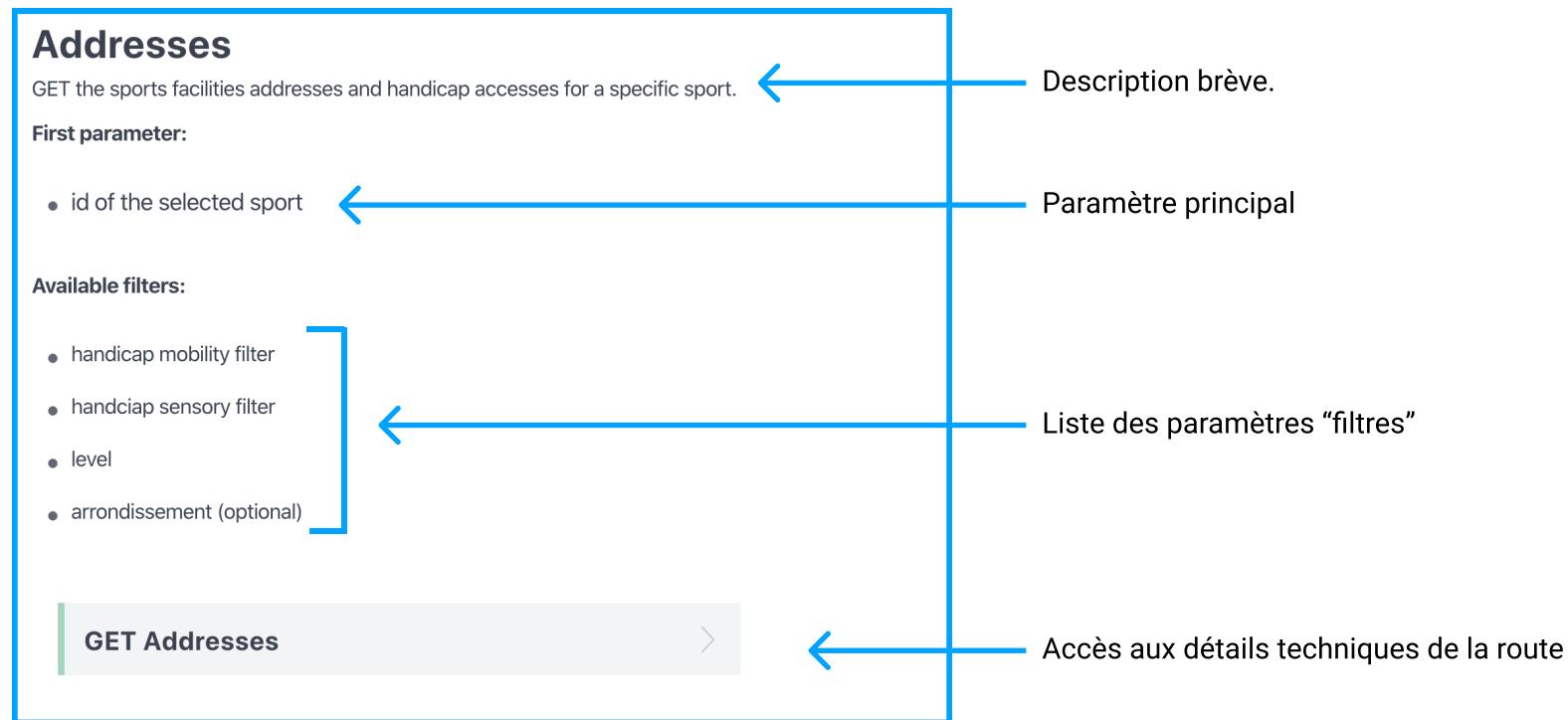
$stmt = $conn->prepare($sql);
$stmt->bindValue('id', $id);
$stmt->execute();
$result = $stmt->fetchAll();
return $result;
```

## 2.2 Documentation de l'API

Pour réaliser la documentation de l'API, nous avons utilisé **Apiary**.

Notre documentation est accessible via le lien : <https://jo2024g7.docs.apiary.io/#>

Chaque route est décrite comme suit :



Lorsque l'on clique sur une route, nous avons alors accès à ce panneau reprenant les différents paramètres de la route :

**GET** [http://localhost:8080/addresses/id\\_practice/handicap\\_mobility/handicap\\_sensory/level/arrondissement](http://localhost:8080/addresses/id_practice/handicap_mobility/handicap_sensory/level/arrondissement)

### Parameters

Parameter	Description	Type
<b>id_practice</b>	● ID of the selected sports practice	Integer
<b>handicap_mobility</b>	● handicap mobility accessibility filter	Boolean
<b>handicap_sensory</b>	● handicap sensory accessibility filter	Boolean
<b>level</b>	● level of practice of a sport (indicate 'false' as parameter value if you don't want to use this filter) Example: <code>Entrainement</code> .	String
<b>arrondissement</b>	○ Paris arrondissement number Example: <code>15</code> .	Integer

### Paramètres

 Obligatoire

 Optionnel

Un aperçu de la réponse est disponible plus bas :

### Response

**200**

#### HEADERS

Content-Type:application/json

#### BODY

```
01 [
02   {
03     "facilityType": "Bassin sportif de natation",
04     "facilityName": "Centre Municipal Armand Massard",
05     "addressNumber": 66,
06     "addressStreet": "Boulevard Du Montparnasse",
07     "arrondissement": 75115,
08     "handicappedAccessMobilitySportArea": false,
09     "handicappedAccessSensorySportArea": false,
10     "handicappedAccessSensoryLockerRoom": false,
11     "handicappedAccessMobilityLockerRoom": false,
12     "handicappedAccessMobilitySwimmingPool": true,
13     "handicappedAccessSensorySanitary": false,
14     "handicappedAccessMobilitySanitary": false
15   },
16 ]
```

Enfin, la requête peut être testée via la console de test de requêtes :

### Request

Production Raw Try

GET http://localhost:8080/addresses/5201/false/false/Entrainement/  
15

URI Parameters Headers Body Reset Values

<input checked="" type="checkbox"/> id_practice	5201
<input checked="" type="checkbox"/> handicap_mobility	false
<input checked="" type="checkbox"/> handicap_sensory	false
<input checked="" type="checkbox"/> level	Entrainement
<input checked="" type="checkbox"/> arrondissement	15

+ Add a new query parameter

Show Code Example Production Call Resource

```
graph TD; A[Try] --> B[Call Resource]
```