Bouinot Thomas

Formation AFPA développeur logiciel

Stage de développement web

(PHP Symfony, JavaScript)

ManageChart

Application web de représentation de données scientifiques sous forme de graphiques

Maître de stage : Mathias ROUAN

Stage réalisé du 03 Octobre au 25 novembre 201 Soutenance : 01 décembre 2016

Remerciement

Je tiens tout d'abord à remercier mon maître de stage Mathias Rouan pour m'avoir permis de travailler sur ce projet. Je lui suis également très reconnaissant pour sa disponibilité et ses explications qui m'ont éclairé sur de nombreux problèmes.
J'aimerais remercier tous les stagiaires qui m'ont précédé, et qui n'ont pas oublié de commenter judicieusement le code.
Je tiens également à remercier Cyril, qui en plus de Mathias, a accepté de partager son bureau avec moi. Pour conclure je remercie tous les membres du laboratoire LETG/Brest pour leur accueil très sympathique.

Bonne lecture

Sommaire

Abstract....6
1. Introduction....7
2. Analyse des spécifications fonctionnelles....8
2.1. L'application ManageChart....8
2.2. Analyse du cahier des charges....9
3. Modélisation fonctionnelle....10
3.1. Cas d'utilisations....10
3.2. Les vues....11
3.3. Diagramme d'activités....15
3.4. Modélisation du domaine....16
3.4.1. Dictionnaire de données....16
3.4.2. Modèle conceptuel de données....18
3.4.3. Modèle logique relationnel....19
3.4.4. Modèle logique objet....20
4. Modélisation technique....21
4.1. Choix technologiques....21
4.2. Framework et librairies....21
4.3. Outils et plateformes....22
5. Conception générale....23
5.1. Patterns utilisés....23
5.2. Appel de la base de données....25
5.2.1. Création de graphique....25
5.2.2. Visualisation....25
6. Conception détaillée....26
6.1. Ajout de l'entité flag et ses conséquences....26
6.1.1. Ajout de l'entité....26
6.1.2. création des formulaires....26
6.1.3. Modification du contrôleur....27
6.2. Requête SQL....27
6.2.1. Modification de la base de données....27
6.2.2. Modification du JavaScript....28
6.3. Divers....29
7. Diagrammes....30

7.1. Diagramme de classe.....30
7.1.1. Entités.....30
7.1.2. Contrôleur.....31
8. Codage.....32
8.1. L'enregistrement en base de données via le contrôleur.....32
8.2. Visualisation d'un graphique.....33
9. Conclusion.....35

Abstract

As part of a training as Software developer at the AFPA in Brest, I completed an 8 weeks internship from October 03 to November 25, 2016 under the supervision of Mathias Rouan, Research Engineer at CNRS.
This course took place within the LETG laboratory (Littoral - Environnement - Télédétection - Géomatique) / Brest Geomer, specialized in the coastal environment of temperate and tropical ecosystems.

The subject of the internship was to continue the development of an existing web application "ManageChart" (https://www-iuem.univ-brest.fr/wapps/managechart/), allowing a representation of scientific data in the form of graph. This application is coded in PHP and mobilizes the free JavaScript library "*Highcharts*". At the beginning of the internship, the application is already built and operational. It allows graphs to be made from databases.

To explain the scientific context, a model has been developed in Teradclim project. It provide information on climatic variables (variability of cumulative temperatures, water stress data...) and variables on the phenology of the vine and the agronomic actions carried out by the vine growers.
It is envisaged to produce dynamic representations of this information in the form of graphs.

The main objective of this internship is to enrich the *ManageChart* application by exploring other graphical representation modes that can be used in this project.
After discussion between the various speakers, it is agreed to integrate the representation in the form of a marker, called flag in the Highcharts library. These flags allows to highlight on an axis or on a series of data (temporal or not) information by means of a letter or a pictogram.

In the project, the goal is to be able to display the different phenological stages of the vine as well as the agronomic actions of the vine growers as a time function of time and compare them with bioclimatic factors.

1. Introduction

Dans le cadre d'une formation au titre de développeur Logiciel à l'AFPA de Brest, j'ai réalisé un stage en entreprise de 8 semaines, du 03 octobre au 25 novembre 2016, sous la supervision de Mathias Rouan, Ingénieur de Recherche au CNRS.

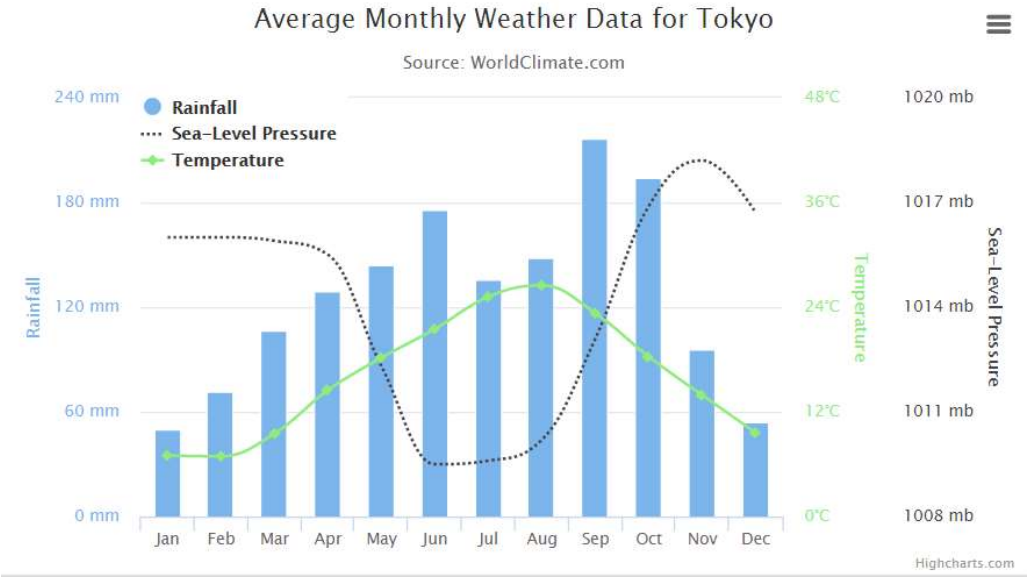
Ce stage s'est déroulé au sein du laboratoire LETG (Littoral - Environnement - Télédétection - Géomatique)/Brest Geomer, spécialisé sur l'environnement littoral des écosystèmes tempérées et tropicaux.

Le sujet du stage consistait à poursuivre le développement d'une application web déjà existante, principalement en PHP et JavaScript.

2. Analyse des spécifications fonctionnelles

2.1. L'application ManageChart

L'application « **ManageChart** » est une application web (https://www-iuem.univ-brest.fr/wapps/managechart/) permettant une représentation de données scientifiques sous la forme de graphique.



Exemple de graphique simple possible avec ManageChart

Cette application est codée en PHP et mobilise la librairie « **Highcharts** », une librairie JavaScript libre de droit à but non commercial. Cette application a été créée au sein du laboratoire LETG/Brest Geomer.

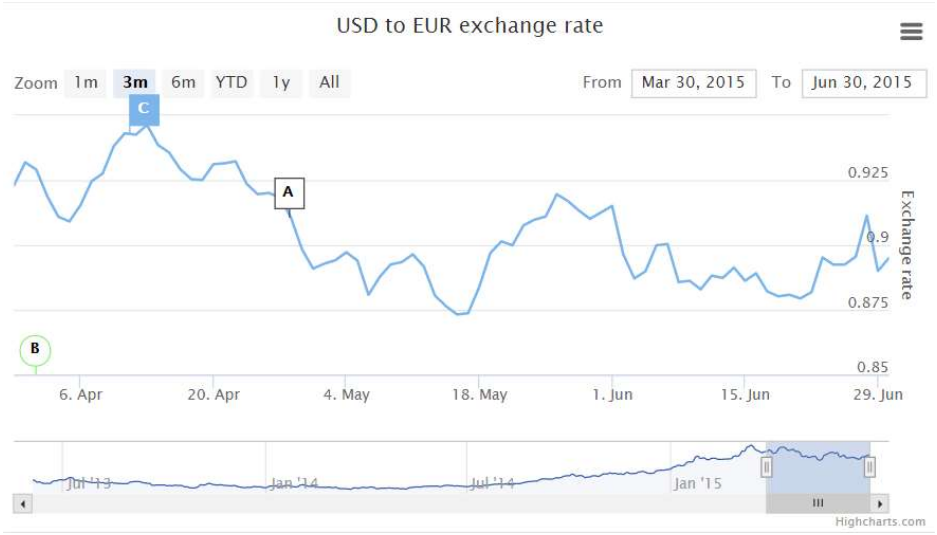
Au début du stage, l'application est déjà construite et opérationnelle. Elle permet de réaliser des graphique à partir de bases de données. Les différents types de graphique disponible sont :

- Graphique simple
- Graphique temporelle
- Graphique temporelle multi axes
- Graphique dynamique
- Graphique polaire

2.2. Analyse du cahier des charges

Dans la cadre du projet GICC Teradclim, un modèle permettant de relier les itinéraires agro-techniques à l'évolution des contraintes d'environnement de la vigne a été développé. Les résultats produits sont administrés au sein d'une base de données relationnelle. Ils fournissent des informations sur des variables climatiques (variabilité des cumuls de températures, données de stress hydrique...) et des variables sur la phénologie de la vigne et les actions agronomiques menées par les viticulteurs.
Il est envisagé de produire des représentations dynamiques de ces informations sous la forme de graphiques dans le cadre du projet européen ADVICLIM. Ce projet pose la question du transfert d'outils et de connaissances auprès des acteurs de la profession viticole.

L'objectif de ce stage est de permettre d'enrichir l'application *ManageChart* en explorant d'autres modes de représentation graphique utilisables dans le cadre de ce projet.
Après discussion entre les différents intervenants, il est convenu d'intégrer à l'application la représentation sous forme de marqueur, appelé *flag* dans la librairie *Highcharts.*



Exemple de graphique contenant des flags

Ces flags permettent de mettre en valeur sur un axe ou sur une série de données (temporelle ou non) des informations à l'aide d'une lettre ou d'un pictogramme. Ils permettent également lors du survol de ces données d'afficher du texte.
Dans le projet ADVICLIM, l'objectif est de pouvoir afficher en fonction du temps les différents stades phénologiques de la vigne ainsi que les actions agronomiques des viticulteurs et de les comparer avec des facteurs bioclimatiques.

3. Modélisation fonctionnelle

3.1. Cas d'utilisations

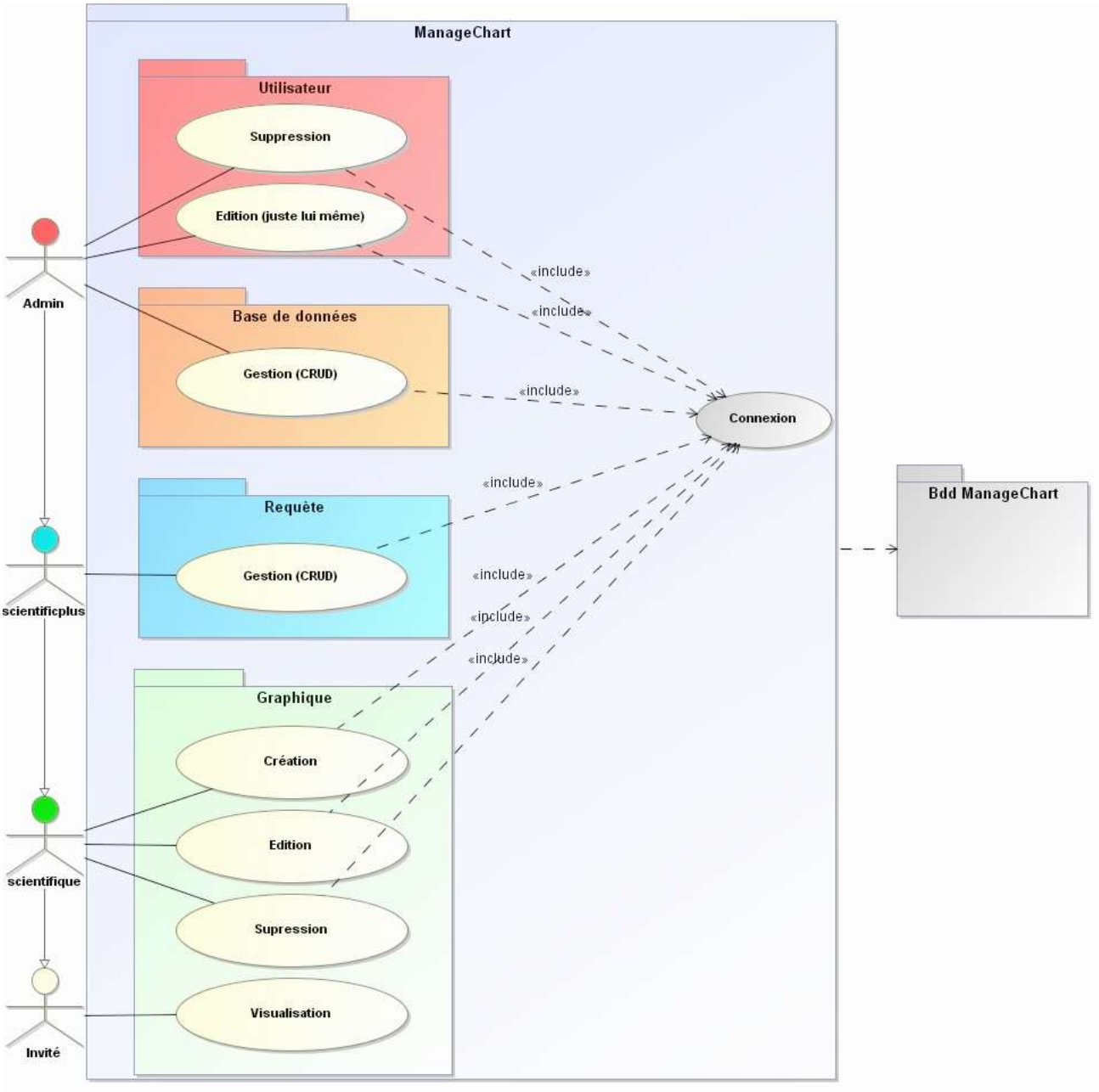


Diagramme d'utilisation de ManageChart

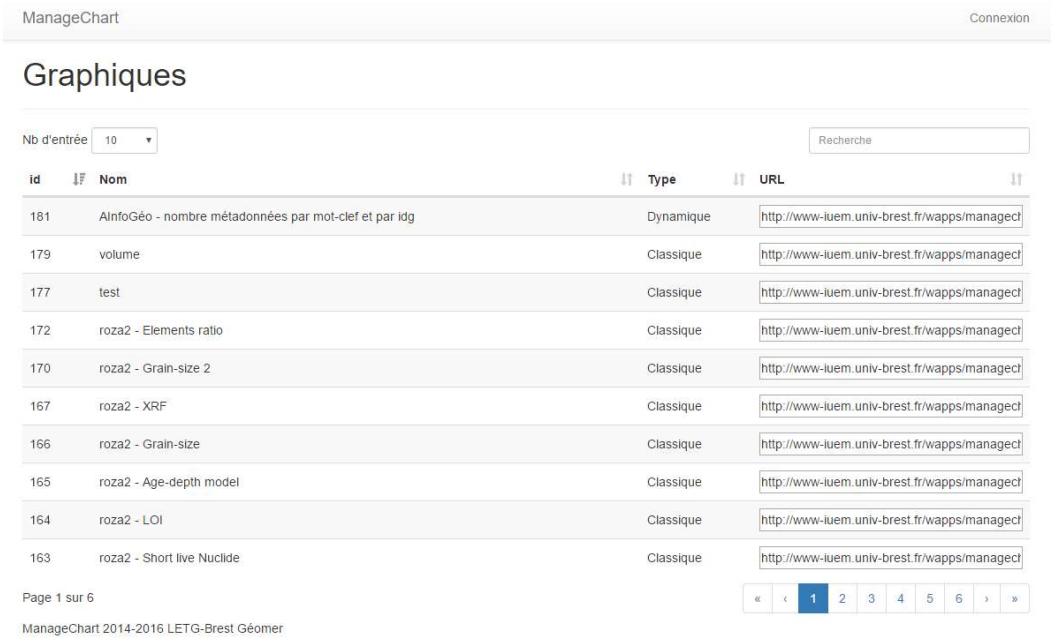
Il existe 4 types d'utilisateurs différents :

- L'*invité* : Il peut visualiser les graphiques existant sur l'application.
- Le *scientifique* : Il peut créer, éditer ou supprimer des graphiques.
- Le *scientifique plus :* Il a la possibilité de créer, éditer, supprimer ses propres requêtes. Il nécessite une connaissance en base de données.
- L'*administrateur* : Il gère les bases de données ainsi que les utilisateurs. Il peut supprimer d'autres utilisateurs mais ne peut éditer que lui-même.

3.2. Les vues

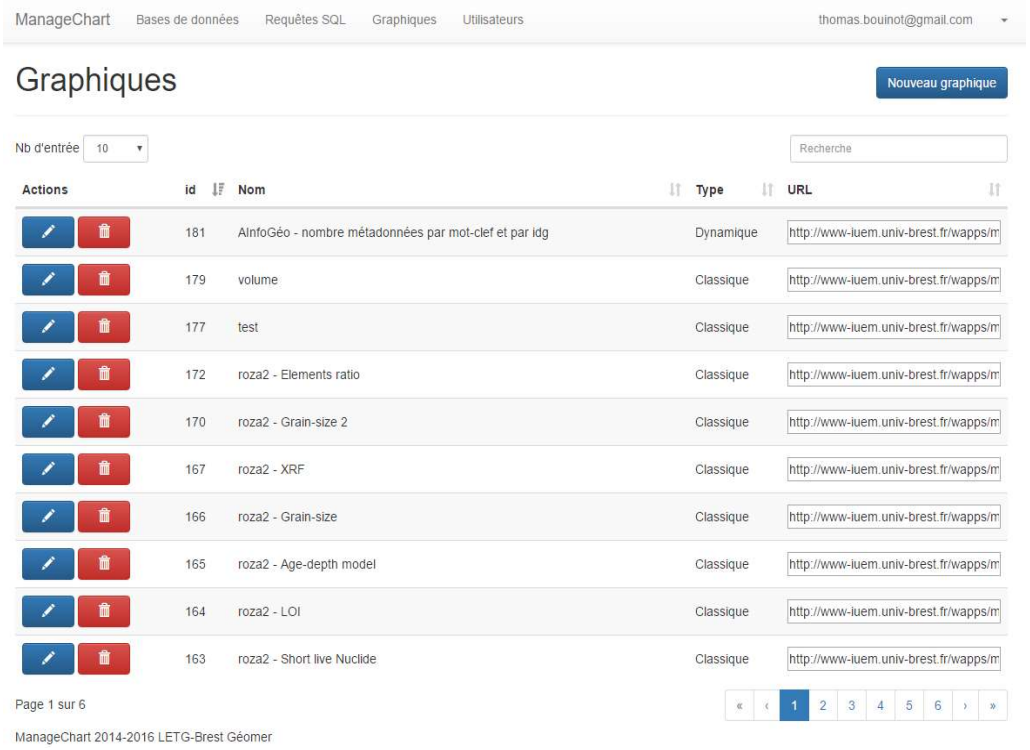
L'application web *ManageChart* est constitué d'une barre de menu comportant les principales fonctionnalités. Cette barre de menu est différente selon le profil d'utilisateur :

- Session invité => Choix de l'identification
- Session Scientifique => Ajout du menu graphique
- Session Scientifique plus => Ajout du menu requête
- Session Administrateur => Ajout du menu base de données et utilisateur

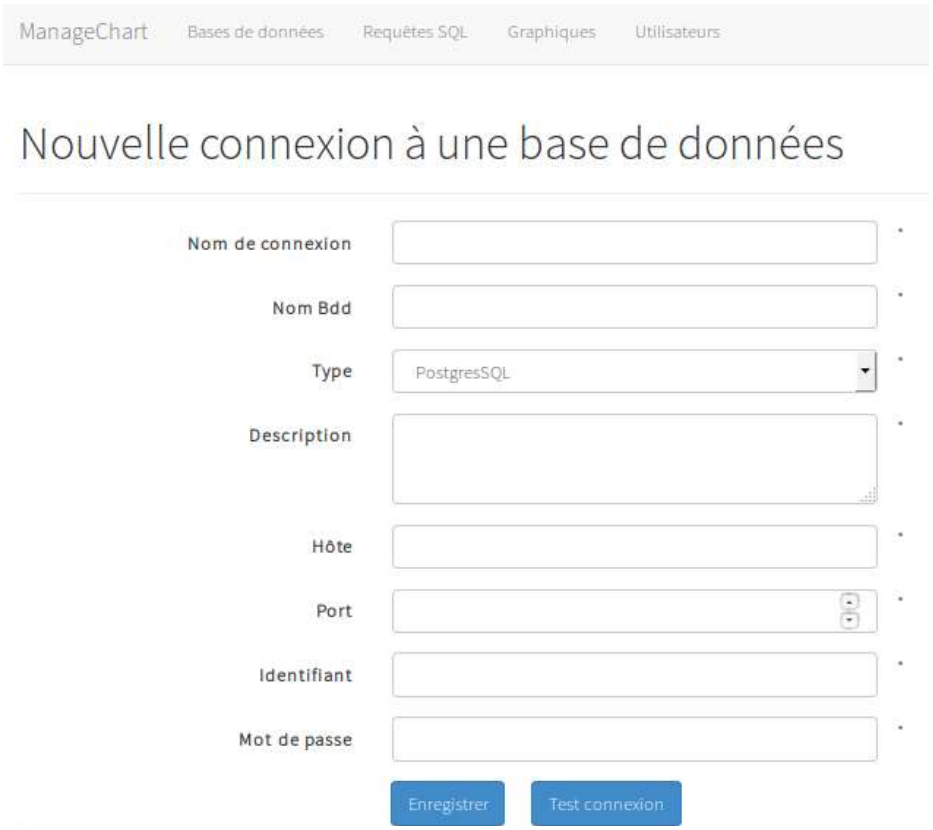


<=*Menu*

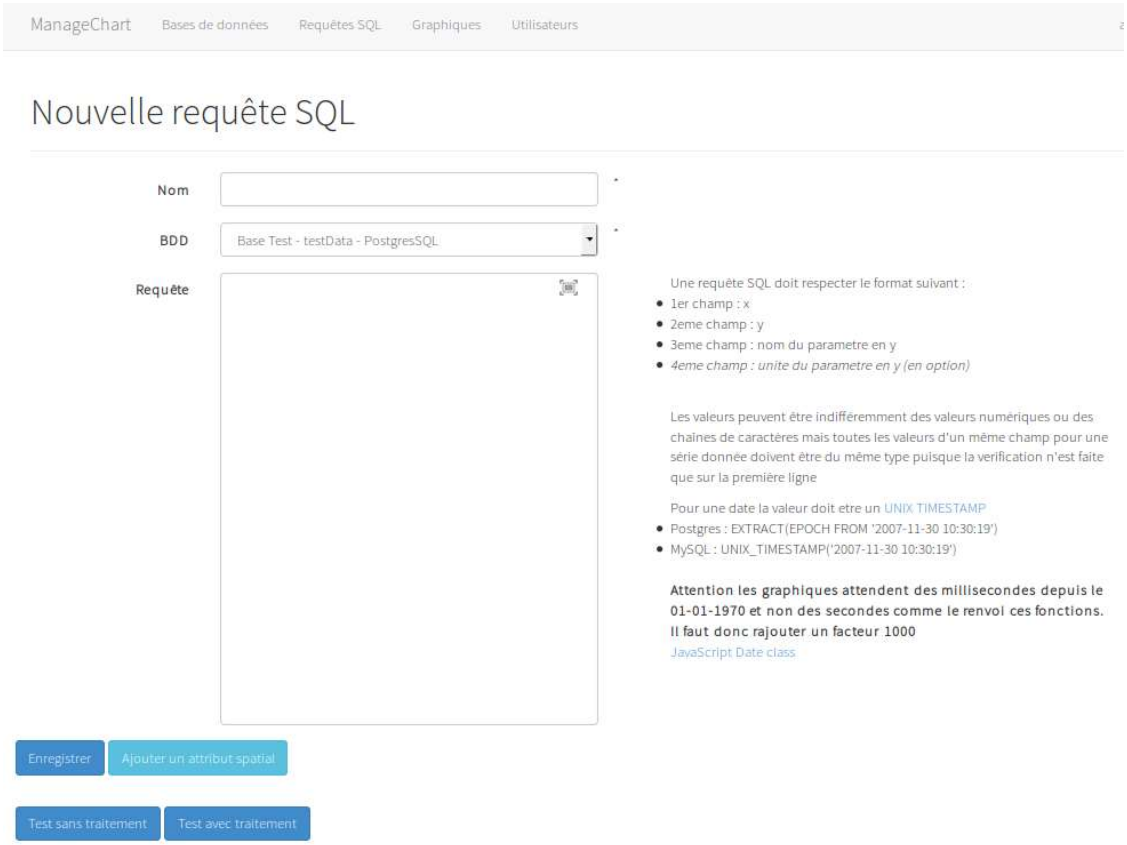
Page d'accueil invité



Page d'accueil session administrateur

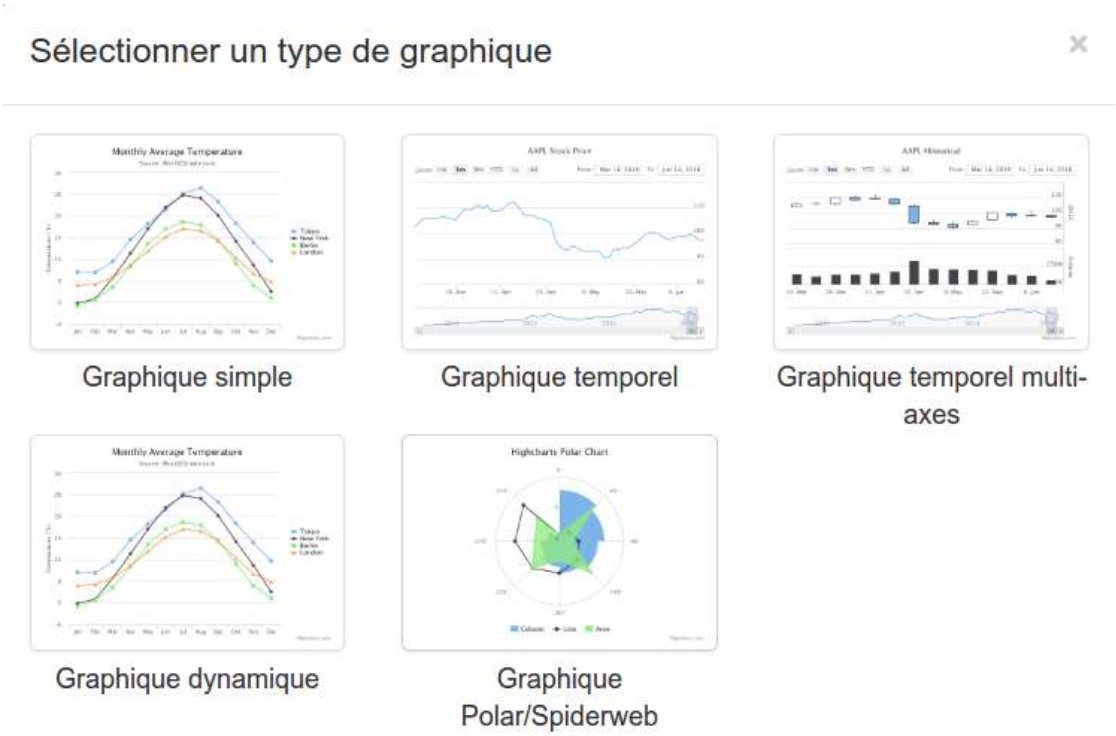


Page de création d'une connexion à une base de données SQL



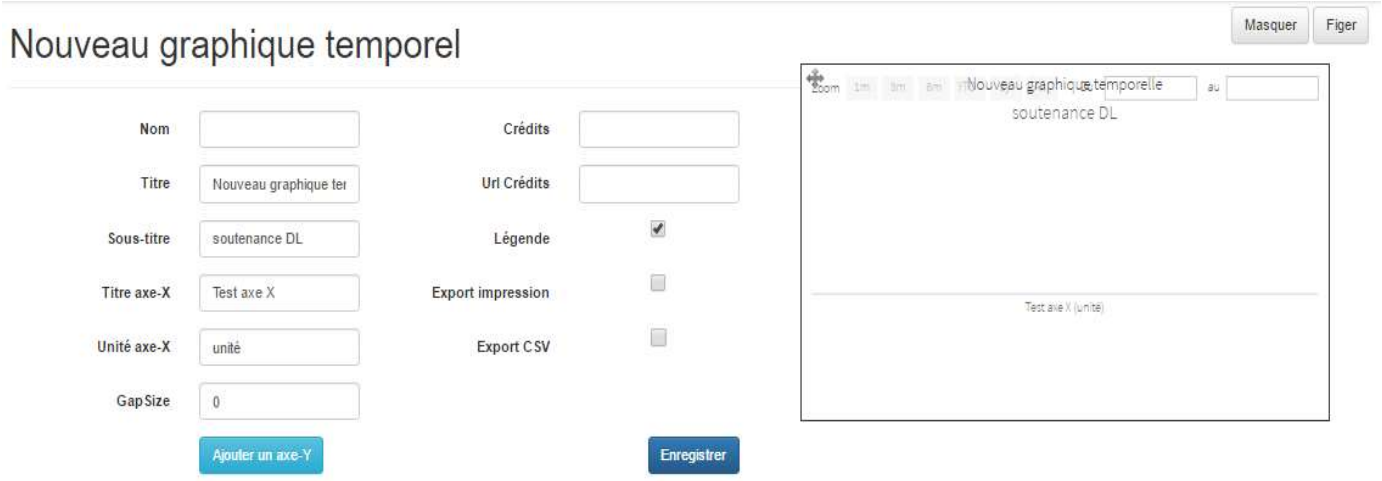
Création d'une requête SQL

L'identification permet la création de différents types de graphique :



L'étape de création nécessite une succession de formulaire pour chaque entité utilisée dans la base de donnée.

➔ Le formulaire principale va créer l'entité *Chart* qui correspond au graphique général (son nom, titre, …).



Lors de la création ou l'édition d'un graphique, une prévisualisation se fait directement à l'aide d'une Iframe et d'appel AJAX comme le montre la capture d'écran ci-dessus.

➔ Le formulaire suivant est pour l'ajout d'un axe verticale *Yaxis*.

➔ Puis viennent les derniers formulaires correspondant aux séries.

3.3. Diagramme d'activités

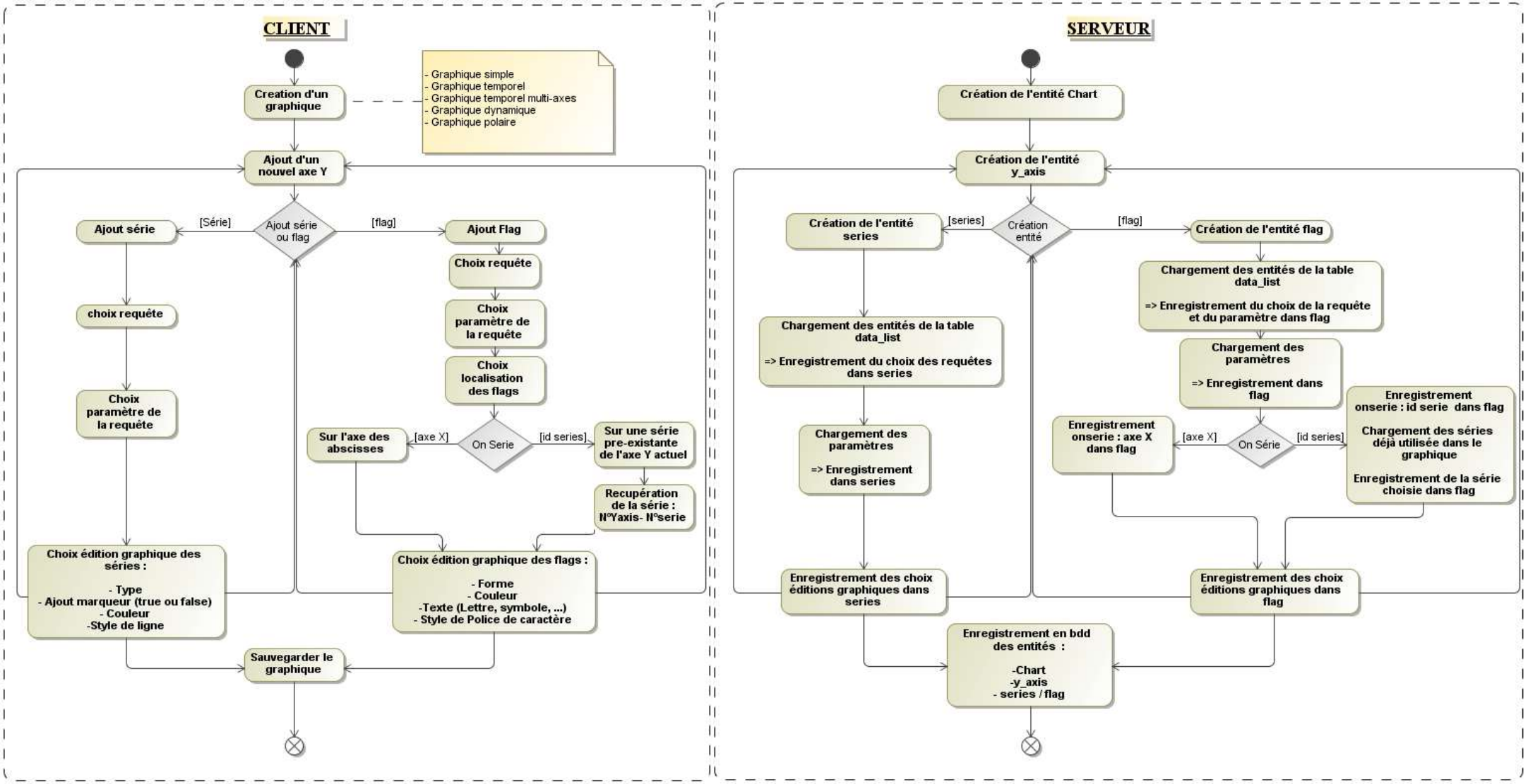


Diagramme d'activité de l'ajout d'une série ou d'un flag

3.4. Modélisation du domaine

3.4.1. Dictionnaire de données

La base de données est initialement composée de 8 tables.

- **chart** représente le graphique général.
- **yAxis** représente les axes verticaux du graphique chart
- **series** représente les séries de données
- **data_list** représente les requêtes
- **data_source** représente les bases de données externes où sont situées les données à visualiser
- **attribut_spatial** représente un paramètre de la requête pouvant être injecté via l'URL du graphique et notamment utilisé pour filtrer les données sur leur localisation géographique
- **account** représente les utilisateurs. L'entité étant géré par le bundleFOSUserBundle qui sert à la gestion des utilisateurs, elle n'a aucune interaction avec le reste de l'application.

Ce stage a induit la création d'une table supplémentaire :

- **Flag** représente les séries de données sous forme de balise

Contraintes d'intégrité et cardinalités :

Les cardinalités sont principalement définies par les cardinalités 0,n (OneToMany) et 1,1 :

- 1 graphique peut contenir plusieurs axes verticaux (OneToMany). 1 axe vertical est dans 1 seul graphique.

- 1 axe vertical peut contenir plusieurs séries (OneToMany). 1 série est dans 1 axe vertical.

- 1 série ne représente qu'une seule requête. 1 requête peut générer plusieurs séries (OneToMany).

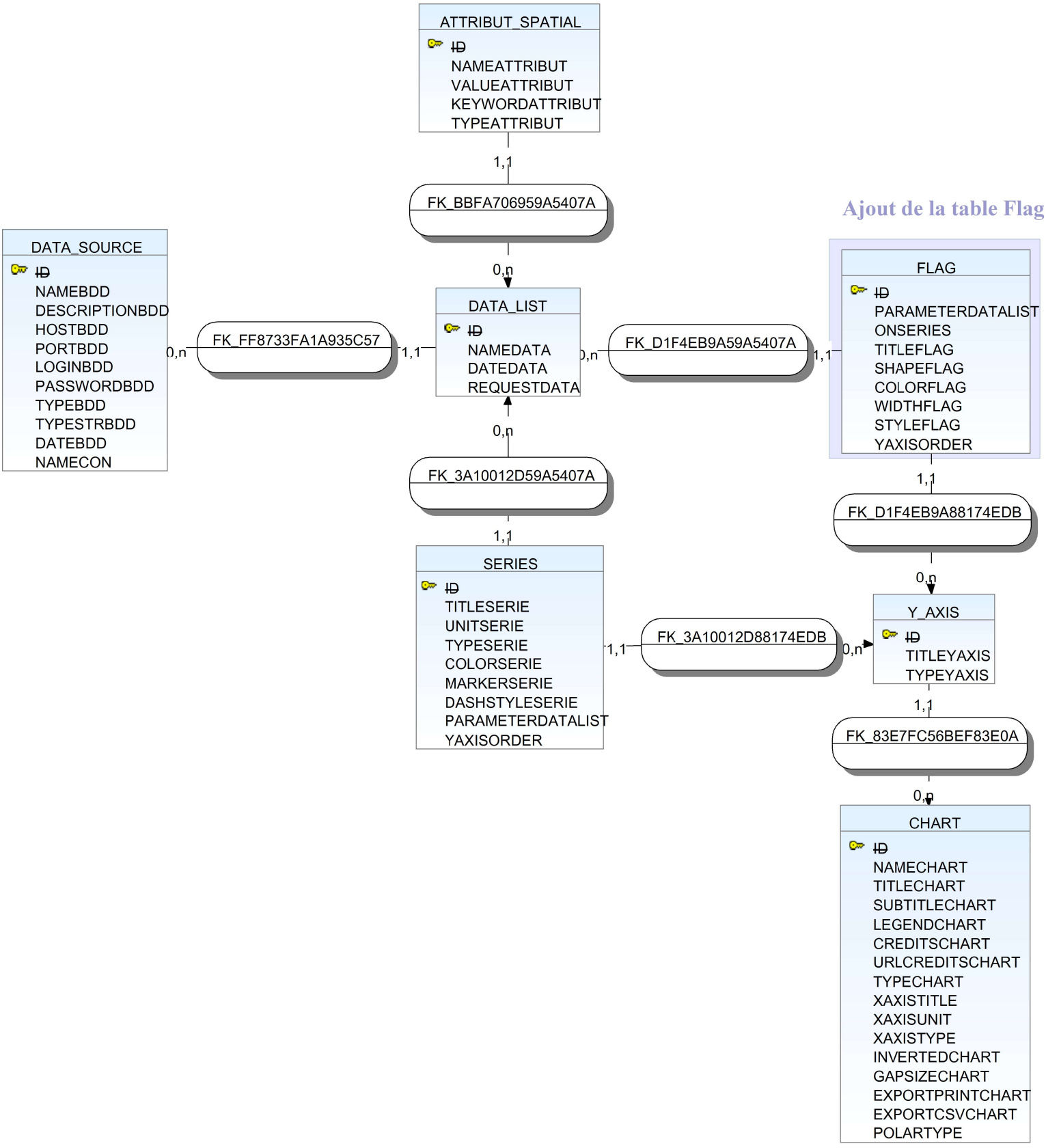
- 1 requête ne provient que d'une seule base de données. 1 base de données peut générer plusieurs requêtes (OneToMany).

- 1 requête peut contenir plusieurs attributs spatiaux (OneToMany). 1 attribut spatial est contenu dans une requête.

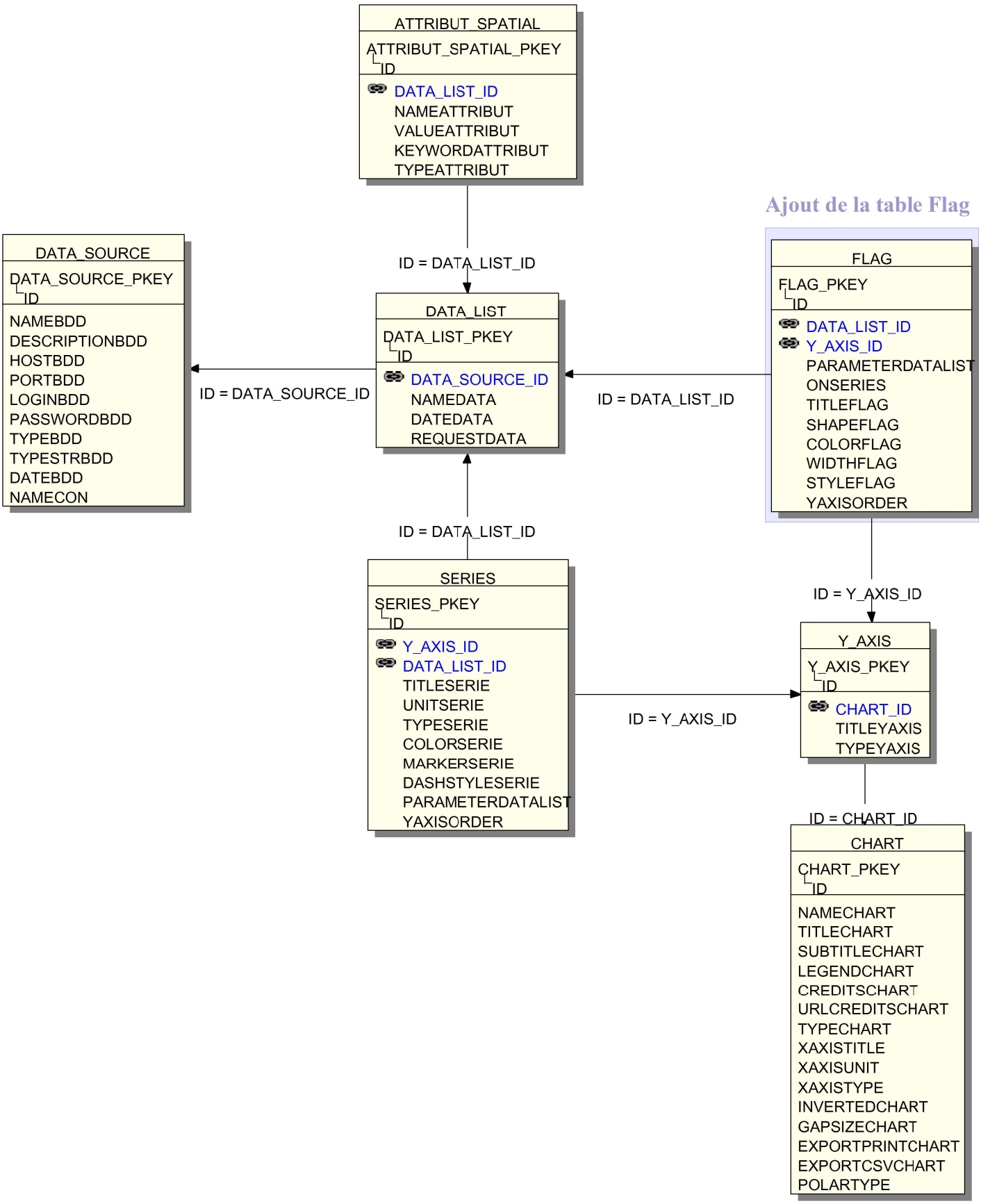
Les *flags* ont le même rôle que *séries*, la représentation des données, cependant ils possèdent des caractéristiques qui leur sont propres d'où la création d'une nouvelle entité mais possédant des associations similaires.

TABLE FLAG			
Nom	*Signification*	*Type*	*Non nul*
ID	Identifiant de la personne (Automatique en Symfony)	INTEGER	true
DATALIST	Base de données sélectionnée	Entité DATALIST	false
PARAMETERDATALIST	Requête sélectionnée	STRING	false
TITLEFLAG	Titre de la série de flag	STRING	true
COLORFLAG	Couleur du flag	STRING	false
SHAPEFLAG	Forme du flag	STRING	false
ONSERIES	Localisation de la série Axe X ou série pré-existante	STRING	true
WIDTHFLAG	Largeur du flag	INTEGER	false
STYLEFLAG	Police d'écriture du flag	STRING	false
YAXISORDER	Ordre de création de la série sur l'axe Y	INTEGER	true

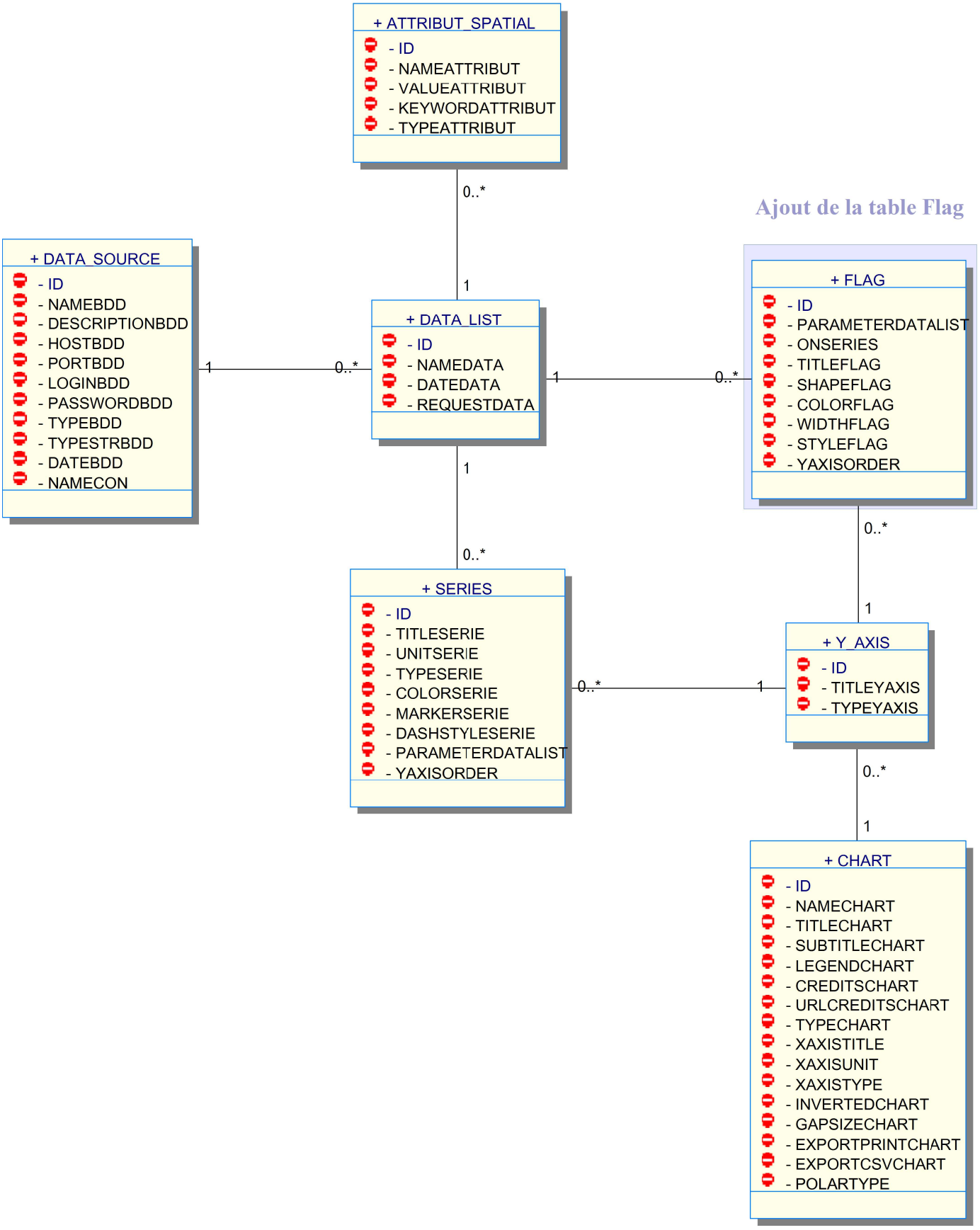
3.4.2. Modèle conceptuel de données



3.4.3. Modèle logique relationnel



3.4.4. Modèle logique objet



4. Modélisation technique

4.1. Choix technologiques

L'application *ManageChart* est projet web dynamique codé en PHP5 et JavaScript.

La base de données utilisée est une base de données PostgreSQL.

4.2. Framework et librairies

Le Framework *Symfony* 2.8 a été choisi. Afin d'accroître la rapidité de programmation, les librairies *jQuery* pour apporter le dynamisme et *Bootstrap* pour l'apparence générale de l'application ainsi que pour l'aspect « Responsive » de l'application.

Le but de l'application *ManageChart* étant de fournir des représentations graphiques, la librairie **HighCharts**, permettant la création de graphiques interactifs, a été sélectionnée car elle offre une grande variété de graphiques et d'options d'exports.

Elle se décline en quatre sous-librairies :

- ✓ **highcharts** permettant la génération de graphiques.
- ✓ **highstocks** permettant la génération de graphiques spécifiquement temporel. Cette librairie a été la principale utilisée car les flags sont inclus dans cette librairie, ainsi que la librairie **highcharts**.
- ✓ **exporting** permettant l'ajout d'options d'export au format image et PDF, ainsi que l'impression papier.
- ✓ **export-csv** permettant l'ajout d'options d'export au format CSV, XLS, ainsi que la visualisation dans le navigateur. Celle-ci dépend de la précédente.

4.3. Outils et plateformes

L'application a été principalement codée dans un environnement Linux (Ubuntu) grâce au logiciel Sublime Text 3 et ses nombreux plugins. Afin d'effectuer un suivi, le logiciel de gestion de versions Git a également été utilisé.
La modélisation du domaine a été réalisée avec *WinDesign*, les digrammes de cas d'utilisation et d'activité avec *MagicDraw* et les autres diagrammes avec *VisualParadigm*. J'ai également utilisé le logiciel *DbVisualiser* qui permet comme son nom l'indique de visualiser rapidement n'importe quelle base de données, et leur association.

5. Conception générale

5.1. Patterns utilisés

Le framework Symfony permet de créer une application en respectant totalement le pattern MVC. Il permet également une plus grande facilité avec la maintenance de l'application. Ce framework intègre par défaut un contrôleur frontal qui capte les requêtes et les redistribue.

Dans Symfony, la partie modèle est entièrement gérée par Doctrine. Doctrine est un ORM (*object-relational-mapping*), une couche d'abstraction à la base de données pour PHP. Il est l'ORM par défaut du framework Symfony. Il va gérer automatiquement les entités dans la base de données *ManageChart*.

L'architecture Symfony repose sur le principe des *bundles*. Chaque bundle est structuré selon le modèle MVC, modifiable si besoin. L'application ManageChart a été crée en utilisant le modèle triple Héritage pour chaque bundle (*Rapport de stage Maxime Collin, 2014*).

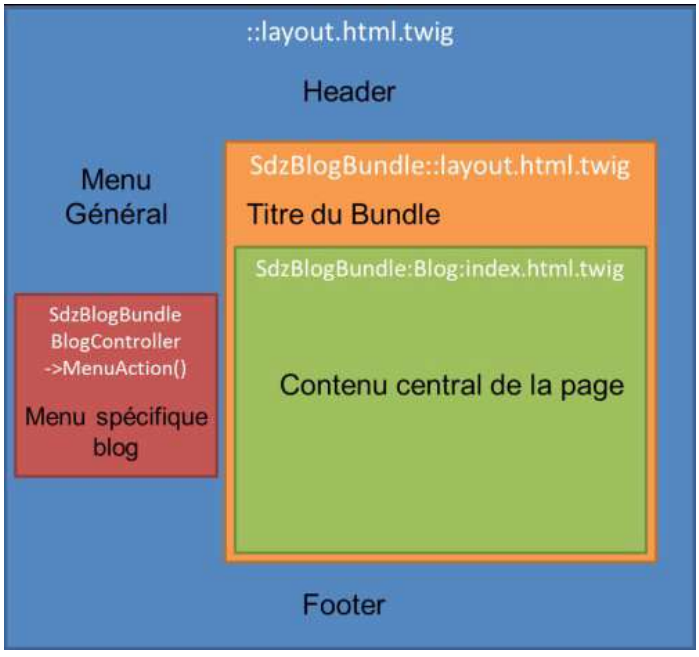
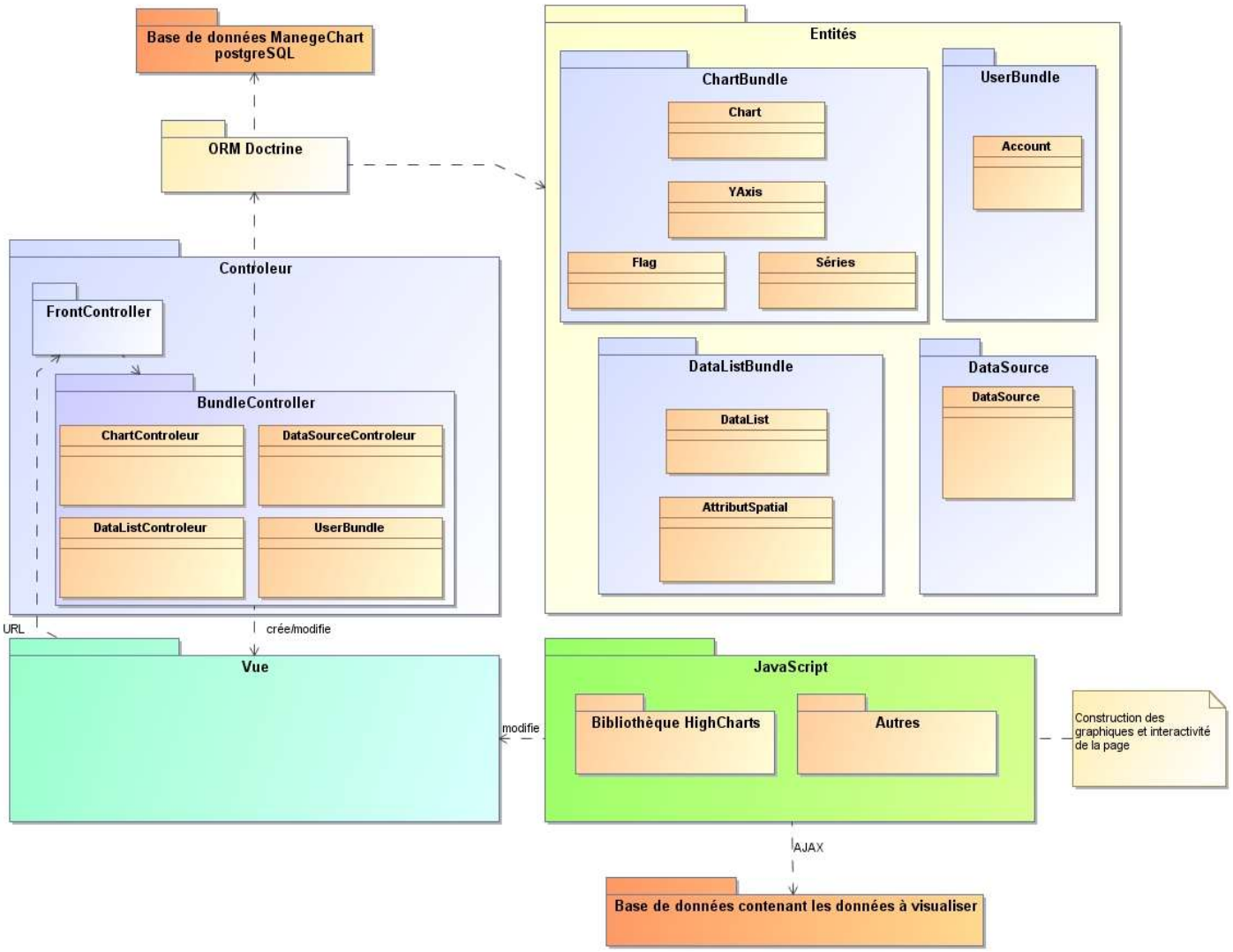


Schéma du modèle triple héritage

La partie vue est générée avec le moteur de template *Twig*. Cette vue est modifiée en JavaScript avec des requêtes AJAX.



Architecture général de ManageChart

5.2. Appel de la base de données

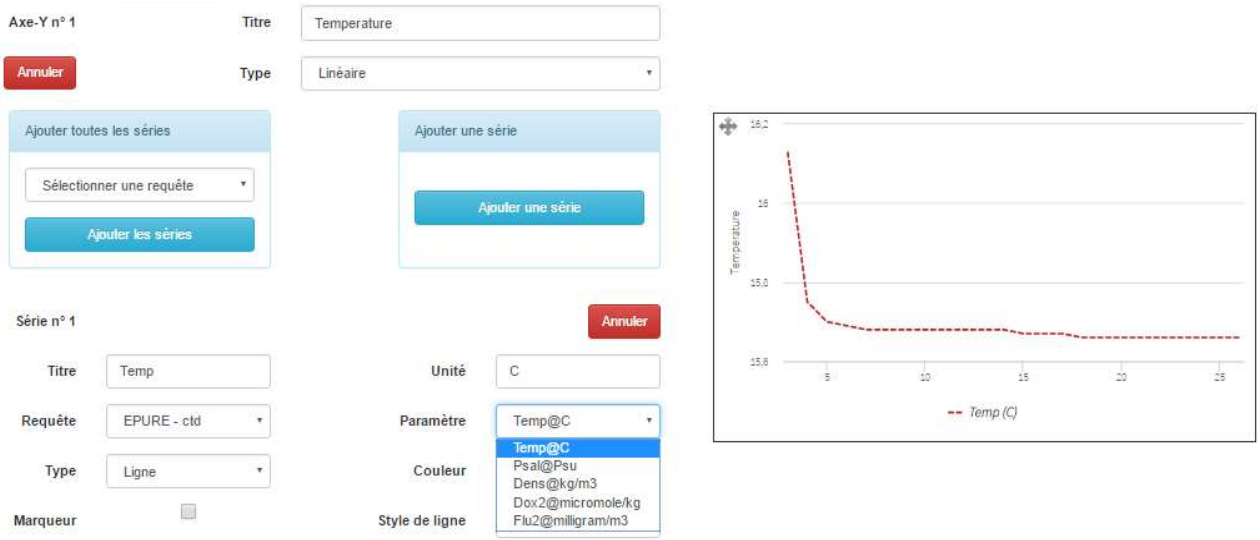
5.2.1. Création de graphique

Lors de l'appel des vues des formulaires d'ajout de série, on interroge la base de données en PHP afin de générer la liste des requêtes ainsi que celle des paramètres .

Dans chaque Bundle se trouve des « repositories ». Il en existe 1 par entité. Ces repositories sont des objets issus de Doctrine qui servent à récupérer les entités. Pour la liste des requête, on fait appel au repository de *DataList* : *DataListRepository* et on trie les requêtes en fonction de leur nom et par ordre chronologique.

```
function(DataListRepository $dr) {
        return $dr->createQueryBuilder('d')->orderBy('d.nameData', 'ASC');}
```

Pour les paramètres, un simple *getParameter()* de la requête choisie suffit.



Liste de requête et prévisualisation lors de création d'un graphique

Ensuite, les données résultantes des requêtes sont obtenues par un appel AJAX (Asynchronous JavaScript And XML) en JQuery. Ces données sont ainsi visualisées directement lors de la création et la modification des graphiques :

Fonction getJSON(url, [data]) ;

5.2.2. Visualisation

Les graphiques enregistrés sont appelés directement dans la page HTML par du code JavaScript avec le moteur de template *Twig*. Ce dernier permet d'appeler directement les entités grâce a une syntaxe simplifiée. Exemple d'une boucle for sur les entités Yaxis contenu dans le graphique :

```
{% for yAxis in chart.YAxis %}
        ...
{% endfor %}
```

6. Conception détaillée

6.1. Ajout de l'entité flag et ses conséquences

6.1.1. Ajout de l'entité

Avec Symfony, les entités sont générées grâce à l'ORM doctrine en console :

```
php app/console generate:doctrine:entity
```

Il faut ensuite créer les associations directement dans le fichier PHP de l'entité générée. Les associations sont construites à l'aide de commentaires spécifiques à doctrine :

```
/**
 * @ORM\ManyToOne(targetEntity="Mc\ChartBundle\Entity\YAxis", inversedBy="flag")
 * @ORM\JoinColumn(nullable=false)
 */
private $yAxis;
```

Cet exemple montre que plusieurs flags sont possibles sur *yAxis* avec l'ajout de ManyToOne suivis de l'entité. « *InversedBy* » signifie que l'association est de type bidirectionnel et non plus unidirectionnelle.
« *nullable=false* » indique que cet attribut ne peut être nul.

Après cet étape on signifie à doctrine qu'il peut créer la table *flag* dans la base de données :
1) Comparer l'état actuel de la base de données avec ce qu'elle devrait être en tenant compte de toutes nos entité

```
php app/console doctrine:schema:update –dump-sql
```

2) Exécuter les requêtes

```
php app/console doctrine:schema:update –force
```

6.1.2. création des formulaires

Afin de remplir l'entité, on construit le formulaire en PHP.

Formulaire d'une série de type flag

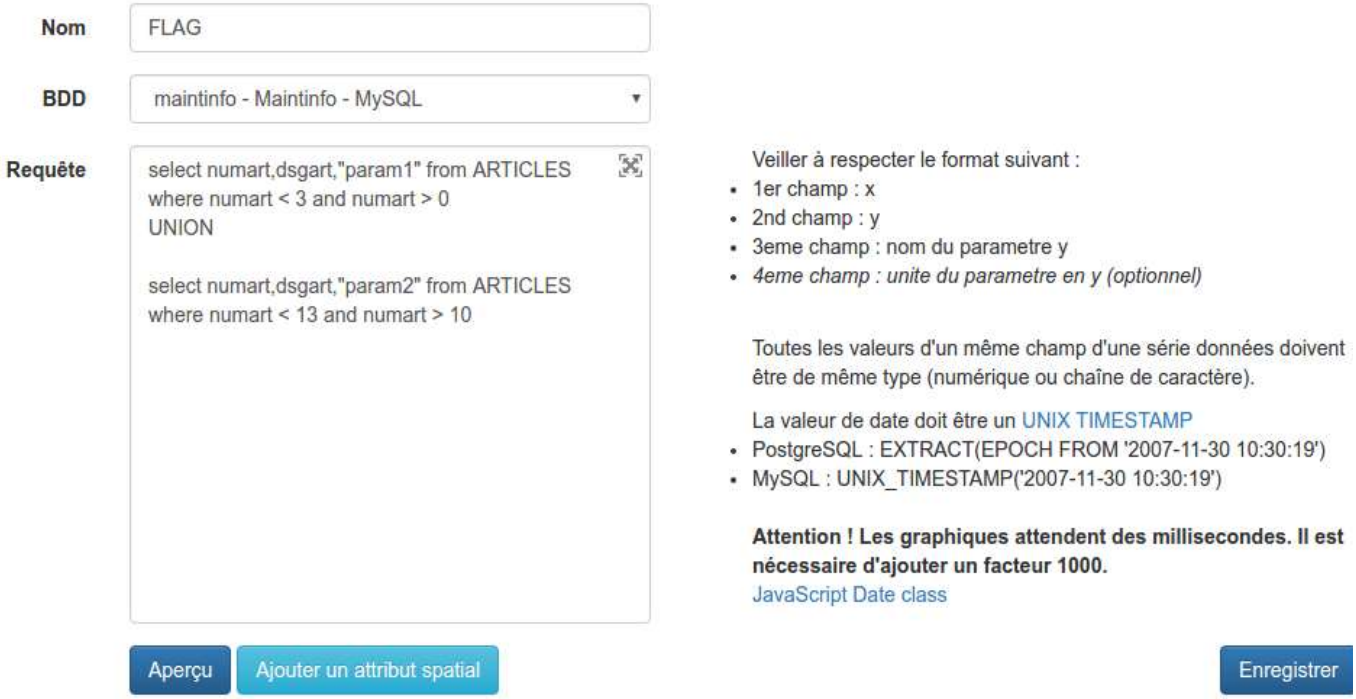
Doctrine se sert de ce formulaire pour construire automatiquement les objets. L'entité est créée en base de données lors de l'enregistrement.

6.1.3. Modification du contrôleur

L'ORM Doctrine de Symfony s'occupe de la gestion des entités. Il faut toutefois lui notifier dans le contrôleur l'ajout de l'entité flag (Cf. Codage 8.1).

6.2. Requête SQL

Le format des requêtes est stricte dans *ManageChart*. Le 1er champ représente l'axe X, le 2ème, l'axe Y, le 3ème, un nom de paramètre et le 4ème, facultatif, l'unité du paramètre.



Vue de la création d'une requête SQL

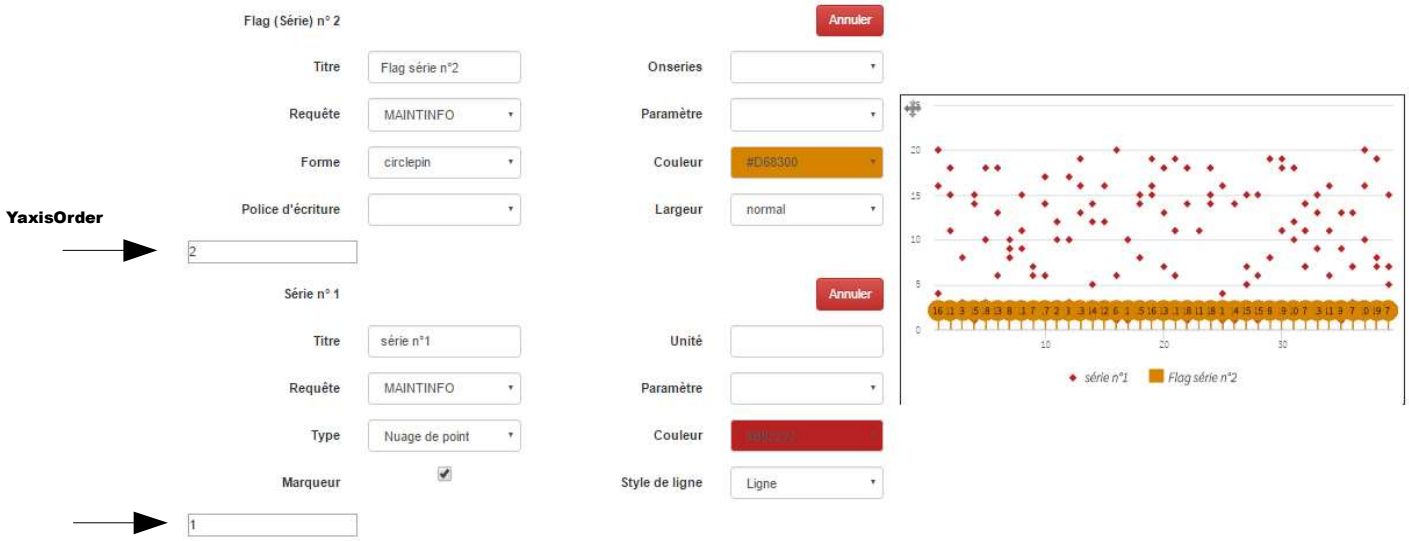
6.2.1. Modification de la base de données

Les séries normales ne peuvent avoir en axe vertical que des entiers numériques. Les *flags*, au contraire, permettent l'utilisation de chaîne de caractères (le cas le plus fréquent). Ces chaînes de caractère vont être utilisées pour remplir ces *flags*.

Lors de l'enregistrement, il faut donc bien différencier comment représenter les données, soit avec des flags soit avec des séries.

J'ai ajouté un attribut **YaxisOrder** aux tables *séries* et *flag*, indiquant l'ordre de création de la série ou du flag sur l'axe vertical. Cet ordre est primordial pour une visualisation ou une édition des graphiques après enregistrement.

Ce paramètre est modifié en JavaScript dans un formulaire afin de pouvoir être enregistré en base de donnée. Ce formulaire est toutefois caché car il n'apporte rien de concret à la création du graphique.



Vue des formulaires séries et flag avec le champ YaxisOrder non caché

6.2.2. Modification du JavaScript

Il ne faut pas confondre les entités PHP et JavaScript. En PHP les entités *séries* et *flags* sont complètements différentes. En JavaScript, les flags sont des séries d'un type particulier, le type flag.

Les différentes fonctions utilisées en JavaScript pour remplir et visualiser un graphique ont du être modifiées. J'ai ajouté un paramètre booléen *flag* qui permet de traiter les données différemment suivant le type de série. Si le booléen est vrai, alors la série de données est de type flag et les données ont la possibilité d'être des chaînes de caractère.

Afin de rendre le code plus lisible et plus facile à corriger, j'ai également changé le mode de lecture des données en JavaScript.

Voici la modification de paramètre contenu dans un graphique :

- ✓ Avant modification : **chart.series(Idseries).update(data) ;**

Les séries sont définies en fonction d'un numéro de série particulier. Il ne permet pas de connaître à quel axe vertical il appartient.

- ✓ Après modification : **chart.yAxis(IdYaxis).series(Idseries).update(data) ;**

Chaque série, que ce soit une série normale ou un flag, est désormais défini en JavaScript par son appartenance à son axe Y et son numéro de série.

Résolution d'un bug récurrent de cette application :

Si par exemple le graphique contenait plusieurs axes verticaux et que l'on voulait supprimer le 1er, l'application ne fonctionnait plus. En effet, la bibliothèque *Highcharts* nécessite que le 1er axe soit le numéro 1. L'ordre des axes Y n'était pas mis à jour. Cet ordre est dorénavant changé automatiquement en JavaScript. L'ajout de cet attribut relatif à l'ordre des séries dans l'axe Y a donc permis la résolution de cette erreur.

6.3. Divers

La première partie de mon stage a été consacrée à la résolution de bug. Cela m'a permis de prendre en main l'application. Ces bugs ont été de sources diverses et j'ai pu commencé à me familiariser avec les différentes technologies employées dans *ManageChart*.

Liste non exhaustive des bugs corrigées :

- Correction sur l'export en csv et échappement des guillemets
- Ajout d'un Gif indiquant que le fichier exporté est en cours de préparation
- Correction sur l'ouverture URL du crédit
- Ajout du menu graphique pour le rôle *scientifique plus*
- Correction de l'ordre des séries (inversé entre la création et l'ouverture)
- Correction d'une non pris en compte du changement de type d'axe horizontal pour les graphiques dynamiques et polaires.

7. Diagrammes

7.1. Diagramme de classe

7.1.1. Entités

Le diagramme de classe des entités est proche de la modélisation des tables de la base de données. Les entités sont regroupés dans 4 bundles :

- **ChartBundle** => *Chart*, *Yaxis*, *Séries* et *Flag*
- **DataListBundle** => *AttributSpatial* et *Datalist*
- **DataSourceBundle** => *DataSource*
- **UserBundle** => *Account*

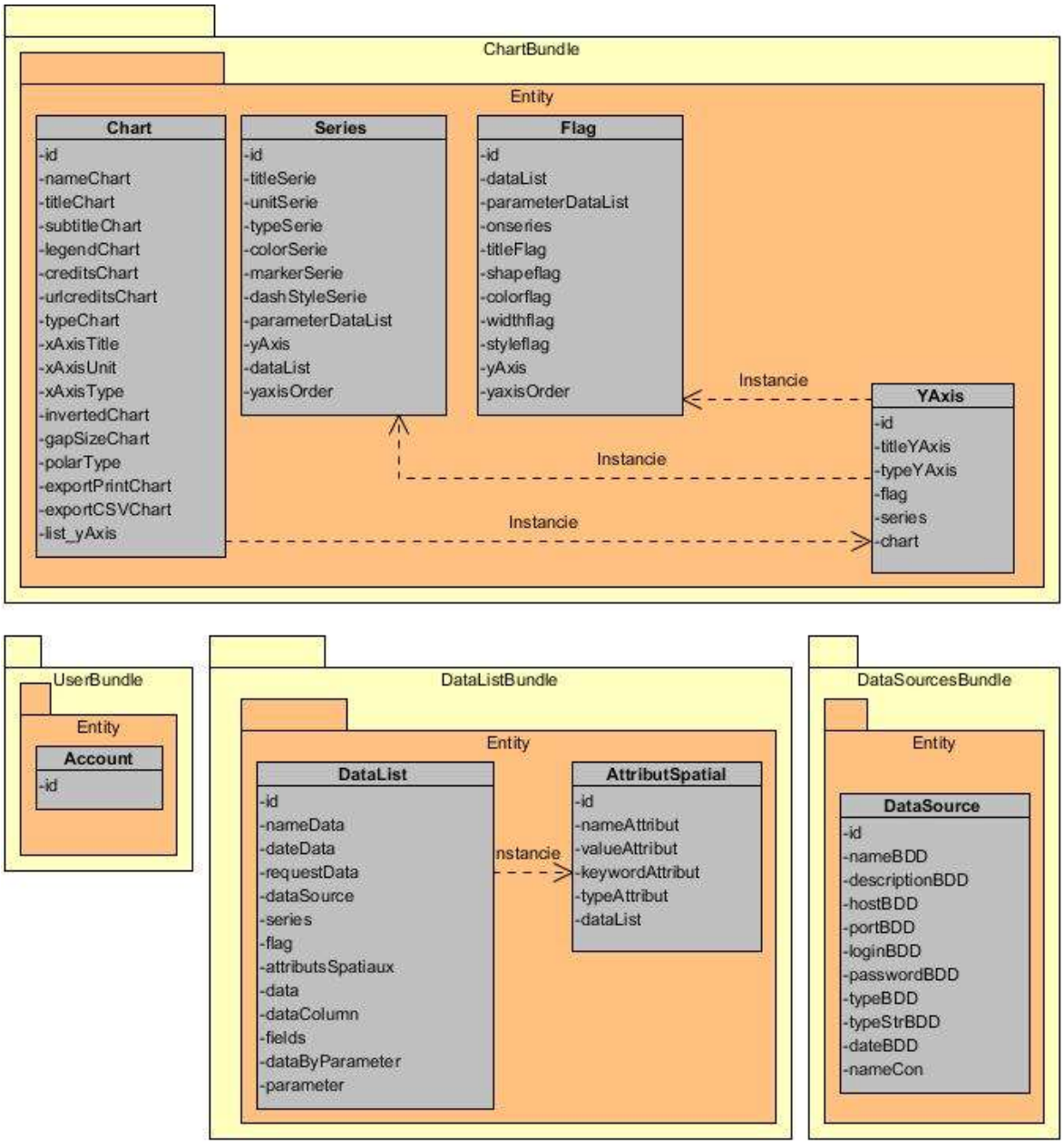
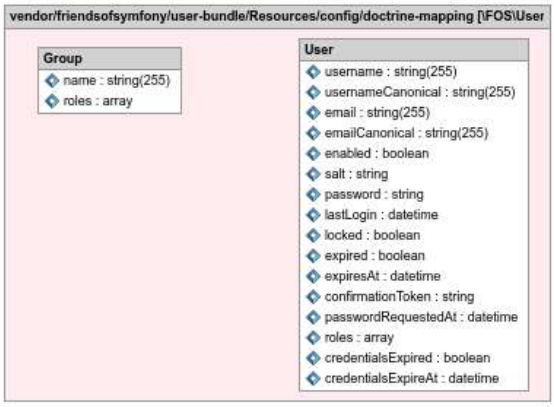


Diagramme de classe des entités

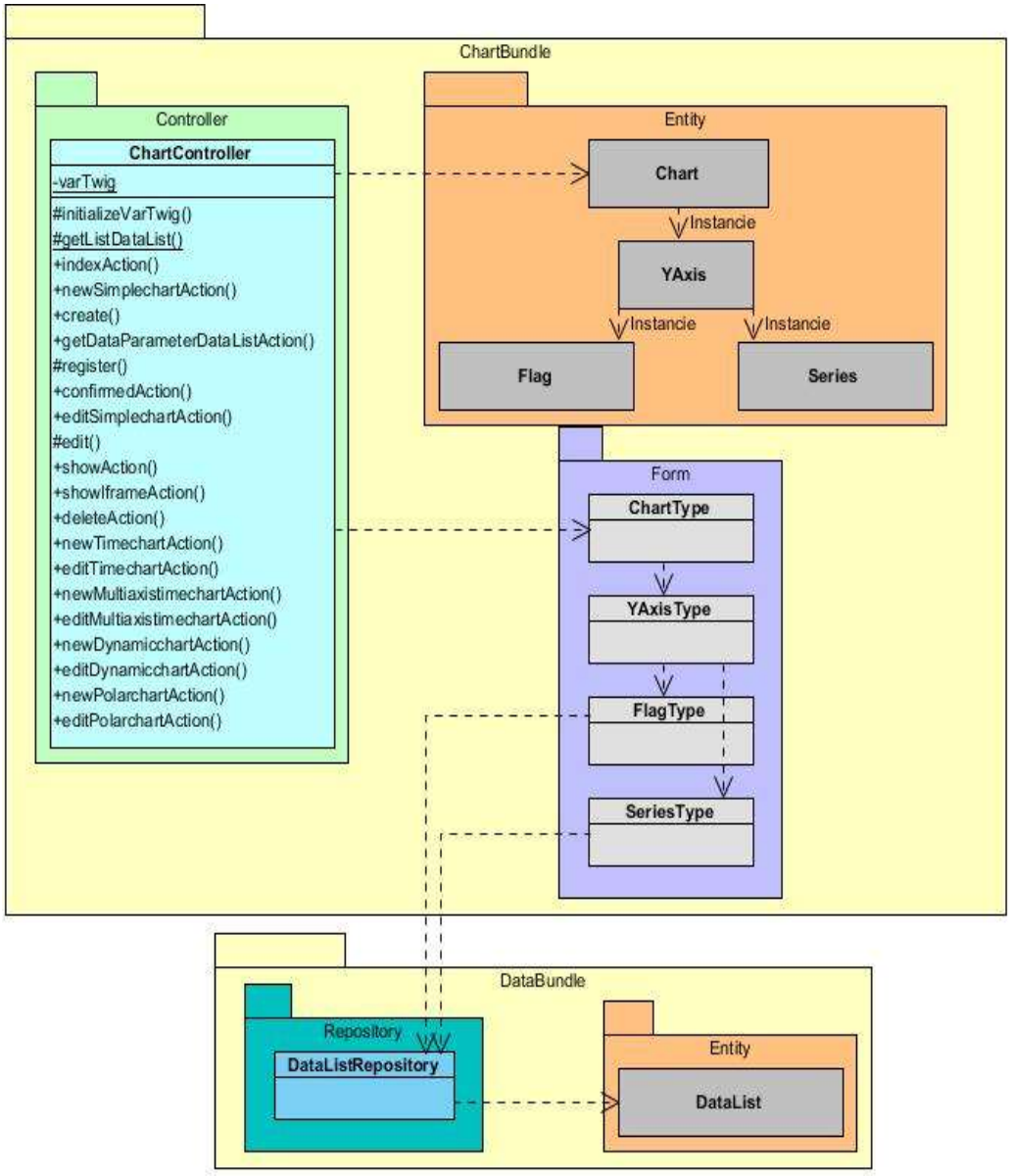


L'entité *account* est gérée par le bundleFOSUserBundle.

Diagramme de classe bundleFOSUserBundle

7.1.2. Contrôleur

Voici le diagramme de classe montrant les interactions du contrôleur du bundle de construction de graphique : ***ChartController***.



8. Codage

8.1. L'enregistrement en base de données via le contrôleur

L'ORM Doctrine de Symfony s'occupe de la gestion des entités. Il faut cependant qu'il enregistre les séries et les axes verticaux dans le bon ordre. La modification de la fonction d'enregistrement dans le contrôleur a ainsi été très importante.

```
/**
    * Enregistrement du graphique en BDD
    */
   protected function register($chart, $edit) {
      $em = $this->getDoctrine()->getManager();

      /* Enregistrement en trois étapes
         - Enregistrement du chart vide,
         - Enregistrement de ces axes Y vides,
         - Enregistrement des séries de données pour chaque axe Y
       */
      $listYAxis = $chart->getListYAxis()->toArray(); //tableau des axes Y sous la forme clé=> valeur

      $chart->getListYAxis()->clear();

      if (!$edit) {
         /* Enregistrement du chart */
         $em->persist($chart);
         $em->flush();
      }

      $listSeries = array();
      $listFlag = array();

      foreach ($listYAxis as $yAxis) {
         $yAxis->setChart($chart);

         $series = $yAxis->getSeries()->toArray();
         $flag= $yAxis->getFlag()->toArray();
         $yAxis->getSeries()->clear();
         $yAxis->getFlag()->clear();

         $listSeries[] = $series;
         $listFlag[] = $flag;
         $em->persist($yAxis);
      }
      /* Enregistrement des axes Y */
      $em->flush();

      // Enregistrer un tableau indice => clé de $listYAxis
      $cleYaxis = array_keys($listYAxis);

      $index = count($cleYaxis)-1; // index max
      foreach ($listSeries as $series) {
         $cle = $cleYaxis[$index]; // clé de l'indice
```

```
        $yAxis = $listYAxis[$cle]; //axe avec la clé

        foreach ($series as $serie) {
            $serie->setYAxis($yAxis);
            $em->persist($serie);
        }
        $index--;
    }

    $index = count($cleYaxis)-1;
    foreach ($listFlag as $series) {
        $cle = $cleYaxis[$index];
        $yAxis = $listYAxis[$cle];

        foreach ($series as $serie) {
            $serie->setYAxis($yAxis);
            $em->persist($serie);
        }
        $index--;
    }
    /* Enregistrement des séries */
    $em->flush();
}
```

8.2. Visualisation d'un graphique

Pour visualiser un graphique, il faut à l'aide des Twig parcourir en JavaScript pour chaque graphique les axes verticaux, et suivant la série, regarder si le graphique est de type *séries* ou *flag* :

```
{% for yAxis in chart.listYAxis %}

        // calcul du max YaxisOrder entre serie et flag
        {% set maxSerieOrder = ' ' %}
        {% for serie in yAxis.series %}
                {% set maxSerieOrder =  max(maxSerieOrder,serie.yaxisOrder) %}
        {% endfor %}
        {% for flag in yAxis.flag %}
                {% set maxSerieOrder =  max(maxSerieOrder,flag.yaxisOrder) %}
        {% endfor %}

        // bouche suivant l'ordre des series/flags
        {% for indexSerie in 1..maxSerieOrder %}
                {% for serie in yAxis.series %}
                        {% if indexSerie == serie.yaxisOrder %}
                                {

                                yAxis: 'yAxis' + {{ serie.yAxis.id|e('js') }},
                                color: '{{ serie.colorSerie|e('js') }}',
                                type: '{{ serie.typeSerie|e('js') }}',
                                data: []
                                },
                        {% endif %}
                {% endfor %}

                {% for flag in yAxis.flag %}
```

```
                        {% if indexSerie == flag.yaxisOrder %}
                                {
                                        yAxis: 'yAxis' + {{ flag.yAxis.id|e('js') }},
                                        color: '{{ flag.colorFlag|e('js') }}',
                                        fillColor: '{{ flag.colorFlag|e('js') }}',
                                        type: 'flags',
                                        name: '{{ flag.titleFlag|e('js') }}',
                                        shape: '{{ flag.shapeFlag|e('js') }}',
                                        data: []
                                },
                        {% endif %}
                {% endfor %}
        {% endfor %}
{% endfor %}
```

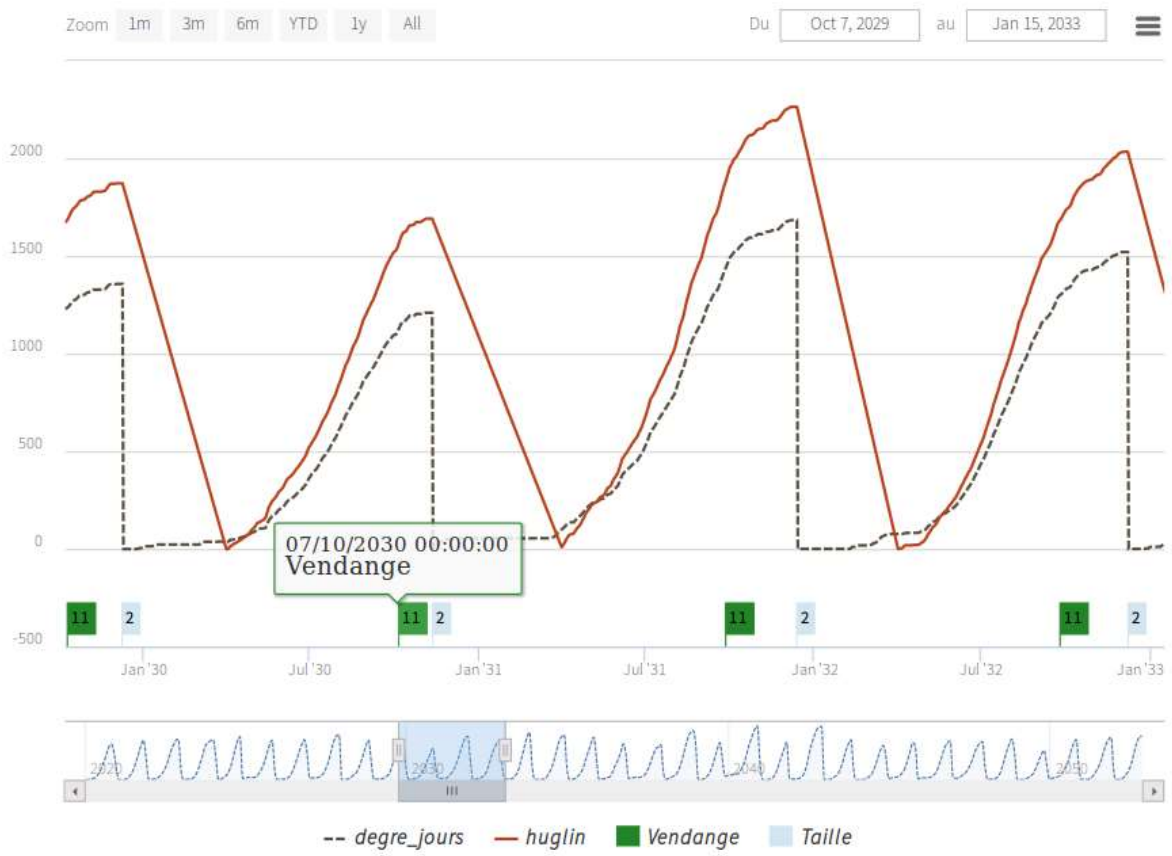
Puis on rempli le graphique avec la fonction setSerie qui récupère les données :

```
{% set indexY = 1 %}
{% for yAxis in chart.listYAxis %}
        {% for serie in yAxis.series %}
                {% set indexSerie = serie.yaxisOrder %}
                setSerie({{ indexSerie }}, {{ serie.dataList.id }},
                {{ serie.parameterDataList }}, {{ indexY }}, false, false,
                '{{ attributsSpatiaux|e('js') }}', {{ test }}, false);
        {% endfor %}

        {% for flag in yAxis.flag %}
                {% set indexFlag = flag.yaxisOrder %}
                        setSerie({{ indexFlag }}, {{ flag.dataList.id }},
                {{ flag.parameterDataList }}, {{ indexY }}, true, false,
        '{{ attributsSpatiaux|e('js') }}', {{ test }}, false);
        {% endfor %}
        {% set indexY = indexY + 1 %}
{% endfor %}
```

9. Conclusion

Durant mon stage au sein du LETG, j'ai ajouté une nouvelle fonctionnalité à l'application web *ManageChart* de visualisation de données scientifiques : l'ajout de marqueurs appelés aussi flags. Ces flags vont permettre une meilleure visualisation des stades phénologiques de la vigne ainsi que les actions agronomiques des viticulteurs en fonction des paramètres bioclimatiques (Cf. image ci-dessous).



Exemple de Flags (actions agronomiques sur la vigne) et de séries temporelles (indices bioclimatiques)

L'ajout de cette nouvelle entité à fait surgir des incompatibilités avec l'ancien code. J'ai du réaliser de nombreuses modifications comme l'ajout d'un attribut pour les entités *séries* et *flag* : Y*axisOrder*. Cet attribut permet d'indiquer l'ordre des séries sur chaque axe vertical, ce qui permet d'identifier précisément quelle entité choisir et donc une lecture et une ré-édition sans erreur.

J'ai également pu corriger quelques bugs, le plus important étant le bon enregistrement en base de données malgré des suppressions de série ou d'axes verticaux.

L'ensemble de ces modifications a été effectué en local sur la machine de développement et testé sur chaque type de graphique possible (graphique simple au polaire). Il est prévu de déployer ces modifications très prochainement sur l'application en production .

D'un point de vu personnel, ce stage a été extrêmement enrichissant. En effet, le stage m'a permis de ma familiariser avec de nombreuses technologies comme le framework Symfony (PHP) et jQuery (JavaScript).
L'application n'étant pas une nouvelle création, j'ai du reprendre un programme écrit par plusieurs personnes. Cet exercice montre fortement l'utilité de commenter efficacement son code.

Améliorations (possibles itérations) :

- Les flags ont la possibilité de pouvoir être affiché sur une série pré-existante. Je n'ai pas eu le temps d'implémenter cette fonctionnalité. Il faudrait enregistrer dans un tableau temporaire l'id des séries déjà visualisées dans le graphique et l'indiquer dans le JSON qui va générer les données :

```
chart.addSeries({
        id: idFlag,
         yAxis: 'yAxis' + i,
         type: 'flags',
         data:[],
         onSeries: IdSeries,
         color: firstColorSerie,
         fillColor: firstColorSerie
});
```

  Il faudrait également ajouter une association entre les entités *séries* et *flag*.

- Au stade actuel de l'application, il n'y a pas de lien entre les graphiques créés et l'utilisateur : toute personne connectée peut supprimer ou éditer n'importe quel graphique. Il faudrait ajouter à l'entité *Chart* l'utilisateur et son rôle par exemple.

- Lors du choix des requêtes, il faudrait au préalable faire un tri de ces requêtes en fonction du format du second paramètre, numérique ou chaîne de caractère.