RAPPORT DE STAGE

PROJET MANAGECHART

STAGE LETG Brest GEOMER

du 11 /05/2015 au 24 /07 /2015

Présenté par : **Robel KAKOA**

Spécialisation : Concepteur Développeur Informatique

Maitre de Stage :

Mathias ROUAN

Tuteur de Stage :

Gilbert DELEPECH

REMERCIEMENTS

Je souhaite remercier l'équipe LETG Brest Géomer pour leur accueil et leur bonne humeur.

Notre responsable du Projet, Mathias ROUAN pour leurs suivi attentif, régulier et leurs aides dans la compréhension complète du sujet; pour leur soutien permanent; pour les efforts importants qu'ils ont fourni malgré leur lourde charge de travail pour nous permettre et pour nous relire toutes les informations dont nous avions besoin pour la rédaction de ce rapport.

Notre encadrant du projet, Gilbert DELPECH pour sa pédagogie, son aide, ses corrections.

A tous ceux pour nous avoir fait confiance, pour leur investissement dans le projet, pour leurs spécifications claires, précises, pour leur synergie, pour leur motivation, et pour nous avoir permis de passer 10 semaines de projet déjà mis en place et exceptionnel, intéressants et formateurs.

Enfin merci à vous pour le temps que vous allez accorder à notre ouvrage.

Mes parents qui mon donne leur soutien moral, à tous ceux qui ont de prêt et de loin à la réalisation de cet œuvre.

Merci encore à vous tous.

TABLES DES MATIERES

LISTES DES FIGURES

I. Introduction

Dans le cadre de la préparation du titre professionnel niveau II Concepteur développeur informatique, à L'AFPA de Brest, j'ai réalisé un projet de mise en pratique accompagnée.

Nous sommes donc tourné vers l'application MANAGECHART ou la mission devrait permettre la gestion des connexions aux bases de données afin de pouvoir en ajouter, en modifier ou en supprimer et afin d'en permettre la visualisation graphique .Le dit développement nous a permis d'effectuer différentes taches dans le domaine informatique.

Ainsi, j'ai effectué ce stage, du 11 Mai 2015 au 24 juillet 2015, cette mission répondait à nos attentes en nous apportant une approche concrète du monde de l'entreprise et des différents acteurs en interaction avec le système informatique.

Celui-ci nous a permis d'obtenir une réelle expérience professionnelle et un aperçu de que peuvent être les missions confiées à un ingénieur informatique.

1) Présentation du LETG Brest Géomer

LETG-Brest Géomer est un des cinq laboratoires de l'UMR LETG (Littoral, Environnement, Télédétection, Géomatique). La partie brestoise est pluridisciplinaire (géographie humaine, géographie physique, géomatique), et ses axes de recherche se déclinent en sept grands thèmes :

- dynamiques de l'occupation des sols
- dynamiques géomorphologiques des littoraux
- risques côtiers
- fréquentation et usages (espaces littoraux et insulaires)
- gestion intégrée des zones côtières
- modélisation des activités humaines
- géomatique.

De plus, dans le cadre de l'OSU-IUEM, LETG Brest - Géomer alimente aussi une base de données sur l'évolution du trait de côte, grâce à un suivi morpho sédimentaire des plages du Finistère.

2) La mission du Stage

Il s'agit de développer et tester l'application web. L'application qu''on appelle Managechart s'inscrit dans les évolutions de l'Infrastructure de Données Spatiales (IDS) Indigeo.Cette IDS est destinée aux scientifiques désirant faire de la recherche et de l'observation sur l'environnement dans l'Ouest de la France. Elle se présente sous la forme d'un visualiseur cartographique, d'un catalogue de métadonnées et d'un serveur de données géo spatialisées adossés à un visualiseur cartographique.
L'application ManageChart permet la création de graphiques à partir de sources de données différentes, avec une gestion des permissions pour les comptes utilisateurs Ces graphiques sont visualisables dans une application externe telle que Indigeo qui peut les afficher dans une iframe avec son URL.

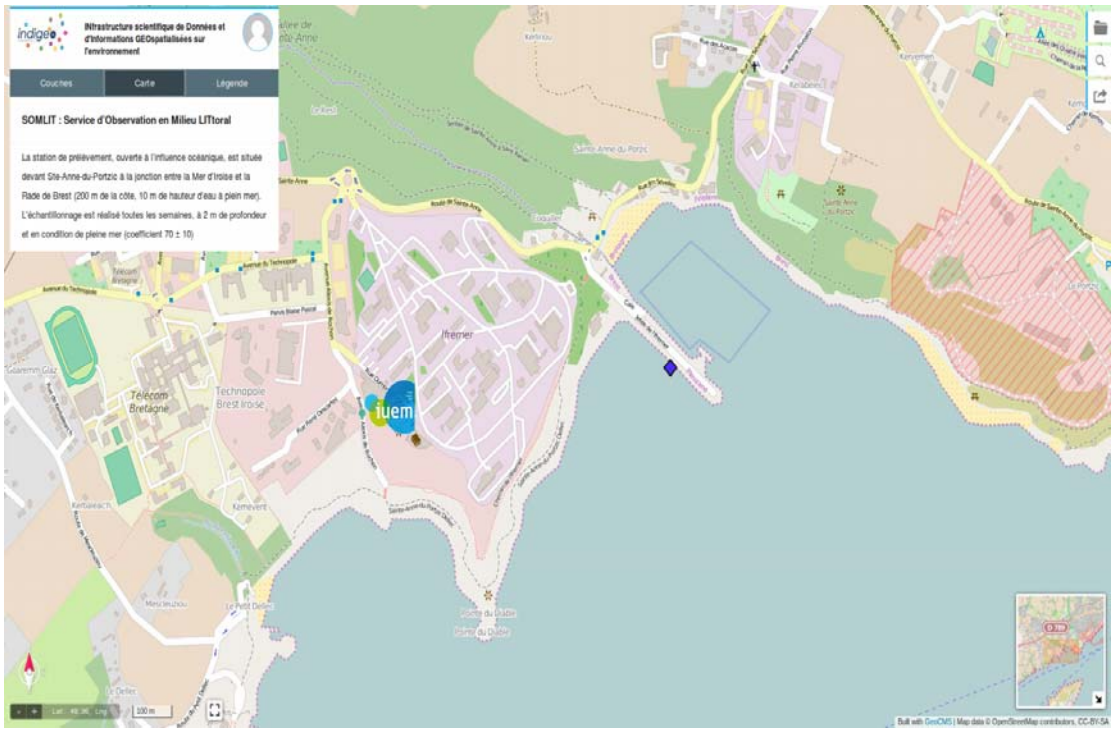


Figure 1 : Visualiseur d'Indigéo zoomé sur le technopôle et la plage de Sainte Anne avec la station somilit

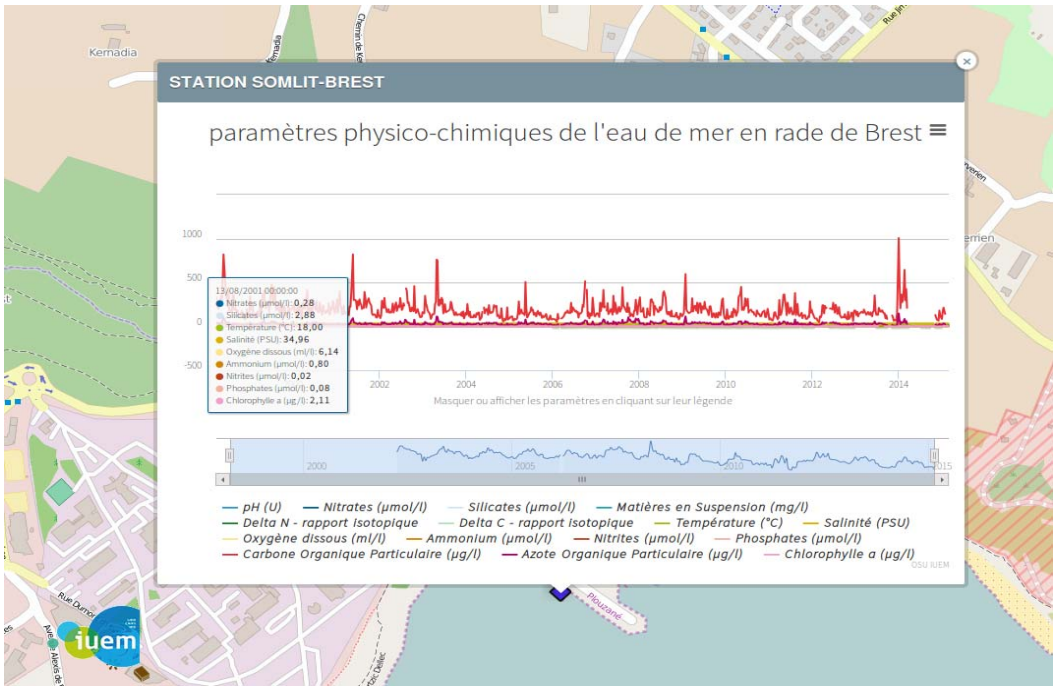


Figure 2: affichage d'une graphique station somlit-brest après l'interrogation

ManageChart s'inscrit dans les évolutions d'Indigéo. Une première application web de création et de visualisation de graphiques a déjà été développée par la société DotGee.

Cependant les graphiques créés par cette application sont insuffisamment paramétrables et notamment figés sur une abscisse temporelle. ManageChart doit donc reprendre les fonctionnalités de cette application et permettre le paramétrage des graphiques.

Préalablement à mon stage, trois étudiants (Amir Khnissi et Redouane Oulla) du Master 2 SIAM de l'UBO et Maxim Collin du Master 1 TIIL ont mené un projet exploratoire sur le parent direct de ManageChart. Ils ont développé une application permettant de montrer la faisabilité du projet et de mieux le définir.

Ils ont étudié une librairie de génération de graphique (HighChart) qui offre les fonctionnalités désirées. Ils ont aussi intégré les données produites par la FOSIT dans une base de données, et permis la génération de rapport. Cela dite, cette application n'est pas sécurisée et met en évidence la nécessité de développer de nouvelles Fonctionnalités, à la base de ce travail de stage.

Une partie de mon travail concerne l'amélioration de l'application ManageChart déjà existante. Elle permet de créer des graphiques à partir de sources de données différentes et de pouvoir afficher ceux-ci dans une application externe au travers d'une iframe. Pour utiliser une base de données il est donc nécessaire d'appliquer un prétraitement.

Les améliorations apportées concernent la mise à jour des librairies graphiques JavaScript HighChart et Highstock et la vérification de leur compatibilité avec l'application ainsi que l'ajout de 2 fonctionnalités à l'application. Le champ 'urlcredits' qui permet de retourner le lien hypertexte depuis le champ crédit et le champ 'Inverser les axes' qui permet d'intervertir les axes sur le graphe highChart.

Enfin l'Agrandissement des polices (date&heure) sur le graphe pour permettre une meilleur lisibilité.

II. Contexte du Projet

II-1 Le Projet existant : MANAGECHART

Le rôle de cette application est de pouvoir créer des graphiques à partir de sources de données différentes, et de pouvoir afficher ceux-ci dans une application externe au travers d'une iframe.

II-1.1 Cahier des Charges & Etudes

ManageChart permet la gestion des connexions. Une connexion à une base de données a besoin de connaître l'adresse du serveur de la base de données, son port, son nom, son type
et un identifiant ainsi qu'un mot de passe pour s'y connecter.

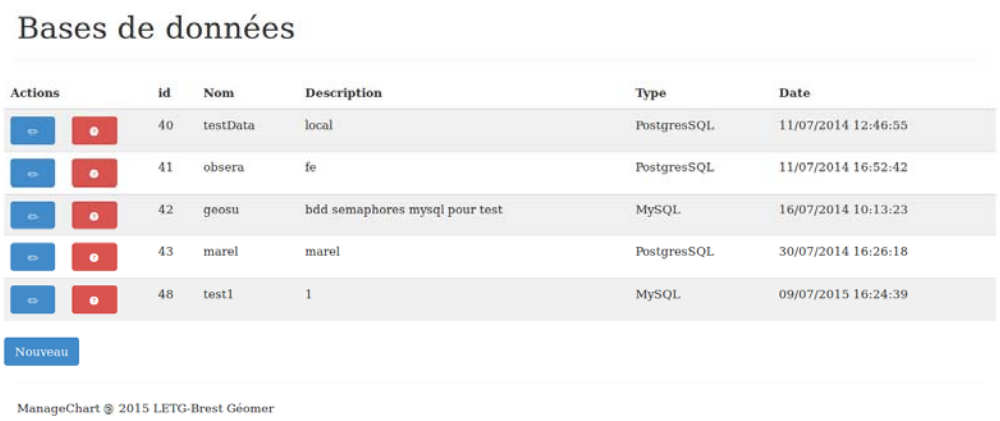


Figure 3 : maquette gestion des connexions aux bases de données

ManageChart géré la récupération des données

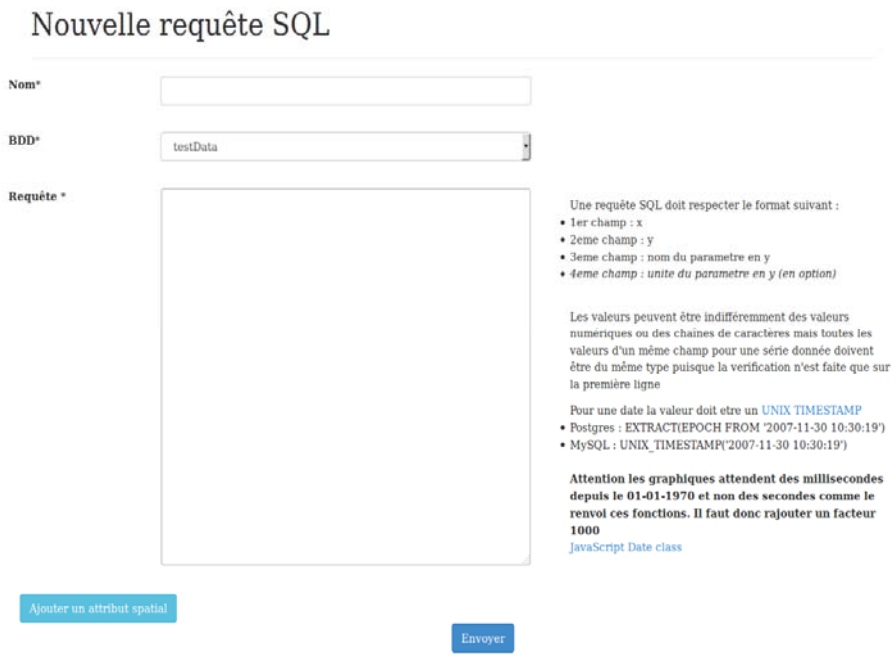


Figure 4 : maquette création d'une requête SQL

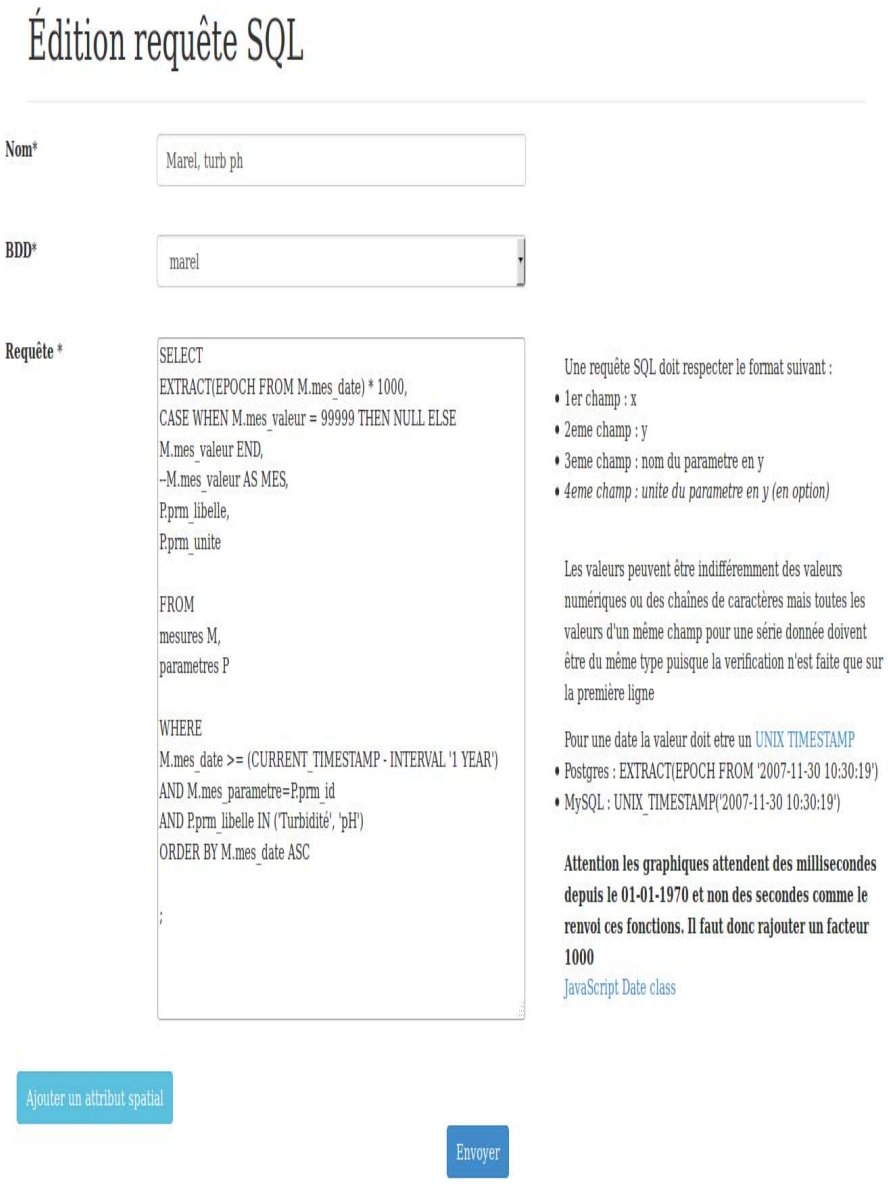


Figure 5 : modification d'une requête SQL

ManageChart permet ajouter, modifier, supprimer et visualiser des graphiques. Ces graphiques respectent une charte graphique définie par le laboratoire de LETG-Brest Géomer. On doit aussi pouvoir les paramétrer de façon à choisir :

- le titre et le sous-titre du graphique ;
- la requête SQL et le paramètre à afficher pour chaque série de données ;
- le type de graphique et sa couleur pour chaque série de données ;
- leur nom et leur unité ;
- leur axe en y (un existant ou un nouveau) et son type ;
- les sélecteurs d'échelle (uniquement pour l'axe x et si c'est un axe temporel)
- les labels, la légende, les crédits, les urlcredits et inverser les axes.

Lors de la création ou de la modification d'un graphique, celui-ci doit pouvoir être visualisé et rafraîchi à chaque modification de paramètre.
Si une requête SQL avec paramètres est choisie, des champs de saisie doivent être ajoutés au formulaire afin de renseigner les paramètres de sélection de l'objet.

Cela permettra de visualiser son jeu de données dans le graphique le temps de sa création. Par la suite ces paramètres devront être renseignés au travers de l'URL qui appellera ce graphique. Ces mêmes champs pourront être rajoutés si besoin dans l'interface de visualisation du graphique.

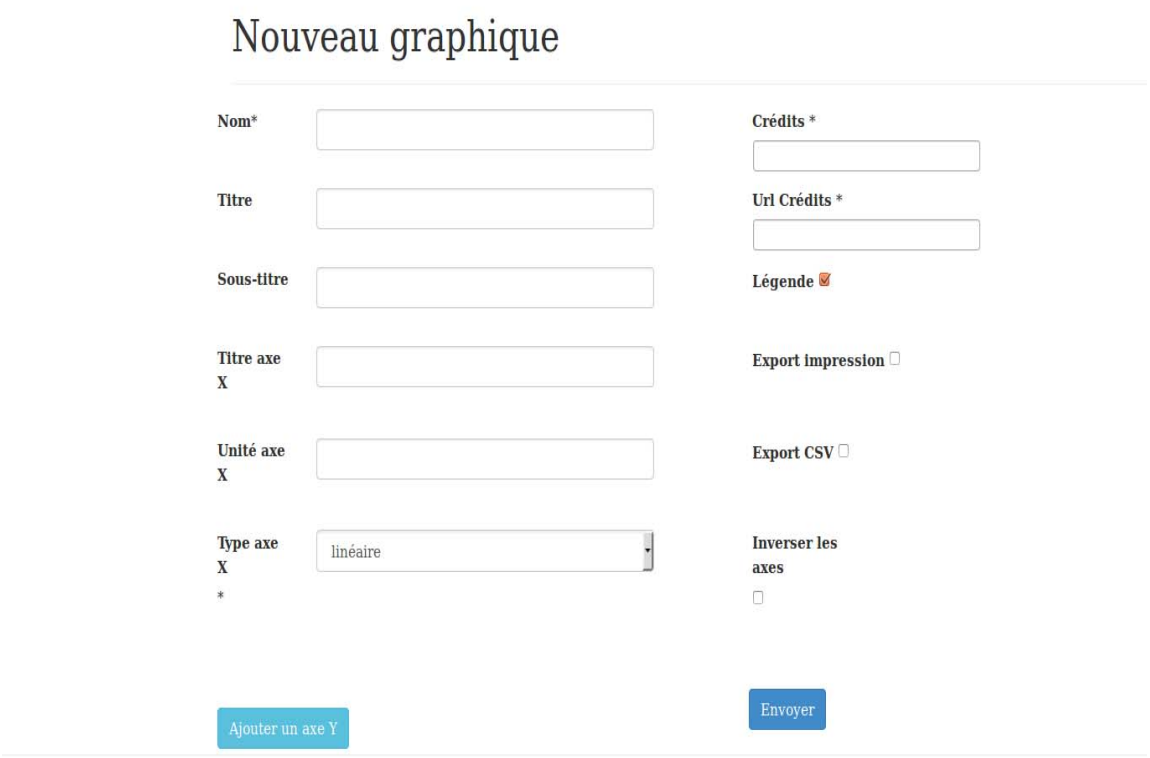


Figure 6: maquette création et modification d'un graphique HighChart

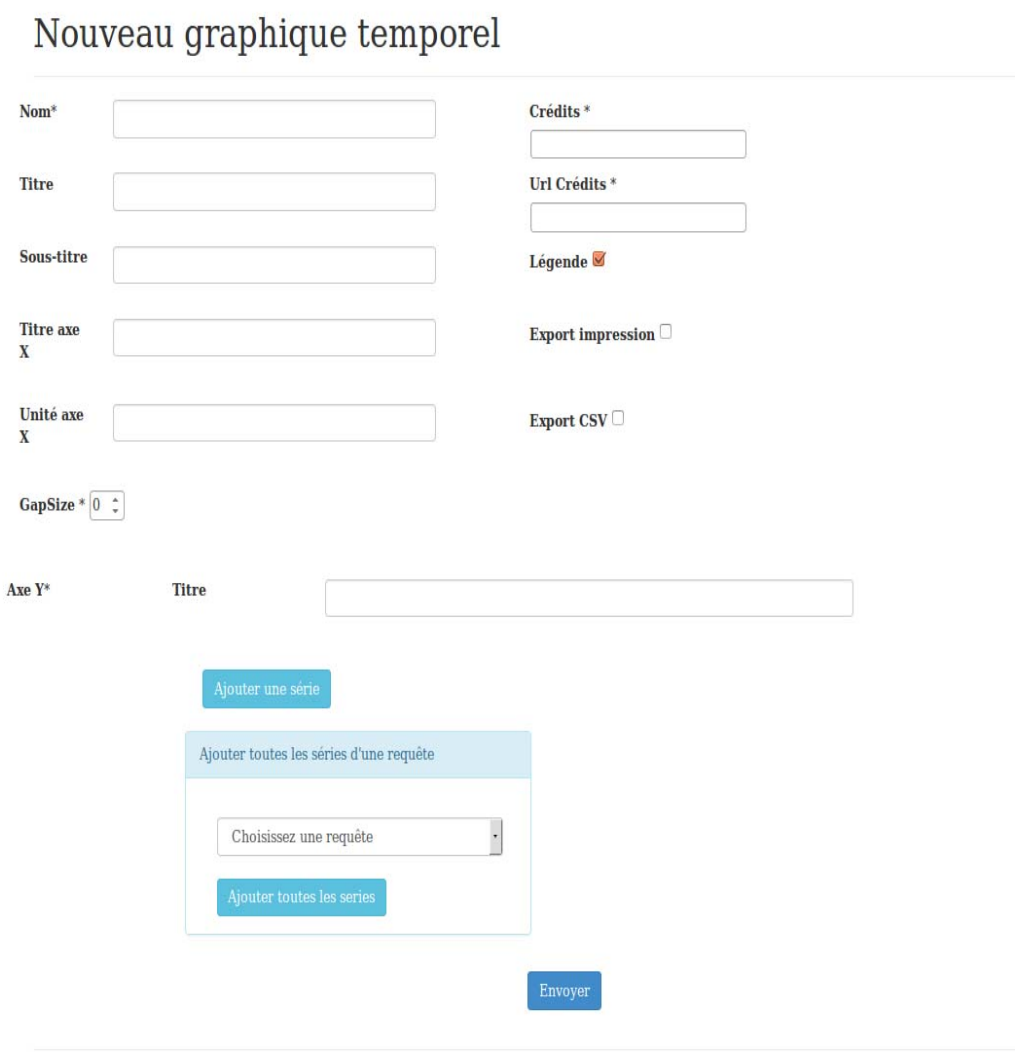


Figure 7 : maquette création et modification d'un graphique HighStock

La dernière fonctionnalité de ManageChart est la gestion des comptes utilisateurs et des permissions. Il doit être possible d'ajouter, de modifier, de supprimer et de visualiser un compte. Un compte a besoin d'une adresse e-mail comme identifiant, d'un mot de passe, et d'un type de compte (Administrateur ou Scientifique). Un Scientifique peut voir et modifier son compte, ainsi que créer, modifier ou visualiser les graphiques. Un Administrateur peut quant à lui tout faire à l'exception de modifier un autre compte.

Figure 8 : maquette création et modification d'un compte

b-Diagramme de Séquence

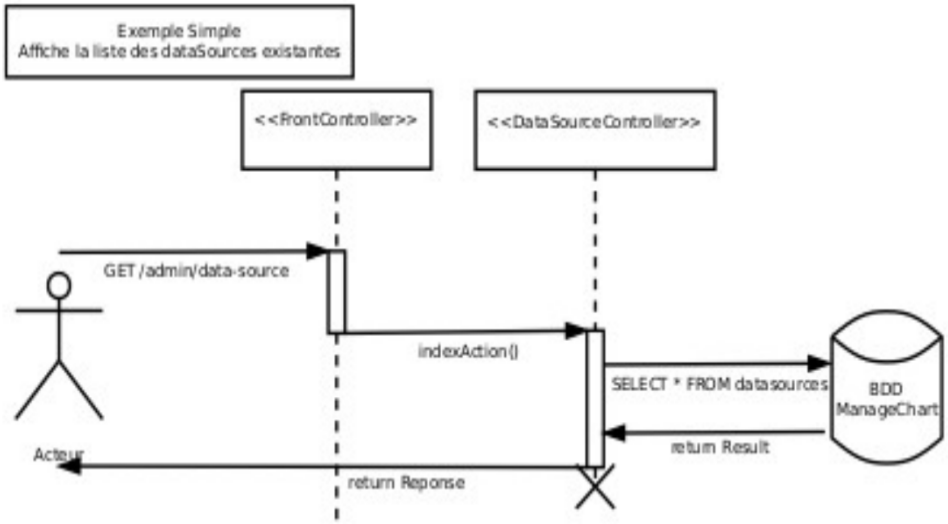


Figure 9 : Copie d'écran Maxim Collin Diagramme de séquence

Ici le client effectue une requête pour afficher la liste des connexions aux bases de données. Le Front Controller lit la requête et appelle le bon contrôleur avec la bonne méthode. En l'occurrence cette méthode (indexAction()) effectue une requête SQL sur la base de données de ManageChart afin de récupérer la liste des connexions aux bases de données distantes existantes.
Elle renvoie ensuite la réponse au client.
Cet exemple fonctionne aussi pour afficher les listes :

- des graphiques
- des requêtes SQL
- des utilisateurs
- et également les détails de chaque élément, mais à la différence d'avoir l'identifiant de l'élément dans l'URL.

L'accès à la base de données s'effectue alors au niveau du Front Controller, comme montré dans l'exemple suivant :

c-Diagramme de déploiement

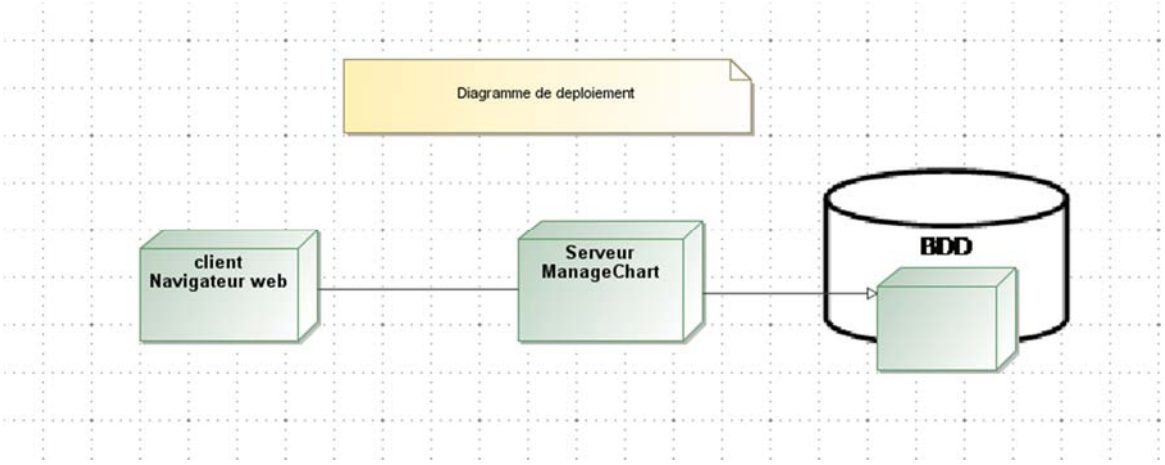


Figure 10 : diagramme de déploiement

d-Diagramme de Cas D'Utilisation

Le diagramme de Cas D'Utilisation montre toute la partie des gestions du graphe sur l'application ManageChart. L'Administrateur peut se connecter, puis accéder sur la visualisation des graphes. Il peut choisir soit la création ou édition du nouveau graphique (HighChart)/nouveau graphique temporel (HighStock).

Voici le schéma qui montre le diagramme :

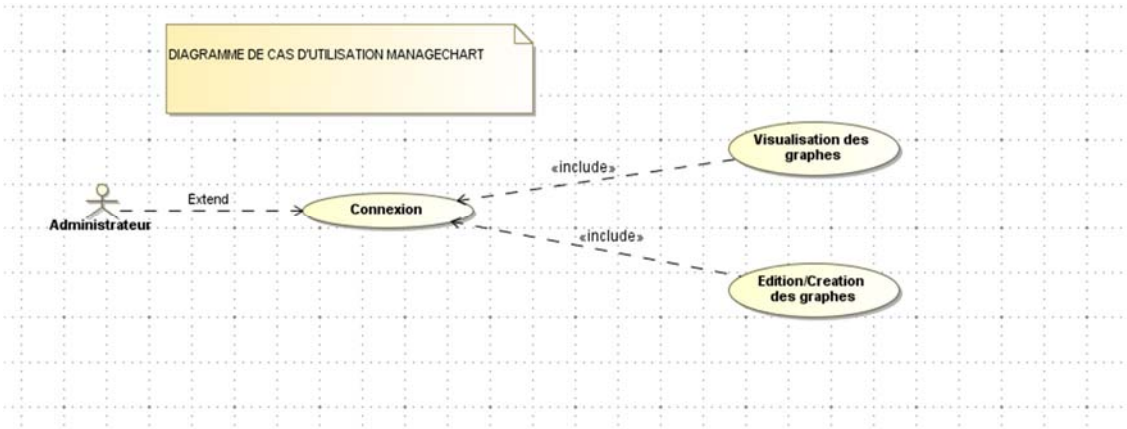


Figure 11: diagramme de cas d'utilisation

e-Diagramme d'activités

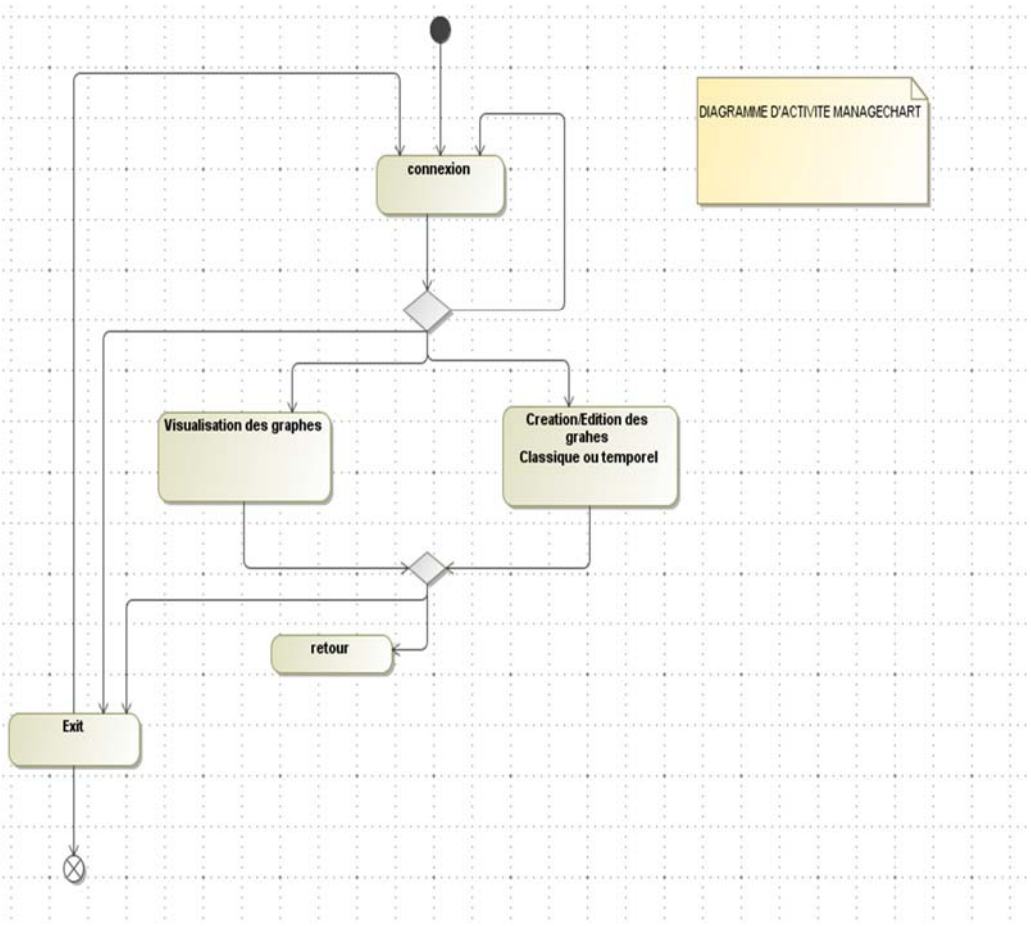


Figure 12 : Diagramme d'activités

f-Diagramme de composants

On a choisi une application du modèle MVC permettant de structurer le code.
Le diagramme de composants montre le principe MVC.

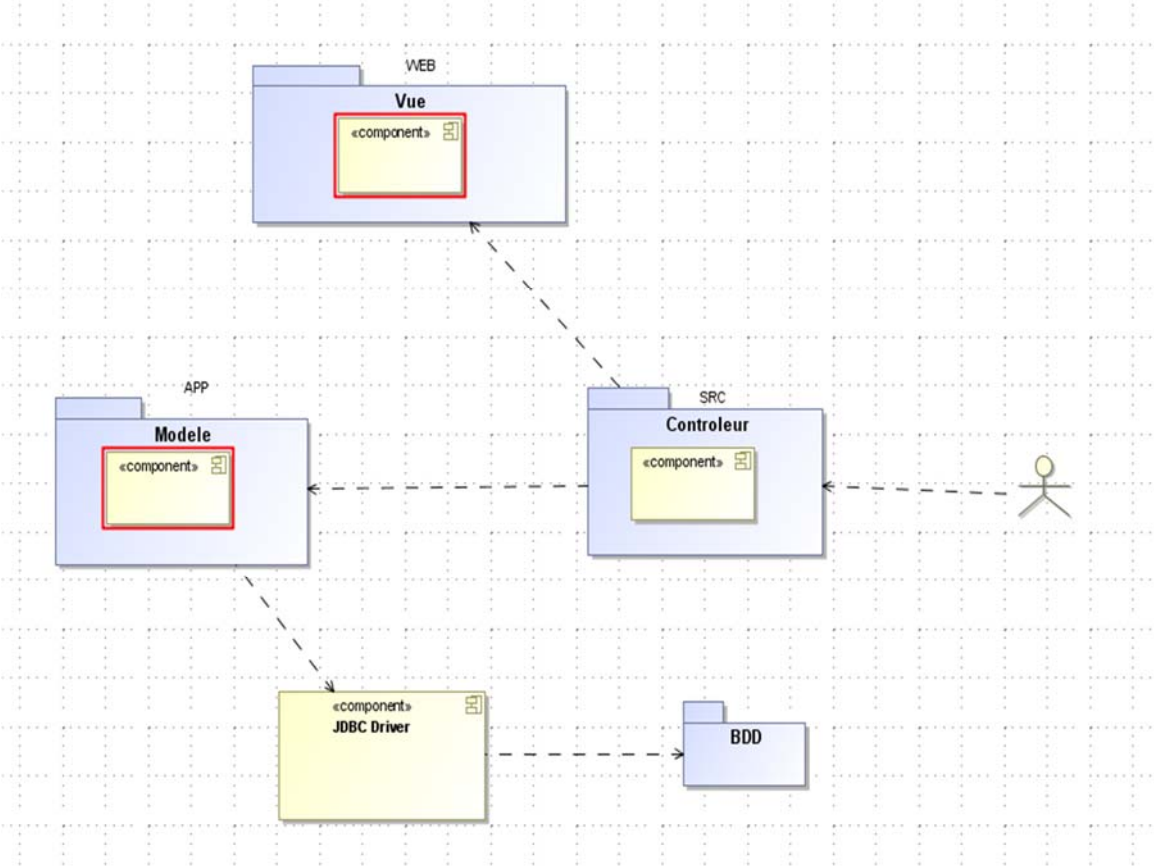


Figure 13 : exemple diagramme de composants

III. Conception et Architecture système

III-1 La base de données

Voilà le schéma général de la Base de données de ManageChart :

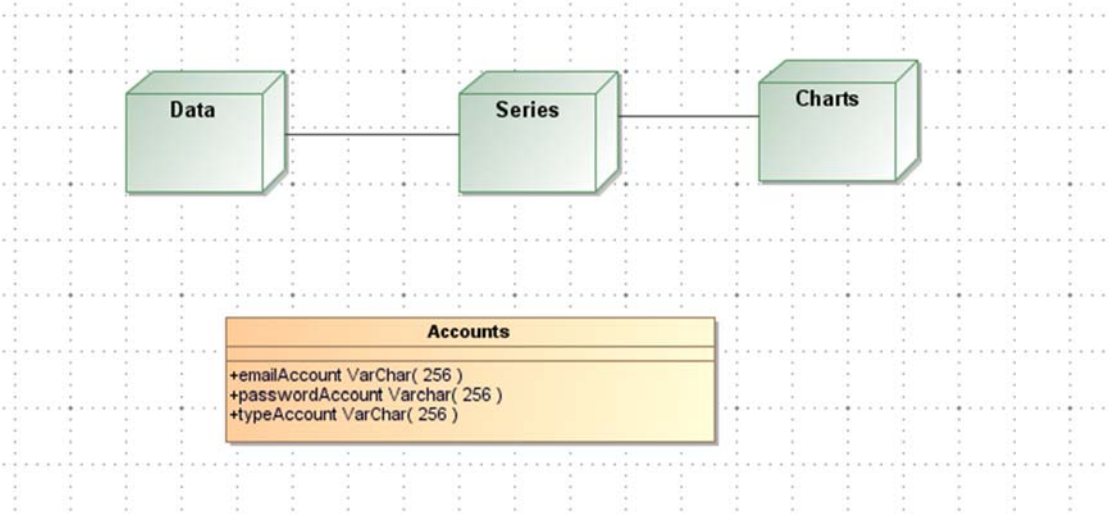


Figure 14 : schéma général de la Base de données de ManageChart

Un graphique est relié aux données au travers des séries, et les comptes ne sont pas reliés au reste des tables.

III-2 L'application ManageChart

Symfony 2 est idéal pour une architecture REST et une conception Modèle-Vue-Contrôleur (MVC), ce qui convient parfaitement aux besoins de Géomer.
Ce framework intègre par défaut un contrôleur frontal qui capte les requêtes, et les renvoie au bon contrôleur. L'architecture repose sur le principe des Bundles qui sont des modules du site. Chaque bundle est structuré selon le design pattern MVC et vit indépendamment des autres. Cependant on peut parfaitement modifier la structure d'un bundle pour utiliser une conception différente de celle du MVC.

Je ne présenterai pas l'ensemble de l'architecture de ManageChart, puisque c'est assez répétitif et classique (un contrôleur frontal et un ensemble de bundle en MVC). Toutefois le design pattern MVC ne convient pas pour tous les bundles de ManageChart, je vais donc détailler ceux-ci davantage. Ces bundles implémentent des fonctionnalités utilisées par d'autres bundles et ne sont pas accessibles Directement depuis une URL. Il en existe deux :

- ✓EncryptBundle : qui crypte et décrypte des chaînes de caractères avec La méthode du OU-exclusif. Cette méthode a été choisie pour la simplicité d'implémentation. Il est cependant envisagé de migrer sur la méthode AES256 avec un bundle existant. EncryptBundle est utilisé pour crypter les identifiants et mots de passe des comptes des bases de données d'où sont issues les données des graphiques. Ces identifiants et mots de passe sont stockés en base de données et ont donc besoin d'être cryptés. Ils aussi besoin d'être décryptés pour que l'application puisse se connecter aux bases de données en question. Ce bundle ne possède qu'un seul contrôleur qui crypte ou décrypte la chaîne de caractères passée en paramètre en fonction d'une chaîne connue de lui seul.

- ✓ BddBundle : ce bundle gère les connexions aux bases de données des graphiques, et les opérations de requête et de récupération des résultats. Php a pour chaque base de données ses propres fonctions et l'objectif de BddBundle est d'offrir une couche d'abstraction au type de base de données.
Bien entendu cela est déjà le cas avec l'extension PDO de PHP. Cependant l'avantage de créer notre propre bundle est d'avoir plus la main sur les fonctions utilisées et donc les performances.

Voici le schéma de la conception bddBundle :

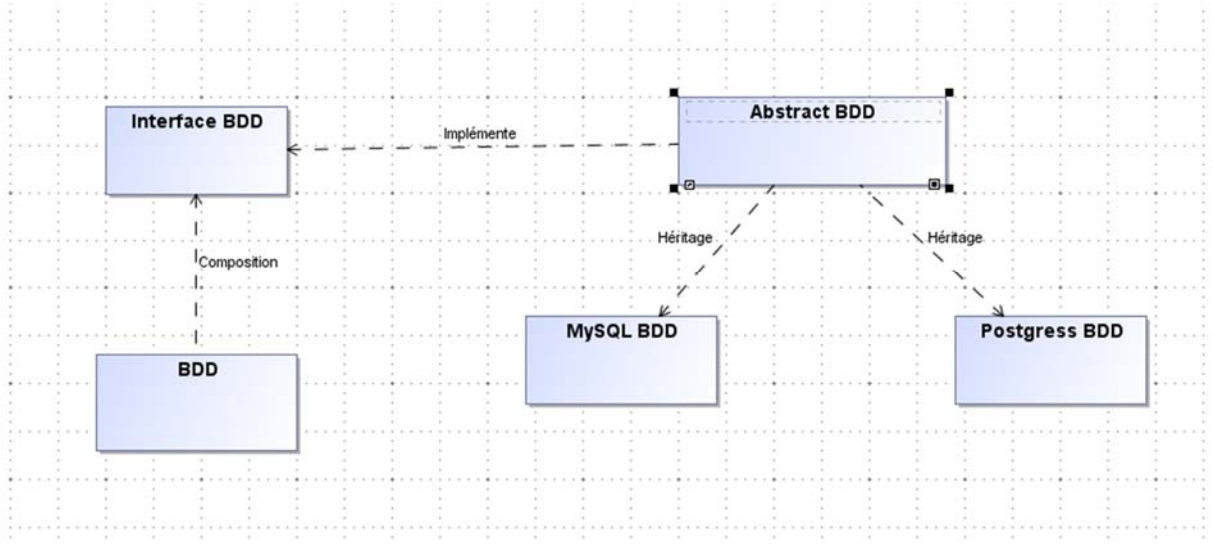


Figure 15 : modèle conception bddbundle Managechart

✓La structure d'un bundle :

- Controller : contient tous les contrôleurs
- DependencyInjection : contient des informations sur les bundles (chargement automatique de la configuration)
- Entity : contient les modèles
- Form : contient les éventuels formulaires
- Config : contient les fichiers de configuration du bundle
- Public : contient les fichiers publics du bundle : fichiers CSS et JavaScript, images, etc.
- Views : contient les vues du bundle, les templates twig

La commande pour générer un nouveau bundle est :

```
php app/console generate:bundle
```

✓La DataListBundle :

Ce bundle gere les requetes SQL effectuées sur la DataSources .Il est reservé aux administrateurs.

III-3 Outils et technologies utilisés

Voici la liste des outils que j'ai utilisé :

Modélisation	MagicDraw

Base de données	▪ PostegresSQL 9.3 ▪ MYSQL
Serveur	Apache 2.2
Langages de développements	▪ PHP5(voir detail ci-apres) ▪ Javascript(voir detail ci-apres) ▪ CSS3(voir detail ci-apres) ▪ HTML5(voir detail ci-apres)
Librairies	▪ JQuery(voir detail ci-apres) ▪ HighChart(voir detail ci-apres)
Framework	▪ Symfony2(voir detail ci- apres) ▪ Bootstrap(voir detail ci-apres)
IDE	Eclipse
Debogage	▪ Firebug ▪ Web debug toolbar de symfony
Tests	visuels

Mon stage s'est déroulé dans l'environnement de développement Eclipse.

La gestion de la base de données s'est effectuée avec PostgreSQl et MySQL.
Pour la modélisation, le logiciel MagicDraw fournit tous les outils nécessaires grâce à son large choix de diagrammes. Il est aisé à prendre en main et permet l'export sous différents formats.
PHP : est un langage de programmation qui se connecte à une base de donnée, et produire des pages web dynamiques via des requêtes sur un serveur http.
Ce sont PHP5 et JavaScript qui ont été retenus comme langages de développement, combinés bien sûr à HTML5 et CSS3.
JavaScript et jQuery : est un langage de programmation orienté objet, il permet d'interagir localement avec des pages web et produire des animations et des contrôles,…etc.
On utilise jQuery sur toute l'application. On peut aussi noter que Bootstarp à fichier javascript.
J'utilise les librairies Highchart, JQuery et le framework CSS Bootstrap pour sa grille adaptable à la différente taille d'écran, ainsi que pour ses nombreuses classes mettant rapidement en forme les éléments. Il s'agit d'un ensemble de composants structurés qui sert à créer les bases et organiser le code informatique pour faciliter le travail des programmeurs, que ce soit en terme de productivité ou simplification de la maintenance. Il existe des frameworks côte serveur (désignés backend en anglais) et d'autre côté client (désignés frontend en anglais).
Comme environnement de développement j'utilise Eclipse avec Symfony 2.
J'ai choisi ce framework parce qu'il est très efficace dans les développements PHP, respectant le design pattern MVC et augmentant nettement le gain de temps pour le développeur et les performances de l'application. Le projet est aussi très bien organisé ce qui en facilite la maintenabilité. J'emploie également Eclipse parce qu'il est conseillé avec Symfony 2, ainsi qu'en raison de mes compétences avec cet IDE.
L'arborescence de Symfony se présente comme ce ci sur l'application ManageChart, il y a 4 répertoires importantes :

- app/
- Src/
- Vendor/
- Web/

Vendor : contient toutes les librairies extérieures.
J'utilise Microsoft Office Word et Libre Office pour la bureautique.
Le dépôt est lié au site tucuxi (forge Redmine de l'IUEM) qui permet le suivi du projet avec un partage de fichier et un diagramme de Gantt en accord avec mon maître de stage Mathias Rouan.

IV. Réalisation

a-Faisabilité

J'ai travaillé de développer ManageChart en PHP avec le framework symfony2, c'est un langage et la nouvelle technologie pour moi.
Javascript est utilisé pour les graphiques avec les librairies Highchart.
Highchart est toutes suffisante pour les besoins de LETG-Brest Géomer. Elle permet d'afficher les données au fur et à mesure de la construction du graphique.
Elle possède des fonctionnalités par défaut qui réduisent le code.
Les graphiques doivent pouvoir être exporter au format PNG ; JPEG, PDF, SVG, PDF

Voici l'exemple de L'highChart

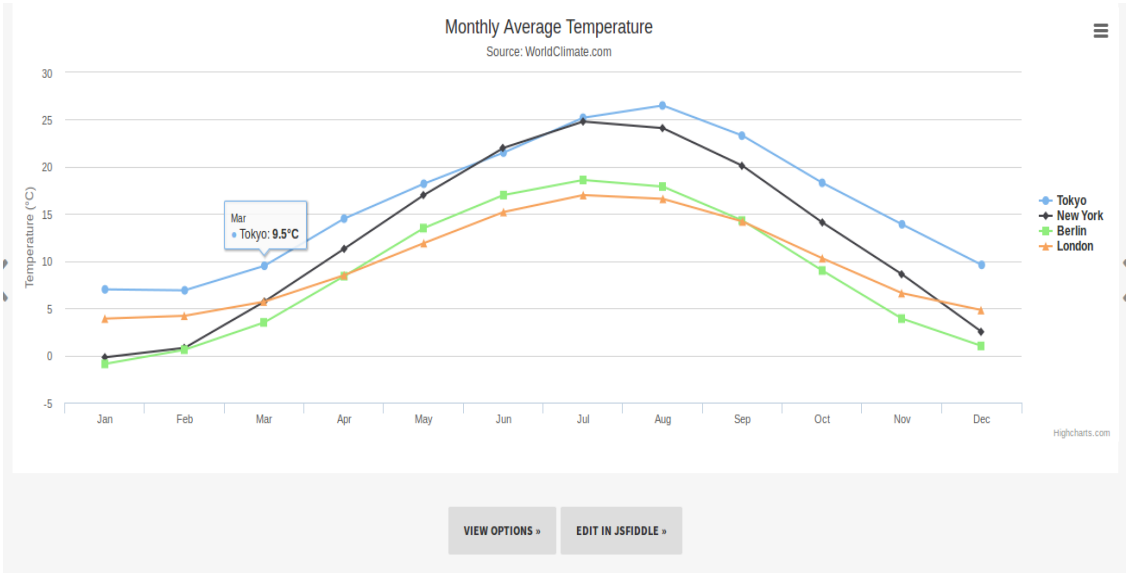


Figure 16 : exemple Highchart

IV-1 Planification du projet

Elle se présente sous la forme d'un diagramme de Gantt réalisé sur le site tucuxi

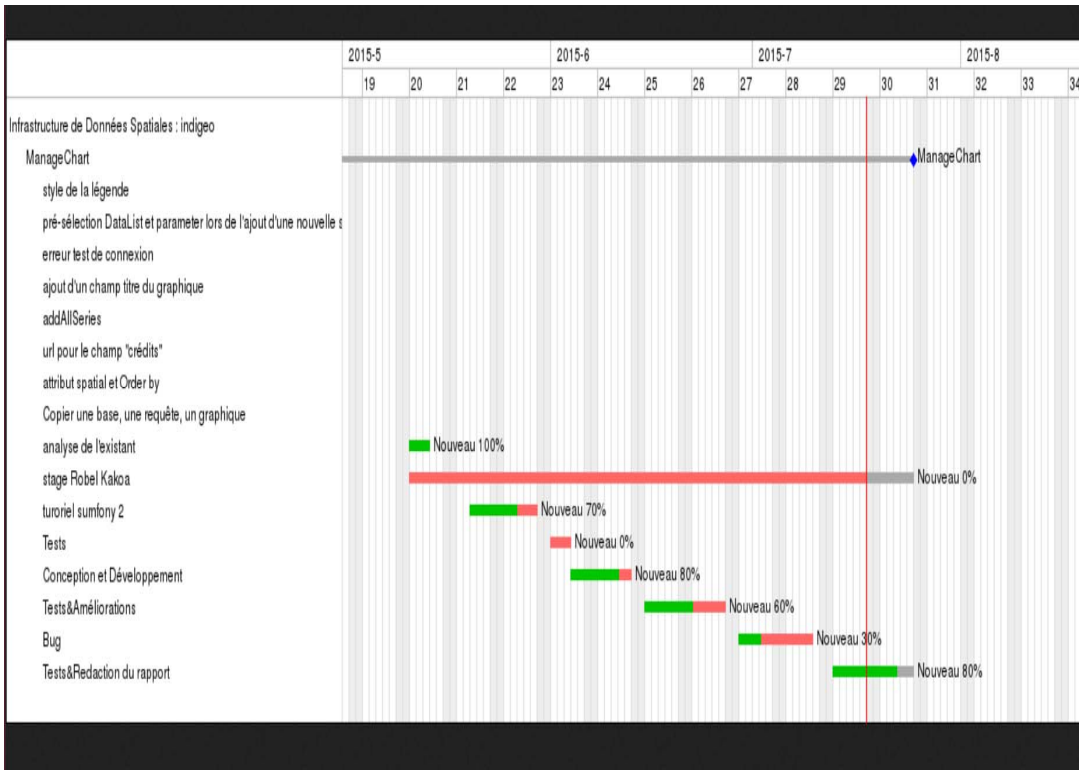


Figure 17 : diagramme de Gantt pour ManageChart

Parce que le Framework Symfony est une nouvelle technologie pour moi, j'ai commencé à l'apprendre à l'aide d'un tutoriel sur openclassroom, avant de commencer la réalisation des améliorations du projet ManageChart,

J'ai mis du temps à comprendre l'organisation de l'application surtout au niveau des composants structurés qui sert à créer les bases.

On a fait des tests visuels et quelques Conceptions (voir le détail sur l'exemple).

Voici un exemple de tache que j'ai réalisé:

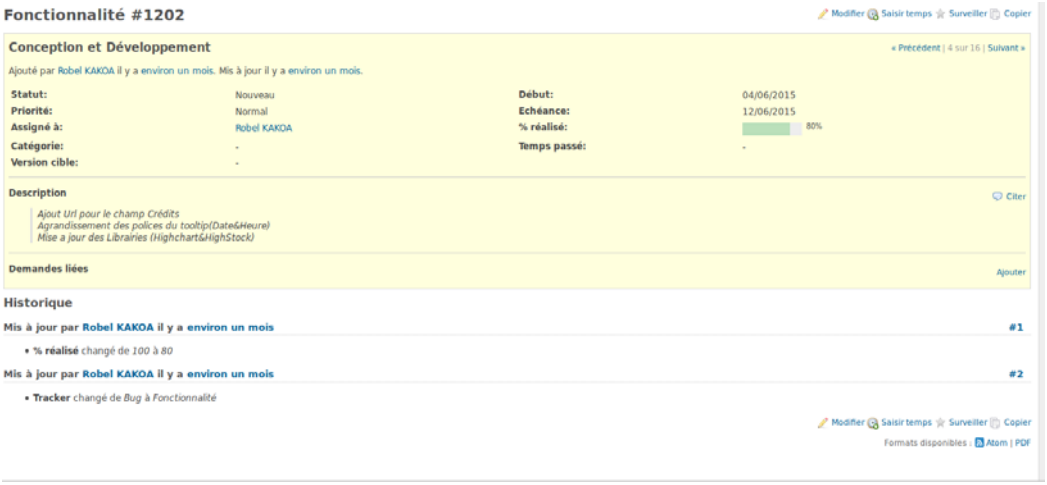


Figure 18 : exemple de tâche à réaliser

IV-2 Améliorations

Création des entités Url Crédits et Inversion des axes dans ManageChart
Voici l'explication des modifications des fichiers sources sur ManageChart :

1. Etapes d'explication d'ajout URL pour le champ 'Crédits' et Inversion les axes:

a. Cas Url Crédits :

Voici le répertoire des fichiers à modifier :

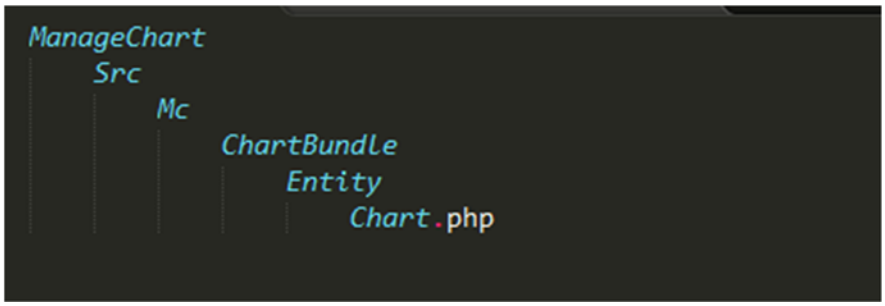


Figure 19 : Exemple ChartBundle entity urlcredits

- **La modification de l'entité urlcredits avec l'annotation :**

Pour modifier une entité, il suffit de lui créer un attribut et de lui attacher l'annotation correspondante.
Le type doctrine text permet de saisir le champ et le paramètre nullable égal true permet à la colonne de ne pas contenir des valeurs NULL.

Voici l'objet urlcredits avec commentaire

```php
<?php

namespace Mc\ChartBundle\Entity;

use Doctrine\ORM\Mapping as ORM;
use Symfony\Component\Validator\Constraints as Assert;


/**
 * Chart
 *
 * @ORM\Table()
 * @ORM\Entity(repositoryClass="Mc\ChartBundle\Entity\ChartRepository")
 */
class Chart
{
    // Creation l'objet urlcreditchart en base de données

    /**
     * @var string
     *
     *@ORM\column(name="urlcreditsChart", type="text", nullable=true)
     */
    private $urlcreditsChart;
    /**
     * Set urlcreditsChart
     *
     * @param string $urlcreditsChart
     * @return Chart
     */
    public function setUrlcreditsChart($urlcreditsChart)
    {
        $this->urlcreditsChart = $urlcreditsChart;

        return $this;
    }

    /**
     * Get urlcreditsChart
     *
     * @return string
     */
    public function getUrlcreditsChart()
    {
        return $this->urlcreditsChart;
    }

}
```

Figure 20: création entités urlcredits

Il faut exécuter la commande doctrine2 pour mettre à jour la base de données et la faire correspondre à l'état actuel des entités qu'on a créé en base de données.
La commande qui gère l'entité UrlCredits en fonction du mapping que Doctrine connait, et elle va générer ce qu'il manque comme la méthode constructeur :
Le getter et le setter.

Voilà la commande à exécuter :

```
##Mise a jour entity(sauvegarde)
php app/console doctrine:generate:entities McChartBundle:Chart
##Mise a jour BDD(affichage requêtes)
php app/console doctrine:schema:update --dump-sql
## Mise à jour BDD(execution requêtes)
php app/console doctrine:schema:update --force
```

Figure 21 : mise à jour entités et BBD

a-1 Création de formulaire « Url Crédits »:

Voici le répertoire des sources à modifier :

```
ManageChart
    Src
        Mc
            ChartBundle
                Form
                    ChartType.php
```

Figure 22 : sources formulaire Url Crédits

A partir du formbuilder, je génère le formulaire.
On va rajouter dans le constructeur du formulaire ‘ buildform ‘le type de formulaire afin qu’il existe en base de données.
J’ajoute le champ de l’entité urlcredits dans le formulaire de type text et l’argument de la méthode $builder ->add( ) qui correspond aux options du champ. Les options se présentent sous la forme d’un simple tableau. Pour le rendre le tableau facultatif, je l’ai défini l’option required à false, comme suit :
Ce schéma ci -dessous montre l’explication du champ de l’entité dans le formulaire :

```
Managechart
    Src
        ChartBundle
            form
                ChartType.php:

public function buildform(formBuilderInterface $builder,array $options)
{
    $builder
        ->add('urlcreditsChart',    'text',     array(
                                                    'label' =>'formchart.title',
                                                    'required   =>false'
                                                    ))
}
```

Figure 23 : formulaire champ Url Crédits

a-2 Mise à jour de la Translations (paramètre d'affichage des langues) :

Voici l'arborescence du répertoire source à rajouter la translation :

```
ManageChart
    Src
        Mc
            ChartBundle
                Ressources
                    translations:
                        messages.en.yml
                        messages.fr.yml
```

Figure 24 : bundle de translations

On va rajouter les deux paramètres ci-dessous dans la translation puisqu'il va traduire les messages sur les navigateurs.

Urlcredits : Credits Url c'est le message en anglais

```
Managechart/src/Mc/ChartBundle/Resssources/translations/messages.en.yml
formChart:
urlcredits: Credits Url
```

Figure 25 : translations en anglais

Urlcredits : Url Crédits c'est le message en français

```
Managechart/src/Mc/ChartBundle/Resssources/translations/messages.fr.yml
formChart:
urlcredits: Url Credits
```

Figure 26 : translations en français

Voici la commande pour mettre à jour les fichiers de traduction :

```
##Mettre a jour fichiers de traduction
php app/console translation:update --dump-messages en McChartBundle
php app/console translation:update --dump-messages fr McChartBundle
```

Figure 27: fichiers mise à jour traductions

J'ai choisi l'exemple EcoFlux pour montrer le lien HyperText depuis le champ crédit.

Résultat visuel ci-dessous prouve le lien HyperText :

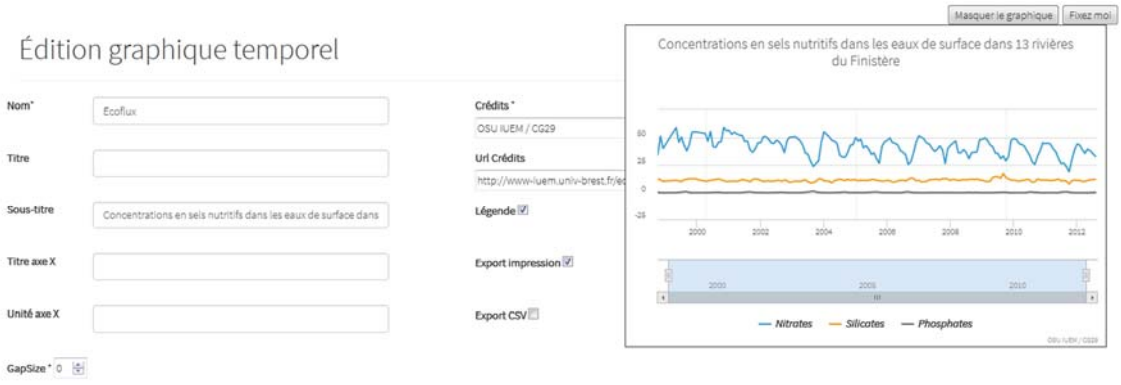


Figure 28 : exemple UrlCredits

b. Cas Inversion les axes :

IL permet d'intervertir les axes sur le graphe HighChart.

Voici le répertoire des fichiers à modifier :

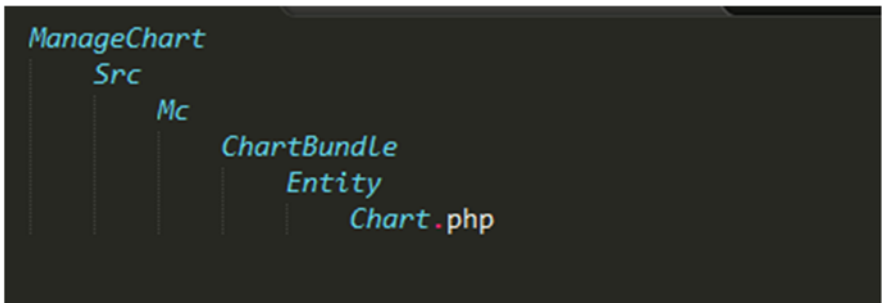
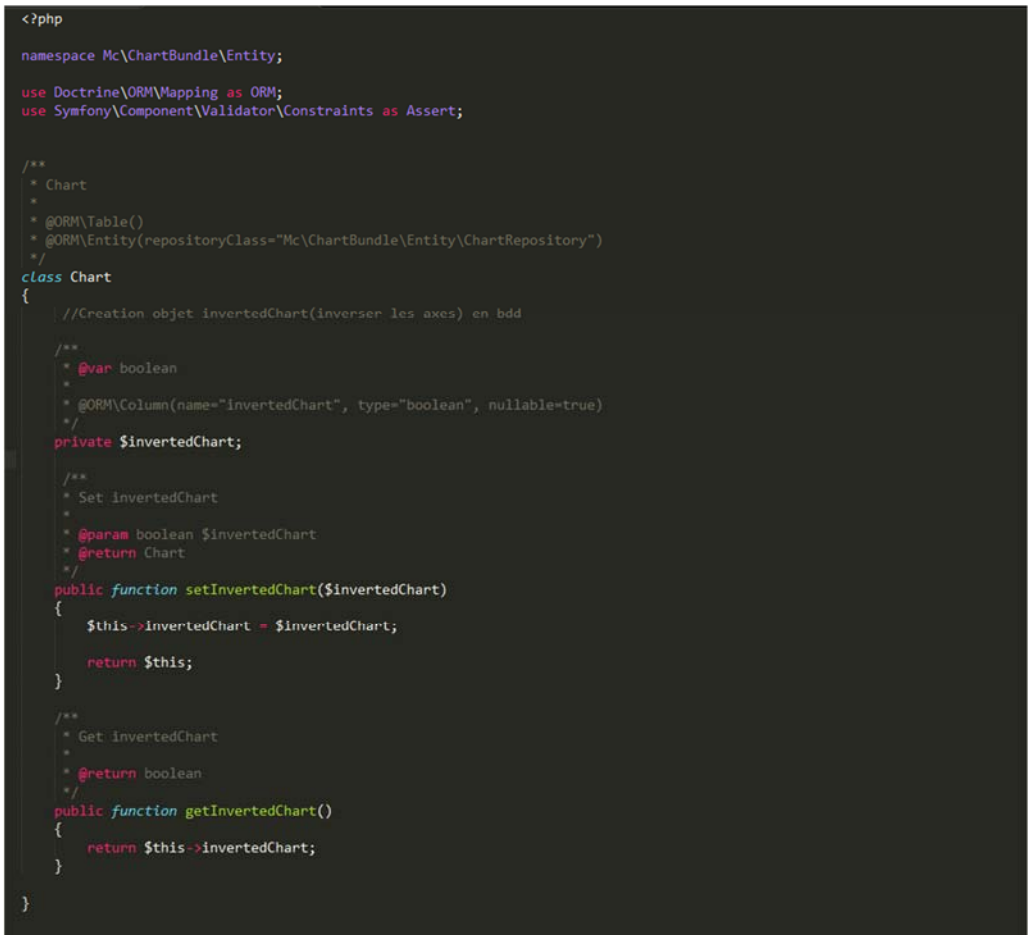


Figure 29 : Exemple ChartBundle entity inversion les axes

- **La modification de l'entité inverser les axes avec l'annotation :**

Je l'ai modifié l'entité 'inverser les axes 'et il suffit de lui créer un attribut et de lui attacher l'annotation correspondante. C'est le même fichier de Chart.php que 'urlcredits 'juste j'explique en dessous la différence entre les 2 fonctionnalités.

Voici l'objet urlcredits avec commentaire



```php
<?php

namespace Mc\ChartBundle\Entity;

use Doctrine\ORM\Mapping as ORM;
use Symfony\Component\Validator\Constraints as Assert;


/**
 * Chart
 *
 * @ORM\Table()
 * @ORM\Entity(repositoryClass="Mc\ChartBundle\Entity\ChartRepository")
 */
class Chart
{
    //Creation objet invertedChart(inverser les axes) en bdd

    /**
     * @var boolean
     *
     * @ORM\Column(name="invertedChart", type="boolean", nullable=true)
     */
    private $invertedChart;

    /**
     * Set invertedChart
     *
     * @param boolean $invertedChart
     * @return Chart
     */
    public function setInvertedChart($invertedChart)
    {
        $this->invertedChart = $invertedChart;

        return $this;
    }

    /**
     * Get invertedChart
     *
     * @return boolean
     */
    public function getInvertedChart()
    {
        return $this->invertedChart;
    }

}
```

Figure 30 : création entité inverser les axes

Il faut exécuter la commande doctrine2 pour mettre à jour la base de données et la faire correspondre à l'état actuel des entités qu'on a créé en base de données.
Voilà la commande à exécuter :

```
##Mise a jour entity(sauvegarde)
php app/console doctrine:generate:entities McChartBundle:Chart
##Mise a jour BDD(affichage requêtes)
php app/console doctrine:schema:update --dump-sql
## Mise à jour BDD(execution requêtes)
php app/console doctrine:schema:update --force
```

Figure 31 : mise à jour entités et BDD

b-1 Création de formulaire « invertedChart »

C'est le même contenu que celui d'avant le ChartType.
Sauf, il y a la différence par rapport le formulaire urlcredits et inverser les axes. Ce champ permet d'intervertir les axes sur le graphe HighChart. On va le définir le constructeur du formulaire ' buildform ' afin qu'il existe en base de données.
J'ajoute le champ de l'entité 'inverser les axes' dans le formulaire de type text et

l'argument de la méthode $entity égal $builder ->add( ) qui correspond aux options du champ. Il a la condition if qui précise si on a un graphique HighChart.

Ce schéma ci -dessous montre l'explication du champ de l'entité dans le formulaire

```
Managechart
    Src
        ChartBundle
            form
                ChartType.php:

public function buildform(formBuilderInterface $builder,array $options)
{
//Si on a u graphique highchart on peut soit choisir le type d'axe X
    $entity = $builder->getData();
    if ($entity->getTypeChart() == 'highchart')
        ->add('invertedChart',  'checkbox',     array(
                                                'label' =>'formchart.inverted',
                                                'required   =>false'
                                                ))
}
```

Figure 32 : formulaire champ inversion les axes

Je prends l'exemple graphique roza_foram montre le résultat Visuel:

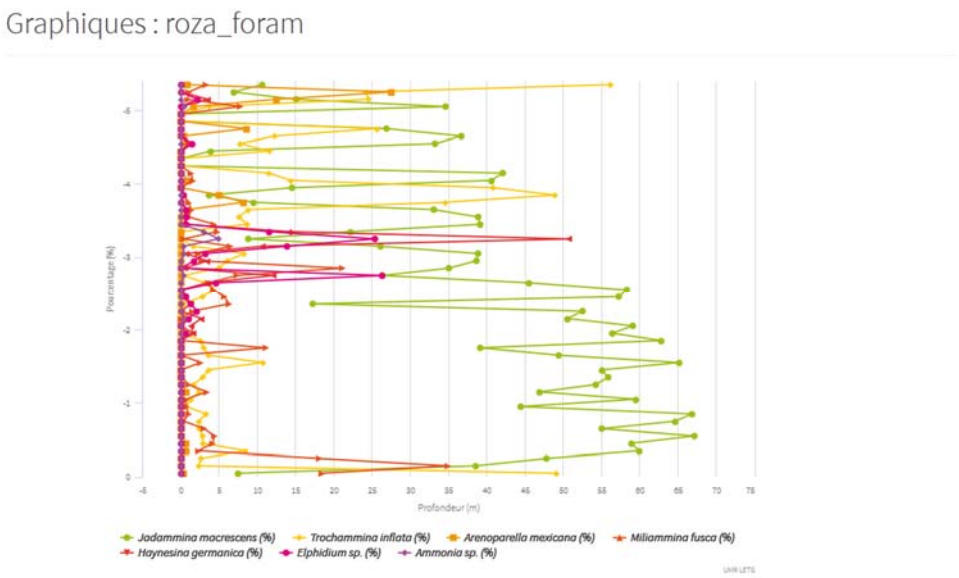


Figure 33 : exemple inversion les axes

2. Mise à jour des librairies :

J'ai mis à jour les librairies JavaScript Highchart 3.0.1 vers la nouvelle version 4.1.5 et Highstock 2.0.3 vers 2.1.5.

Je les ai vérifié en premier temps leur compatibilité avec l'application, il y a aucun bug au niveau de l'affichage et sur le console de test donc la nouvelle version des librairies a fonctionné bien.

Au niveau de fonctionnalité sur la nouvelle version Highstock permet de créer multiple axe Y.

3. Agrandissement des polices du tooltip (Date&Heure)

Ce code permet d'agrandir l'affichage date&heure sur l'infobulle du graphe :

Je présente d'abord le paramètre tooltip d'avant que je fasse la modification. Ce fichier ressemble rien sur l'option, d'où le graphe a une mauvaise lisibilité

```
tooltip: {
                    },
```

Figure 34 : Paramètre tooltip avant la modification

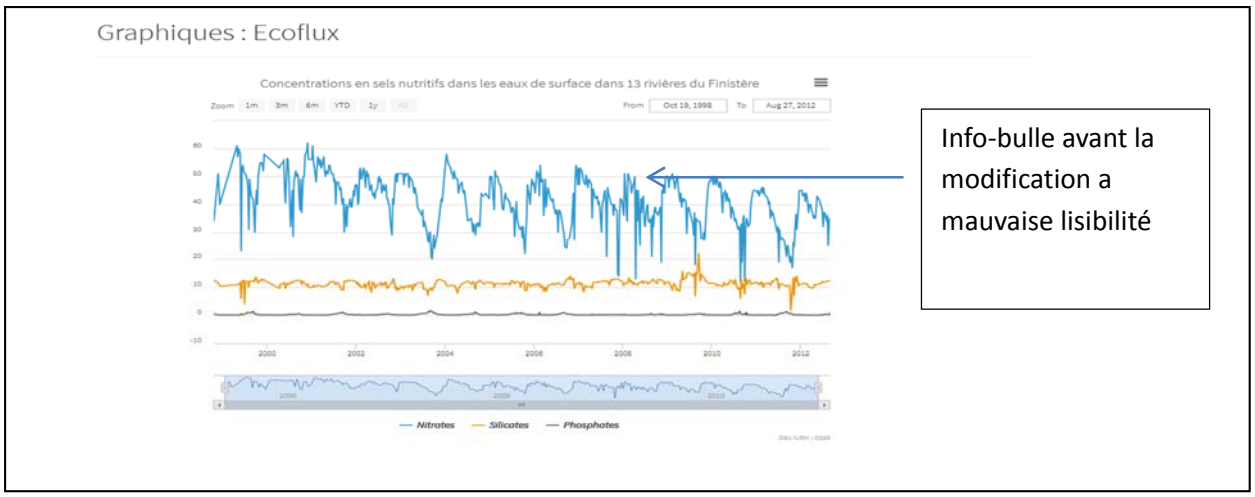


Figure 35 : info-bulle avant la modification

La seconde partie montre la meilleure lisibilité après la modification quand on affiche le graphe.

```
ManageChart
    Src
        Mc
            ChartBundle
                Ressources
                    Chart
                        showIframe.html.twig

                                tooltip: {
                                    style:{
                                        color:'#303030',
                                        fontSize:'12px',
                                        fontWeight: 'normal',
                                        fontFamily: 'Serif',
                                        fontSizeAdjust: '0.70',
                                        padding: '8px'
                                    },
                                }
```

Figure 36: paramètre tooltip après la modification

Surtout on va ajuster la taille du fontSizeAdjust dans l'option style sur le paramètre tooltip

IV-3 Tests

Les tests ont été réalisés à l'aide de :

- Firebug
- Web debug toolbar de Symfony

J'ai préféré plutôt utilisé debug Symfony pour la gestion des erreurs à la base de données, aux contrôleurs, aux routes ….etc

V. Conclusion

Pour clore ce rapport, le projet ManageChart nous a appris à développer à l'aide de la technologie PHP avec Symfony et de Doctrine pour le lien avec la base de données.
Ce stage a été une expérience très enrichissante autant sur le plan personnel que technique, notamment quant à l'appréhension d'un projet vaste comme ManageChart.
On constate que la formation nous donne effectivement les outils nécessaires à notre adaptation dans l'univers professionnel.
Les améliorations sont possibles surtout au niveau du Code.

V-1 Apport du projet

Ce projet s'est déroulé en parfaite adéquation avec mes désirs professionnels
La réalisation de ces taches nous ont permis de développer certaines compétences nouvelles tel que le domaine technique.
Ces 12 semaines du projet nous ont permis d'étudier les nouvelles technologies variées tel que le langage PHP et du Framework Symfony.

V-2 Difficultés rencontrés

J' ai rencontré quelques difficultés concernant le langage PHP, malgré cela j'ai réussi à m'approprier les outils.

J'ai eu des difficultés à la création des multiples Axes Y sur le graphe Highstock, il y existe sur la nouvelle version de la librairie.

Figure 37: création multiple axe Y sur le graphe temporel

La fonctionnalité permet d'ajouter axes Y autant qu'on veut avec le Titre, Type et les Séries Imbriqué.

Donc j'ai rajouté le code JavaScript FormYAxis par exemple :

```
{% javascripts
            '@McChartBundle/Resources/public/js/formYAxis.js'
        '@McChartBundle/Resources/public/js/formSerie.js'
        '@McChartBundle/Resources/public/js/highstock.js'
    %}
        <script type="text/javascript" src="{{ asset_url }}"></script>
  {% endjavascripts %}
```

J'ai difficulté de supprimer Axe Y cree automatiquement par highStock pour ajouter ceux formulaire :

```
{#      // supprime le premier axe cree automatiquement par highstock pour ajouter ceux du formulaire#}

        chart.yAxis[0].remove();
```

L'objectif est de le mettre même la fonctionnalité du code HighChart car c'est du JavaScript en plus je l'ai commenté quelques lignes des containers:

Exemple :

// supprime les deux premieres series cree automatiquement par highstock pour ajouter celles du formulaire :

```
chart.series[0].remove();
chart.series[0].remove();
```

Il faut commenter les lignes précédentes.

Beaucoup de recherche été effectué à fin d'avoir un travail professionnel et propre.

V-3 Travail restant

- Finir la partie multiple axe Y sur Highstock, déjà bien entamé
- Rajouter l'épaisseur du trait (lineWidth) lors de la création graphique sur la partie Série au niveau du style de line (Nuage de Point).

V-4 Evolutions

Plusieurs évolutions pour ManageChart ont été envisagées.

Voici les évolutions possibles :

- A la création des nouveaux graphes se sera mieux de mettre un seul bouton sur le formulaire ce qu'on appelle nouveau graphique
  Il pourra se diriger vers une nouvelle fenêtre afin de choisir entre les deux graphes avec un menu déroulante.
- Lors de création ou Edition d'une connexion a une base de données, ce mieux de pouvoir spécifier le Nom et le type Description de la base de données soit en PostgreSQl ou MySQL.

VI. Annexes

1) Annexe : Conception de la BDD de ManageChart

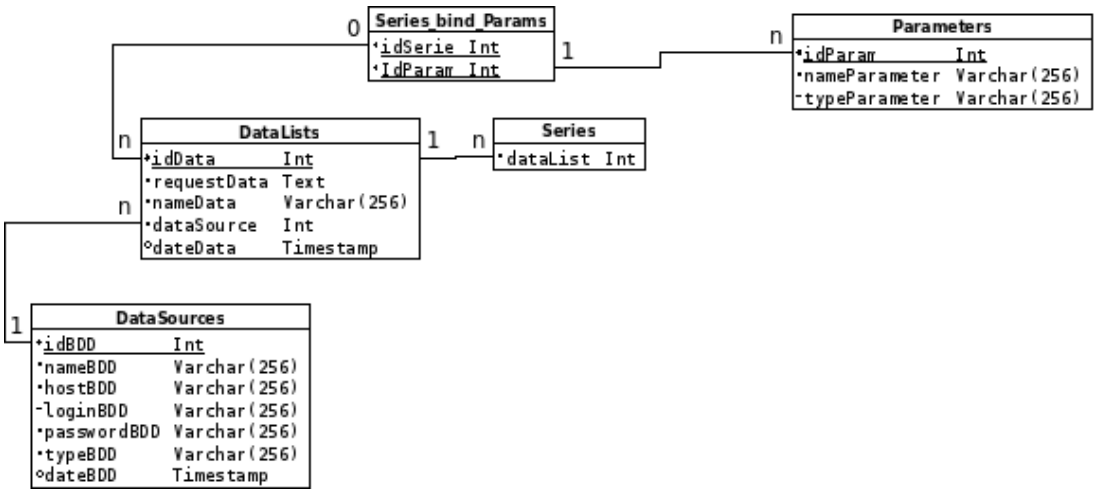


Figure 38: schéma détaillé des données

Les DataSources représentent les bases de données elle-même, et contiennent les informations nécessaires pour créer la connexion. Les DataLists représentent les requêtes SQL effectuées sur ces bases de données. Elles sont reliées par une relation un-plusieurs aux DataSources, pour enregistrer l'identifiant de la dataSource sur laquelle va s'effectuer la requête. En elles-mêmes, elles ne contiennent que le nom de requête et la requête proprement dite. Une table Series_bind_Params fait une association entre les paramètres pour sélectionner un objet dans la table et la requête. Il faut renseigner le nom et le type du paramètre. Enfin les DataLists sont reliées aux Séries par une relation un-plusieurs.

VII. Bibliographie

1-J.Steffe – FICHE PHP – 09 pages.
2-F.Y.Villemin – Le langage PHP & MYSQL – année 2004-2005 – 21 pages
3- J. Steffe - F.Prim – EXERCICES PHP – 76 pages.
4-GuillaumeRossolini (Tutoriels web/ SEO / PHP) – Cours de PHP 5 – 12 mai 2008 – 164 pages
5-document intitulé « PHP – créer un moteur de recherche» issu de comment ça marche informatique – modifier le mardi 14 octobre 2008 à 17 :40 :30 par Jeff, disponible sur www.comment ça marche.net
6-document intitulé « construire une base de données simple avec PHP et MYSQL» issu de Valhalla- publié le 25 décembre 2005 – disponible sur « www.valhalla.fr»
7-Rapport Maxim Collin Master1 TILL à UBO

http://fr.wikipedia.org
http://letg.univ-nantes.fr
http://inspire.ign.fr/
http://menir.univ-brest.fr
http://www.indigeo.fr/
http://www-iuem.univ-brest.fr/zabri/fr/documents/journee-za-18-janvier-2013/diaporama_theme3_peuziat
http://www.highcharts.com/
http://d3js.org/
http://www.symfony.com
http://fr.openclassrooms.com/informatique/cours/developpez-votre-site-web-avec-le-framework-symfony2
http://wapps/semaphores/tableau_de_bord/index.php?page=rapport&sem=1&type=global&annee=2011