Maxime Collin
Master 1 TIIL

Rapport de stage

Stage LETG Brest GEOMER

du 31/03/2014 au 29/08/2014

Maitres de stage :
Ingrid Peuziat
Mathias Rouan
Iwan Le Berre

Tuteur de stage :
Mounir Lallali

Référence du document	Rapport_SLBG_MC_2-5
Version du document	2.5
Date de version	20/06/14
Rédacteurs	Maxime Collin
Relecteurs	Iwan LeBerre Mathias Rouan Mounir Lallali

Historique des Modifications

Version	Commentaire	Date de modification	Auteur
1.0	Création du document	06/05/14	Maxime Collin
1.1	1ère partie : *ManageChart*	07/05/14	Maxime Collin
1.2	Ajout Réalisation	11/06/14	Maxime Collin
1.3	Relecture et corrections + ajouts Figures et diagrammes séquences et Gantt	16/06/14	Maxime Collin
2.0	Deuxième relecture	17/06/14	Maxime Collin
2.1	Relecture	17/06/14	Iwan LeBerre
2.2	Relecture et ajout partie réalisation	18/06/14	Maxime Collin
2.2.1	Relecture	18/06/14	Mathias Rouan
2.3	Ajout conclusion	19/06/14	Maxime Collin
2.4	Relecture et corrections	19/06/14	Mounir Lallali & Maxime Collin
2.5	Relecture	20/06/14	Maxime Collin

Je tiens tout d'abord à présenter mes remerciements à mes maîtres de stage, à savoir Ingrid Peuziat, Iwan LeBerre et Mathias Rouan, pour leur disponibilité et leur suivi régulier de mon travail. Je les remercie également pour leur soutien ainsi que pour les réponses rapides à mes questions et le travail de relecture de ce rapport.

Je remercie ensuite mon tuteur pédagogique, Mounir Lallali, pour ses réponses efficaces, son intérêt pour mon travail et ses propositions d'aide ainsi que de relecture de ce rapport.

Merci infiniment à l'administrateur système et réseaux de LETG-Brest Géomer, Christophe Martin, pour son aide précieuse sur des points techniques et la mise en place d'un environnement de travail adéquat. Merci aussi pour m'avoir aidé à déboguer, pendant parfois de long moments.

Un grand merci à toute l'équipe du LETG-Brest Géomer pour la bonne ambiance, leur écoute de mon travail et leurs idées pertinentes.

Et une sincère reconnaissance à mes proches pour leur avis sur ce rapport.

Merci encore à vous tous.

Table des matières

Index des Figures

I) Introduction

J'effectue mon stage de Master 1 TIIL (Technologies de l'Information et Ingénierie du Logiciel) à LETG-Brest Géomer qui est le laboratoire de géographie de l'IUEM (Institut Universitaire Européen de la Mer) à l'UBO (Université de Bretagne Occidentale).

Le sujet en est le développement d'une application de visualisation de données statistiques dans l'IDG *Indigéo*. Je suis encadré par Ingrid Peuziat, Iwan Le Berre et Mathias Rouan avec Mounir Lallali comme tuteur pédagogique du département d'informatique de l'UBO.

Ce sujet a retenu mon intérêt car je souhaite faire le Master 2 SIAM (Systèmes Informatiques et Applications Marines) et il correspond parfaitement à mon projet professionnel.

Il se déroule du 31 mars 2014 au 29 août 2014, soit pour une durée totale de 23 semaines. Ma soutenance ayant lieu le 25 juin 2014, je mettrais ce rapport à jour fin août pour y intégrer le travail effectué pendant l'été.

I.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.

I.2) Sujet du stage

Il s'agit de spécifier, concevoir, développer et tester deux applications web distinctes. Une première pour permettre la création de graphiques à partir de données provenant de bases de données distantes et leur visualisation dans une application web nommée *Indigéo*, et une deuxième plus spécifique au projet CARTAHU (CARTographie des Activités HUmaines en mer côtière), pour permettre la création de graphiques mais également la génération de rapport au format pdf et leur édition. Ce projet s'inscrit dans les deux thématiques : « fréquentation et usages » et « gestion intégrée des zones côtières ».

La première application qu'on appellera *ManageChart* s'inscrit dans les évolutions de l'Infrastructure de Données Spatiales (IDS) *Indigéo*. Cette IDS est destinée aux scientifiques désirant faire de la recherche et de l'observation sur l'environnement 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 géospatialisées. *Indigéo* respecte la directive INSPIRE de la Direction générale de l'environnement de la Commission européenne. *ManageChart* permettra la création de graphiques paramétrables à partir de sources de données différentes, avec une gestion des permissions pour les comptes utilisateurs. Ces graphiques seront visualisables dans une application externe telle que *Indigéo* qui pourra l'afficher dans une *iframe* avec son URL.

Figure 1 : le visualiseur d'Indigéo zoomé sur le technopôle et la plage de Sainte Anne, avec la station Somlit

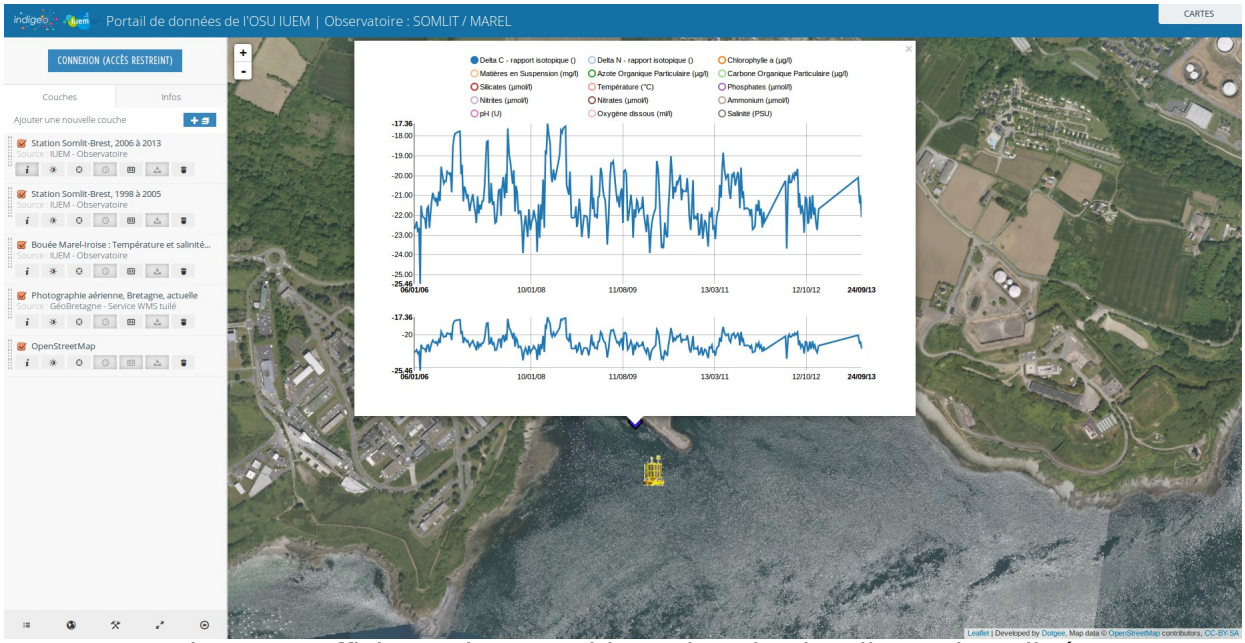


Figure 2 : affichage d'un graphique dans le visualiseur de Indigéo

La deuxième application qu'on appellera *CartahuTools* vise à fournir des outils pour aider les scientifiques à visualiser leurs données, et à générer des rapports à partir des données des sémaphores de l'Iroise dans le cadre du projet CARTAHU. Ce projet cherche à inventorier et évaluer les méthodologies permettant l'analyse spatiale et dynamique des activités en mer côtière. La partie consacrée aux sémaphores s'intéresse plus particulièrement aux données produites par la FOSIT (formation opérationnelle de surveillance et d'information territoriale), dont la mission est la surveillance des activités maritimes le long des côtes françaises et l'organisation du sauvetage en mer. La FOSIT administre notamment le réseau de sémaphores.

CartahuTools doit donc avoir les mêmes fonctionnalités que *ManageChart*, réduites à la base de données de CARTAHU-SEMAPHORES, mais en permettant en plus la génération de rapports.

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, deux étudiants (Amir Khnissi et Redouane Oulla) du Master 2 SIAM de l'UBO ont mené un projet exploratoire sur le parent direct de CartahuTools. 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 (*Highcharts*) 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 dit, cette application n'est pas sécurisée et mis en évidence la nécessité de développer de nouvelles fonctionnalités, à la base de ce travail de stage. Cette application s'inscrit dans le projet Cartahu, et *CartahuTools* peut être considérée comme son descendant.

Une partie de mon travail concerne l'intégration des données produites par la FOSIT. Cela mérite d'expliciter davantage le contexte dans lequel sont produites ces données. Les veilleurs consignent différentes informations lorsqu'un navire entre dans leur zone de surveillance, telles que l'heure, l'immatriculation du navire, sa trajectoire... Ces données sont stockées dans un tableur et sont saisies à la main, ce qui amène à des fautes de frappe ou des façons différentes d'indiquer par exemple qu'un champ n'a pas pu être renseigné. De plus, les champs renseignés sont en évolution. On peut en effet noter l'absence de certains champs et l'apparition de nouveaux dans les données produites en 2012 et celles en 2013. Pour utiliser une base de données il est donc nécessaire d'appliquer un pré-traitement qui est actuellement développé par Annalisa Minelli (post-doctorante) et par mon maître de stage Mathias Rouan.

II) 1ère partie : *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*.

Il faut donc prévoir d'une part la gestion des accès aux données, et d'autre part la génération des graphiques eux-mêmes. À cela il faut ajouter la gestion des utilisateurs et de leurs droits de lecture/écriture des graphiques.

II.1) Cahier des charges & Étude

ManageChart doit 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. 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.

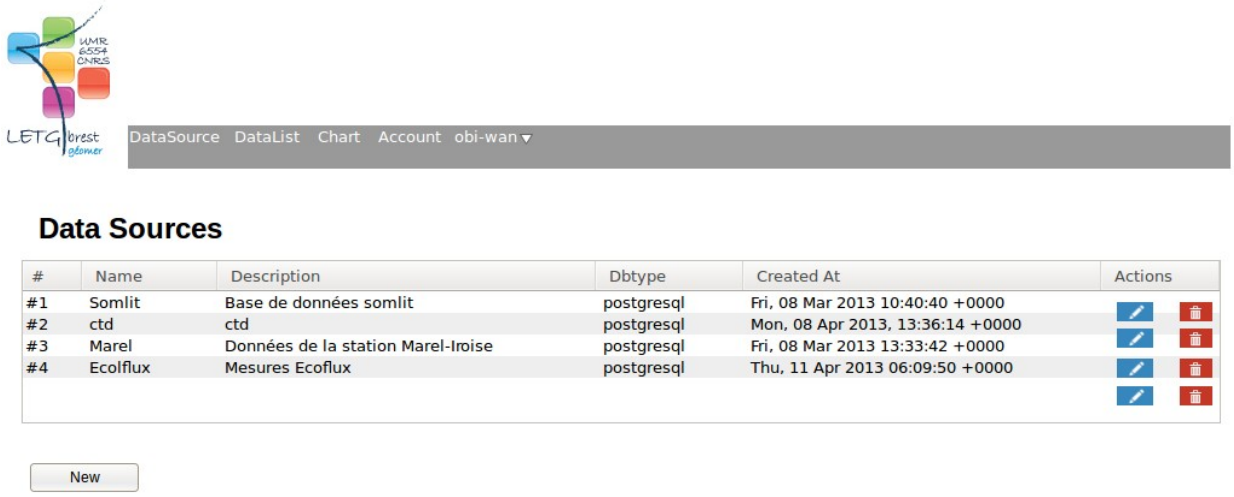


Figure 3 : Maquette gestion des connexions aux bases de données

ManageChart doit pouvoir aussi gérer la récupération des données. Elle doit donc permettre l'ajout, la modification ou la suppression de requêtes SQL sur une base de données précédemment ajoutée, ainsi que leur visualisation. Une requête SQL est caractérisée par un nom, une description, la requête elle-même et l'identifiant de la base de données sur laquelle s'effectuera la requête. Il faut aussi ajouter la possibilité de sélectionner des objet précis dans la table au travers de paramètres passés dans l'URL du graphique. Ces paramètres devront être clairement identifiés avec un nom et un type de donnée, de façon à ce que l'application les reconnaisse et effectue le traitement nécessaire.

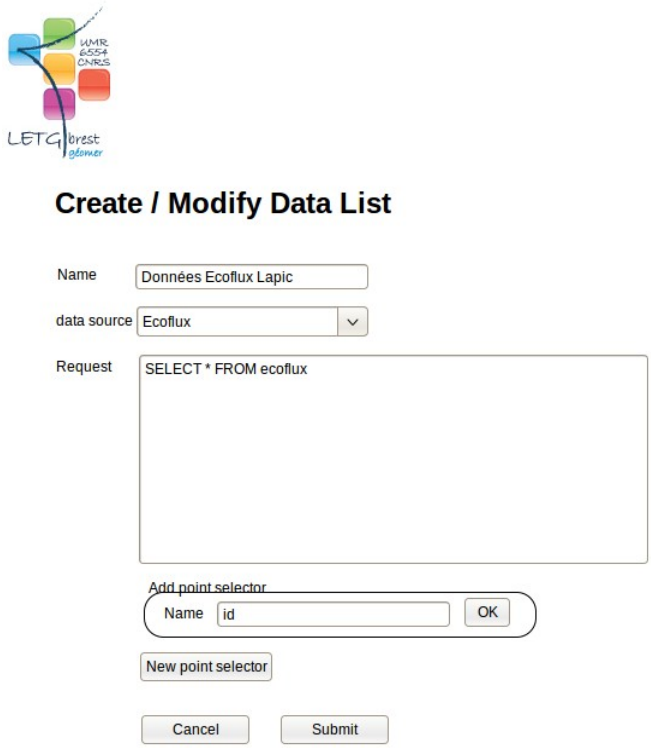


Figure 4 : Maquette création/modification d'une requête SQL

ManageChart doit pouvoir ajouter, modifier, supprimer et visualiser des graphiques. Ces graphiques doivent respecter 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 et les crédits.

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.

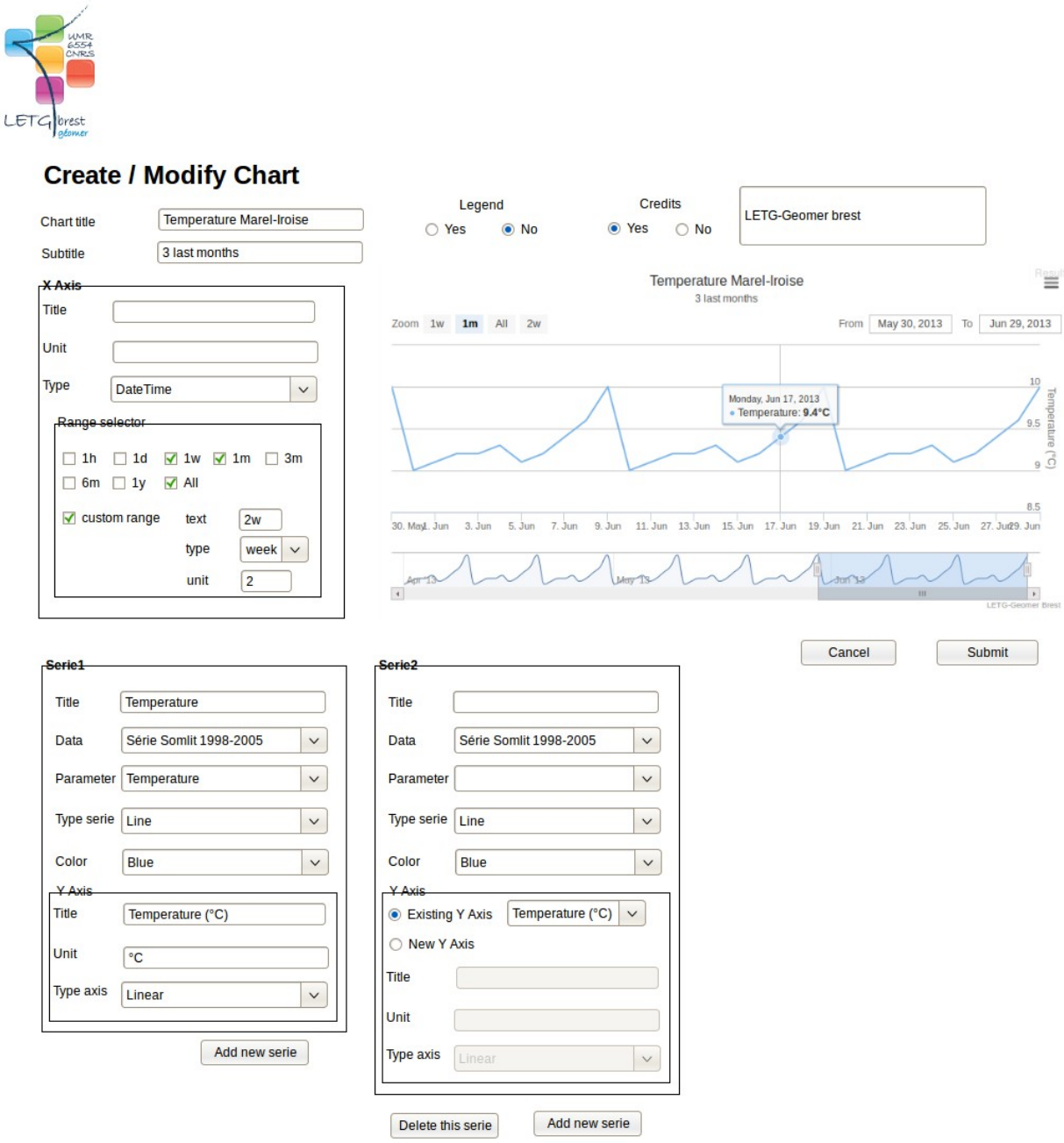


Figure 5 : Maquette création/modification d'un graphique

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 que le sien.

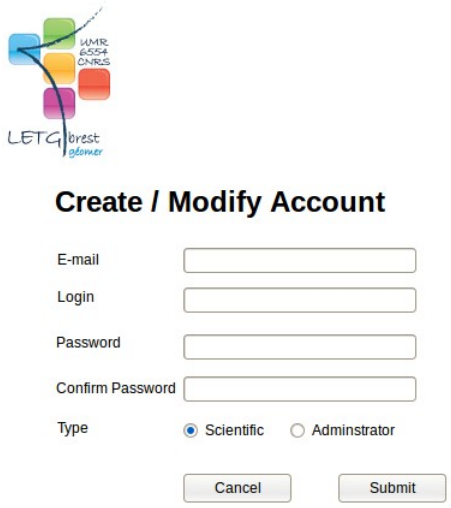


Figure 6 : Maquette création/modification d'un compte

De grands jeux de données seront utilisés, notamment pour les séries temporelles et il est nécessaire que *ManageChart* puisse les afficher rapidement. Les graphiques doivent pouvoir être exporter au format JPG, PNG, SVG et PDF et les données au format CSV.

II.1.1) Faisabilité

J'ai étudié les fonctionnalités, les technologies et les librairies. L'application développée par DotGee qui intègre l'ensemble de ces fonctionnalités à l'exception du paramétrage des graphiques, a été réalisée en *Ruby*. L'étude de ce langage m'a montré qu'il s'agissait d'un langage puissant mais complexe à prendre en main et ayant de faibles performances. Pour la maintenabilité de l'application et ses évolutions ainsi que les performances, j'ai donc proposé de développer *ManageChart* en *PHP* avec le *framework Symfony 2*. C'est un langage nouveau pour moi et avant de faire mon choix, j'ai vérifié que je pouvais le prendre en main sans trop de difficultés.

JavaScript sera aussi utilisé pour les graphiques avec la librairie *Highchart*. Là encore je me suis intéressé aux différentes façons de procéder. *JavaScript* s'est tout de suite imposé pour des raisons de performances. Il est en effet bien plus rapide d'envoyer les données au client et de le laisser les mettre en forme, que de faire sans cesse des requêtes au serveur. Ensuite plusieurs solutions existent : les « clés en mains » et les librairies. Les « clés en mains » sont insuffisamment paramétrables, et trop rigides. Pour les librairies il y a deux librairies qui se sont dégagées du lot : *D3 (Data Driven Document)* et *Highchart*.

D3 est utilisée par l'application développée par DotGee. Elle est très puissante et offre des possibilités quasi infinies de paramétrage. Cependant, elle charge entièrement les données avant d'afficher le graphique, ce qui provoque des temps de chargements longs pour des grands jeux de données. De plus, le concept mis en œuvre par *D3* est complexe à prendre en main, et le grand choix de paramétrage augmente nettement le temps de prise en main de l'API. Enfin, le manque de tutoriel ou d'exemples n'aide en rien l'apprentissage de cette librairie.

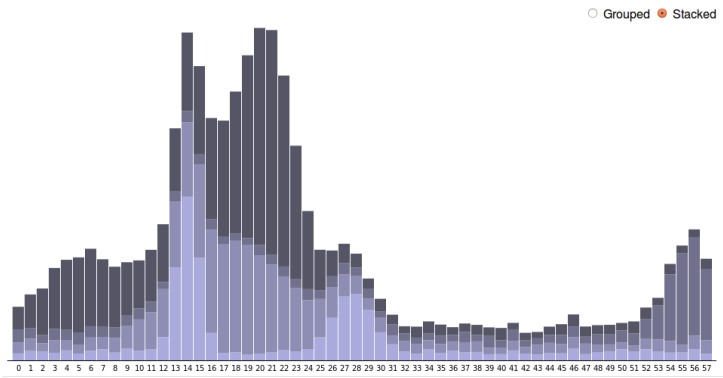


Figure 7 : Exemple de graphique avec D3

Highchart est quant à elle moins paramétrable que *D3*, mais toutefois suffisante pour les besoins de LETG-Brest Géomer. Elle permet d'afficher les données au fur et à mesure de la construction du graphique, répondant au besoin de vitesse pour des grands jeux de données et possède même des fonctionnalités qui permettent d'accélérer encore plus leur affichage. Les nombreuses valeurs par défaut réduisent le code au strict minimum, et les exemples et tutoriels sur leur site abondent. L'API est très lisible et très simple à prendre en main. C'est une librairie stable développée par la société Highsoft AS, avec une licence *creative commons* pour les projets non-commerciaux. Enfin, elle offre plus de compatibilité que *D3* du point de vue des navigateurs.

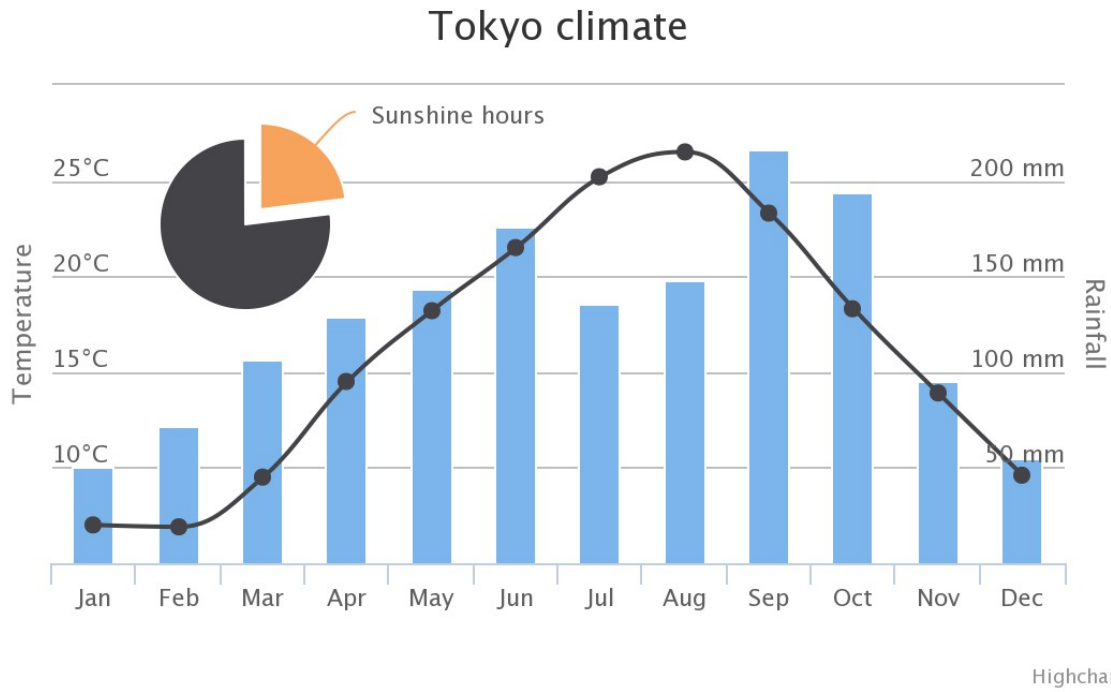


Figure 8 : Exemple de graphique avec HighChart

Enfin, il faut parler des évolutions de cette librairie. *Highchart* a des versions stables et des outils qui se développent, dont notamment un outil « création de graphiques en ligne ». Cet outil ressemble énormément à *ManageChart,* mais ne convient pas exactement aux besoins de LETG-Brest Géomer. En effet, il est encore en version bêta et l'import de données ne se fait que via des fichiers CSV (et vu les volumes de données à grapher cela n'est pas envisageable). De plus, il est « trop » possible de personnaliser le graphique et le LETG-Brest Géomer veut placer son template sur ses graphiques. Cependant, je pourrai me servir de cet outil pour me donner des idées, notamment au niveau de l'ergonomie.

II.1.2) Exigences

Les exigences sont désignées par la lettre E suivi de la première lettre désignant l'application (I pour *Indigéo* et M pour *ManageChart*) et complétées d'un numéro d'ordre.

II.1.2.1) La consultation de graphiques par tout utilisateur sous Indigéo

[E-I-1-1] L'utilisateur consulte un graphique associé à une couche d'informations sous *Indigéo* en affichant les informations relatives à cette carte.

[E-I-1-2] Un graphique s'affiche grâce à une *iframe* qui apparaît sous forme d'infobulle sur la carte.

[E-I-1-3] L'utilisateur a la possibilité de visualiser un graphique et de sélectionner dans la liste proposée les séries de données à afficher. Il peut également exporter ces données au format CSV ou le graphique lui-même au format PNG, JPEG, PDF ou SVG.

II.1.2.2) Les possibilités offertes à un administrateur dans ManageChart

[E-M-2-1] L'administrateur accède à l'interface de *ManageChart* par un portail de connexion.

[E-M-2-2] L'administrateur est déconnecté au bout d'un *timeout* d'inactivité.

[E-M-2-3] L'interface de *ManageChart* a un volet de gestion de sources de base de données, un volet de gestion de séries de données, un volet de gestion de graphiques et un volet de gestion de comptes.

[E-M-2-4] Le volet de gestion de sources de base de données n'est ouvert qu'aux administrateurs. Il permet d'afficher, d'ajouter, de modifier et de supprimer des liaisons à des bases de données existantes.

[E-M-2-5] Le volet de gestion de séries de données n'est ouvert qu'aux administrateurs. Il permet d'afficher, d'ajouter, de modifier et de supprimer des requêtes SQL vers des bases de données présentes dans le volet de gestion de sources de base de données.

[E-M-2-6] Les requêtes SQL du volet de gestion de séries de données sont uniquement des *SELECT* afin de créer des sources de données.

[E-M-2-7] Une requête SQL doit pouvoir sélectionner dynamiquement le jeu de données d'un objet spatial de la série de données.

[E-M-2-8] Une série de données a un ou plusieurs champs.

[E-M-2-9] Le volet de gestion de graphiques est ouvert aux administrateurs et aux utilisateurs. Il permet d'afficher, d'ajouter, de modifier et de supprimer des graphiques, ainsi que de sélectionner l'URL de l'*iframe* associée au graphique. Seul un administrateur peut supprimer un graphique.

[E-M-2-10] Lors de la création ou de la modification d'un graphique l'administrateur sélectionne la/les séries de données, dans celles disponibles du volet de gestion de séries de données.

[E-M-2-11] Il est possible de paramétrer le graphique de façon à choisir :

- le titre du graphique
- son sous-titre
- le type de graphique pour chaque série
- les couleurs de chaque série
- leur axe en y (un existant ou un nouveau)
- les types d'axes
- leur unité
- leur nom
- les sélecteurs d'échelle (uniquement pour l'axe x si c'est un axe temporel)
- les labels
- la légende et les crédits

[E-M-2-12] *ManageChart* doit détecter si la/les requête(s) choisie(s) sélectionne(ent) dynamiquement le jeu de données d'un objet spatial de la série de données. Auquel cas il doit proposer un champ texte pour sélectionner un point le temps de la création du graphique, afin de visualiser le rendu. Par la suite quand ce graphique sera appelé depuis une *iframe*, l'objet désiré sera passé en paramètre dans l'URL pour afficher le bon jeu de données.

[E-M-2-13] Le graphique doit pouvoir être visualisé dans l'interface de création et de modification, et rafraîchi à chaque modification de paramètre.

[E-M-2-14] L'URL de l'*iframe* du graphique doit être générée à la création du graphique. S'il s'agit d'un graphique qui a une/des séries nécessitant un/des paramètre dans l'URL, une valeur par défaut est placée dans l'URL pour chaque paramètre.

[E-M-2-15] Si une *iframe* qui a des paramètres dans l'URL est appelée sans que toutes les valeurs par défaut ne soient modifiées, un message d'erreur est affiché précisant que l'utilisateur n'a pas modifié ces valeurs par celles d'un objet spatial de la série de données.

[E-M-2-16] Le volet de gestion de comptes permet, si l'utilisateur est administrateur, de visualiser les comptes existants, d'en créer de nouveau, ou d'en supprimer.

[E-M-2-17] Tout administrateur peut créer de nouveaux comptes ou en supprimer.

[E-M-2-18] Un administrateur ne peut modifier que son compte. Il est possible de modifier l'adresse e-mail, le mot de passe et le type de compte.

II.1.2.3) Les possibilités offertes à un scientifique dans *ManageChart*

[E-M-3-1] Le scientifique accède à l'interface de *ManageChart* par un portail de connexion.

[E-M-3-2] Le scientifique est déconnecté au bout d'un *timeout* d'inactivité.

[E-M-3-3] Le scientifique ne peut accéder qu'au volet de gestion de graphiques.

[E-M-3-4] Le volet de gestion de graphiques pour un scientifique ne permet pas de supprimer un graphique, mais tout autre fonctionnalité est mise à disposition.

[E-M-3-5] Le scientifique ne peut modifier que son compte via l'onglet de son compte désigné par son nom.

II.1.2.4) Les contraintes de la gestion de graphiques

[E-M-4-1] Les sources de base de données, les requêtes SQL de sélection de séries de données, les paramètres des graphiques et les comptes doivent être enregistrés en base de données.

[E-M-4-2] *ManageChart* doit être compatible avec les navigateurs web Firefox, Chrome, Safari et Internet Explorer si possible.

[E-M-4-3] Les graphiques créés doivent être exploitables dans une *iframe* de dimension maximum de 600px par 600px.

II.1.2.5) Les technologies contraintes

[E-M-5-1] Le serveur utilisé pour *ManageChart* est *Apache 2.2*

[E-M-5-2] La base de données utilisée est *PostgreSQL 9.3*

II.1.2.6) Évolutions de *ManageChart*

[E-M-6-1] L'administrateur a la possibilité de verrouiller un graphique, empêchant toute modification ou suppression. Il peut, éventuellement, être déverrouillé par un autre administrateur.

II.1.3) Diagrammes de séquence

Voici deux diagrammes de séquences : un sur un exemple simple et très récurent et l'autre sur un exemple plus complexe.

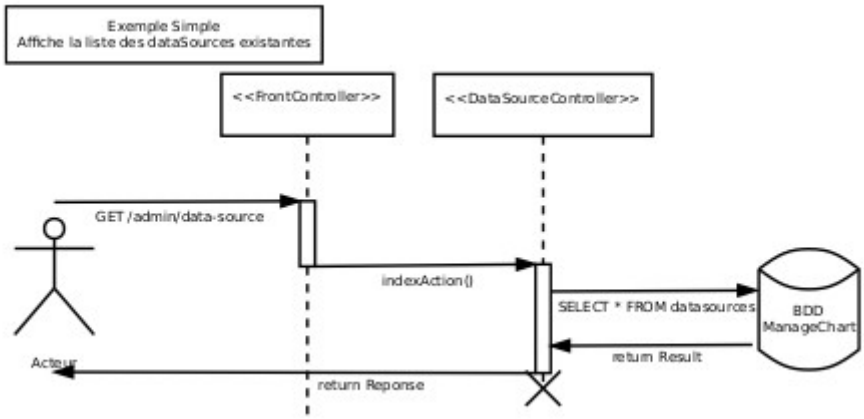


Figure 9 : Diagramme de séquence, exemple simple

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.

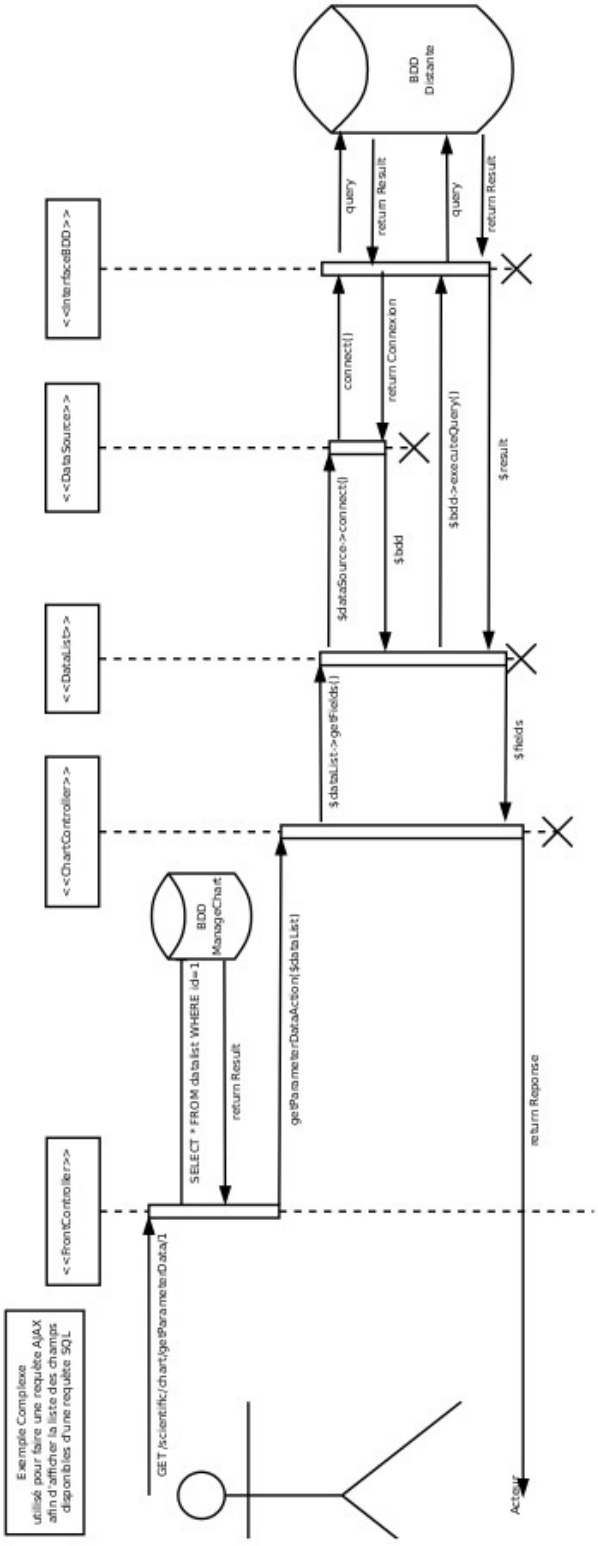


Figure 10 : Diagramme de séquence, exemple complexe

Dans cet exemple, le client effectue une requête *Ajax* pour afficher la liste des champs disponibles pour une requête SQL donnée, dont l'id est passé dans l'URL. Le *Front Controller* effectue le même travail, mais en récupérant automatiquement la requête SQL dans la base de données en se basant sur l'id.

Le contrôleur demande à la requête SQL de lui renvoyer sa liste de champs disponibles. Il faut donc se connecter à la base de données distante, dont les informations de connexions sont dans l'objet *$datasource*. On exécute ensuite la requête SQL associée et on renvoie la liste des champs.

II.2) Conception & Architecture système

II.2.1) La base de données

Voici le schéma général de la Base de données de *ManageChart* :

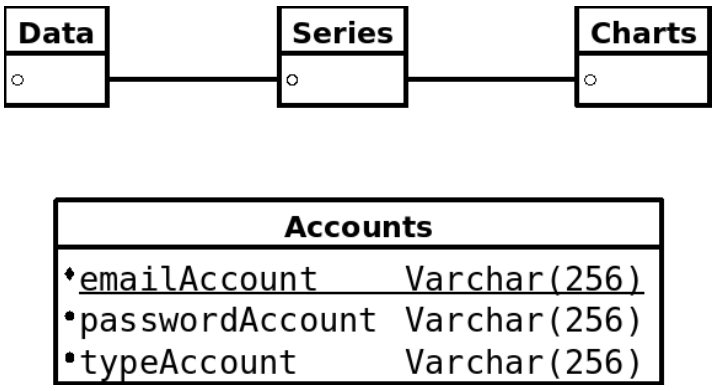


Figure 11 : Schéma général de la Base de Données de ManageChart

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

La conception détaillée de la base de données est fournie en « *annexe 1 : Conception de la BDD de ManageChart* ».

II.2.2) 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 utilisé 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 non 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. Sa conception est fournie en « *annexe 2 : Conception de BddBundle* ». 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.

II.3) Outils & Moyens utilisés

Ci-après la liste des outils que j'utilise :

Serveur	- *Apache 2.2*
Base de données	- *PostgresSQL 9.3*
Modélisation	- *Dia* (voir détail ci-après)
Maquettage	- *Pencil* (voir détail ci-après)
Langages de développement	- *PHP 5* (voir détail ci-après) - *JavaScript* (voir détail ci-après) - *HTML 5* (voir détail ci-après) - *CSS 3* (voir détail ci-après)
Librairies	- *JQuery* (voir détail ci-après) - *HighChart* (voir détail ci-après)
Framework	- *Symfony 2* (voir détail ci-après) - *Bootstrap* (voir détail ci-après)
IDE	- *Eclipse* (voir détail ci-après)
Gestionnaire de version	- *Git* (voir détail ci-après)
Automatisation des tâches	- *Phing*
Débogage	- *JavaScript Debug Toolkit* - *xDebug* - *Firebug* - *Web Debug Toolbar de Symfony*
Tests unitaires	- *JSUnit* - *PHPUnit*
Analyse statique de code	- *JSLint* - *CSSLint* - *PHPLint*
Test performances	- *JSLitmus* - *PageSpeed Insights*

Tests automatisés et montée en charge	- *Selenium* - *Tsung*
Métrique	- *PHPDepend*
Profilage	- *xDebug* - *Web Debug Toolbar*
Compatibilité	- *BrowsersShots*
Minification et mise en cache du code pour le serveur	- *eAccelerator* - *minJS*

Pour la modélisation, le logiciel *Dia* 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.

J'utilise *Pencil* pour la création de la maquette en raison de son ergonomie, de ses fonctionnalités intéressantes et des différents formats d'exports.

Ce sont *PHP5* et *JavaScript* qui ont été retenus comme langages de développement, combinés bien sûr à *HTML5* et *CSS3*. J'utilise les librairies *Highchart* et *JQuery* et le *framework CSS Bootstrap* pour sa grille adaptable aux différentes taille d'écran, ainsi que pour ses nombreuses classes mettant rapidement en forme les éléments.

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.

Du côté du gestionnaire de version, c'est *Git* qui a été choisi en accord avec mon maître de stage Mathias Rouan et l'administrateur système et réseaux Christophe Martin. Le dépôt est lié au site tucuxi qui permet le suivi du projet avec un partage de fichier et un diagramme de Gantt.

Je devrais me former sur un certain nombre de ces outils notamment au niveau des tests. Je pense à *JSLitmus*, *PageSpeed Insights*, *Selenium*, *Tsung*, *PHPDepend*, et *eAccelerator*. Les autres ne devraient pas me demander d'apprentissage puisque, soit je les connais, soit ils sont très intuitifs.

N'ayant pas besoin de fonctionnalités très avancées en bureautique, j'utilise *LibreOffice* puisqu'il a l'avantage d'être gratuit et installé par défaut sous *Ubuntu*.

II.4) Réalisation

Il s'agit d'une architecture *MVC* avec un *Front Controller* qui associe l'URL à une action d'un contrôleur. La couche modèle est entièrement gérée par l'*ORM Doctrine* de *Symfony* à tel point qu'on ne touche en aucune façon à la base de données. *Symfony* embarque une console permettant d'exécuter des commandes, telles que la création d'une classe *Entity* gérée par *Doctrine*. La commande nous demande le nom de la classe, son *namespace* et ses attributs. La classe est alors créée, on peut ensuite mettre à jour la base de données en ligne de commandes également. Il est très déconseillé d'effectuer une requête SQL dans l'application pour manipuler les éléments, mais d'utiliser plutôt *Doctrine*.

Voici un exemple d'enregistrement d'une nouvelle entrée :

```
130                 $em = $this->getDoctrine()->getManager();
131                 $em->persist($chart);
132                 $em->flush();
```

Figure 12 : exemple code Doctrine enregistrement entrée

À la ligne 130 on récupère *l'EntityManager* de *Doctrine* qui nous permettra de réaliser nos actions. À la ligne suivante on indique à *Doctrine* de persister l'élément *$chart*, donc de le modifier au prochain *flush*. Et enfin on l'enregistre en base de données avec *flush()*.

Et voici un exemple pour récupérer l'ensemble des entrée d'une table :

```
50          $repository = $this->getDoctrine()
51          ->getManager()
52          ->getRepository('McChartBundle:Chart');
53
54          $list_chart = $repository->findAll();
```

Figure 13 : exemple code Doctrine lecture table

À la ligne 50, on récupère le *repository* associé à l'entité *Chart*. Le *repository* nous permet de récupérer nos entrées avec une liste de fonctions définies par défaut, et la possibilité de créer nos propres fonctions en utilisant le *Doctrine Query Langage (DQL)*. Ce langage ressemble au SQL et permet justement de ne pas en faire, pour laisser le soin à *Doctrine* de gérer la base de données. Et à la ligne 54 on utilise la fonction *findAll()* du *repository* qui retourne toutes les entrées de la table *chart*.

La couche Vue est générée par le moteur de *template Twig*, qui fournit deux syntaxes puissantes : {{ }} qui dit : « affiche quelque chose » et {% %} qui dit « fais quelque chose ». *Twig* intègre aussi la notion d'inclusion et d'héritage et peut générer des URLs en utilisant le routage de *Symfony*. Pour *ManageChart* j'utilise le modèle triple héritage :

- Un *template* général qui définit la structure de *ManageChart*, il contient le *header*, le menu, le *footer*...
- un *template* fils par *bundle* qui définit une structure interne plus spécifique au *bundle* mais tout en gardant les parties communes à tous les *bundles*
- et le *template* de la page affichée qui contient le contenu central.

Voici un schéma pour bien comprendre le modèle :

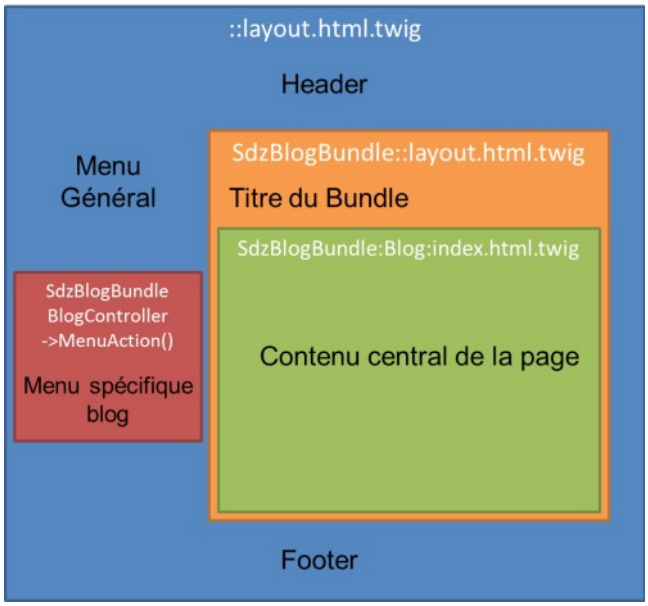


Figure 14 : modèle triple héritage

J'ai également commencé à traduire le site en français et en anglais. Il y a pour cela des fonctions *Symfony* qui permettent de traduire une chaîne de caractères en une autre, à partir d'un catalogue à notre charge. Il peut parfaitement utiliser des mots-clés séparés par des points comme chaîne à traduire, ce qui a l'avantage de réduire le code et de le rendre plus lisible. De plus, avec cette solution, dans le format de catalogue que j'ai choisi (*yaml*) les points peuvent être remplacés par des retours à ligne et indentation, ce qui permet de préfixer toute une série de mots-clés. On a aussi la possibilité de passer des variables avec la syntaxe %var%

Voici un exemple de catalogue :

```yaml
chart:
    table:
        titlecolumn:
            id: id
            nameData: Nom
            nameBDD: Base de données
            dateData: Date
            actions: Actions
        label:
            requestData: Requête SQL
        delete:
            msgconfirm: Voulez-vous vraiment supprimer %nameChart% ?
```

Figure 15 : exemple de code pour un catalogue de traduction

Et voici la fonction de traduction :

```
<tr>
    <th>{{ 'chart.table.titlecolumn.id'|trans() }}</th>
    <th>{{ 'chart.table.titlecolumn.actions'|trans() }}</th>
</tr>
```

```
{{ 'chart.table.delete.msgconfirm'|trans({'%nameChart%': chart.nameChart}) }}
```

Figure 16 : exemple de code pour la fonction de traduction dans Twig

II.4.1) Planification du projet

Voici une estimation du temps de travail pour chaque tâche, sous forme de diagramme de Gantt réalisé sur le site tucuxi.

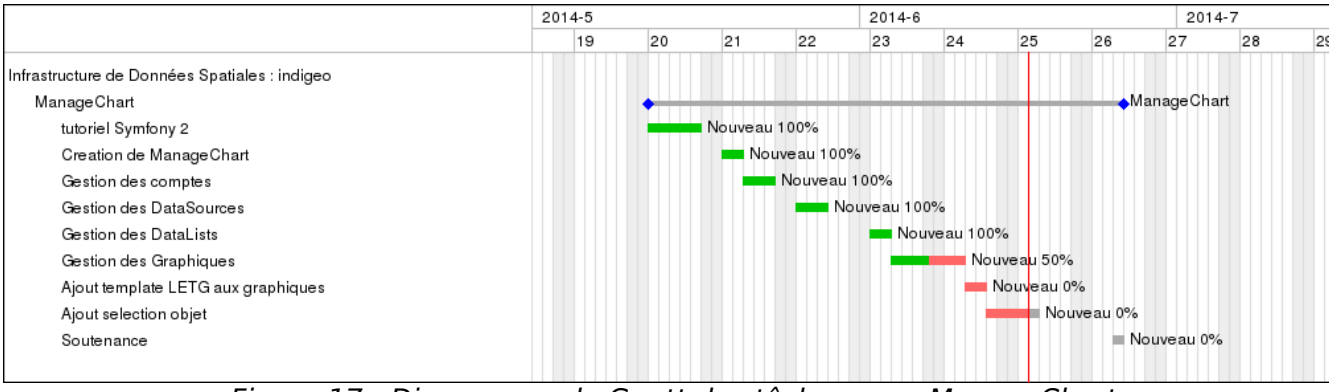


Figure 17 : Diagramme de Gantt des tâches pour ManageChart

J'ai un certain retard dû à une mauvaise estimation des tâches à réaliser. Il y a certains points auxquels je n'avais pas pensé, tel que l'encryptage des mots de passe ou l'utilisation d'un *framework CSS*.

J'ai dû commencer par suivre un tutoriel pour apprendre à me servir de *Symfony*, avant de commencer la réalisation de *ManageChart*. J'ai immédiatement intégré la gestion des utilisateurs, pour ne pas avoir à revenir sur l'ensemble du travail effectué avant afin de sécuriser l'application.

J'ai donc utilisé un *bundle* existant : *FOSUserBundle* ; réputé pour son efficacité et sa personnalisation. L'ensemble fonctionne bien, j'ai eu une difficulté sur un point technique : on voulait que ce soit un administrateur qui puisse créer des nouveaux comptes, or le *bundle* nous connecte automatiquement une fois le compte créé. J'ai donc cherché l'événement qui provoquait cette connexion automatique et je l'ai stoppé avant que la connexion ne se fasse.

J'ai ensuite créé le *bundle DataSource* qui est donc tout le volet qui gère les connexions aux bases de données distantes. À cet endroit j'ai eu du travail que je n'avais pas prévu :

- j'avais oublié le *framework CSS bootstrap*, qu'il a fallu que j'apprenne puis que j'intègre au travail déjà fait.
- je n'avais pas pensé à encrypter les mots de passe et login des bases de données distantes. J'ai donc créé un *bundle* pour cela mais seulement après avoir fait des recherches, et essayer d'installer et d'utiliser un *bundle* déjà existant, sans succès. Ce *Bundle* à l'avantage utiliser la méthode AES256. L'administrateur système et réseaux à réussi à l'installer sur le serveur, on envisage donc d'utiliser ce *bundle* une fois *ManageChart* sur le serveur.
- Enfin, je comptais utiliser l'extension *php PDO* pour gérer les connexions aux bases de données distantes, mais j'avais insuffisamment la main sur les fonctions. J'ai donc créé mon propre *bundle* en reprenant le principe de *PDO*.

Je suis ensuite passé au *bundle DataList*, ressemblant beaucoup au *bundle DataSource*. J'ai eu donc peu de modifications à faire pour le faire fonctionner intégralement. Il me restait alors le *bundle Chart* sur lequel j'ai bien trop sous-estimé le temps nécessaire pour le réaliser. La possibilité d'ajouter et de supprimer des axes Y pour un graphique donné à l'infini, et de faire de même pour les séries à l'intérieur d'un axe Y, a rajouté de la complexité. Cela ajouté à l'affichage des champs de la requête sélectionnée pour chaque série en *Ajax*, m'a beaucoup retardé. Actuellement je n'ai toujours pas fait le lien avec la librairie *highchart*.

II.4.2) Sécurité

Symfony offre la possibilité de gérer la sécurité à deux niveaux : l'authentification avec le pare-feu et l'autorisation avec les rôles. Il est conseillé de n'utiliser qu'un seul pare-feu avec le *bundle FOSUser*, ce que j'ai respecté, pour éviter tout dysfonctionnement. Cela dit j'aurais apprécié utiliser plusieurs pares-feu pour protéger tout un ensemble de l'application. Ensuite il y a la gestion des rôles, il est possible de sécuriser une URL ou un ensemble d'URL, une action d'un contrôleur ou encore des parties de l'affichage. On peut également affecter plusieurs rôles à un même rôle, ainsi un administrateur est un scientifique et un administrateur. Tout cela donne une gestion fine de la sécurité. Enfin J'utilise l'algorithme SHA512 pour encrypter les mots de passe des utilisateurs.

II.5) Tests

J'ai de nombreux outils pour effectuer les tests, dont une bonne partie qui ne pourront être utilisés qu'une fois *ManageChart* sur le serveur. Pour l'instant j'ai préféré bien comprendre comment s'utilise *Symfony*, par rapport aux routes, aux contrôleur, à la base de données... Avant de me lancer dans des tests bien formalisés. Ce qui ne m'empêche pas de tester mon travail en continu, en vérifiant les entrée en base de données, la sécurité, la gestion des erreurs...

II.6) Livrables

Tous les livrables que je fournirai respecteront une charte de rédaction détaillée dans le livrable « *LETG_CM_CRD_1-2* ».

Voici la liste des livrables à produire :

- Charte de Rédaction
- Cahier des Charges
- Outils et Moyens Utilisés
- Architecture Système
- Conception Base de Données
- Maquette IHM
- Plan de tests
- Manuel d'Installation
- Manuel Utilisateur
- Évolutions envisagées

Chaque livrable est désigné par *LETG_CM_??_X-X*, avec ?? qui correspond aux initiales du document (comme CDC pour Cahier Des Charges) et X-X à sa version.

III) 2ème partie : CartahuTools

CartahuTools a pour but de gérer les données fournies par la FOSIT dans le cadre du projet CARTAHU. Cette application doit permettre d'importer les données de la FOSIT fournies sous forme de tableurs en base de données. Elle doit ensuite être capable de créer des graphiques à partir de ces données et de générer des rapports constitués de graphiques et de zones de texte. Ces rapports partent d'un modèle commun et doivent être personnalisables afin de déplacer des éléments, en rajouter et en enlever.

Les données produites par la FOSIT peuvent présenter des fautes de frappes ou des différences sur la convention de saisie des données. Il faut donc vérifier les données fournies et les pré-traiter avant leur import en base de données. Il y a actuellement une chaîne de pré-traitement qui est effectué pour mettre la date dans le bon format, concaténer les fichiers fournis sur un an, et corriger les routes. L'immatriculation et le nom des navires ne sont pas traités. D'un point de vue base de données il peut être intéressant d'avoir une table qui possède une entrée par navire entrant dans la zone de surveillance d'un sémaphore, avec une clé étrangère sur l'immatriculation du navire et leur trajectoire ainsi que l'heure. Une deuxième table répertorie tous les navires, identifiés par leur immatriculation et contient des informations plus détaillées comme le type du navire, son utilisation... Cependant il faut trouver une solution pour l'immatriculation qui n'est pas utilisable en l'état comme identifiant.

Cette application va probablement utiliser *ManageChart* ou reprendre son code pour générer les graphiques. Je développe donc *ManageChart* en premier. Pour ce qui est des rapports, les deux étudiants Amir Khnissi et Redouane Oulla du Master 2 SIAM ont développé le modèle commun comme le montre la figure suivante. Il reste à le rendre personnalisable, sur quoi j'ai fait une étude de faisabilité en utilisant *JavaScript* et *Jquery*.

Je ne peux malheureusement pas détailler encore beaucoup plus cette partie, puisqu'elle a été laissée de côté pour se concentrer sur *ManageChart*.

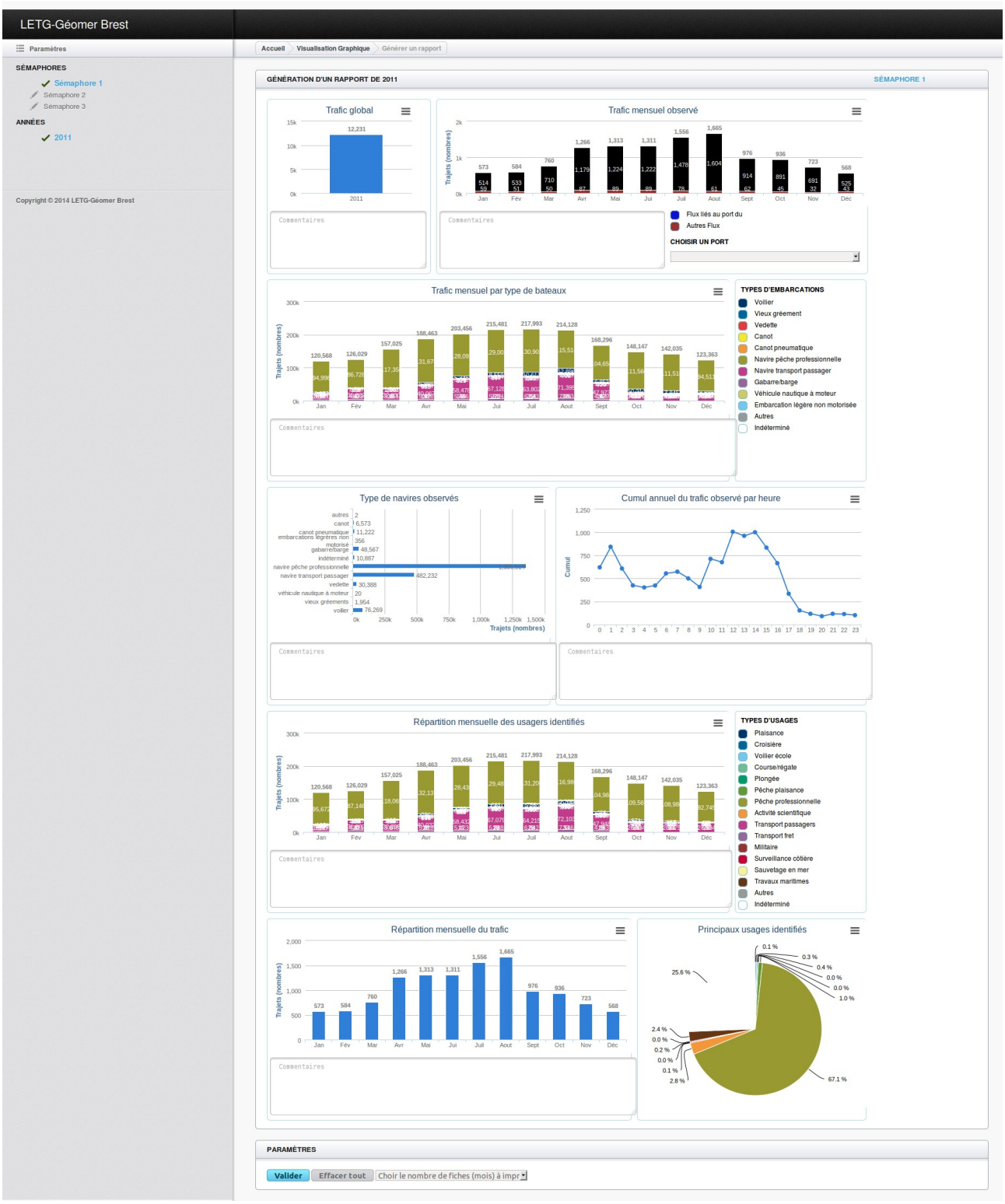


Figure 18 : copie d'écran de l'application semaphores de Amir Khnissi et Redouane Oulla – génération d'un rapport

IV) Conclusion

Ce stage m'aura permis de valider mes acquis et de développer des nouvelles compétences en programmation web. J'aurais aussi vu la conduite d'un projet de sa création à son terme et tout cela en relative autonomie. Enfin j'aurais moi-même réalisé chaque partie, de la spécification à la rédaction des livrables. Tout cela est très formateur, et je pense avoir beaucoup appris de ce stage.

Nombre de mes enseignements m'ont servi pendant ce travail, tels que :

- « services à objets répartis » pour les principes de développement web
- « Ingénierie Développement Logiciel » pour la conduite de projet
- « Sécurité et Administration Système » qui m'a beaucoup sensibilisé sur les failles de sécurité
- Les projets de synthèse ont aussi été très formateurs et comptent pour beaucoup dans mes acquis.

Bien entendu, tout cela repose sur mes bases de licence et sur mon stage précédent, qui m'a aidé à développer l'autonomie nécessaire pour ce stage.

Ce stage aura aussi été pour moi l'occasion de découvrir les *frameworks* et leur apport dans un projet. S'ils ne sont pas indispensables, ils me semblent en tout cas très conseillés pour la plupart des projets.

Il m'aura aussi permis de préciser mon projet professionnel, ce qui me semble inestimable.

IV.1) Travail restant

Je vais commencer par finir *ManageChart*, à savoir :

- finir la partie création de graphiques, déjà bien entamée
- rajouter les exigences concernant la sélection d'un objet spatial dans la série de données soit les exigences [E-M-2-7], [E-M-2-12], [E-M-2-14] et [E-M-2-15]
- et intégrer le *template* du LETG-Brest Géomer

Ensuite je pourrais effectuer une batterie de tests, et déployer l'application sur le serveur. Je terminerai avec quelques améliorations et la rédaction des derniers livrables. Je pourrais alors me concentrer sur *CaratahuTools*.

IV.2) Évolutions

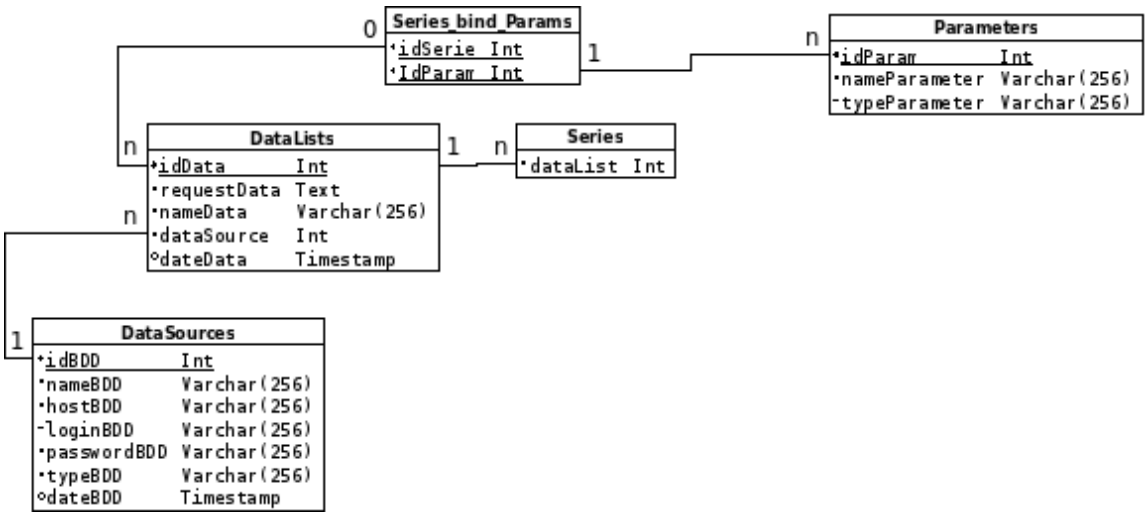
Plusieurs évolutions pour *ManageChart* ont été envisagées et chaque fois inscrite dans un livrable « Évolutions », avec la façon de procéder. Je résume ici les évolutions :

- Sécuriser des URLs en utilisant le protocole HTTPS. Aucune URL de *ManageChart* n'est protégée par le protocole HTTPS. Il peut être intéressant de le faire, pour la page de login par exemple ou la création/modification d'une connexion à une base de données distante.
- Demander un e-mail de confirmation lors de la création d'un utilisateur. Actuellement c'est un administrateur qui inscrit les utilisateurs. Il est notamment demandé une adresse e-mail, mais celle-ci n'est pas vérifiée avec l'envoi d'un e-mail accompagné d'un lien d'activation du compte. Cela est possible avec le *bundle FOSUser*.
- Pré-remplissage des champs libellés et unité d'une série. Il existe dans les bases de données distantes une table qui détaille les informations de chaque champ d'une table « mesures ». Il est envisagé de lire cette table et de pré-remplir les champs libellé et unité d'une série, lors de la sélection d'un des champs. Cependant cette table qui regroupe ces informations n'a pas toujours le même nom, ni les mêmes noms de champ. Il y a une solution avec des ALIAS dans la création de la requête SQL associée.
- N'autoriser que les *SELECT* sur les bases de données distantes. Actuellement on peut exécuter des requêtes autres que *SELECT* sur une base de données distantes. Cela n'est pas gênant dans la mesure où les comptes saisis pour se connecter à ces bases de données n'ont que des droits en lecture seule. Mais il serait plus sûr de vérifier que la requête SQL est bien une *SELECT* et non autre chose.
- Verrouiller un graphique. On pourrait laisser la possibilité à un administrateur de verrouiller un graphique empêchant toute modification ou suppression. Cela se traduirait par une case à cocher en face de chaque graphique, et la modification de la table *chart* pour intégrer ce champ.
- Créer un éditeur de requête SQL. Actuellement il n'y a aucune aide pour créer des requêtes SQL. On cependant envisager d'afficher la liste des tables pour une base de données, et pour chacune la liste de leurs champs. La liste des fonctions exécutables serait un plus et l'insertion de ces champs aussi.

V) Annexes

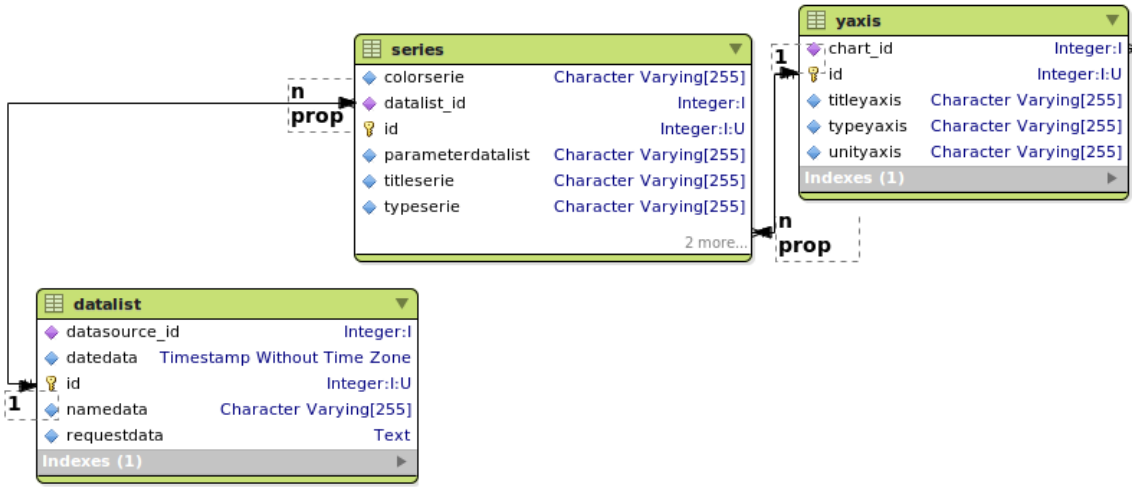
V.1) Annexe 1 : Conception de la BDD de ManageChart

V.1.1.1) Schéma détaillé des données



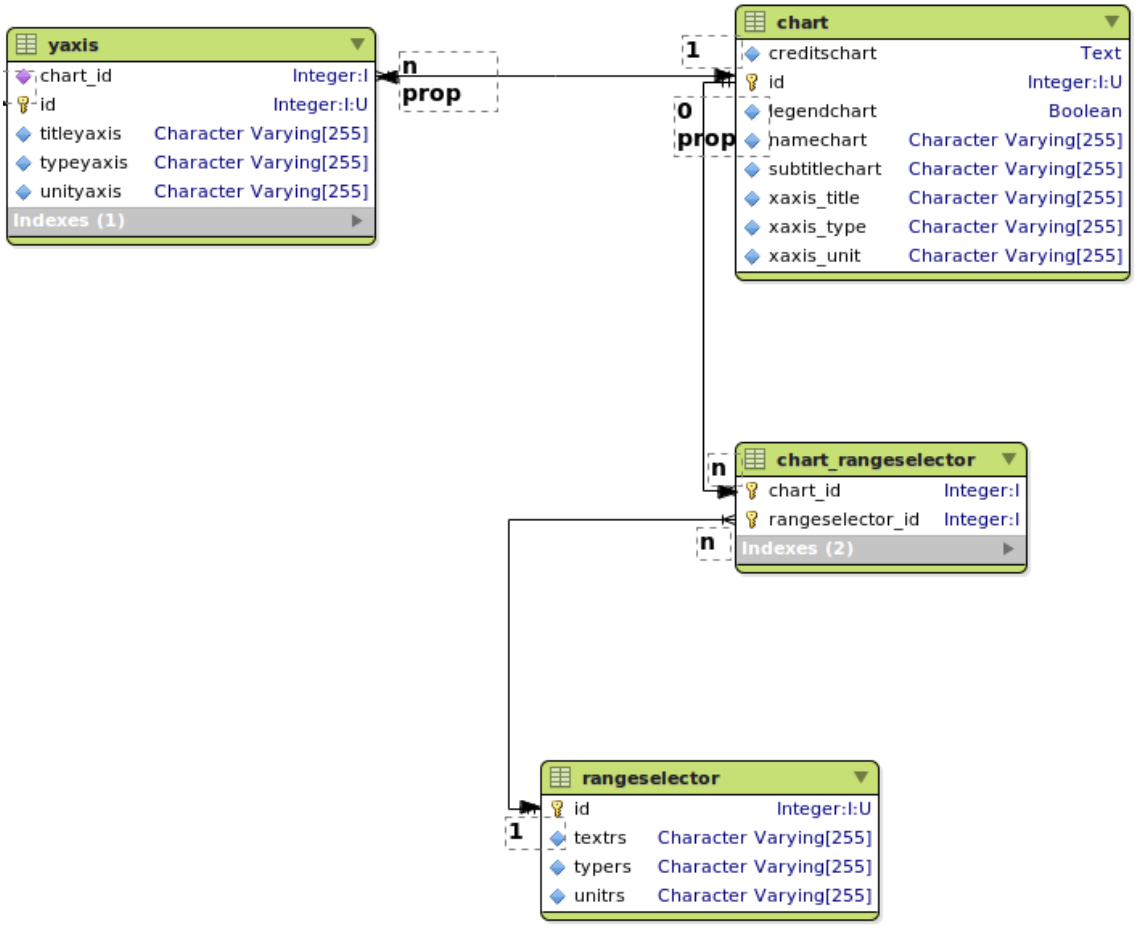
Les *DataSources* représentent les bases de données elle-mêmes, et contiennent les informations nécessaires pour créer la connexion. Les *DataLists* représentent les requêtes SQL effectuées sur ces bases de données. Elles sont reliées par une relation un-plusieurs aux *DataSources*, pour enregistrer 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 spatial dans la table et la requête. Il faut renseigner le nom et le type du paramètre. Enfin les *DataLists* sont reliées aux *Series* par une relation un-plusieurs.

V.1.1.2) Schéma détaillé des séries



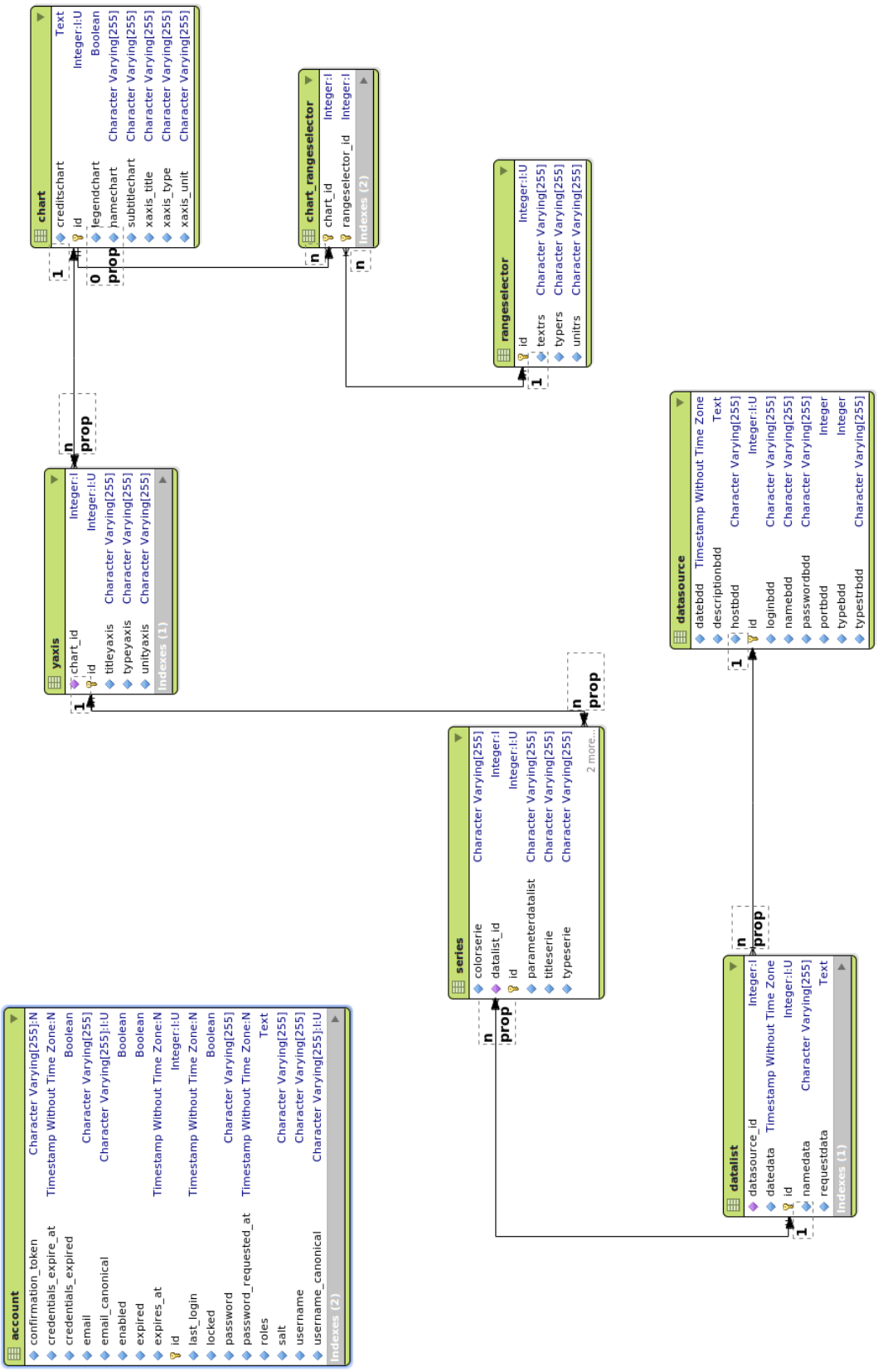
Une série de données est reliée d'une part à un *dataList* pour récupérer les données, et d'autre part à un axe des ordonnées. Ce dernier est relié au graphique, ce qui permet de partager une ordonnée avec d'autres séries ou d'avoir sa propre ordonnée.

V.1.1.3) Schéma détaillé des graphiques

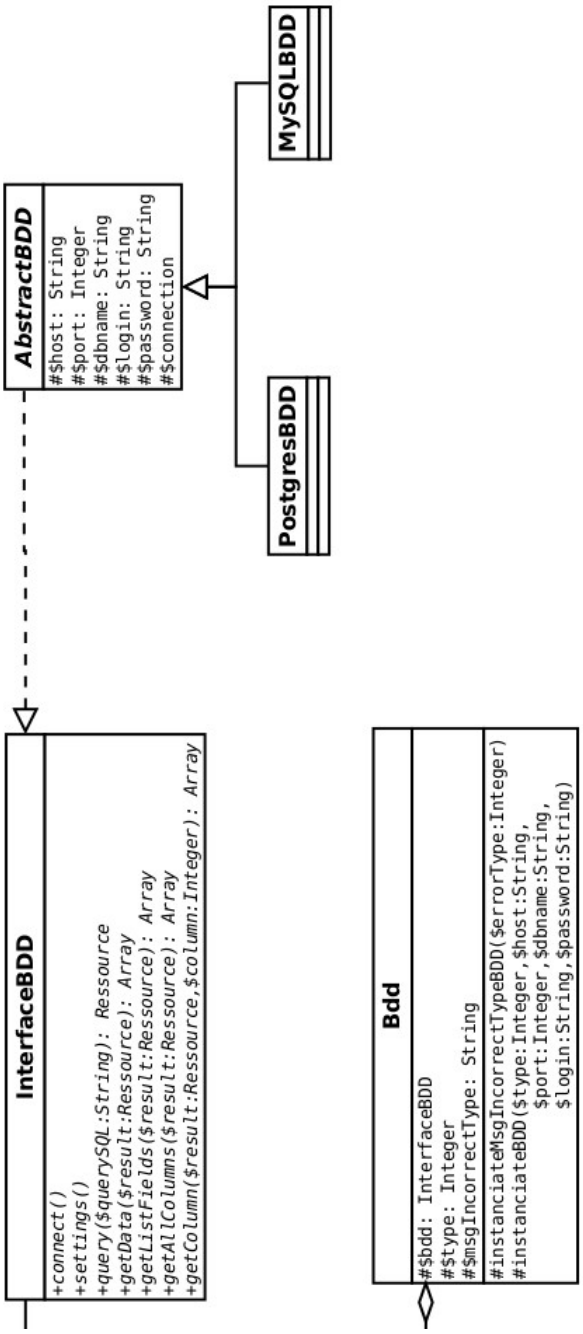


Un graphique contient toutes les informations pour le créer, et est relié aux sélecteurs d'échelle grâce à la table *Charts_bind_RangeSelectors* pour les graphiques temporels.

V.1.1.4) Schéma complet



V.2) Annexe 2 : Conception de BddBundle



L'interface définit les fonctions communes et permet d'abstraire leur implémentation. La classe abstraite, elle, définit les attributs et fonctions communs à toutes les classes qui implémentent l'interface. Les classes filles implémentent donc l'interface en utilisant les extensions *php* spécifiques aux fabricants des bases de données. Enfin la classe *Bdd* est optionnelle. Elle rajoute simplement une couche d'abstraction sur le type de base de données à instancier.

VI) Références

- **Liens :**

  http://letg.univ-nantes.fr

  http://inspire.ign.fr/

  http://menir.univ-brest.fr

  http://symfony.com

  http://fr.openclassrooms.com/informatique/cours/developpez-votre-site-web-avec-le-framework-symfony2

  http://fr.wikipedia.org

- **Ressources :**

  http://www.indigeo.fr/

  http://www.highcharts.com/

  http://d3js.org/

  http://wapps/semaphores/tableau_de_bord/index.php?page=rapport&sem=1&type=global&annee=2011

- **Documents :**

  http://www-iuem.univ-brest.fr/zabri/fr/documents/journee-za-18-janvier-2013/diaporama_theme3_peuziat

- **Rapport :**

  Rapport de Amir Khnissi et Redouane Oulla pour leur projet exploratoire du M2 SIAM de l'UBO