|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **VERSION** | **AUTEUR** | **DATE** | **FONCTION** | **REFERENCE** |
| 1.1 | Sébastien Deschamps | 07/04/2017 | Resp. BU | SD\_SPECS\_PORTAIL\_BI\_LIGHT\_V1.1DOCX |

|  |  |  |
| --- | --- | --- |
| **DESTINATAIRES** | **FONCTION** | **STRUCTURE** |
|  |  |  |

|  |  |
| --- | --- |
| **ANNEXE** | **DOCUMENT** |
|  |  |

OBJECTIF DU DOCUMENT :

Ce document décrit l’architecture et les fonctionnalités attendues dans la V1du portail « Apologic BI Light»

TABLE DES MATIERES :

[1. GENERALITES 3](#_Toc479320777)

[1.1. OBJECTIF DU PROJET 3](#_Toc479320778)

[1.2. TECHNOLOGIES UTILISEES 3](#_Toc479320779)

[1.3. ARCHITECTURE GENERALE 4](#_Toc479320780)

[2. SERVEUR WEB 5](#_Toc479320781)

[2.1. IDENTIFICATION 5](#_Toc479320782)

[2.2. GESTION DES UTILISATEURS 6](#_Toc479320783)

[2.2.1. CREATION D’UN UTILISATEUR 6](#_Toc479320784)

[2.2.2. MODIFICATION D’UN UTILISATEUR 7](#_Toc479320785)

[2.2.3. VALIDATION DE LA FICHE UTILISATEUR 7](#_Toc479320786)

[2.3. GESTION DES BASES DE DONNEES LIEES AUX UTILISATEURS 9](#_Toc479320787)

[2.4. RAPPORTS 9](#_Toc479320788)

[2.5. ABONNEMENT AUX RAPPORTS 12](#_Toc479320789)

[2.6. WIDGETS 13](#_Toc479320790)

[2.7. REQUETEUR 14](#_Toc479320791)

[2.8. ENVOI DE RAPPORTS PAR EMAIL 14](#_Toc479320792)

[2.9. OUVERTURE DU PORTAIL BO 14](#_Toc479320793)

[2.10. WEBSEVICES 14](#_Toc479320794)

[3. GESTION DES JETONS 16](#_Toc479320795)

[3.1. PRINCIPE 16](#_Toc479320796)

[3.2. SUIVI DES JETONS ACHETES PAR LE CLIENT 16](#_Toc479320797)

[3.3. SUIVI DES JETONS CONSOMMES PAR LE CLIENT 17](#_Toc479320798)

[4. SERVEUR DATAS 18](#_Toc479320799)

[5. SERVEUR DE GENERATION 18](#_Toc479320800)

[5.1. COMMUNICATION AVEC LE PORTAIL 18](#_Toc479320801)

[6. SPECIFICITES DE DEVELOPPEMENT DU PROJET 20](#_Toc479320802)

[6.1. SURCHAGE DU MODELE 20](#_Toc479320803)

[6.2. SURCHARGE DU CONTROLLEUR 20](#_Toc479320804)

[6.3. LIBRAIRIES 20](#_Toc479320805)

[6.4. HELPERS 21](#_Toc479320806)

[6.5. CONFIG.PHP 21](#_Toc479320807)

[6.6. DATABASE.PHP 21](#_Toc479320808)

[6.7. ROUTES.PHP 22](#_Toc479320809)

1. GENERALITES
   1. OBJECTIF DU PROJET

Actuellement, la solution décisionnelle proposée par Apologic repose sur l’utilisation de Business Objects couplé à un infocentre. Cette solution peine à se vendre compte tenu des coûts de licence et d’exploitation (maintenance et administration de BO chronophage). L’automatisation du déploiement d’un nouveau client sous BO n’est également pas possible, obligeant de multiples interventions manuelles sur le serveur pour chaque nouveau compte client à créer.

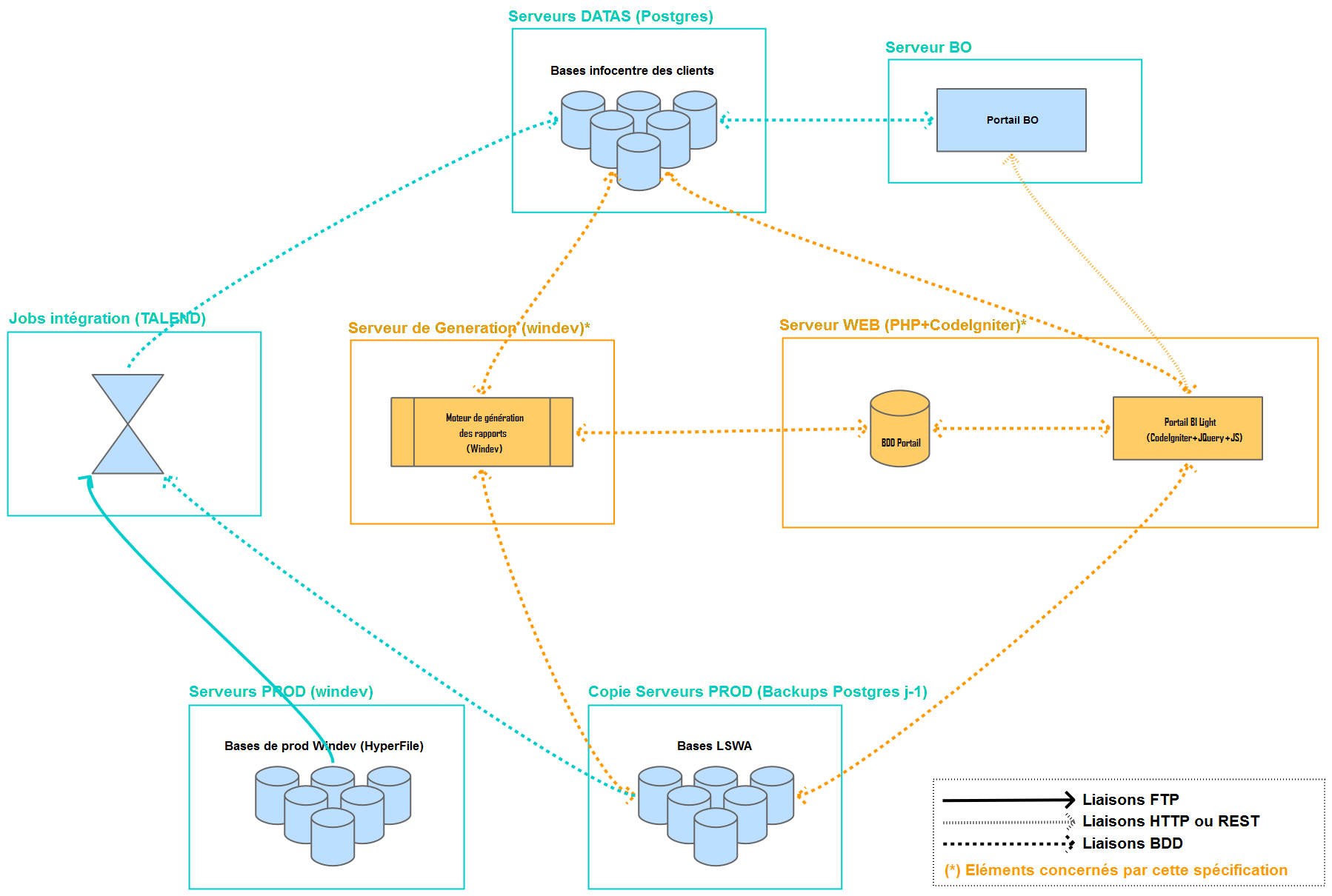
Le projet Apologic BI Light doit permettre de proposer une solution extrêmement simple d’accès et intuitive, permettant à l’utilisateur final de lancer la génération de rapports ou l’affichage de widgets préétablis en quelques clics.

Le facturation de la solution se fera en fonction du consommé de chaque client, via un système de jetons.

* 1. TECHNOLOGIES UTILISEES

Les technologies utilisées pour ce projet sont les suivantes :

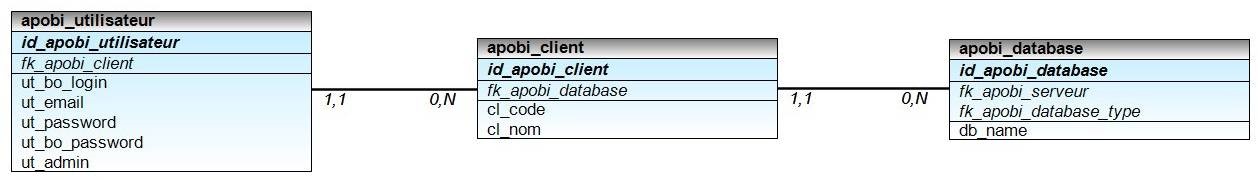
* Framework PHP CodeIgniter pour le portail
* JavaScript et JQuery pour certains traitements AJAX et certaines parties de l’UI
* PostgreSQL pour les bases de données (portail et infocentres)
* WinDev pour le moteur de génération des rapports
* D3.JS pour la génération des widgets
* HTML5 + CSS
* IIS pour le serveur web
  1. ARCHITECTURE GENERALE



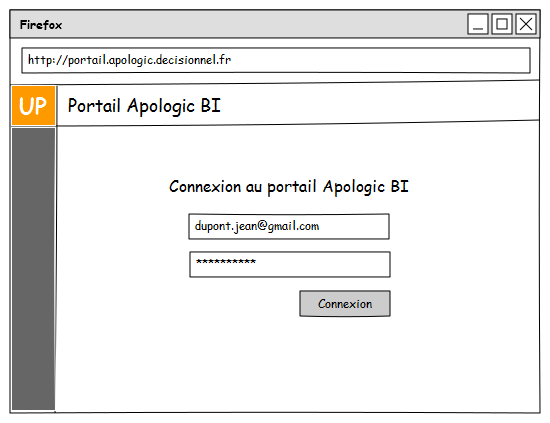
1. SERVEUR WEB

Le serveur WEB va héberge le portail PHP (serveur apache sous linux), ainsi que la base de donnée du portail (contenant le paramétrage, les comptes utilisateurs, etc…).

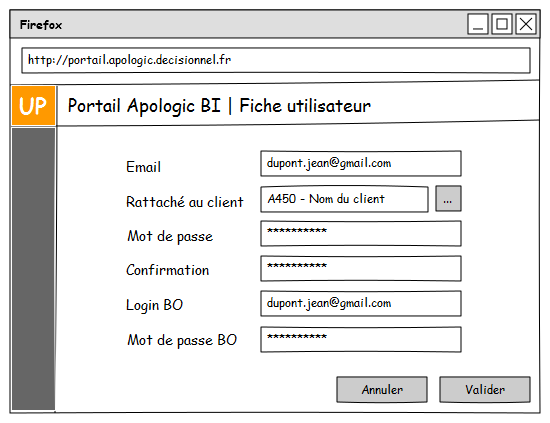
* 1. IDENTIFICATION
* Lors de la connexion au portail, l’utilisateur doit impérativement s’identifier via un « login/mot de passe ». C’est l’utilisateur qui va déterminer sur quel infocentre les données doivent être lues. Il est impératif que dans 100% des cas, l’utilisateur connecté puisse visualiser uniquement les données de l’infocentre qui lui est rattaché.
* La modélisation de la base de données prévoit que l’utilisateur soit lié à 1 ou n client(s), et le client lié à 1 ou n infocentre(s), ceci afin d’anticiper des cas particuliers futurs (par ex : Utilisateur ayant les droits sur plusieurs clients, ou même client disposant de plusieurs infocentres).



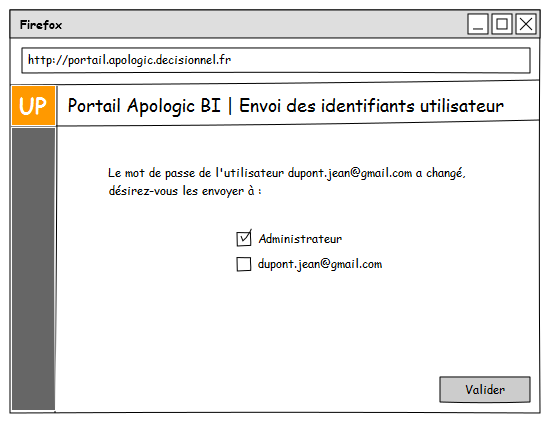
* Dans la V1 du portail, nous considérons que l’utilisateur est toujours lié à un seul client, et que chaque client possède toujours un unique infocentre.
* Le mot de passe n’est pas stocké en clair dans la base, mais sous forme hashée (Au minimum hashage SHA avec inclusion d’un grain de sel).
* L’utilisateur, une foi connecté, peux modifier son mot de passe via un menu dédié (double confirmation du mot de passe, sur 8 caractères minimum, comprenant 3 classes de caractères au moins). Le changement de mot de passe est loggé (Voir librairie logging : LIBRAIRIES).
* La fenêtre de connexion est de ce type :



* 1. GESTION DES UTILISATEURS
* La gestion de l’ensemble des utilisateurs (Création / Modification / Suppression / Changement de mot de passe) est accessible aux administrateurs via l’interface d’administration (bouton spécifique dans la barre latérale de gauche).
* La gestion des clients « apobi\_client » et des infocentres « apobi\_database » n’est pas accessible depuis le portail, mais uniquement depuis le moteur WinDev, ou via la réalisation manuelle de requêtes SQL (à terme la création des client et des infocentres sera totalement automatisée, aucune administration manuelle ne sera donc nécessaire).
  + 1. CREATION D’UN UTILISATEUR
  + En création d’un utilisateur, l’administrateur peux saisir les éléments suivants :
    - L’email de l’utilisateur (faisant office de login)
    - Un mot de passe (optionnel : si aucun mot de passe n’est spécifié, le système doit en générer un automatique respectant un niveau de sécurité minimum : 8 caractères minimum combinant 3 classes de caractères)
    - Le login/Mot de passe du compte BO lié à l’utilisateur (optionnel). Le mot de passe BO doit être stocké de manière cryptée dans la base du portail, mais non hashée, afin de permettre son décryptage.



* + 1. MODIFICATION D’UN UTILISATEUR
* En modification d’une fiche utilisateur, si le mot de passe n’est pas spécifié, il ne doit pas être re-généré.
  + 1. VALIDATION DE LA FICHE UTILISATEUR
* Après la validation d’une fiche utilisateur, dans le cas où le mot de passe a été changé manuellement ou généré automatiquement, une fenêtre doit apparaitre, permettant d’envoyer les nouveaux identifiants de l’utilisateur par mail.



Un mail de ce type est alors envoyé :

Sujet :

*Vos identifiants de connexion Apologic BI*

Message :

*Bonjour,*

*Ci-dessous vous trouverez vos identifiants de connexion à la plateforme Apologic BI :*

*URL : http://?????????.apologic.fr*

*Login : xxxxxxxx*

*Mot de passe : xxxxxxxxx*

*Cordialement,*

*Sébastien Deschamps*

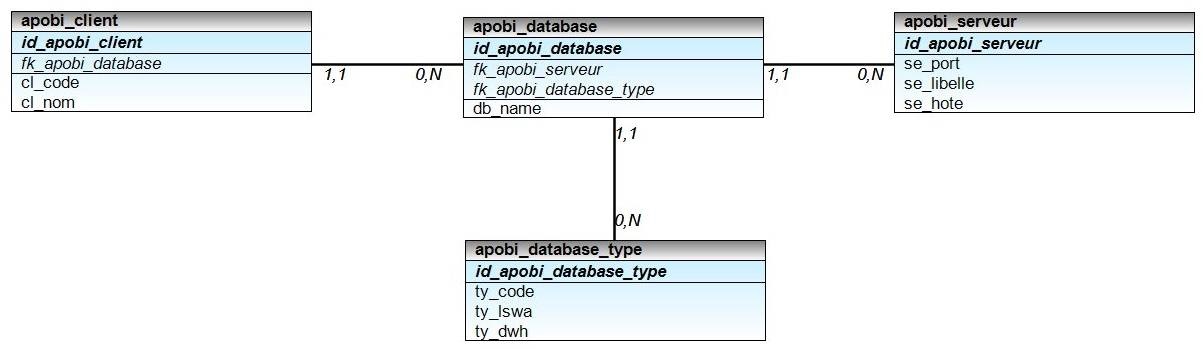
* 1. GESTION DES BASES DE DONNEES LIEES AUX UTILISATEURS

Le portail ainsi que le moteur de génération doivent être en mesure d’identifier la base de données sur laquelle sont executées les requête d’extraction des données (pour la génération des exports, diagrammes, rapports, etc…).

Comme expliqué précédemment, chaque utilisateur est lié à un client, et chaque client est lié à une ou plusieurs base de donnée. Ces bases possèdent un type :

* « DWH » pour Datawarehouse : ce sont des infocentres.
* « LSWA » (La Solution Web Apologic) : ce sont les copies des bases de production LSWA.

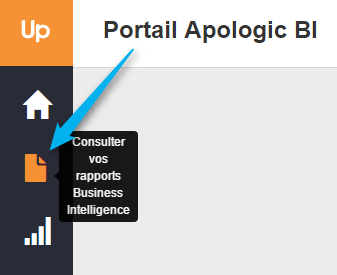
Dans 90% des cas, les rapports et requêtes d’extraction sont basées sur les données d’un infocentre. Nous conservons néanmoins la possibilité de réaliser des requêtes sur une base LSWA afin de répondre à certaines demandes spécifiques clients (export comptable spécifique à seul de nos clients, etc…).



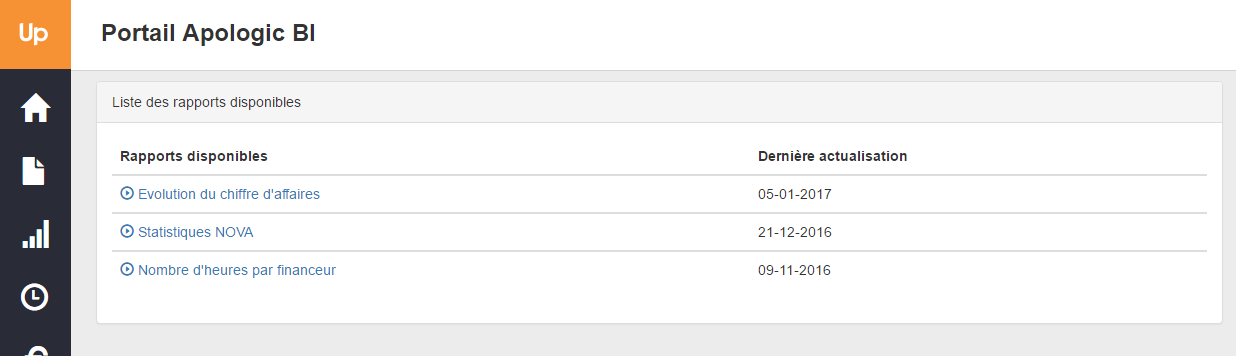
* Chaque client possède 0 ou N databases « apobi\_database ».
* Chaque database est liée à 1 serveur.
* La gestion des clients, databases et des serveurs se fait via le module Windev ou requête SQL écrite manuellement. A terme, ceci sera totalement automatisée. Il n’est donc pas nécessaire de développer le backoffice d’administration de ces éléments dans le portail.
* Dans tous les cas, les requêtes d’extraction de données ne peuvent être jouées que sur la/les base(s) liée(s) à l’utilisateur connecté.
  1. RAPPORTS

La génération des rapports est « sous-traitée » au moteur de génération WinDev (voir chapitre : SERVEUR DE GENERATION).

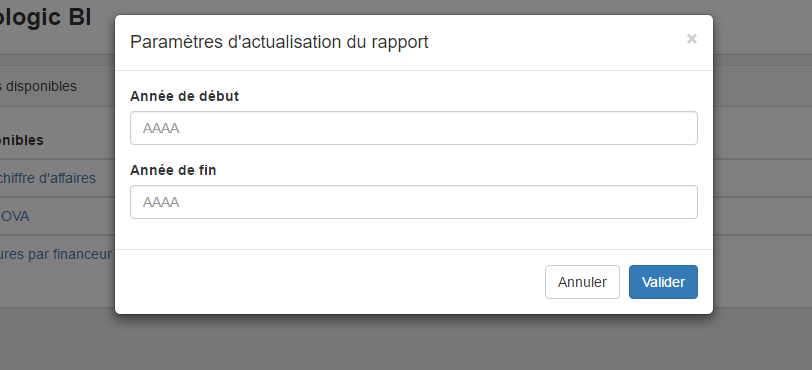
L’utilisateur peut accéder à la liste des rapports auxquels il a souscrit à partir de l’icône suivante du menu latéral de gauche :



L’utilisateur visualise alors les rapports disponibles :

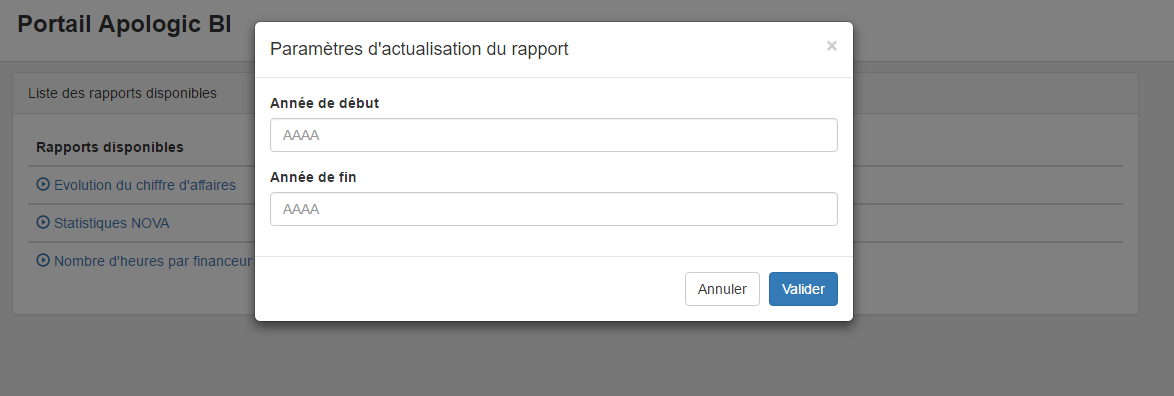


Lors du clic sur un rapport, un pop-up permettant à l’utilisateur de sélectionner les valeurs d’invite s’ouvre (selection d’une année d’analyse, d’un utilisateur, etc…) :



Les invites peuvent être statiques (valeur saisie par l’utilisateur), ou dynamique (sélection d’une valeur parmi une liste de propositions établie depuis la lecture des données de l’infocentre).

Lorsque l’utilisateur demande la génération d’un rapport, une boite de dialogue doit apparait, permettant à l’utilisateur de spécifier les valeurs d’invite à passer au moteur de génération (Code du secteur à afficher, Pèriode de filtrage, etc….).



Les invites peuvent avoir les types suivants :

* CHAINE
* BOOLEEN
* DATE
* ENTIER
* Dans le cas d’une date, un sélecteur de type « calendrier » doit apparaitre (voir <https://jqueryui.com/datepicker/>)
* Dans le cas d’un booléen, un sélecteur de type « case à cocher » doit apparaitre
* Dans le cas d’une chaine, et si une liste de valeurs prédéfinies est spécifiée, un sélecteur de type « combo » doit apparaitre

Modélisation de la base de données :

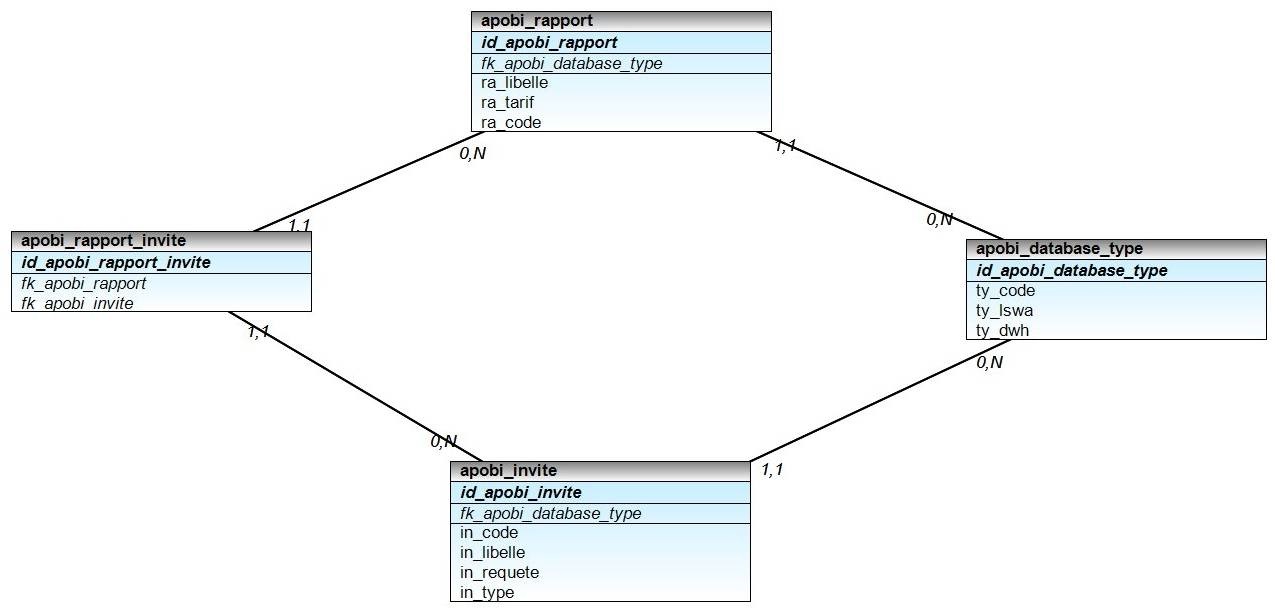


Table « apobi\_invite » :

Cette table contient la description de toutes les invites disponibles sur la plateforme.

|  |  |
| --- | --- |
| Colonne | Descriptif |
| fk\_apobi\_database\_type | Clé étrangère vers le type de base sur laquelle l’invite peux fonctionner (Infocentre ou LSWA) |
| in\_code | Code unique de l’invite |
| in\_libelle | Libellé de l’invite qui est affichée à l’utilisateur dans la boite de dialogue |
| in\_requete | Requête qui est jouée sur la base liée au rapport, pour permettre la récupération des valeurs préalimentées de la combo |
| In\_type | Type de donnée de l’invite (CHAINE,ENTIER,BOOLEEN,DATE), déterminant quel composant web doit être utilisé pour afficher le champ |

Table « apobi\_rapport\_invite » :

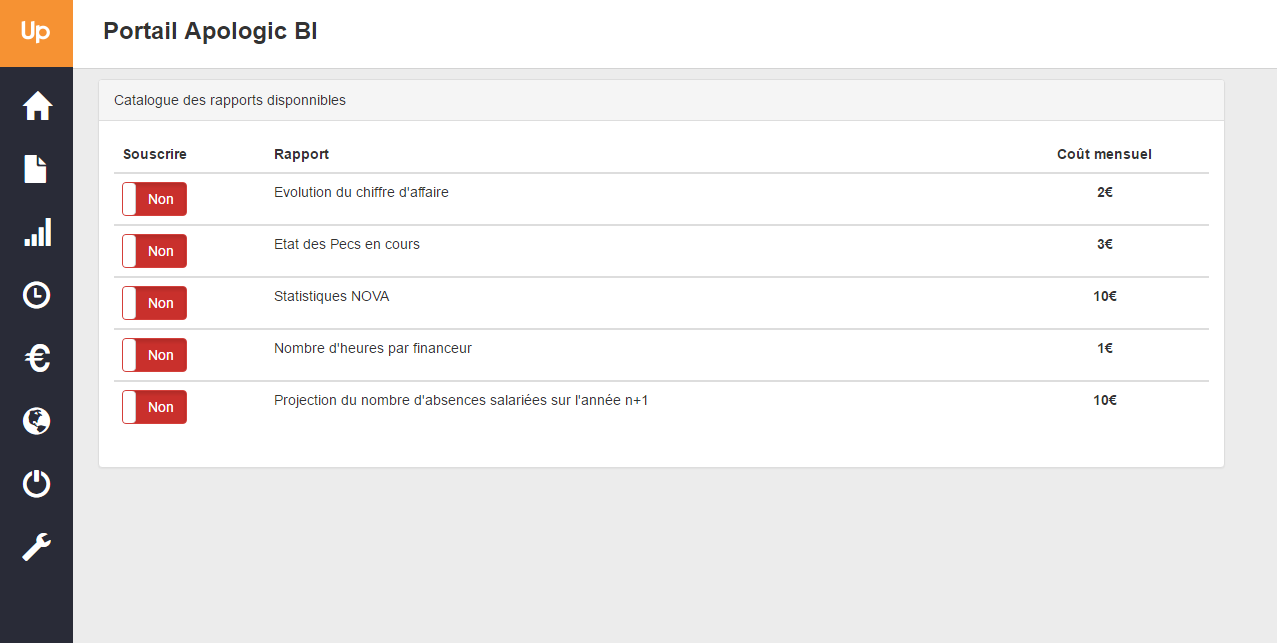
Cette table contient le détail des invites utilisées par chaque rapport de la plateforme.

|  |  |
| --- | --- |
| Colonne | Descriptif |
| Fk\_apobi\_rapport | Clé étrangère vers le rapport |
| Fk\_apobi\_invite | Clé étrangère vers l’invite reliée au rapport |

* L’administration des rapports de la plateforme est réalisée par le module Windev ou directement par requête SQL. Il n’est pas nécessaire de prévoir d’interface de gestion des rapports.
* Lors de la demande de génération du rapport auprès du moteur windev, le code du rapport « ra\_code » est passé en paramètre, permettant au moteur windev d’identifier le rapport à générer. Les valeurs d’invites sont également passée en paramètre au moteur de génération.
  1. ABONNEMENT AUX RAPPORTS

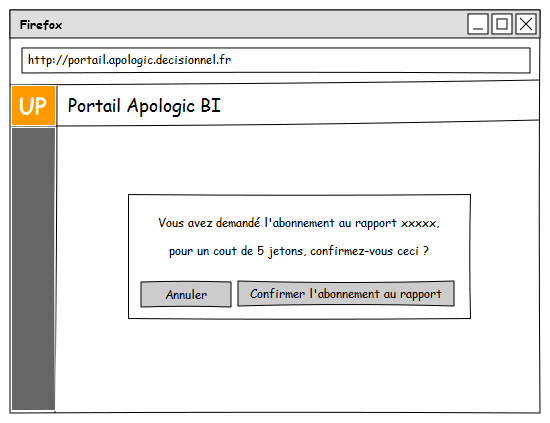
L’utilisateur doit pouvoir s’abonner aux rapports de son choix, afin de les rendre accessibles dans l’écran de génération des rapports. L’abonnement à un nouveau rapport est impossible si le crédit en jetons est insuffisant (voir :

GESTION DES JETONS).



La table apobi\_client\_abonnement contient l’enssemble des options souscrites par le client (abonnement à un rapport, un widget, une planification, etc..).

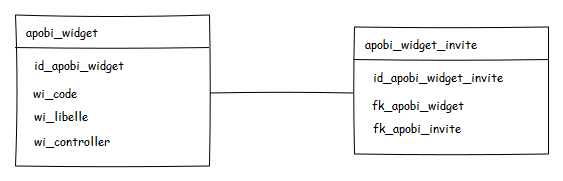
* Lorsque le client clique sur le bouton de souscription, une fenêtre de confirmation doit apparaitre (type popup ou autre), permettant à l’utilisateur de confirmer sa demande d’abonnement au rapport :



* Lors de la souscription d’un rapport, un enregistrement est crée dans la table « apobi\_client\_abonnement » (voir chapitre : SUIVI DES JETONS CONSOMMES PAR LE CLIENT).
* Lors de la désactivation d’un rapport, on recherche la ligne « apobi\_client\_abonnement » pour le client en cours, le rapport en cours, et qui ne possède pas de date de désactivation. Si cette ligne est trouvée, alors on met à jour le date de désactivation avec la date du jour.
  1. WIDGETS

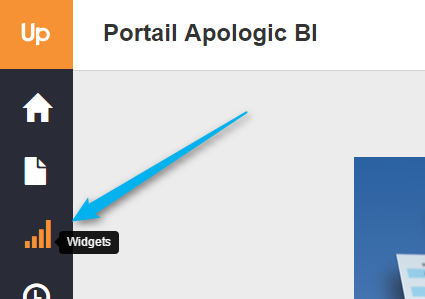
Le portail propose également des widgets, développés à l’aide du framework d3.js.

Comme pour les rapports, une nouvelle table contient la liste des widgets disponibles. Chaque widget pouvant être lié à une ou plusieurs invites :



Chaque widget est développé dans une vue indépendante, et est accessible via un controlleur spécifique. Le controlleur à utiliser pour l’affichage du widget est pécifié dans la colonne « wi\_controller » de « apobi\_widget ».

Comme pour les rapports, les widgets sont accessibles via un bouton dédié de l’interface du portail :



L’utilisateur visualise alors les widget disponibles dans une liste et est alors en mesure d’en afficher un d’un simple clic.

* L’administration des widgets de la plateforme est réalisée par le module Windev ou directement par requête SQL. Il n’est pas nécessaire de prévoir d’interface de gestion des widgets.
  1. REQUETEUR
  2. ENVOI DE RAPPORTS PAR EMAIL
  3. OUVERTURE DU PORTAIL BO
  4. WEBSEVICES

Le portail doit proposer une API Rest, permettant d’échanger certaines données avec d’autres applications. Dans un premier temps, seuls deux webservices sont nécessaires :

* Permettre le renvoi du nom de toutes les bases LSWA déclarée sur le portail
* Permettre le renvoi du nom de toutes les bases DWH déclarées sur le portail

Ci-dessous quelques liens détaillant la méthodologie à appliquer pour générer des webservices Rest sous CodeIgniter 3 :

<https://code.tutsplus.com/tutorials/working-with-restful-services-in-codeigniter--net-8814>

<https://etienner.fr/api-restful-sur-codeigniter-3>

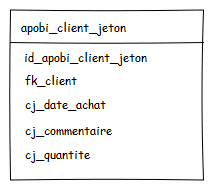
* Un permier web service, accessible uniquement aux administrateurs (coche « ut\_admin » de « apobi\_utilisateur ») ayant droit d’accéder aux API (coche « ut\_api » de « apobi\_utilisateur ») doit être créée. Cette API Rest permet de récupérer un listing de toutes les bases de données référencées sur le portail.
* Le retour JSON doit contenir les infos suivantes :
  + Apobi\_serveur.se\_hote
  + Apobi\_serveur.se\_port
  + Apobi\_database.db\_name
  + Apobi\_type\_database.tdb\_code
  + Apobi\_type\_database.tdb\_libelle
  + Apobi\_client.cl\_code
  + Apobi\_client.cl\_nom

1. GESTION DES JETONS
   1. PRINCIPE

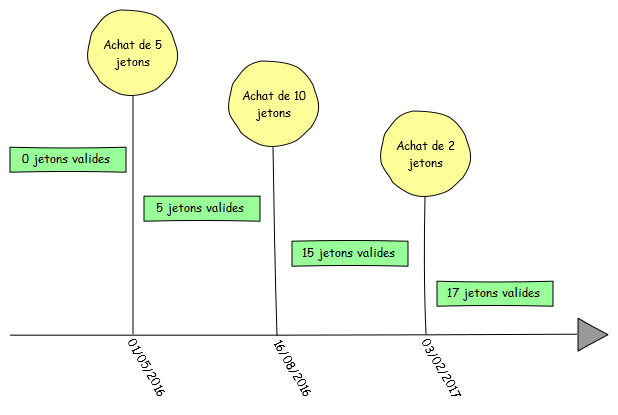
La facturation des rapports mis à disposition sur le portail se fait via un système de jetons :

* Selon le forfait souscrit par le client, un nombre de jetons lui est alloué.
* Chaque éléments activable sur le portail possède un cout en jetons.
* Chaque rapport, widget ou planification activée au moins une fois sur un mois consomme la quantité de jetons correspondante.
* A chaque activation d’une nouvelle option, le système calcule le solde d’utilisation des jetons (SUIVI DES JETONS - SUIVI DES JETONS CONSOMMES). Si le solde est insufisant, l’option ne s’active pas et un message d’information apparait, invitant l’utilisateur à se rapprocher d’Apologic pour commander des jetons supplémentaires.
  1. SUIVI DES JETONS ACHETES PAR LE CLIENT

Afin de permettre le suivi de la croissance de l’offre, la table des jetons possédés par chaque client va permettre de reconstituer l’historique des différentes commandes passées au fil du temps :

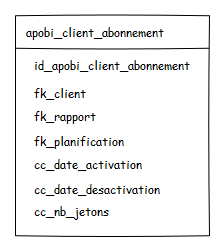


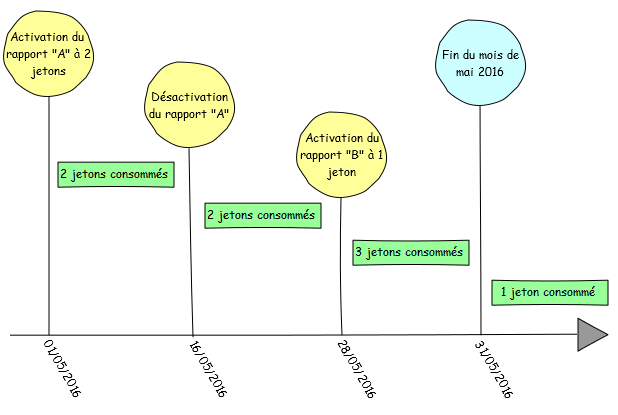
Chaque nouvelle commande donne lieu à une nouvelle ligne dans la table « apobi\_client\_jeton ». Le fonctionnement qui permet de déterminer le nombre de jetons du client à une date donnée est basé sur le principe suivant :



* 1. SUIVI DES JETONS CONSOMMES PAR LE CLIENT

D’une manière similaire au suivi des jetons acheté par le client à un instant T, le portail est en mesure de déterminer le nombre de jetons consommés par à un client à une date donnée, et éviter que le client puisse désactiver/réactiver les rapports au fil de ses besoins (et ainsi contourner la limitation du nombre de jetons acquis).





La requête permettant de retrouver le nombre de jetons consommés à une date donnée est du type :

***SELECT SUM(cc\_nb\_jeton)***

***FROM apobi\_client\_conso***

***WHERE fk\_client = ?***

***AND (cc\_date\_activation, coalesce(cc\_date\_desactivation, current\_date))***

***OVERLAPS (« date\_debut\_du\_mois\_en\_cours », « date\_fin\_du\_mois\_en\_cours »)***

1. SERVEUR DATAS

Les bases de données infocentre se trouvent sur des serveurs distincts, au format PostgreSQL. Ces bases sont alimentées la nuit par des traitements TALEND. Le projet Apologic BI Light ne doit pas intervenir ou venir modifier les données présentes dans ces infocentres.

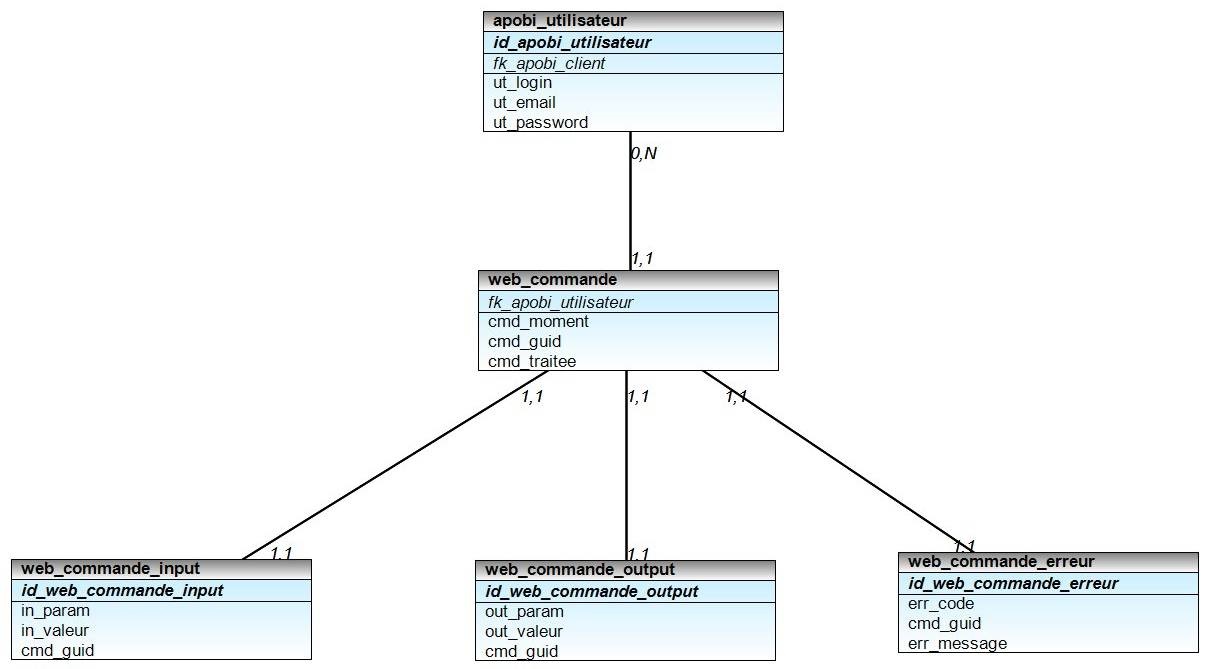
1. SERVEUR DE GENERATION

Le serveur de génération Windev permet la génération de fichiers de reporting (Rapports PDF, Fichiers Excel, Fichiers CSV, etc..). Ce serveur doit pouvoir être découplé du serveur d’hébergement du portail BI Light, afin de rendre la solution scalable.

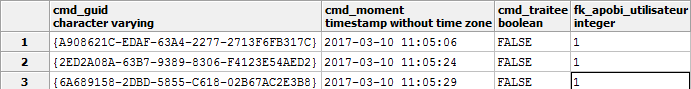
* 1. COMMUNICATION AVEC LE PORTAIL

Le serveur de génération dialogue avec le portail via l’intermédiaire de la base de données. L’exe de génération tournera en tant que service Windows sur le serveur de génération. Les rapports générés sont retournés au portail via la table web\_commande\_output.

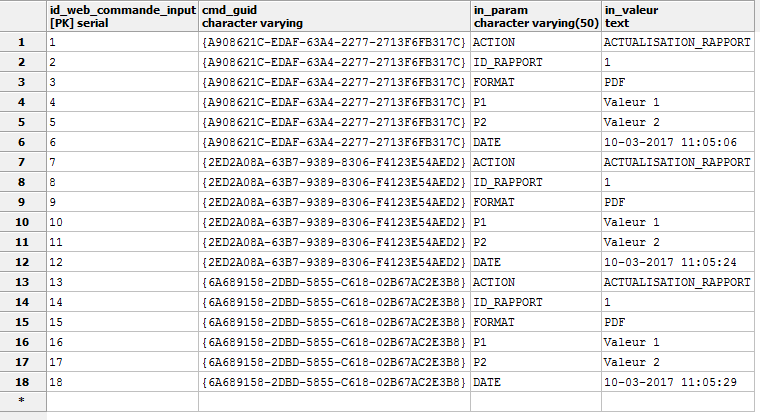
Les tables actuellement en place pour cette communication sont les suivantes :



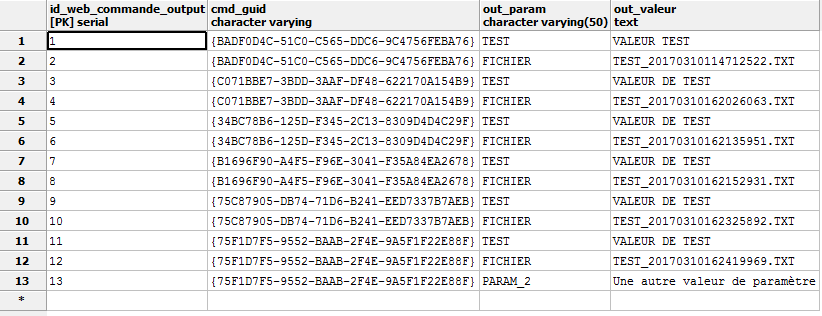
**« web\_commande »**: Contient l’identifiant unique (GUID) de la commande déclenchée par le portail, et à destination du serveur de génération, la date et l’heure de la commande, l’utilisateur l’ayant déclenchée, ainsi que son état de traitement par le moteur de génération.



**« web\_commande\_input »** : Contient les paramètres d’entrée liés à une commande. Selon les commandes, le nombre et le type des paramètres d’entrées est susceptible de varier.



**« web\_commande\_output »** : Contient les paramètres de sortie de la commande. Cette table est alimentée par le moteur de génération Windev.

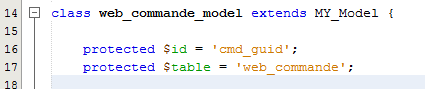


1. SPECIFICITES DE DEVELOPPEMENT DU PROJET
   1. SURCHAGE DU MODELE

La classe « /application/core/MY\_Model » surchage le modèle à utiliser par défaut dans CodeIgniter, et ajoute de nouvelles fonction, en particulier :

* Commit($data) : Permet de sauvegarder les données en base, en gérant automatiquement l’ajout ou la modification de l’enregistrement, selon la présence ou non d’un $this->id valide.
* Delete($id) : Permet de supprimer un enregistrement en se basant sur son id.
* Get($id) : Permet de récupérer un enregistrement en se basant sur son id

Lors de la création d’un nouveau modèle, on utilise donc la syntaxe suivante :



Les deux attributs $id et $table doivent impérativement être initialisés, afin de permettre le bon fonctionnement de la classe parent « MY\_Model ».

* 1. SURCHARGE DU CONTROLLEUR

Comme pour le modèle, le controlleur est surchargé par « /application/core/MY\_Controller ». Actuellement cette classe ne propose rien de plus que le controlleur classique, mais doit être utilisée en prévision de la gestion des droits (utilisateur non connecté, utilisateur simple, administrateur).

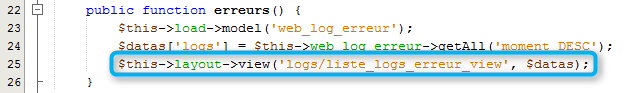
Lors de la création d’un nouveau controlleur, on utilise donc la syntaxe suivante :



* 1. LIBRAIRIES

Quelques librairies personnalisées ont été mises en place sur le projet, dans « /application/librairies/ » :

* Apobie.php : Contient des fonctions permettant de lancer le moteur de génération windev apobie.exe.
* Logging.php : Contient des fonctions permettant d’insérer facilement des logs dans les tables « web\_log\_erreur » et « web\_log\_info ».
* Layout.php : Permet la gestion d’un système de template, permettant définir l’architecture globale du site via un modèle présent dans « /application/templates/up/template.php ». Chaque ouverture de vue doit se faire via « $this->layout->view(…) » et non plus « $this->load->view(…) ».



* 1. HELPERS

Deux helpers personnalisés sont présents dans le projet :

* General\_helper.php : Contient diverses petites fonctions utiles (conversion de dates, génération de GUID, etc…).
* Bootstrap\_helper.php : Contient des fonctions permettant de faciliter et uniformiser l’utilisation du framework Bootstrap. Pour créer une table, nous utiliserons par exemple les fonctions suivantes du helper :



* 1. CONFIG.PHP

Le fichier de configuration CodeIgniter présent « /application/config/config.php » possède certains paramètres spécifiques à ce projet :



* 1. DATABASE.PHP

Le paramétrage de la connexion à la base de données du portail se fait dans le fichier « /application/config/database.php » :



* 1. ROUTES.PHP

Les routes d’URL sont définies dans le fichier « /application/config/routes/php », afin d’éviter les URL du type :

*http://127.0.0.1:8080/edsa-apobi/catalogue\_controller/index*