Manier Gaetan

42 Rue Saint Firmin

59190 Morbecque

Mail : [gaetan.manier@sfr.fr](mailto:gaetan.manier@sfr.fr)

Dossier Projet de VAE

Master informatique parcours

Ingénierie des Systèmes Informatiques Distribués

Jury du 29 juin 2018

Université du Littoral Côte d’Opale

[1 Introduction 5](#_Toc504071467)

[2 Le cahier des charges 6](#_Toc504071468)

[3 Fonctionnalités 7](#_Toc504071469)

[3.1 Introduction 7](#_Toc504071470)

[3.2 L’application WEB 7](#_Toc504071471)

[3.2.1 La page d’accueil 7](#_Toc504071472)

[3.2.2 La gestion des accès utilisateur 8](#_Toc504071473)

[3.2.3 La page « Contact » 8](#_Toc504071474)

[3.2.4 La page « à propos de » 8](#_Toc504071475)

[3.2.5 La page « Liste des produits» 9](#_Toc504071476)

[3.2.6 La page « Recherche d’un produit» 10](#_Toc504071477)

[3.2.7 La page « Bilans produits» 11](#_Toc504071478)

[3.3 L’application mobile 11](#_Toc504071479)

[4 Construction du projet 12](#_Toc504071480)

[4.1 Choix des outils 12](#_Toc504071481)

[4.2 Découpage du projet sous VS2013 13](#_Toc504071482)

[5 Architecture du logiciel 14](#_Toc504071483)

[5.1 Architecture de l’application web 14](#_Toc504071484)

[5.1.1 Les modèles 15](#_Toc504071485)

[5.1.2 Les vues 17](#_Toc504071486)

[5.1.3 Les contrôleurs 17](#_Toc504071487)

[5.1.4 Principe de fonctionnement 18](#_Toc504071488)

[5.1.5 Publication de l’application 20](#_Toc504071489)

[5.1.6 Conclusion 21](#_Toc504071490)

[5.2 Architecture de l’application mobile 22](#_Toc504071491)

[5.2.1 Introduction 22](#_Toc504071492)

[5.2.2 L’application Web API 23](#_Toc504071493)

[5.2.3 L’application Android 24](#_Toc504071494)

[5.2.4 Conclusion 25](#_Toc504071495)

[5.3 Le serveur Web 25](#_Toc504071496)

[6 Architecture de la base de données 27](#_Toc504071497)

[6.1 Configuration du serveur de base de données 27](#_Toc504071498)

[6.2 Introduction aux Data Warehouse 27](#_Toc504071499)

[6.3 Choix du schéma 29](#_Toc504071500)

[6.4 Alimentation du Data Warehouse avec SSIS 30](#_Toc504071501)

[6.4.1 Configuration de l’ETL 30](#_Toc504071502)

[6.4.2 Déploiement de l’ETL 34](#_Toc504071503)

[6.4.3 Exécution de l’ETL 36](#_Toc504071504)

[6.4.4 Conclusion 38](#_Toc504071505)

[6.5 Création et gestion du cube avec SSAS 38](#_Toc504071506)

[6.5.1 Connexion à la base de données 39](#_Toc504071507)

[6.5.2 Vue des sources de données 39](#_Toc504071508)

[6.5.3 Création des dimensions 40](#_Toc504071509)

[6.5.4 Création du cube 41](#_Toc504071510)

[6.5.5 Déploiement du cube 42](#_Toc504071511)

[6.5.6 Exécution du cube avec Excel 43](#_Toc504071512)

[6.5.7 Conclusion 45](#_Toc504071513)

[6.6 Création des rapports avec SSRS 45](#_Toc504071514)

[6.6.1 Création des rapports 45](#_Toc504071515)

[6.6.2 Configuration de Reporting Service 48](#_Toc504071516)

[6.6.3 Déploiement des rapports 50](#_Toc504071517)

[6.6.4 Consultation des rapports 50](#_Toc504071518)

[6.6.5 Conclusion 51](#_Toc504071519)

[7 La sécurité 52](#_Toc504071520)

[7.1 Les comptes utilisateurs 52](#_Toc504071521)

[7.1.1 Compte pour l’application Web 53](#_Toc504071522)

[7.1.2 Compte pour le SSAS 53](#_Toc504071523)

[7.2 Protection des pages web 54](#_Toc504071524)

[7.3 Conclusion 54](#_Toc504071525)

[8 Qualité du code, gestion des versions et des fiches recettes 56](#_Toc504071526)

[8.1 Choix de la gestion des versions 56](#_Toc504071527)

[8.1.1 Copie et renommage de répertoires 56](#_Toc504071528)

[8.1.2 Utilitaire de gestion de code sources 56](#_Toc504071529)

[8.1.3 GitHub 57](#_Toc504071530)

[8.2 Les fiches de recettes 59](#_Toc504071531)

[8.2.1 Les fiches de recette 59](#_Toc504071532)

[8.2.2 Les tests unitaires 60](#_Toc504071533)

[8.2.3 Conclusion 62](#_Toc504071534)

[9 Conclusions 65](#_Toc504071535)

[10 Annexes 66](#_Toc504071536)

[10.1 Annexe 1 : Arborescence des vues 66](#_Toc504071537)

[10.2 Annexe 2 : Architecture globale 67](#_Toc504071538)

[10.3 Annexe 3 : Architecture de l’application 68](#_Toc504071539)

[10.4 Annexe 4 : POSTMAN, test de la page d’accueil 69](#_Toc504071540)

# Introduction

Ce document est mon rapport de projet de VAE pour le titre de Master informatique parcours Ingénierie des Systèmes Informatiques Distribués.

Mon projet consiste en la création d’une application informatique comme décrit dans le chapitre du cahier des charges.

Les chapitres décrits dans mon rapport vont nous amener à comprendre cette application de part :

* ses fonctionnalités
* son architecture logicielle
* son architecture de base de données
* sa sécurité
* la qualité de son code, des versions et des recettes

Pour chaque chapitre une conclusion donne un aperçu des autres outils existant sur le marché, des difficultés rencontrées ainsi que du temps passé pour réaliser chaque étape de ce projet.

Les concepts vus dans ce projet sont pour moi entièrement nouveaux, car ils ne répondent pas à ce jour aux standards utilisés dans ma société.

En fin de rapports vous trouverez la conclusion du projet ainsi que les différentes annexes possibles pouvant être poursuivi.

# Le cahier des charges

L’application informatique devra répondre aux exigences suivantes : Modèle, Vue et Contrôleur.

Le langage de programmation sera de type JEE, mais pourra être également en C#, C++ ou DotNet suivant ce qui est utilisé dans l’entreprise mais devra dans tous les cas répondre à l’architecture MVC. L’utilisation des API Rest (get, delete, post) est demandée.

La base de données SQL devra être de type Data Warehouse et structurée comme telle sur le serveur. Le client aura une connexion web au serveur où il faudra gérer la sécurité de la connexion.

Le schéma de l’application sera le suivant :



La gestion du suivi des versions se fera avec l’un des outils suivants : git/sun, github ou bitbucket.

Au niveau du rapport et des attendus pour la soutenance orale, les points suivants seront demandés :

* Une explication des fonctionnalités de l’application
* Une architecture du logiciel au format MVC
* Une architecture de la base de données ainsi que sa structure
* Quels sont les points clés au niveau sécurité informatique qui ont été mis en œuvres ou qui auraient pu être mis en places.
* La qualité du code avec la gestion des versions et des fiches de recettes (Test applicatif)

Le choix du sujet de l’application à développer est laissé à l’étudiant.

Il s’agira d’une application permettant de consulter les fiches des bobines d’acier et de suivre leurs manutentions dans les halles du site.

# Fonctionnalités

## Introduction

L’application permet aux utilisateurs de consulter des fiches de bobines à partir d’une interface web, de rapports BI et d’une application sur mobile.

L’application serveur permet de consulter ces informations à partir de pages web. Il est possible de modifier ces fiches produit grâce à divers actions : création, modification et suppressions.

Ces fiches contiennent les informations relatives à leur historique et leurs localisations dans les halles. L’utilisateur a la possibilité de faire des recherches des bobines soit par numéro à partir d’une liste déroulante et ainsi effectuer les actions de création, d’édition et de suppression du produit.

Pour pouvoir accéder aux différentes vues, l’utilisateur doit s’identifier via une interface. Ces données sont sauvegardées en base de données avec un cryptage de son mot de passe.

Une application mobile permet de consulter également ces informations par smartphone afin de pouvoir faire une vérification des produits directement dans les halles.

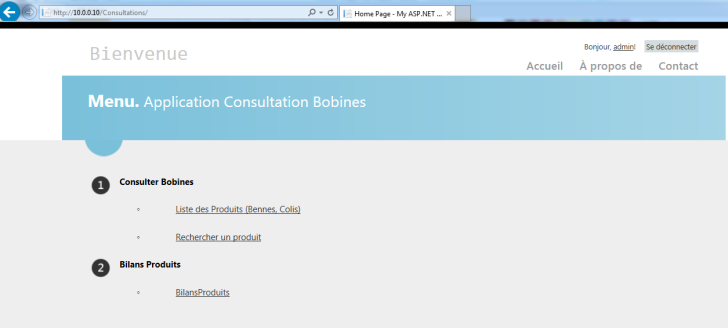
## L’application WEB

L’arborescence complète de l’application web est décrite en annexe 1.

### La page d’accueil

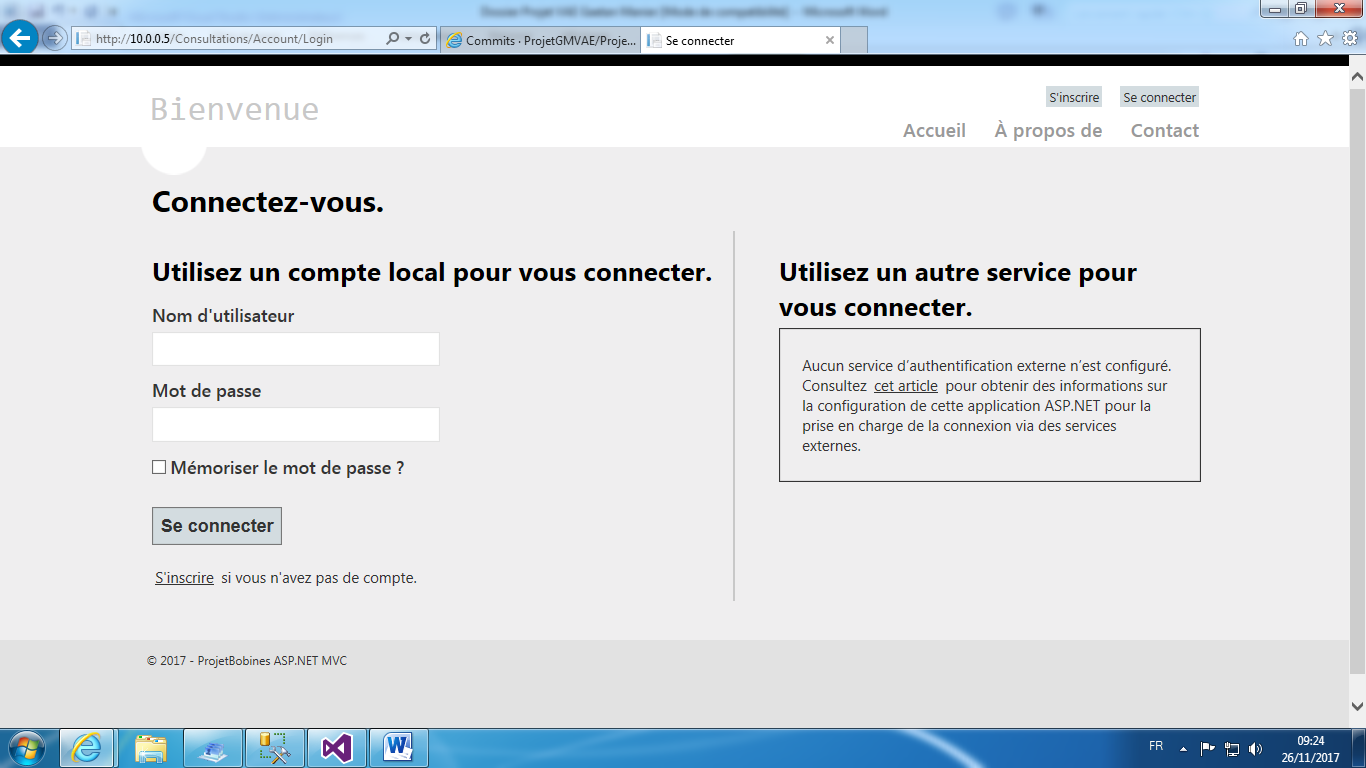
Cette page s’ouvre lors de l’arrivée de l’utilisateur sur le site web (ex : http://10.0.0.6/Consultations/). Elle permet d’accéder aux différentes fonctionnalités de l’application.

* consultation des produits
* bilans des produits
* gestion des accès
* avoir des infos divers



### La gestion des accès utilisateur

Lors de son arrivée sur la page d’accueil, l’utilisateur a la possibilité de s’inscrire ou de créer un nouveau compte.

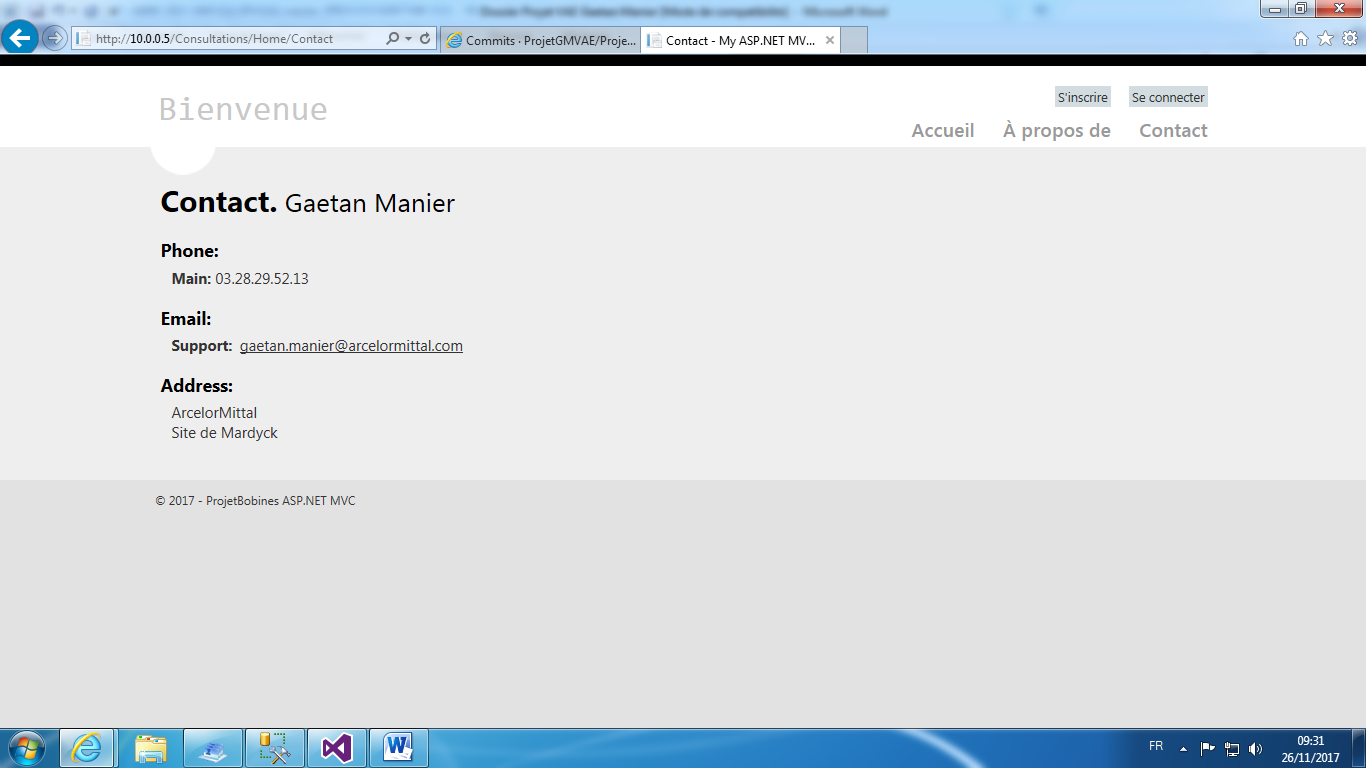
 

Lors de la création d’un nouveau compte, l’utilisateur doit envoyer un mail à l’administrateur du site afin de définir le rôle de son compte. L’accès aux différentes vues et aux fonctionnalités (création, suppression, mise à jour) se fait grâce à l’attribution de rôles sur les comptes.

Ces rôles sont les suivants :

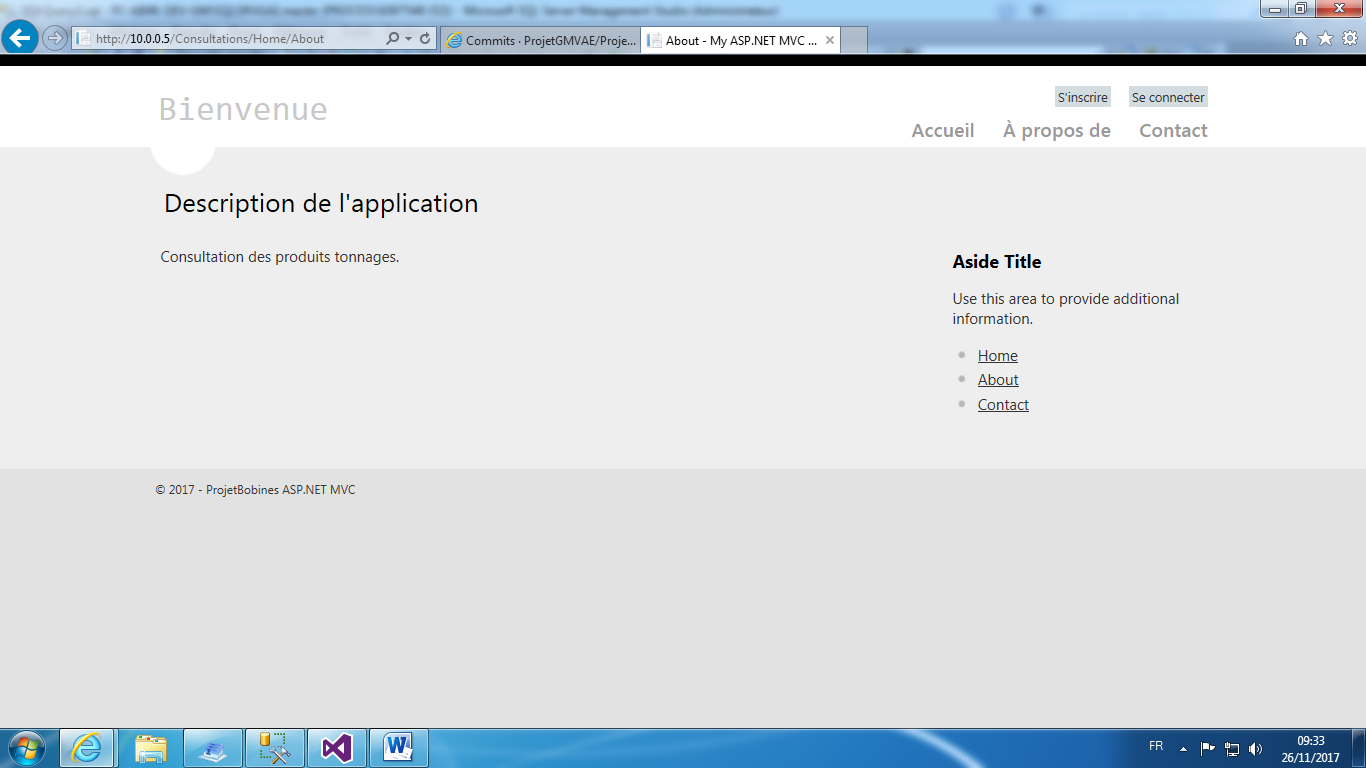
* Admin : il permet d’avoir accès à l’ensemble des fonctionnalités du site
* Modificateur : il permet d’accès aux fonctionnalités de mise à jour des informations.
* Visiteur : il permet d’accéder uniquement en mode visualisation des vues

### La page « Contact »



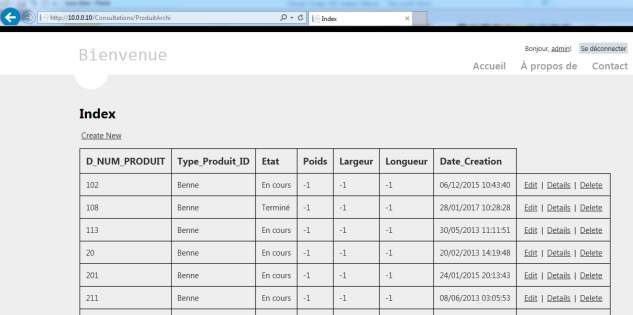
Cette page permet d’avoir les coordonnées de l’administrateur du site afin de faire les demandes d’ajout de rôle sur le compte qui est créé.

### La page « à propos de »



Cette page permet d’avoir un aperçu des fonctionnalités du site, de connaitre la version en cours.

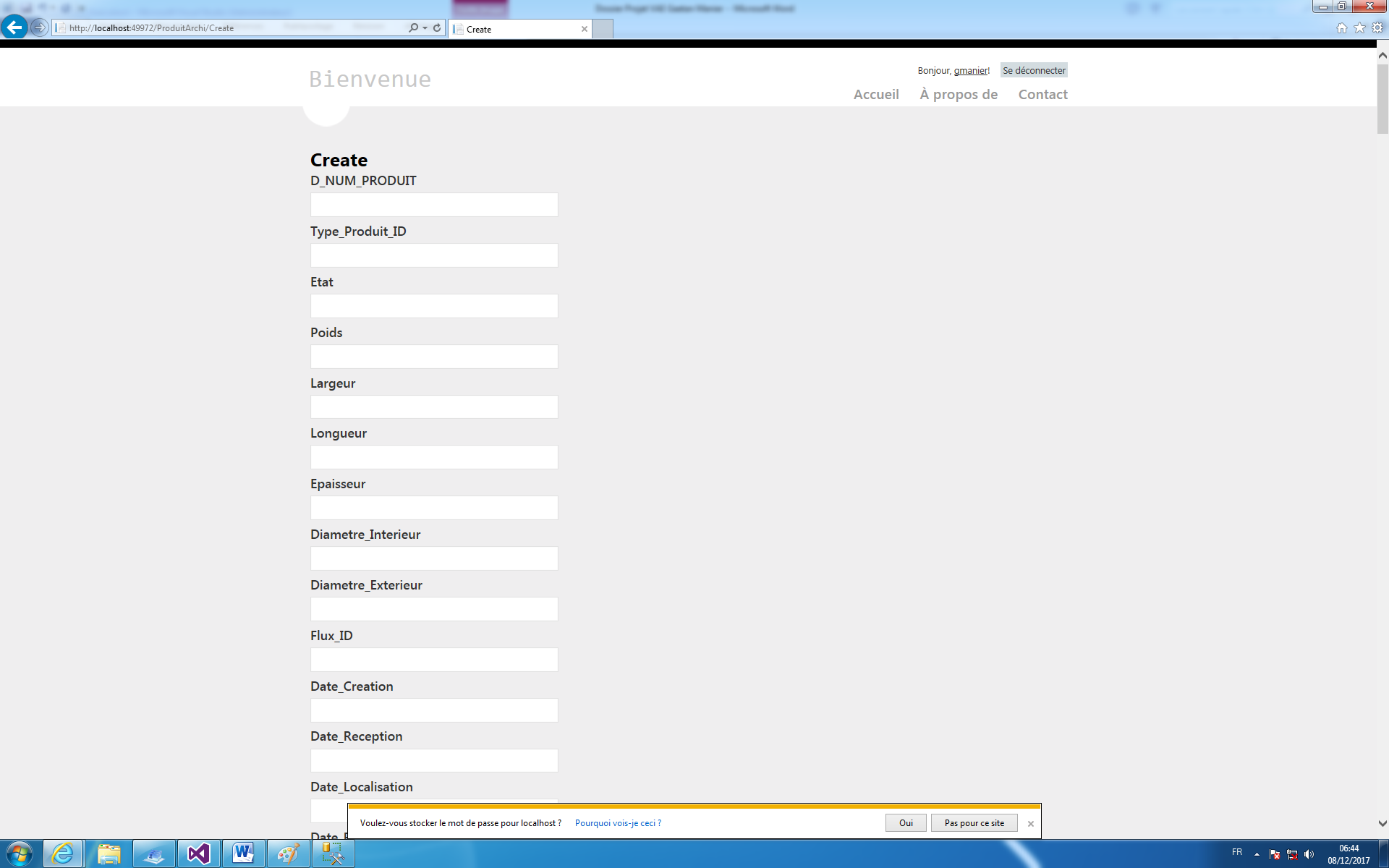
### La page « Liste des produits»



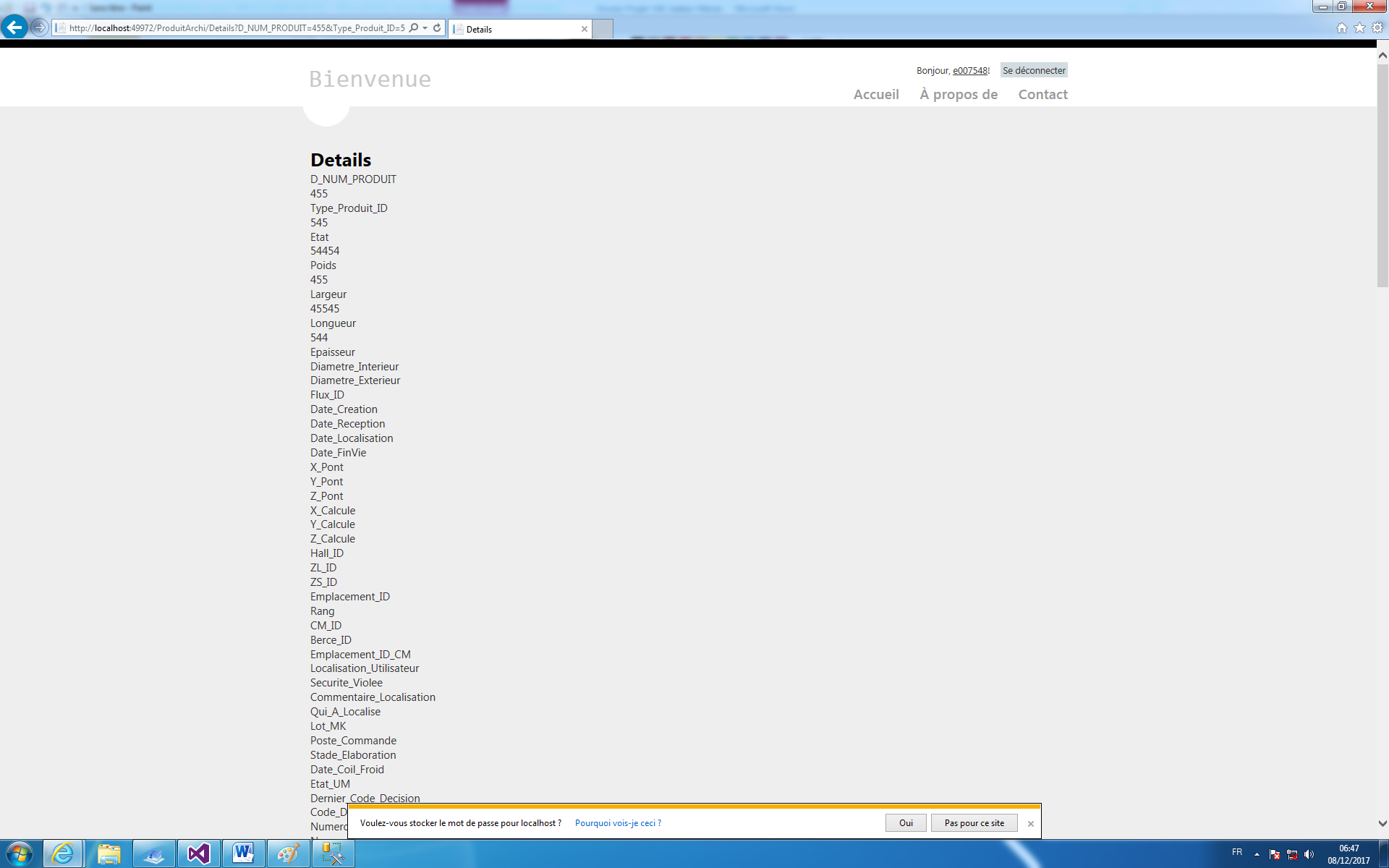
Cette page permet de visualiser les données des produits (bennes et colis) en base de données.

Elle se présente sous la forme d’une liste de produits présents dans la table « produits » sur laquelle l’utilisateur (suivant le rôle qui lui est attribué) peut :

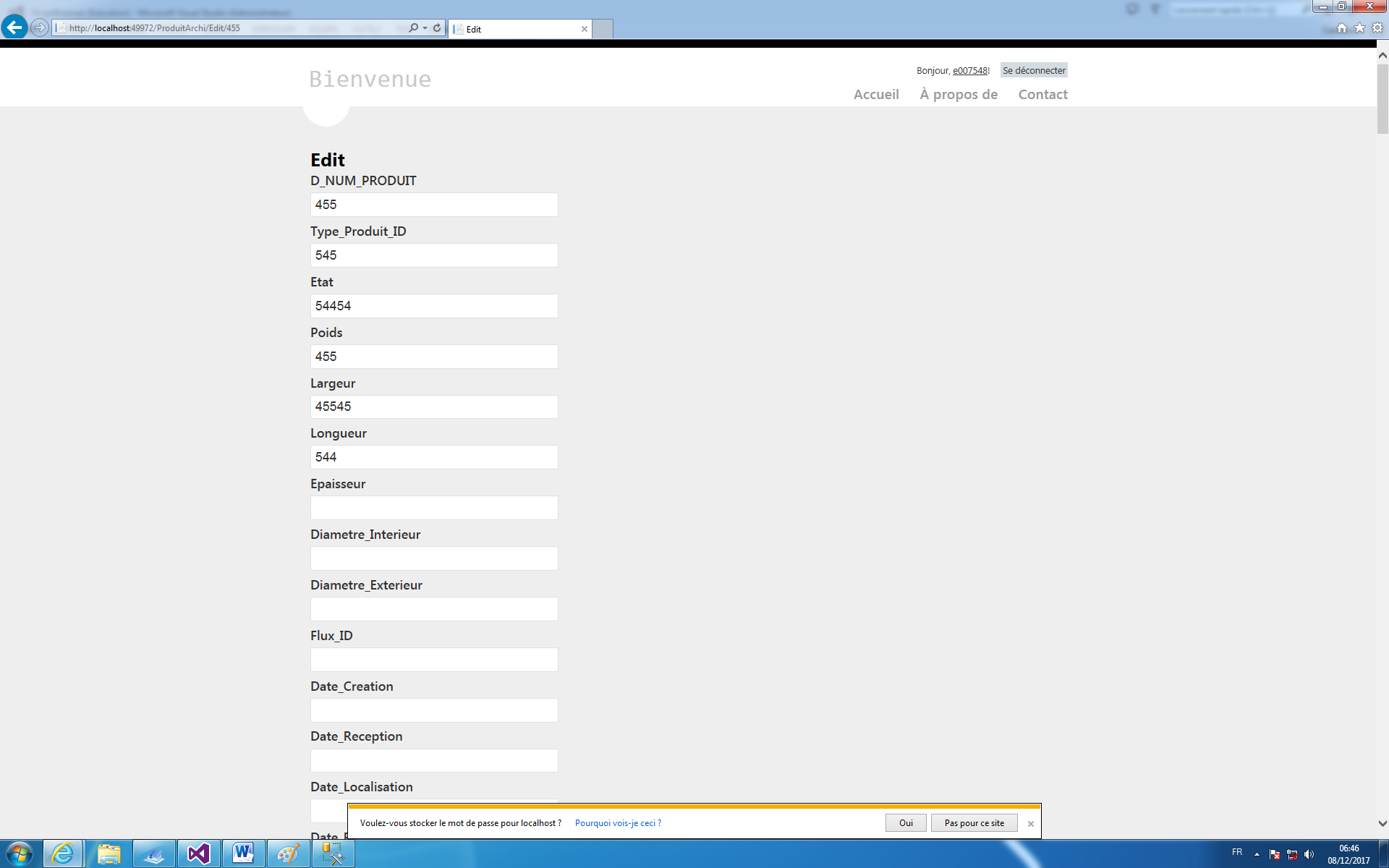
* Créer un nouveau produit



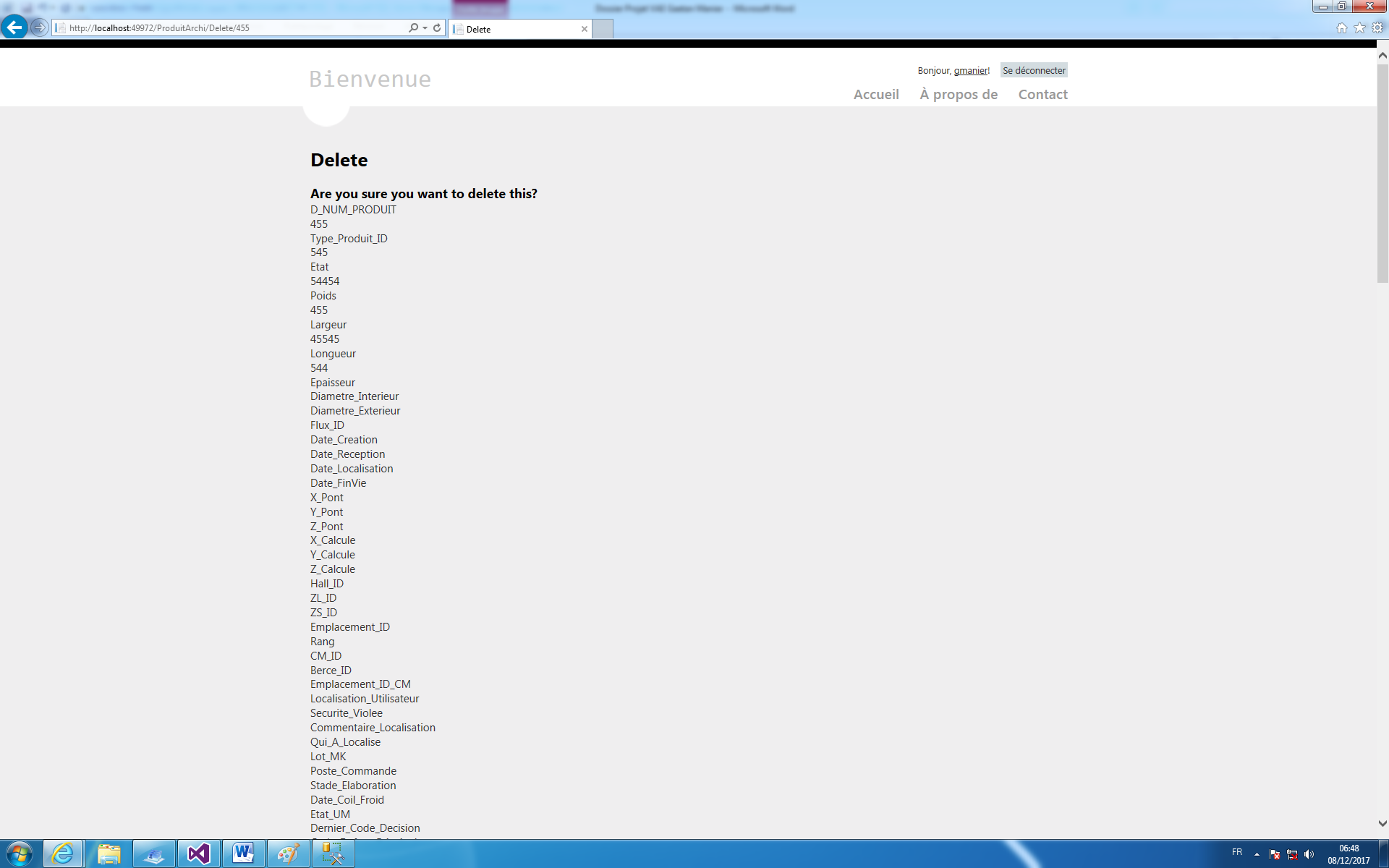
* Consulter le détail d’un produit



* Modifier un produit



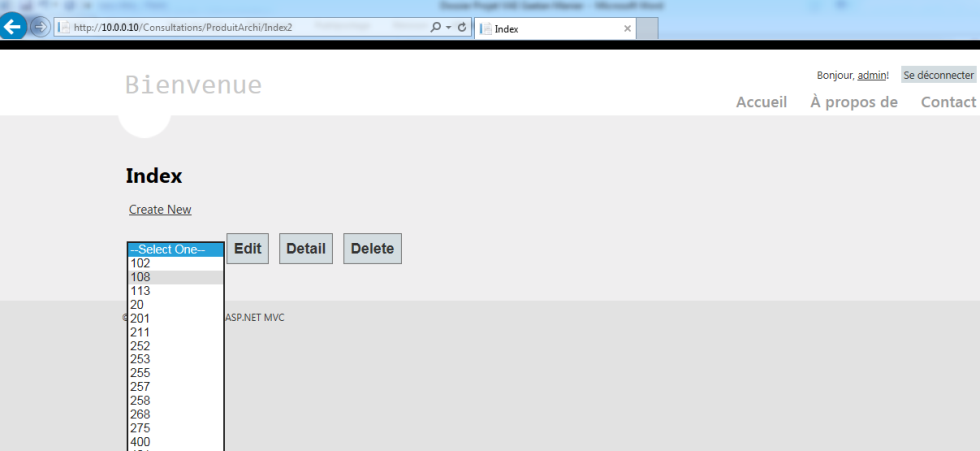
* Supprimer un produit



Ces données font parties de la table produits qui sert à l’alimentation du Data Warehouse par un ETL (décrit chapitre 6).

### La page « Recherche d’un produit»

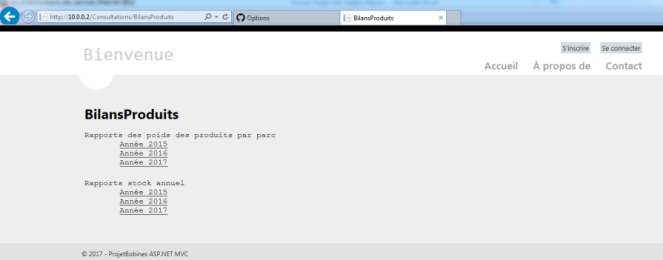
Cette page permet après avoir sélectionné un numéro de produit dans la liste déroulante d’effectuer les mêmes opérations que dans la page « Liste des produits ». C’est à dire la modification, la visualisation et la suppression d’un produit.



Ceux sont d’ailleurs les mêmes écrans et les mêmes méthodes qui sont appelées. Ceci grâce au découpage de l’application au format MVC qui permet la réutilisabilité de ces éléments.

### La page « Bilans produits»

Cette page permet de consulter différents bilans bobines basés sur des rapports créés en automatiques et se basant sur les cubes du Data Warehouse.

## L’application mobile

Cette application propose à l’utilisateur les fonctionnalités de visualisation de la liste des produits et la consultation d’un produit après saisi son numéro.

TODO.

# Construction du projet

## Choix des outils

Afin de répondre au cahier des charges et à l’architecture logicielle demandée, j’ai dû faire différents choix.

D’abord pour l’application, j’avais le choix entre le langage de programmation Java et le c#. C’est deux types de langage permettent de développer en respectant l’architecture MVC (modèle Vue Contrôleur). J’ai donc choisi le c# par commodité car c’est l’environnement de développement informatique que j’utilise au quotidien avec Visual Studio 2013. Je n’avais donc pas de nouveau langage informatique à apprendre hormis les concepts imposés par l’architecture MVC et les applications web.

Visual Studio 2013 permet également de développer des applications Web API et des applications mobiles utilisant les API Rest (get, delete, post …).

Pour rappel, voici les principaux verbes HTTP que nous utilisons pour des architectures REST :

* GET : Accède à une ressource
* HEAD : Accède aux métadonnées d’une ressource
* POST : Ajoute une ressource
* PUT : Met à jour une ressource complète en la remplaçant par une nouvelle version (99% des cas).
* PATCH : Met à jour une partie d’une ressource en envoyant un différentiel (une sorte de ‘git diff’)
* DELETE : Supprime une ressource

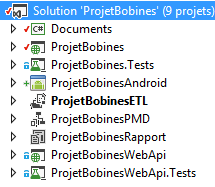
Pour la base de données j’ai choisi SQL Serveur 2012 car c’est l’environnement de base de données qui est utilisé dans mon entreprise. De plus l’ensemble des fonctionnalités BI (Business Intelligence) est natif avec cette base de données qui sera utile pour traiter la partie Data Warehouse et alimenter le cube.

Pour la partie web, j’utilise les fonctions IIE pour l’hébergement de mon application Web moyennant l’ajout des fonctions ASP.Net.

Enfin pour la gestion de l’archivage et l’historisation des fichiers sources au cours du développement, j’ai choisi GitHub parmi la multitude d’applications existantes qui s’interface directement avec Visual Studio 2013. Cela m’a permis de retrouver la souplesse de ce que je connais avec TFS (Team Foundation Services) au quotidien.

Bien sur d’autres outils existent sur le marché et l’architecture logicielle aurai été différentes. Je reviendrais sur ces différents points dans les conclusions des différents chapitres de ce rapport.

## Découpage du projet sous VS2013



La solution est constituée de plusieurs projets distincts se trouvant dans la solution « ProjetBobines » sous Visual Studio.

Le projet « Documents » contient les documents comme ce rapport, un exemple de fichier Excel mappé sur le cube du projet

Le projet « ProjetBobines » contient l’ensemble des sources de l’application web au format MVC.

Le projet « ProjetBobines.Tests » contient l’ensemble des sources de l’application web au format MVC pour la gestion des tests et des recettes de l’application.

Le projet « ProjetBobinesETL» contient l’ensemble des fichiers de l’application Integration Services permettant la gestion des flux et l’intégration de données hautes perfomances. L’extraction, la transformation et l’entreposage (ETL) des données pour le Data Warehouse est décrits dans ce projet.

Le projet « ProjetBobinesPMD» contient l’ensemble des fichiers de l’application Analysis Services permettant la création des modèles multidimensionnels et d’exploration de données pour le cube de l’application.

Le projet « ProjetBobinesRapport» contient l’ensemble des fichiers de l’application Report Server pour la création des rapports utilisant le cube de l’application.

Le projet « ProjetBobinesWebApi» contient l’ensemble des fichiers de l’application Web Api au format MVC permettant de communiquer avec l’application mobile.

Le projet « ProjetBobinesAndroid» contient l’ensemble des fichiers de l’application Mobile.

# Architecture du logiciel

Le projet concerne l’application web (repère en A sur le schéma de l’annexe 2 page TODO) qui respecte le standard MVC (Modèle Vue Contrôleur). Elle permet à l’utilisateur de consulter les données via une page web qui a pour effet d’envoyer une requête à une routine qui interroge le contrôleur. Celui par l’intermédiaire du modèle récupère les données pour les envoyer à la vue et ensuite les afficher sur la page web. L’accès à la base de données du modèle se fait en utilisant Entity Framework qui est un ensemble de technologie ADO.NET.

Le second projet concerne la base de données (repère en B sur le schéma de l’annexe 2 page TODO) devant être un Data Warehouse afin de répondre au cahier des charges. Elle possède une table « produits » dans laquelle on retrouve de façon brute les données de bobines provenant d’une base de données classique de type SGBDR (systèmes de gestion de base de données relationnel).

Afin de créer le Data Warehouse, la table produit a ensuite été décortiquée afin d’être sur un schéma de base de données de type OLAP (OnLine Analytical Processing). Une table de fait et plusieurs tables de dimension ont ainsi été créées sur le principe d’un schéma en étoile.

Ensuite un ETL (Extarct Transform Load) a été implémenté en utilisant SSIS (SQL Server Integration Services) afin de récupérer et de transformer les données de la table produit pour alimenter le Data Warehouse.

Un cube OLAP (Online Analytical Processing) a été créé en utilisant SSAS (SQL Server Analysis Services).

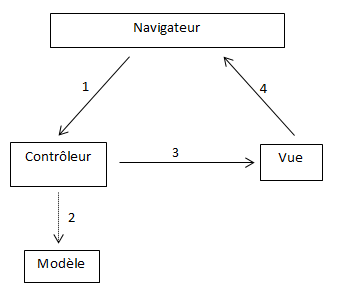
Enfin la création et l’édition de rapports et de bilans est rendu possible par l’utilisation de SSRS (SQL Server Reporting Services). Les rapports se basent sur le cube précédemment créé et sont déployés sur un site web. Les rapports sont consultables depuis l’application web en utilisant les raccourcis adéquats.

Le troisième projet concerne l’application mobile permettant à l’utilisateur de consulter les données des produits sur son téléphone. Cela s’appuie sur une application mobile qui va s’appuyer sur une application WebAPI coté serveur et une application Android …..TODO

## Architecture de l’application web

L’architecture de mon application web est basée sur le Modèle Vue Contrôleur (MVC) qui est un patron d’architecture logicielle pour implémenter des interfaces utilisateurs et des applications web par exemple. Le découpage de l’application sous Visual Studio se trouve en annexe 3 page TODO .

Elle est composée de trois types de modules ayant trois responsabilités différentes : les modèles, les vues et les contrôleurs.



Le principe de fonctionnement est le suivant :

1. Action utilisateur via une requête http dans le navigateur
2. Consultation et/ou mise à jour du modèle (facultatif)
3. Le contrôleur décide de la vue à afficher
4. La vue renvoi le HTML au navigateur

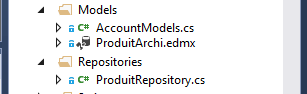
Après le descriptif de ces différents modules nous verrons un exemple de fonction de l’application et des imbrications de ces différents modules.

### Les modèles

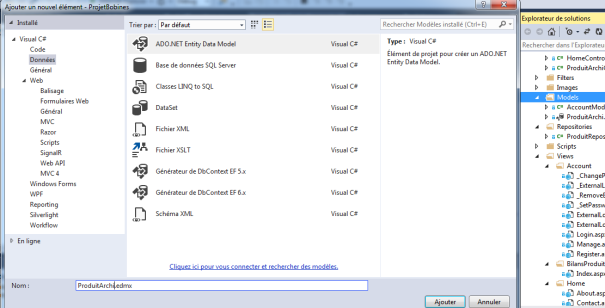
Les modèles permettent à l’application web d’accéder aux données et d’effectuer les actions de validation, de lecture et d’enregistrement. Dans mon projet j’utilise en plus Entity Framework qui est un ORM (Object Relational Mapping) pour faciliter l’accès aux données. Il s’agit d’une technique de programmation informatique qui permet d’établir des correspondances entre une base de données relationnelle et un modèle objet de classes.

Dans mon projet il y a 2 modèles :

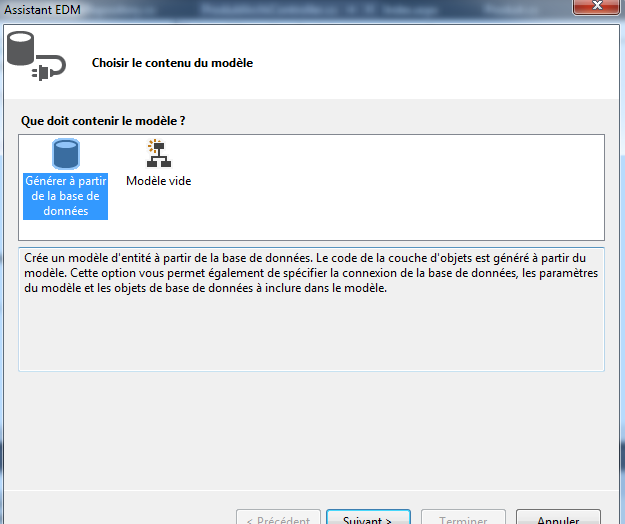
* Le « AccountModel » qui a été généré automatiquement lors de la création du projet et qui permet d’accéder aux différentes tables de gestion des utilisateurs et des droits.



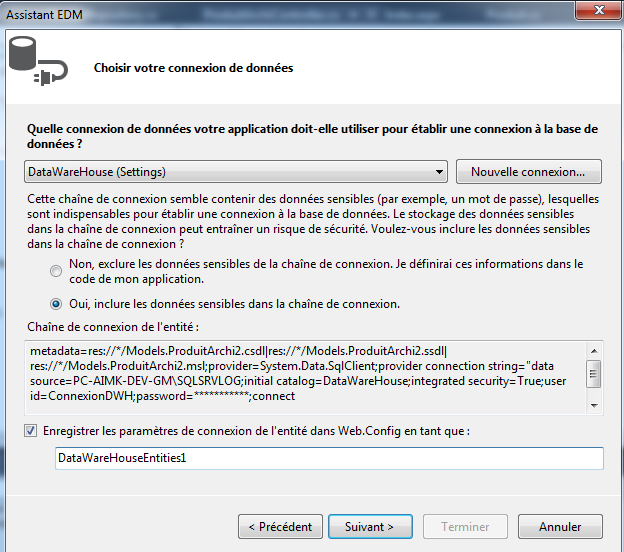
* Le « ProduitArchi.edmx » correspond à la couche d’accès aux données de la table « produits.



Pour le créer il faut d’abord ajouter un nouvel élément et sélectionner dans la partie « données » l’objet ADO.NET Entity Data Model puis le nommer.

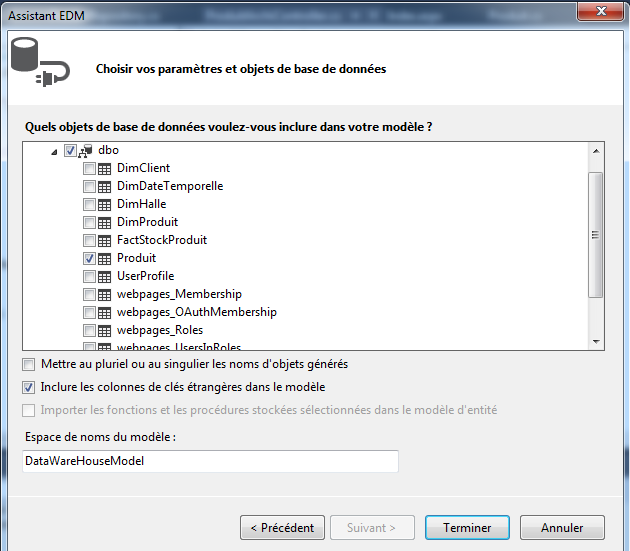


Puis choisir la création à partir de la base de données.



Il faut définir la configuration au serveur de base de données et cocher « oui » pour inclure les données sensibles à la connexion. Cela a pour effet de crypter le mot de passe.

Il est important d’enregistrer les paramètres de connexion dans le fichier web.app afin de les avoir lors du déploiement de l’application



Il faut ensuite sélectionner la ou les tables à mettre dans la vue. Le plus propre étant de créer une vue par table.

L’espace de nom est à spécifier également.

Les informations de ce modèle seront ensuite utilisées lors de la création des vues et des contrôleurs.

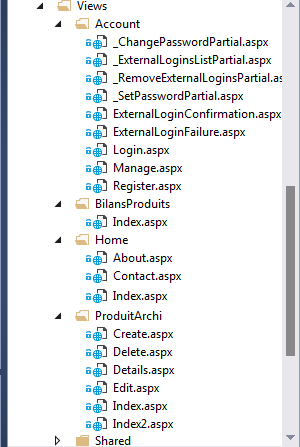
Afin de faciliter l’utilisation de ces méthodes, j’ai rajouté ensuite un dossier « repositories » qui permet d’encapsuler ces méthodes dans un pattern «ProduitRepository.cs ». Son but principal est d'isoler la couche d'accès aux données de la couche métier. Il expose diverses méthodes s'appuyant sur un modèle CRUD (Create, Read, Update, Delete). Ce modèle fournis une centralisation de transaction aux données.

Cela présente plusieurs intérêts :

* Cela fournis un code de meilleure qualité.
* Les méthodes de lecture, de mise à jour et de suppression de données sont accessibles sans forcément connaitre l’utilisation du modèle et les commandes SQL.
* Si au lieu d’avoir une base de données SQL Server j’utilise un fichier Excel ou une base de données Oracle, je n’aurais que le modèle à modifier et le fichier de « repositories » à modifier sans devoir refaire une passe complète sur l’application.
* Cela permet d’effectuer les tests unitaires en passant par des objets nommés "mock" à notre repository.

Enfin pour améliorer le code et limiter les dépendances entre les classes, j’ai créé une interface à la classe «ProduitRepository.cs ».

### Les vues



C’est la partie visible d'une [interface graphique](https://fr.wikipedia.org/wiki/Interface_graphique). La vue se sert du modèle, et peut être un diagramme, un formulaire, des boutons, etc.

Une vue contient des éléments visuels ainsi que la logique nécessaire pour afficher les données provenant du modèle. Dans une application web une vue contient des balises [HTML](https://fr.wikipedia.org/wiki/Hypertext_Markup_Language).

Pour mon projet les vue « ProduitArchi » ont été créées automatiquement lors de la création du contrôleur.

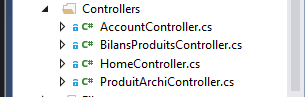
Le fichier Index2.aspx est une recopie de Index.aspx pour la recherche de produit par liste déroulante.

### Les contrôleurs

Le contrôleur gère les interactions avec l’utilisateur et détermine quels traitements doivent être effectués pour une action donnée. D’une manière générale, il va utiliser les données du modèle, les traiter en fonction de l’action de l’utilisateur, et les envoyer à la vue afin qu’elle les affiche.

Le contrôleur est au cœur du système, il interprète et « contrôle » les données venant de l’utilisateur, comme des données venant d’un formulaire ou bien simplement une action faite via une URL.

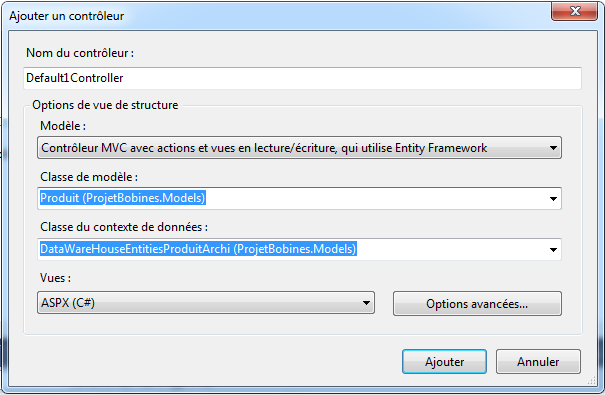
Une bonne pratique est de faire en sorte que le contrôleur fasse le moins de choses possible. Il doit traiter l’action et choisir la bonne vue.



Dans mon application il y a 4 contrôleurs dont 2 générés automatiquement lors de la création du projet sous Visual Studio : accountcontroller (gestion des comptes) et homecontroller (page d’accueil).

Les contrôleurs BilansProduitController (gestion des bilans) et ProduitArchiController (consultation et actions sur les produits) ont été ajoutés au projet en utilisant l’assistant de création des contrôleurs.

Lors de l’ajout d’un contrôleur il faut spécifier son nom et les options de vue de structure. Cela consiste à sélectionner un modèle qui va générer automatiquement ou non les vues et les associer au modèle choisi.

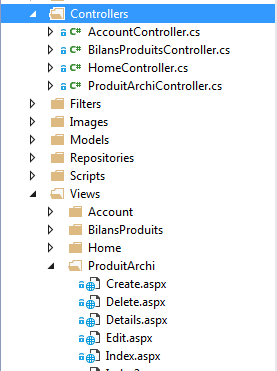


Dans le cas du contrôleur pour les produits j’ai donc opté pour la création des vues actions les actions création, consultation et suppression utilisant Entity FrameWork en se basant sur le modèle précédemment créé sur la table « produits ».

Pour le langage utilisé pour les vues j’ai opté pour l’ASPX (c#).

L’assistant de création de contrôleur de Visual Studio génère automatiquement le dossier et l’ensemble des vues nécessaires.

* Index : pour la page principale ou l’on retrouve la liste des produits
* Create : pour la création de produits
* Delete : pour la suppression de produits
* Détails : pour la consultation de la fiche produit
* Edit : pour la modification de produits

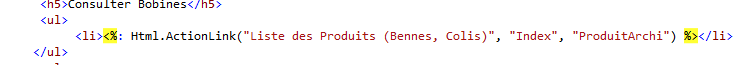


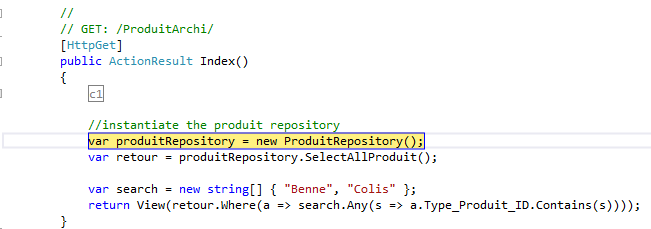
Dans le contrôleur « ProduitArchiController » le code des différentes méthodes est généré également automatiquement. On retrouve les méthodes avec les attributs Get et Post mais également les attributs de sécurité pour lutter contre les contres façons de requêtes.

### Principe de fonctionnement

Afin de mieux comprend le fonctionnement des intéractions dans MVC, je vais maintenant décrire ce qui se passe lorsque l’on ouvre la page « liste des produits » de l’application et que l’on souhaite éditer un produit pour le modifier.

L’utilisateur arrive sur la page d’accueil lors de son accès sur le site, puis il clique sur le lien qui est codé dans la vue Index.aspx du dossier Home:

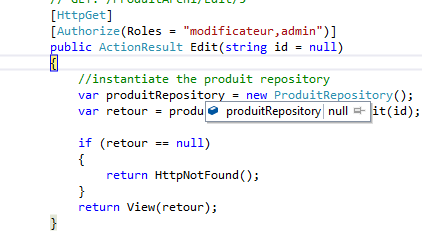
Ce qui a pour conséquence d’appeler la méthode Index.aspx du contrôleur « ProduitArchi ». Elle a pour but de sélectionner les données « produits » en utilisant le repositories et de filtrer sur les bennes et les colis, puis de les renvoyer à la vue portant le nom index.aspx présent dans le répertoire ArchiProduit.



La liste des produits apparaît donc alors.

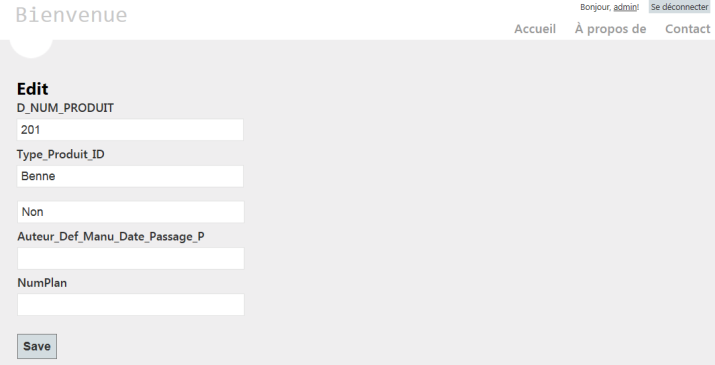


Pour éditer un produit afin de le modifier, le clic sur le lien appelle la ligne de commande du fichier index.aspx. Cette commande ouvre le fichier Edit.aspx du répertoire ArchiProduit, appel la méthode Edit du contrôleur produitArchiController avec l’attribut HttpGet en passant en argument le numéro de produit qui est l’index de la table « produits ».

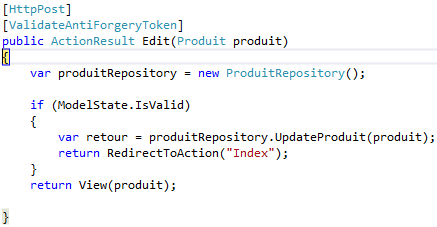


Une vérification des rôles de l’utilisateur est effectuée avec l’attribut « Authorize » qui ouvre automatiquement la page de saisie du login et du mot de passe si nécessaire.

La page d’édition du produit sélectionné s’ouvre alors :

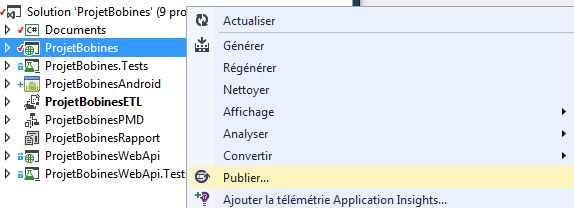


Après avoir modifié des données dans la page d’édition, l’appui du le bouton  appelle la commande. Cela a pour effet d’appeler la méthode la méthode Edit du contrôleur produitArchiController avec l’attribut HttpPost.

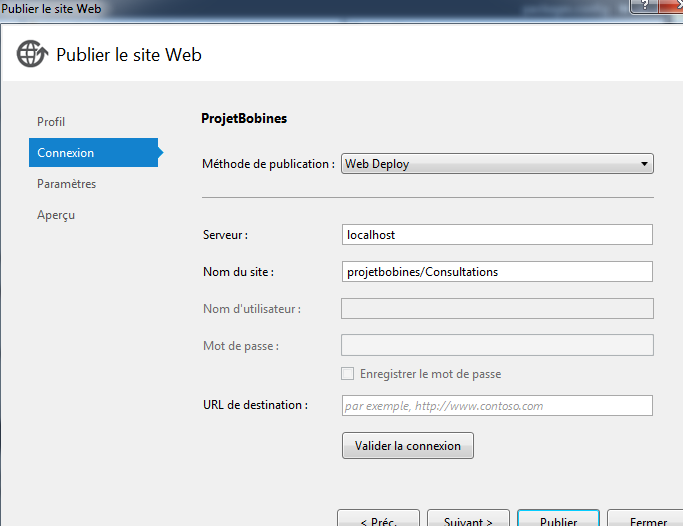


Une fois la vérification effectuée (par modelState.isvalid), l’ensemble des données du produit modifié sont mises à jour en utilisant le repository. En retour de cette mise à jour, on ouvre de nouveau la liste des produits. Si la vérification s’est mal passée, laisse la page d’édition ouverte avec les données du produit sélectionné.

### Publication de l’application



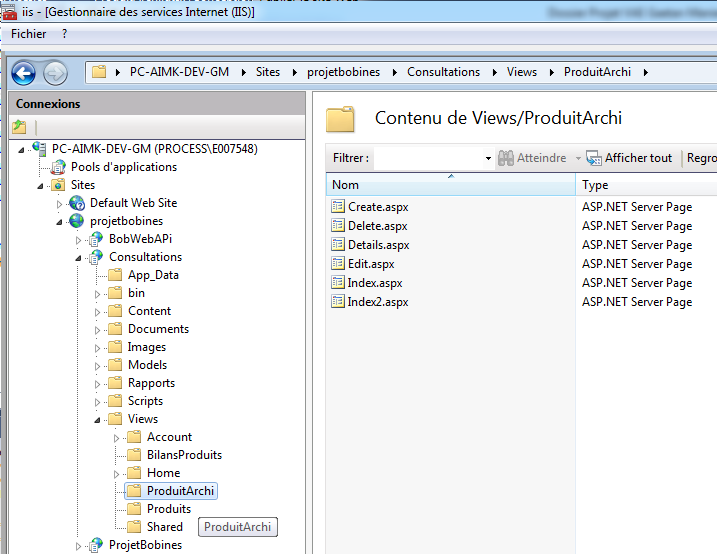
La publication de l’application se fait simplement par un clic droit sur le projet « projetbobines ».



Cela a pour effet d’ouvrir l’assistant de publication de l’application où l’on repli les informations de connexion et de renseigner le nom du site qui a été créé précédemment lors de la configuration IIS.



Ensuite il faut compléter les informations liées aux connexions de base de données.



Dans IIS, nous retrouvons bien l’application web déployée ainsi que les différentes vues par exemple.

### Conclusion

La création de cette application m’a permis de comprendre les mécanismes de fonctionnement d’une application web et de l’architecture MVC. La conception d’application informatique en partant d’un feuille blanche n’a pas été simple pour moi car en temps normal nos applications informatiques sont issues d’un Framework fournit par nos équipes supports de développement.

Il est bien sur possible d’aller plus loin dans mon application :

* améliorer l’ergonomie des IHM
* ajouter des fonctionnalités de création de produit ou de mise à jour des tables de dimensions de notre schéma en étoile du Data Warehouse.
* ajouter de nouveaux rapports consultables

Cette partie du projet a représenté environ 100 heures de travail. Ceci en comptant le temps passé à me documenter sur internet afin d’appréhender le sujet et les multiples essais de création d'une application qui tenait la route et qui respectée le cahier des charges.

## Architecture de l’application mobile

Mon application mobile se base sur une application Web Api pour la partie serveur et une application Android pour la partie mobile.

### Introduction

L’application Web Api doit respecter le standard MVC ainsi que les concepts REST.

REST (pour REpresentational State Transfer) ne décrit pas une implémentation mais un style d’architecture qui suit différents principes :

* 1/ client-serveur: les responsabilités sont séparées entre le client et le serveur. L’interface utilisateur est séparée de celle du stockage des données. Cela permet aux deux d’évoluer indépendamment (contrairement à des applications qui réalisent des requêtes directement en base de données par exemple…).
* 2/ sans état: la requête envoyée par le client doit être auto-suffisante : ne pas nécessiter de sauvegarder un état sur le serveur
* 3/ mise en cache : la possibilité pour le serveur de spécifier les réponses pouvant être mises en cache
* 4/ une interface uniforme : chaque ressource est identifiée unitairement, les ressources ont des représentations définies
* 5/ un système hiérarchisé par couche: les états de l’application sont représentés par des ressources individuelles. L’ensemble de l’information n’est pas envoyée dans une ressource unique.

À noter qu’on assimile **«**API REST**»** avec l’application de ce style d’architecture sur le protocole HTTP et en utilisant le format d’échange de données JSON.

Quels avantages trouve-ton dans l’utilisation d’API REST ?

* Un protocole REST permet d’être simplement accédé, aussi bien par un autre élément du SI, que par une application cliente. On parle alors d’architecture WOA (Web Oriented Architecture), qui est un sous-ensemble des SOA. il n’y a pas de « préjugé » côté client.
* Avec son approche orientée ressources (au lieu de traitement), et le fait que les serveurs sont sans état, la tenue de la charge est facilitée par la possibilité de passer à l’échelle une partie du SI qui serait particulièrement sollicitée.

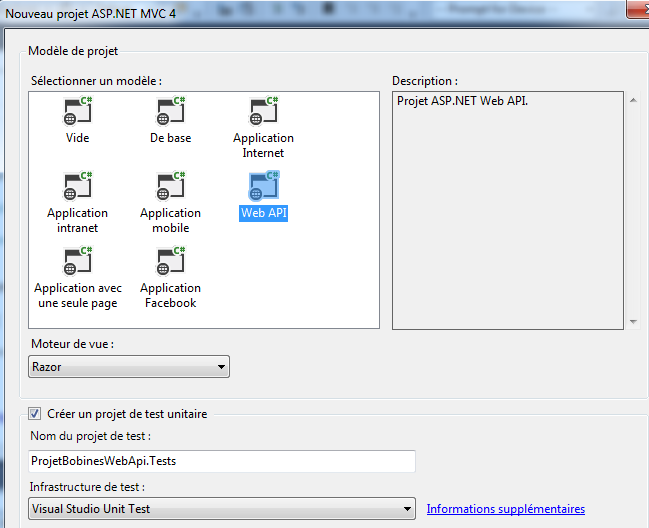
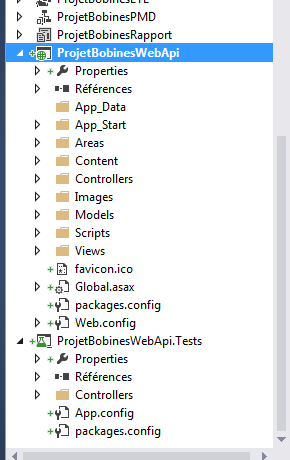
Le protocole HTTP est un protocole déconnecté, extrêmement maîtrisé par tous les administrateurs, les développeurs…. Les outils de protection (firewalls), de tenue de la qualité de service, ainsi que les librairies de développement d’API existent et sont omniprésents dans de nombreux environnements: la plupart des organisations utilisent aujourd’hui ces API REST pour permettre l’accès aux données de leurs différents systèmes.

### L’application Web API

Comme pour l’application Web vue précédemment elle se base sur l’architecture MVC (Modèle Vue Contrôleur).

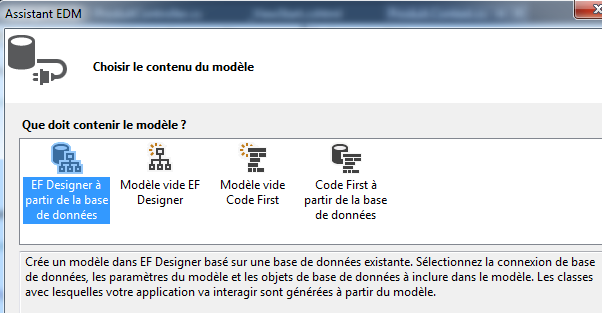
Pour générer cette application dans Visual Studio 2013, j’ai donc créé un nouveau projet avec l’assistant comme pour le projet de l’application web. Mais cette fois ci, j’ai sélectionné le modèle Web Api.

Les projets ProjetBobinesWebAPi et ProjetBobinesWebAPi.Tests sont donc créés dans la solution.

#### Les modèles

Ensuite comme pour l’application web, j’ai créé un modèle permettant le requettage des données de la table des produits et la connexion au serveur de base de données.



La procédure de création du modèle est la même que pour l’application web hormis le choix de EF designer afin de pouvoir sélectionner la table produit de la base de données.

Comme pour l’application Web, j’ai créé le fichier de repositories ainsi que les interfaces associées afin de décolérer la couche d’accès aux données du contrôleur.

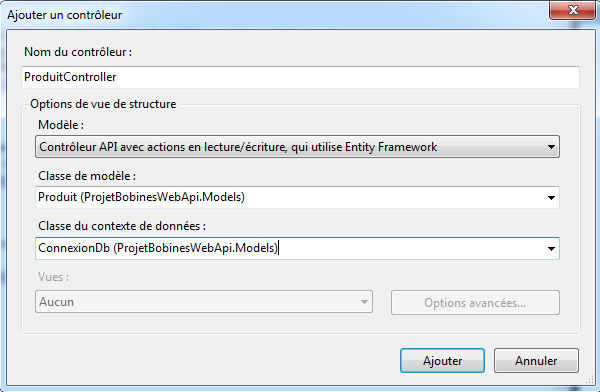
#### Les vues

Dans l’application web API il n’y aura pas de vues pour les IHM ….. ?

TODO.

#### Les contrôleurs

Comme pour l’application web, j’ai créé le contrôleur qui va permettre d’utiliser les informations de la vue.



Comme précédemment j’ai choisi le modèle pour lequel le contrôleur allait servir. Mais cette fois ci il a fallu sélectionner « contrôleur API avec action … ».

### L’application Android

TODO

### Conclusion

Le développement de l’application WebAPi m’a pris beaucoup moins de temps que l’application web elle-même, car j’avais déjà acquis les compétences techniques pour cette technologie. Le temps passé sur cette partie a été de 30 heures.

L’intérêt d’utiliser les WebAPi est que l’on fait une pierre 2 coups. Contrairement à l’application Web vue précédemment, ici on développement l’application Web mais on rend également accessible les méthodes pour les applications mobiles.

*Json.NET*

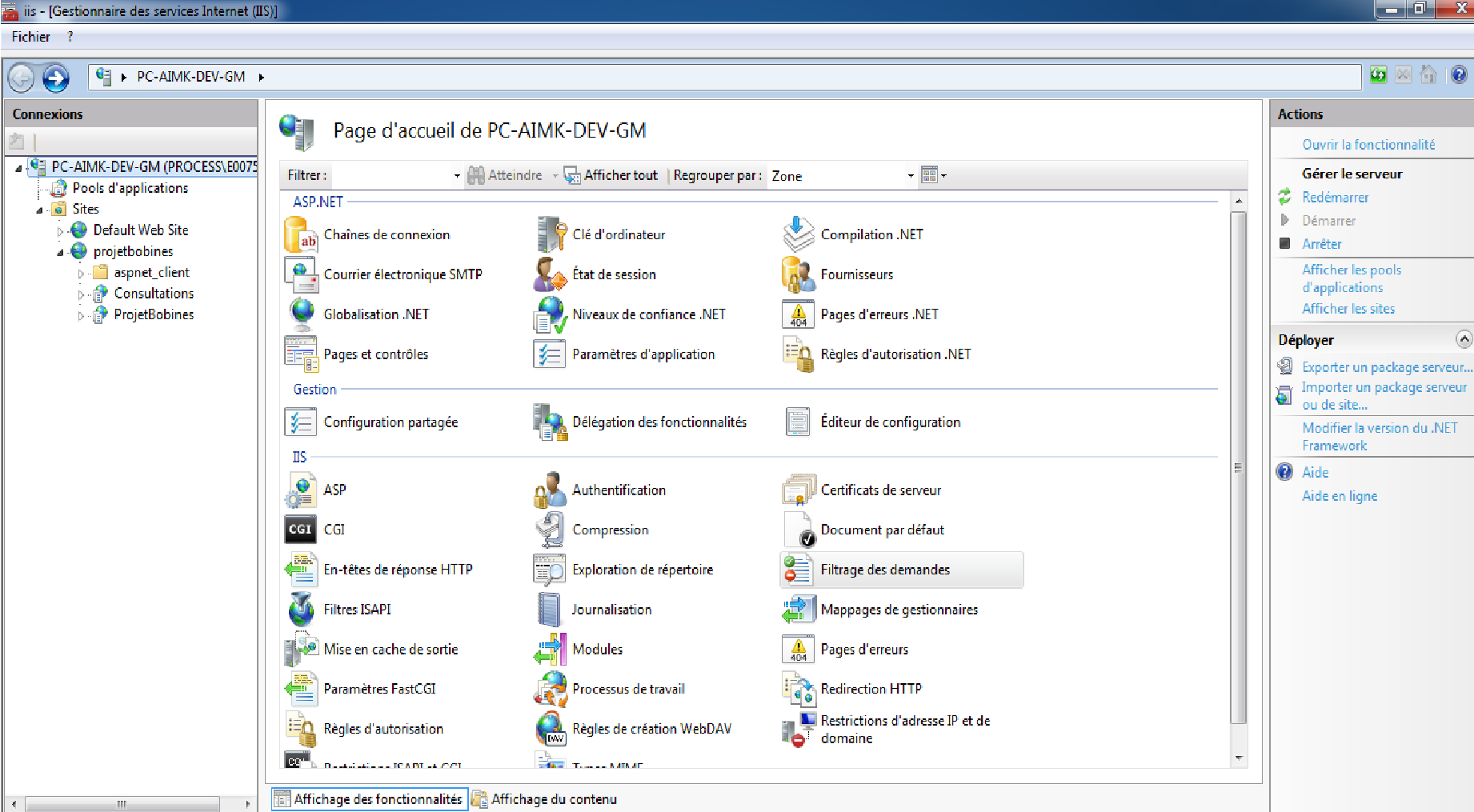
*Microsoft HTTP Client Libraries*

TODO

## Le serveur Web

Afin de pouvoir répondre au cahier des charge il m’a fallu créer un nouveau site web en utilisant IIS (Internet Information Services) sur mon PC. Une première pour moi car cela ne fait pas parti de la technologie que nous utilisons actuellement dans nos applications informatiques.

Après mettre documenté et effectué une série d’essais plus ou moins infructueux, j’ai créé le site web « projetbobines » qui contient les applications « ProjetBobines » et «Consultations ».



Lors de mes différents essais, j’ai été confronté à des problèmes liés au déploiement de mon application C# vers le service IIS.

* L’application développée en C# utilisait le Framework 4.5 alors que le site web et le projet d’application pointé sur le Framework 2.0 nativement. J’ai donc dû modifier ces paramètres dans IIS.
* La version de ASP n’était pas la bonne (*erreur : HTTP Error 500.21 - Internal Server Error Handler "ScriptHandlerFactory" has a bad module "ManagedPipelineHandler" in its module list*) , j’ai dû mettre la version de ASP en téléchargement la version ASP.NET v4.
* Une fois ces points réglés, je ne pouvais toujours pas déployer mon application C# de VS2013 vers IIS. Ceci aussi bien en utilisant l’option de publication que l’importation d’un package de déploiement (*erreur : No option to Import site package using IIS 6.1 & web deploy 3.5*). Pour cela j’ai dû mettre à jour la version web deploy 3.5.

# Architecture de la base de données

## Configuration du serveur de base de données

Le choix de la base de données s’est porté sur SQL Server 2012 qui est utilisé en standard dans nos applications informatiques.

Les bibliothèques de business intelligence ont été rajoutées afin de pouvoir utiliser l’ensemble des outils nécessaires à la construction et la gestion du Data Warehouse.

* Integration Services : permet l’alimentation des tables du Data Warehouse à partir des tables « brutes » de l’exploitation.
* Analysis Services : permet de créer et de gérer le cube qui s’interface sur le Data Warehouse.
* Reporting Services : permet de générer des rapports automatiques à partir de la base de données en se basant sur le cube.

La configuration et l’utilisation de ces différents services seront vues plus en détail dans les chapitres suivants.

## Introduction aux Data Warehouse

Dans le cahier des charges, il m’a été demandé de réaliser un Data Warehouse (entrepôt de données) afin de structurer l’archivage des données et d’en faciliter l’interrogation.

En temps normal j’utilise des bases de données de type SGDB (système de gestion de base de données relationnelles) dans nos applications car elles répondent aux besoins de stockage de données bidimensionnelles de production.Ce type de base de données est fortement utilisé dans ce cas, car les mises à jours de données sont régulières et en temps réel.

Un entrepôt de données, ou data Warehouse, est une vision centralisée et universelle de toutes les informations de l'entreprise. C'est une structure (comme une base de données) qui a pour but, contrairement aux bases de données, de regrouper les données de l'entreprise pour des fins analytiques et pour aider à la décision stratégique. La décision stratégique étant une action entreprise par les décideurs de l'entreprise et qui vise à améliorer, quantitativement ou qualitativement, la performance de l'entreprise.

Les informations sont épurées, organisées, historisées et provenant de plusieurs sources de données, servant aux analyses et à l'aide à la décision.

Les Data Warehouse permettent également de modéliser de l'information pour des fins d'analyse.

Les bases de données pour les Data Warehouse sont d’ailleurs basées sur le modèle OLAP (On-Line Analytical Processing) principalement utilisé en lecture seule.

Il s’agit également d’un premier pas vers les Big Data, un concept, tel qu'il est défini actuellement, englobe un ensemble de technologies et de pratiques destinées à stocker de très grandes masses de données et à les analyser très rapidement.

Il existe 3 possibilités pour créer un Data Warehouse:

* Relational OLAP (ROLAP)
  + Données sont stockées dans un SGBD relationnel
  + Un moteur OLAP permet de simuler le comportement d'un SGBD multidimensionnel
* Multidimensional OLAP (MOLAP)
  + Structure de stockage en cube
  + Accès direct aux données dans le cube
* Hybrid OLAP (HOLAP)
  + Données stockées dans SGBD relationnel (données de base)
  + + structure de stockage en cube (données agrégées)

Pour mon projet le Data Warehouse sera basé sur le modèle MOLAP qui est permis grâce à l’utilisation de SQL Server 2012 et des outils BI intégrés.

L’alimentation en données du Data Warehouse se fait par l’intermédiaire d’un ETL (Extract-Transform-Load) qui permet de récupérer les donnes de base de données (ex : SGBD), de fichiers Excel ou de fichiers à plat (ex : cvs). L’ETL transforme donc les données brutes en données réparties dans les différentes tables du Data Warehouse en fonction du ou de schémas choisis (en étoile ou en flocon).

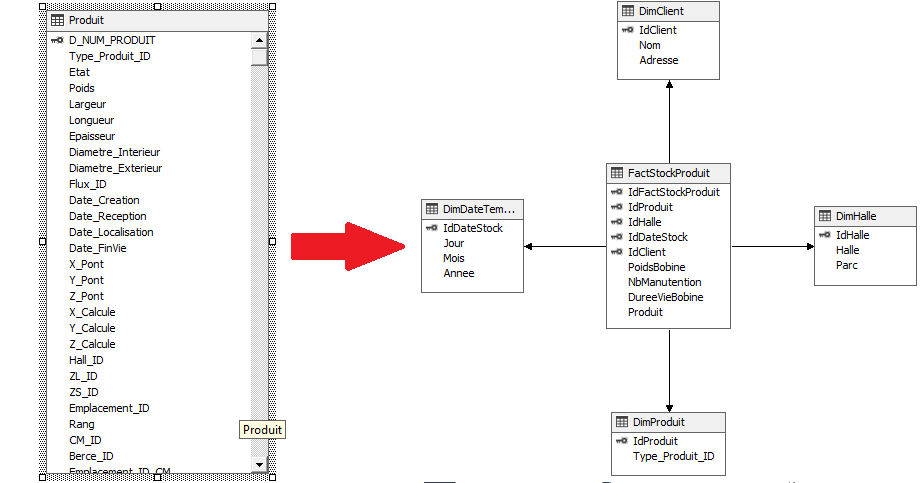
Afin de pouvoir accéder aux données il faut créer un cube OLAP. C’est est une méthode de stockage de données sous forme multidimensionnelle, généralement à des fins de génération de rapports. Les données (ou mesures) sont classées par dimensions. Les cubes OLAP sont souvent pré-synthétisés entre les dimensions, ceci afin d'accélérer considérablement l'interrogation par rapport aux bases de données relationnelles.

Les cubes permettent de :

* Analyser des données déjà agrégées et stockées selon les besoins de l’utilisateur.
* Modéliser les données sous forme de dimensions et de mesures.
* Capacité à manipuler de gros volumes de données agrégés en croisant différentes dimensions.
* Créer des rapports directement dans Excel en utilisant toute la puissance du cube.
* Utiliser les fonctions classiques d’agrégation : Min, Max, Count, Sum, mais peut être enrichi avec des calculs spécifiques complexes.

## Choix du schéma

A partir de la table produit (contenant l’ensemble des informations bobines), j’ai créé le schéma en étoile suivant. Bien sûr cela ne représente qu’une portion de ce qui est aurait pu être faite à partir de la table d’origine.



L’utilisation du Data Warehouse modifie en profondeur la composition de la base de données. D’abord la base de données est gérée par un serveur OLAP (Online analytical processing).

Pour pouvoir faire ce choix de schéma en étoile, j’ai d’abord analysé les besoins de recherche et de reporting que l’on peut avoir besoin lorsque l’on souhaite manipuler les données de notre base de données actuelle.

Ce que j’en ai déduit est que généralement on souhaitait faire des recherches à partir des halles ou des clients pour en déduire les stocks de produits existant.

J’ai donc créé une table de fait (FactStockProduit) dans laquelle on retrouve les différentes clés en lien avec les tables de dimensions, ainsi que les colonnes comme le poids, le nom des produits ou encore le nombre de manutention. L’alimentation de cette table sera fera lors de l’exécution de l’ETL (Extract-Transform-Load) décrit plus bas.

Pour les tables de dimension, je me suis contenté pour notre exemple des tables clients, produit, halle et date. L’alimentation de ces tables a été réalisée en prenant les données de la table produit lors de la phase de construction de la base de données.

## Alimentation du Data Warehouse avec SSIS

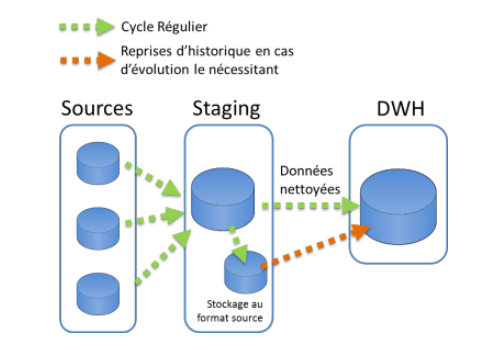
Pour pouvoir alimenter le Data Warehouse j’ai programmée un ETL (Extract-Transform-Load) en utilisation une solution « integration services » du module business intelligence de Visual Studio 2013.

L’ETL permet de synchroniser et de stocker des informations massives provenant principalement de bases de données.

### Configuration de l’ETL

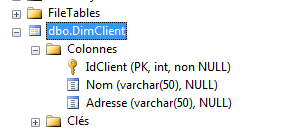
A partir des données de la table produit, l’ETL permet d’alimenter la table FactStockProduit (tables des faits du schéma en étoile) en effectuant différentes recherches dans les tables de dimension.

Les tables de dimension contiennent les index pour les données halle, clients, produit, date. Elles ont été alimentées manuellement en insérant les données nécessaires provenant de la table produit avec en plus une indexation automatique.



Par exemple pour la table DimClient j’ai procédé ceci :

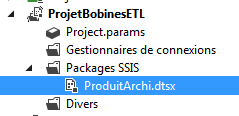
* Création de la table



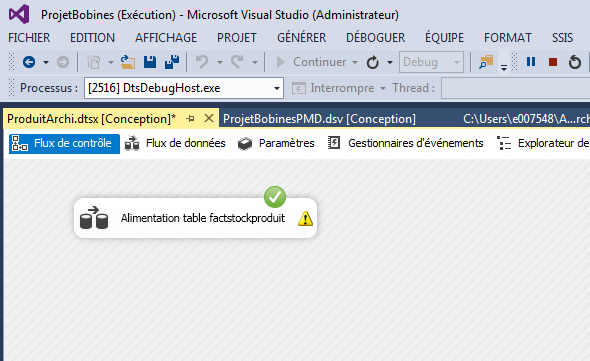
* Alimentation de la table à partir d’un fichier excel (suppression des blancs) qui se base sur le select distinct des champs de la table produits.

Pour alimenter la table FactStockProduit l’ETL est configuré comme ceci :

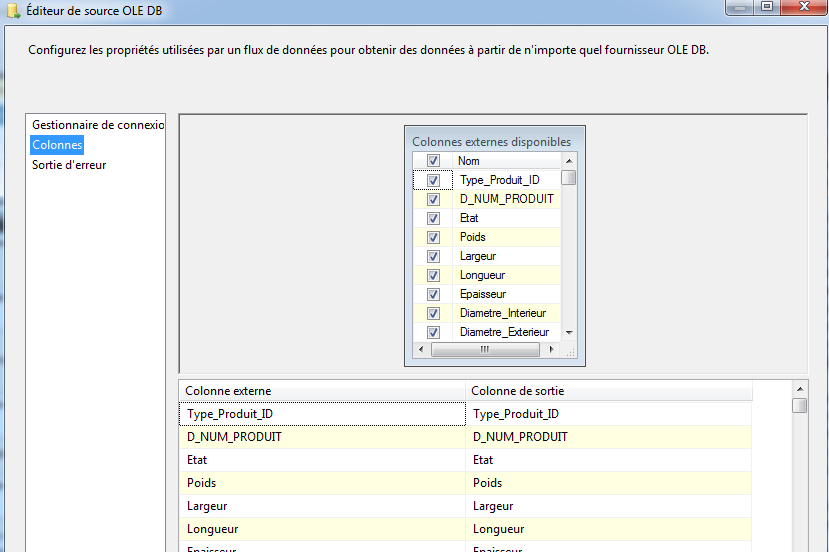
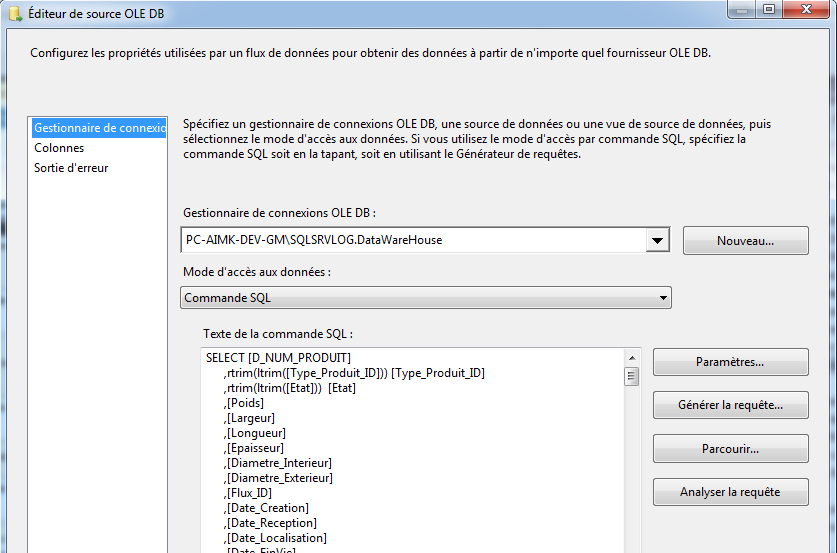
* Dans la solution il faut créer un package SSIS



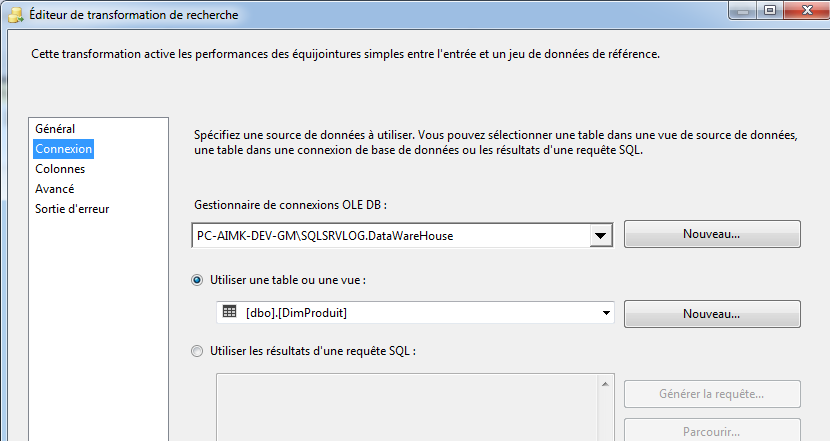
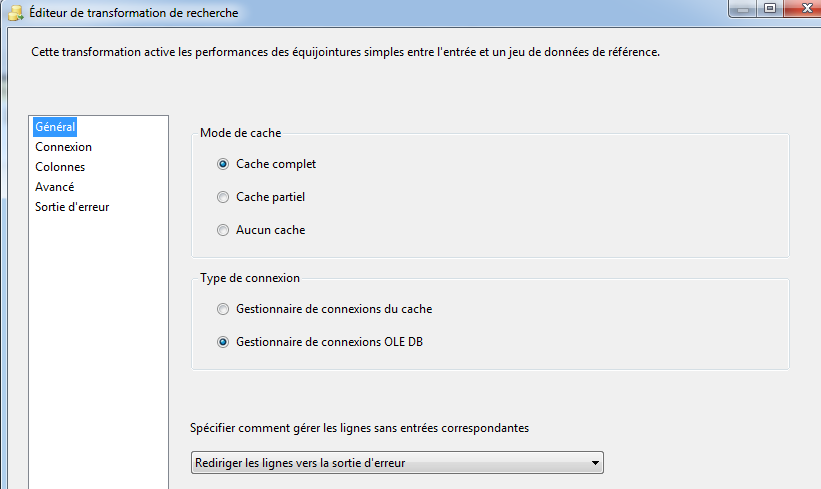
* On crée ensuite un flux de contrôle.

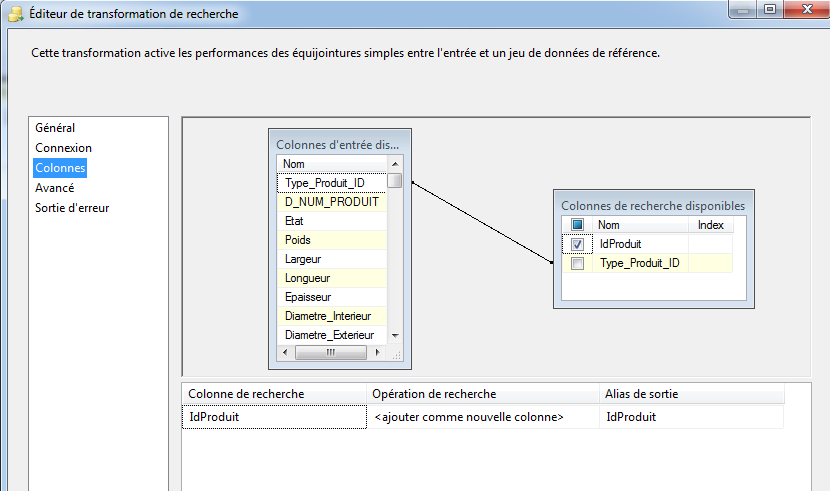


* Dans le flux de contrôle on double clic pour créer le flux de données.
* La source de données est configurée de telle façon que l’on récupère l’ensemble des données de la table produit. Dans la commande SQL les données de type chaînes de caractères utilisées pour effectuer des requêtes dans les tables de dimensions sont traitées de façon à supprimer les blancs devant et derrière les données.

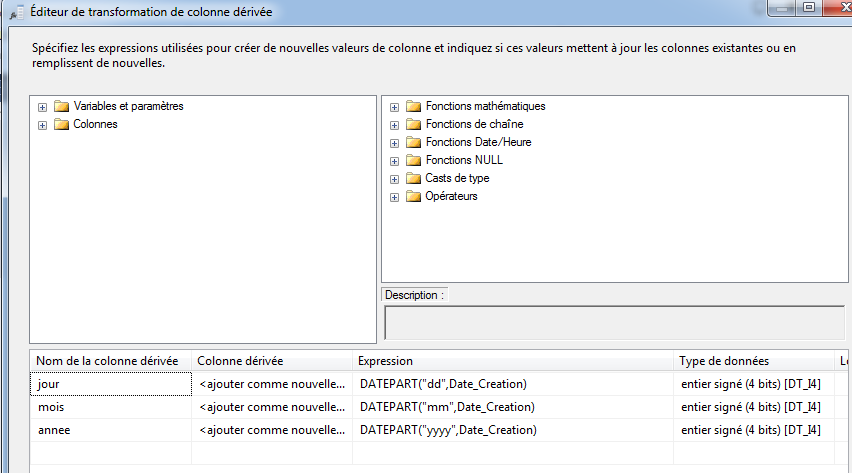


* En sortie de la source de données on vient rajouter un composant de recherche du produit qui va permettre de recherche l’ID de la table de dimension DimProduit en fonction du type de produit.

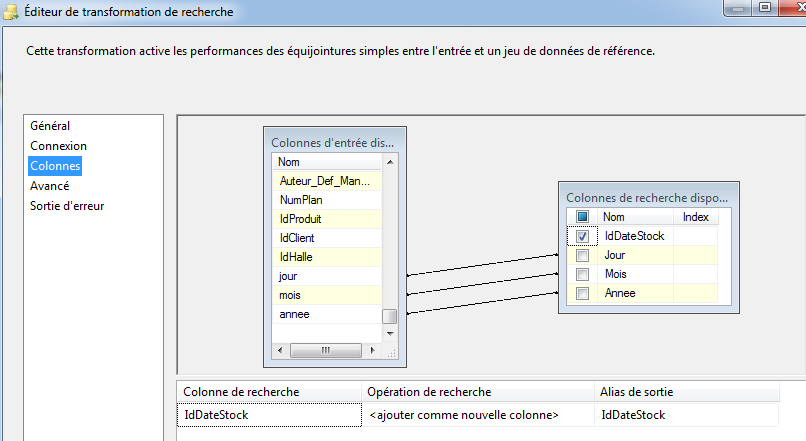




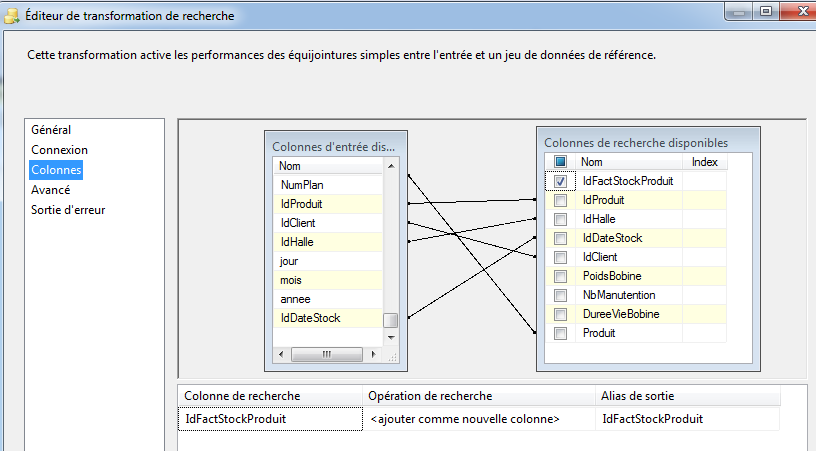
* Lorsque l’on relie la sortie d’un composant de recherche vers un autre, il faut spécifier si la recherche se fait avec ou sans correspondance. Dans notre cas ce fut par correpondance.
* On effectue la même chose pour retrouver les ID clients et Halles des tables DimClient et DimHall.
* Pour la recherche de l’identifiant de la table de dimension DimDateTemporelle, j’ai procédé d’abord en utilisant un composant de colonne dérivée. Il m’a permis de sortir de nouvelles colonnes jour, mois et année.



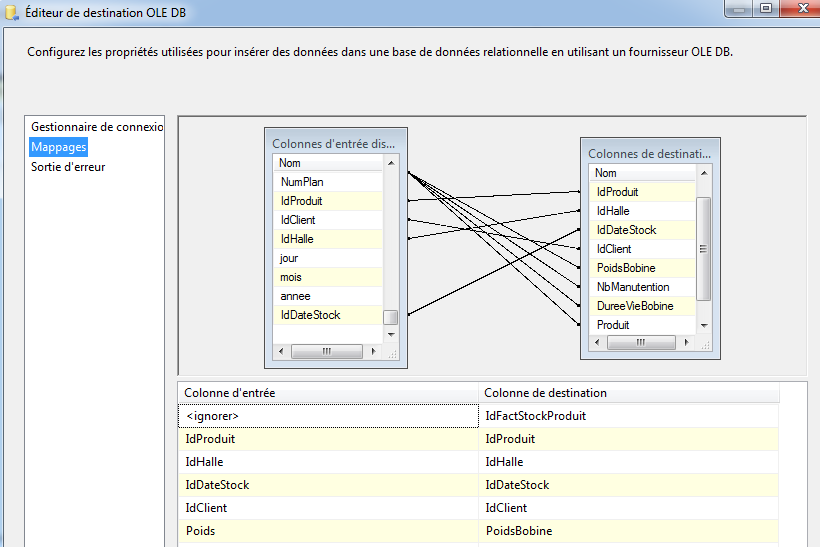
* Les nouvelles colonnes ainsi crées m’ont permis de les affecter au composant de recherche de la table de dimension DimDateTemporelle.



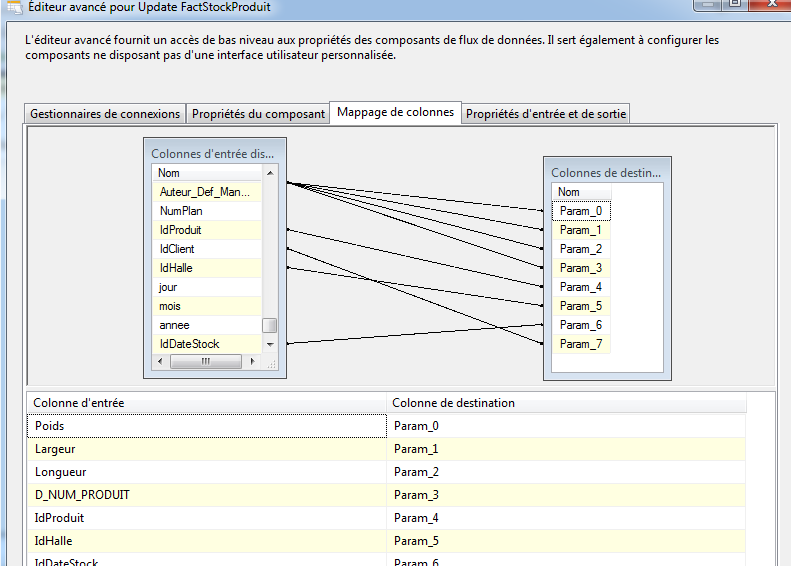
* Avec d’envoyer les données vers la table de destination, il fallait tester si les données étaient déjà présentes dans la table. Pour cela un composant de recherche permet de vérifier si les données pour un même enregistrement IDclient, IdHall, IDDateStock et IDProduit existent dans la table. Si la recherche donne un IDFactStockProduit cela indiqué qu’il existait déjà un enregistrement.



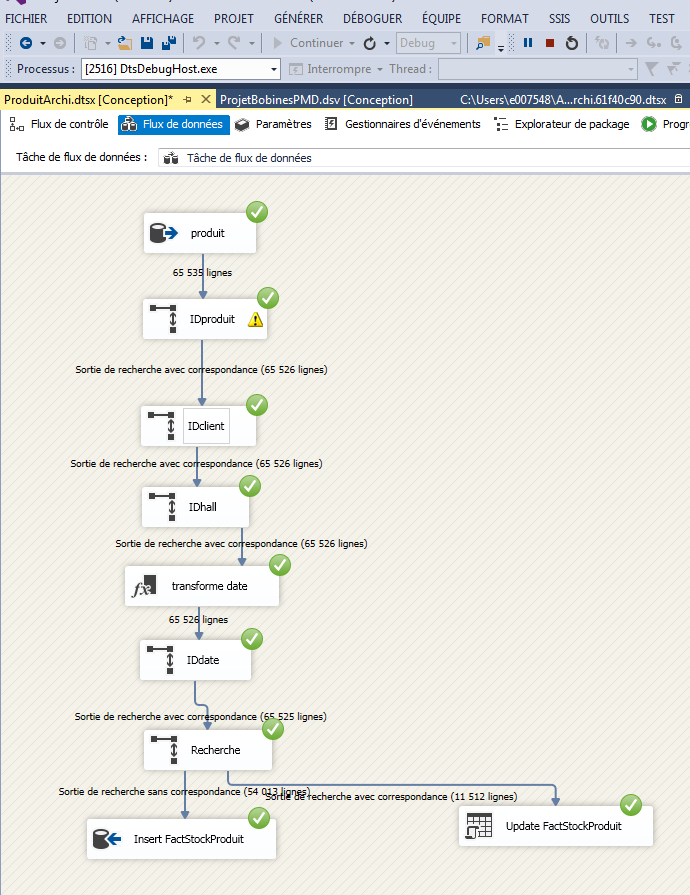
* La sortie sans correspondance du composant de recherche permet d’insérer un nouvel enregistrement dans la table FactStockProduit.



* En sortie avec correspondance du composant permet de mettre à jour la table FactStockProduit.



Lors de l’exécution de l’ETL dans Visual Studio nous avons les différentes étapes qui passent au vert indiquant ainsi que la programmation a bien été effectuée.



Comme nous pouvons constater dans le schéma ci-dessus, l’exécution de l’ETL s’est bien déroulée. Sur les 65525 enregistrements, 54013 enregistrements ont été insérés dans la table FactStockProduit et 11512 ont été mis à jour. Une fois le projet créé, je l’ai déployé sur le serveur de base de données.

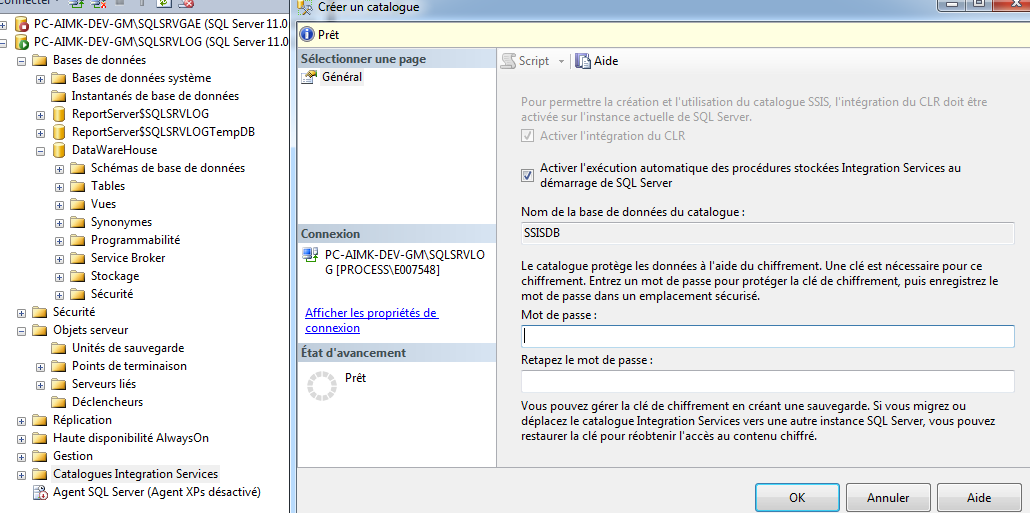
### Déploiement de l’ETL

Sur le serveur j’ai créé le catalogue **SSISDB** afin de pouvoir déployer les différents packages de ma solution de l’ETL que j’ai mis en place.

Pour cela dans SQL Server Management Studio il faut créer un catalogue SSISDB « Catalogues Integration Services ».

Le catalogue SSISDB est l’élément central pour l’utilisation des projets Integration Services Integration Services (SSIS) que j’ai déployé sur le serveur Integration Services Integration Services. Ainsi, c'est dans ce catalogue que j’ai définis les paramètres de projet et de package, j’ai configuré les environnements pour spécifier des valeurs d'exécution pour les packages, exécuté et résolue les problèmes relatifs aux packages, et géré les opérations du serveur Integration Services Integration Services

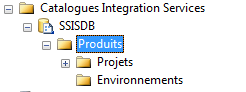
Pour cela un clic droit permet d’afficher la fenêtre d’invite et de remplir les différents éléments.



Il faut cocher les 2 cases dont l’intégration du CLR (Common Language Runtime).

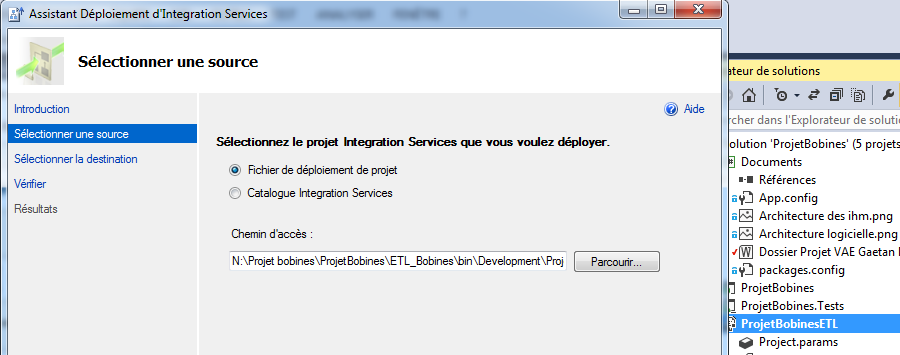
Le Common Language Runtime (CLR) est au cœur de Microsoft .NET Framework et fournit l'environnement d'exécution pour tout le code .NET Framework. Le code qui s'exécute dans le CLR est appelé code managé. Le CLR fournit divers services et fonctions requis pour l'exécution du programme, notamment la compilation juste-à-temps (JIT), l'allocation et la gestion de mémoire, l'application de sécurité de type, la gestion des exceptions, la gestion des threads et la sécurité.

En faisant un clic droit sur SSISDB, j’ai créé un projet avec le nom « produits » qui va contenir les différents objets déployés.

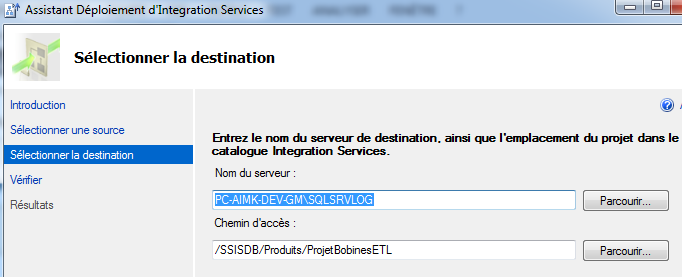


Sous Visual Studio, le clic droit sur le projet ProjetBobinesETL permet de sélectionner «déployer » et d’avoir l’assistant de déploiement d’Integration Services.

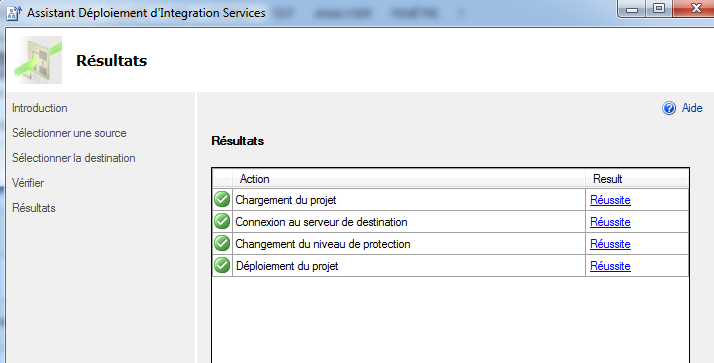
Il faut sélectionner le projet d’Integration Services à déployer. Dans mon cas il s’agit du choix par défaut correspondant au projet sélectionné.



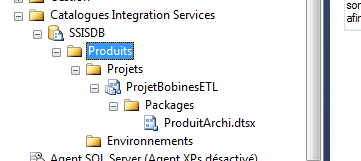
Il faut ensuite sélectionner la destination en sélectionnant le nom du serveur de base de données et le chemin d’accès. Dans notre cas on retrouve le projet précédemment créé dans Sql Server Management Studio.



En cliquant sur le bouton « suivant », on finalise le déploiement et si tout se passe correctement nous avons l’écran suivant :



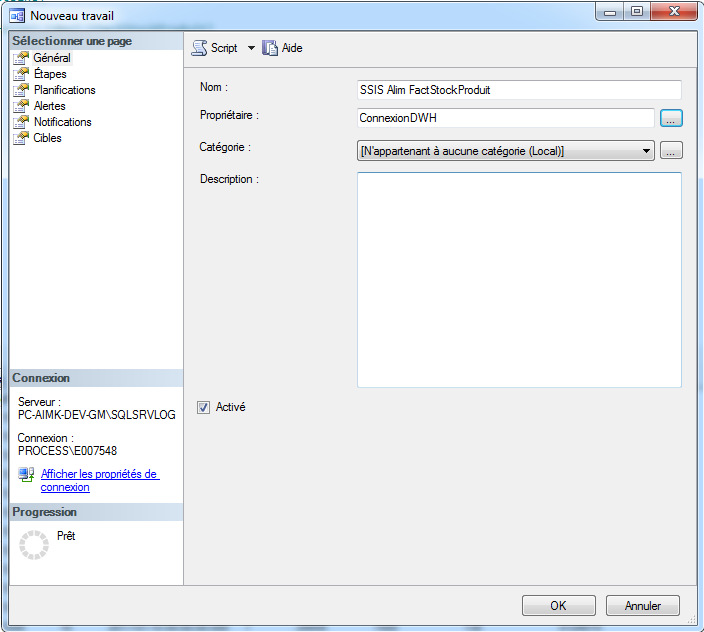
Dans Sql Server Management Studio on retrouve le projet déployé.



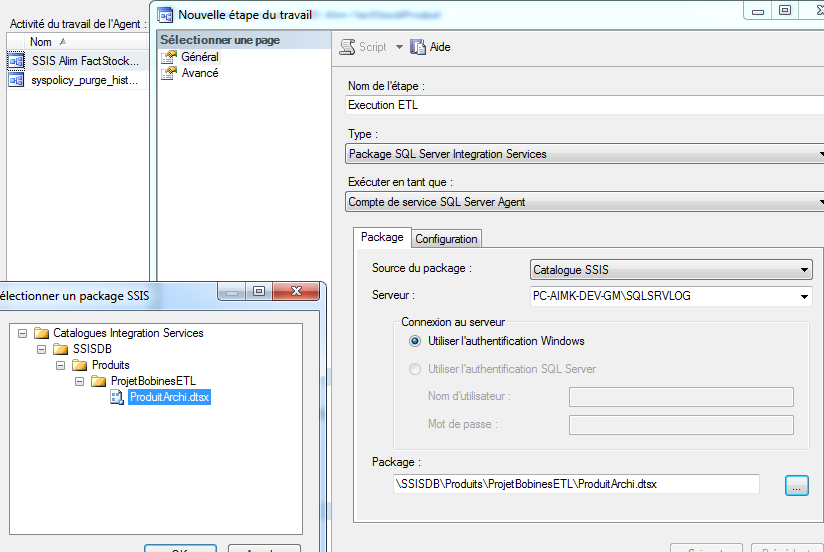
### Exécution de l’ETL

Une fois l’ETL déployé dans le serveur de base de données, j’ai configuré l’agent SQL Server afin de le faire exécuter en automatique.

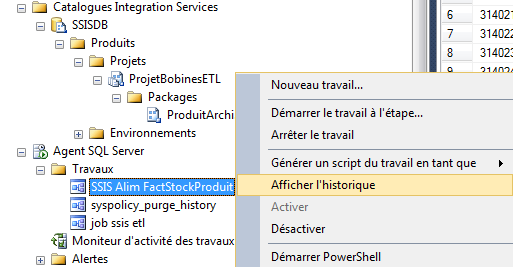
Pour cela il faut nommer le nouveau travail à faire exécuter et affecter un propriétaire.



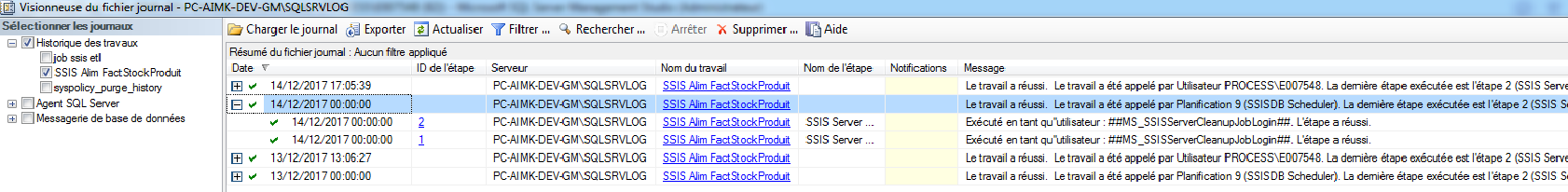
Puis il faut définir les étapes que l’on souhaite faire exécuter.



Afin de vérifier le bon fonctionnement du déclenchement de l’ETL, j’ai vérifié son historique.



On constate que la tâche d’exécution de l’ETL se déroule normalement en manuel et en automatique.



### Conclusion

Cette partie du projet m’a permis de prendre en main SSIS et de créer un Data Warehouse et son alimentation par un ETL. D’ailleurs j’ai déjà eu l’occasion de réutiliser les fonctions de SSIS pour alimenter des tables de production vers un serveur de test en créant un ETL simple de données sources vers des données de destination.

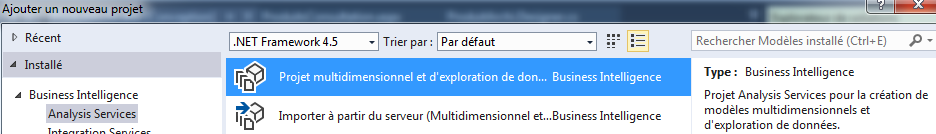
Pour aller plus loin dans mon projet il aurait été possible de rajouter les points suivants :

* En ajoutant un flux de contrôle avant l’alimentation de la table FactStockProduit on aurait pu alimenter en automatique les tables de dimensions.
* Pour la gestion du stock, il est possible de rajouter une transformation de dimension à variation lente afin de gérer les écarts de stock ou les calculs de durée de vie des produits sur l’historique.
* Ajouter d’autres tables de dimensions afin de proposer d’autres possibilités de recherche et d’autres rapports aux utilisateurs.
* La table produit est suffisante à elle-même pour poursuivre les développements de ce projet et de proposer d’autres tables de faits et de dimensions comme le suivi des manutentions de produits.

Cette partie du projet a représenté environ 30 heures de travail. Ceci en comptant le temps passé à me documenter sur internet afin d’appréhender le sujet et les multiples essais de création de l’ETL.

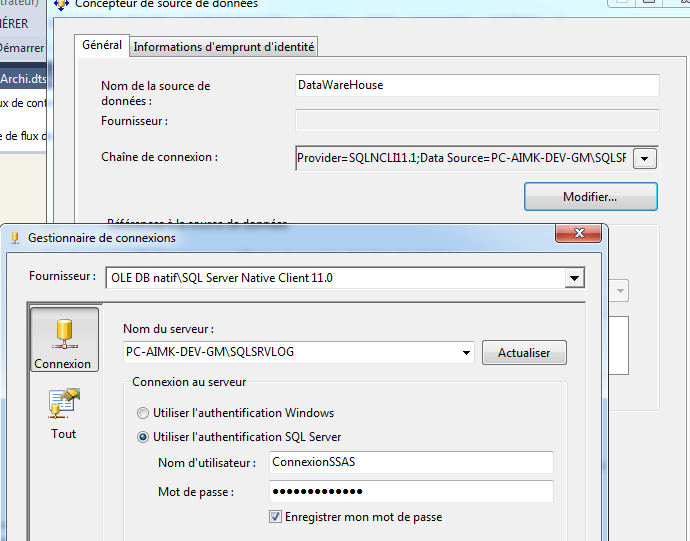
## Création et gestion du cube avec SSAS

Pour le créer le cube, j’ai utilisé les outils SSAS (SQL Server Analysis Services) fournis dans Visual Studio. Pour cela il faut créer un nouveau projet multidimensionnel et d’exploration de données dans notre solution.



### Connexion à la base de données

Il faut d’abord créer la connexion à la base de données. Dans les bonnes pratiques conseillées par Microsoft, il faut créer un compte dédié au cube. C’est le cas avec le compte connexionSSAS vu précédemment.



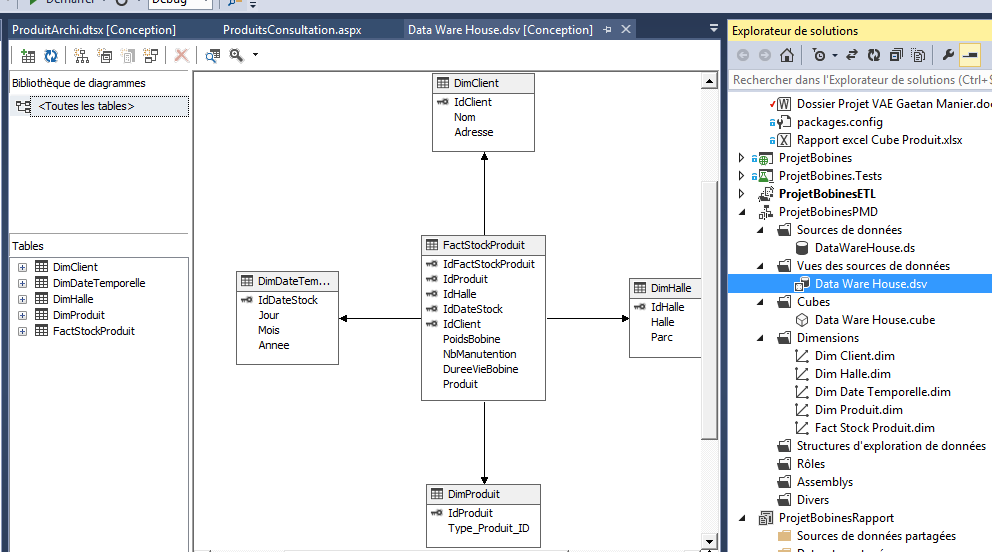
### Vue des sources de données

J’ai ensuite créé la vue des sources de données qui contient le modèle logique du schéma utilisé par le cube, les dimensions et les structure d’exploration de données. Elle représente la définition (stocké au format XML) des métadonnées de ces éléments de schéma par le modèle UDM (Unified Dimentional Model) et par les structures d’exploration de données.

Le rôle d'un UDM consiste à établir une passerelle entre l'utilisateur et les sources de données. Un UDM est construit sur une ou plusieurs sources de données physiques. L'utilisateur émet des requêtes sur cet UDM, à l'aide d'une large palette d'outils clients, tels que Microsoft Excel.

L'UDM ne constitue qu'une fine couche au-dessus de la source de données et offre à l'utilisateur final les avantages suivants : modèle de données plus simple et plus lisible, indépendance par rapport aux sources de données d'arrière-plan hétérogènes et optimisation des performances des requêtes de type résumé. Dans certains scénarios, il est possible de construire automatiquement un simple UDM. Lorsque l'UDM est construit avec des investissements plus conséquents, les avantages peuvent décupler grâce à la richesse des métadonnées que le modèle peut fournir.

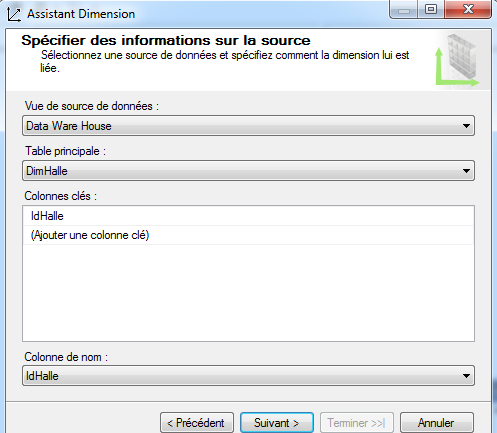
Un assistant de création permet de sélectionner la base de données puis de choisir les différentes tables de dimension et la table de fait.



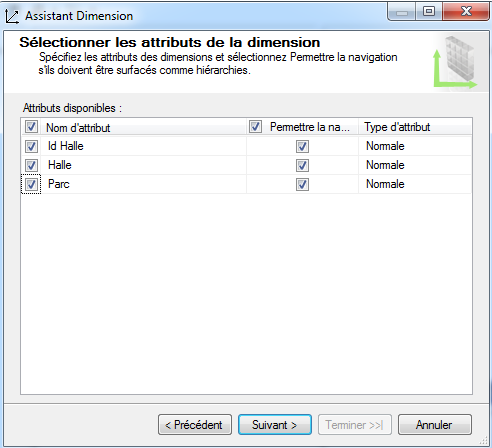
### Création des dimensions

J’ai ensuite créé les dimensions à partir des tables de dimension de la base de données. L’assistant de création de dimensions permet également de générer des tables type comme les tables temporelles, de production, de suivi client etc…

Pour les 4 tables de dimension de mon projet j’ai donc procédé de la même façon. Dans la vue de source de données j’ai sélectionné une table de dimension, la table DimHalle pour notre explication



J’ai ensuite sélectionné les attributs de la table de dimension.

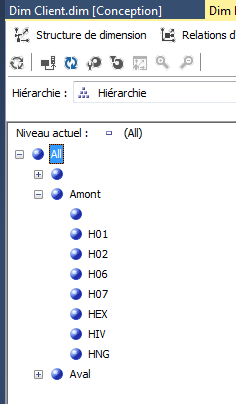


Puis dans l’interface j’ai pu définir la hiérarchie des données pour cette table. Dans notre cas nous avons le parc (amont, aval) puis les halles (E2, P2…).



Le bouton « traiter »  permet de déployer sur le serveur la dimension ainsi créée et de vérifier son bon fonctionnement en l’exécutant.

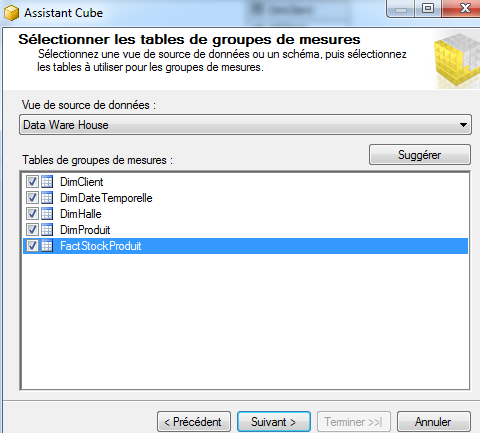
Grace au bouton « navigateur »  il est possible de visualiser le résultat de la création de notre dimension.



On retrouve bien dans notre exemple la hiérarchie souhaitée pour la dimension des halles. Comme dans la table DimHalle il y avait des données avec des nuls pour certains parcs nous les retrouvons donc également.

### Création du cube

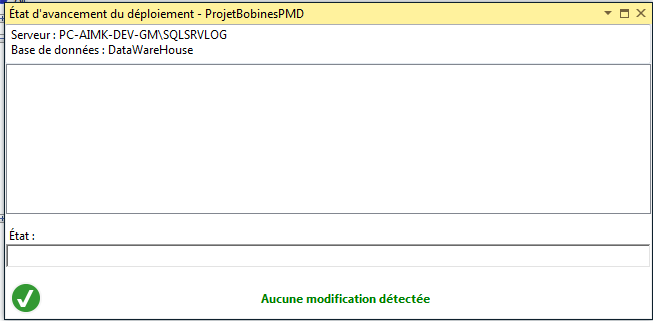
Pour la création du cube à proprement dit, j’ai utilisé l’assistant de création en le paramétrant de la façon suivante. J’ai dû donc choisi la création du cube à partir des tables existantes et en les sélectionnant dans la vue des sources de données précédemment créée.



### Déploiement du cube

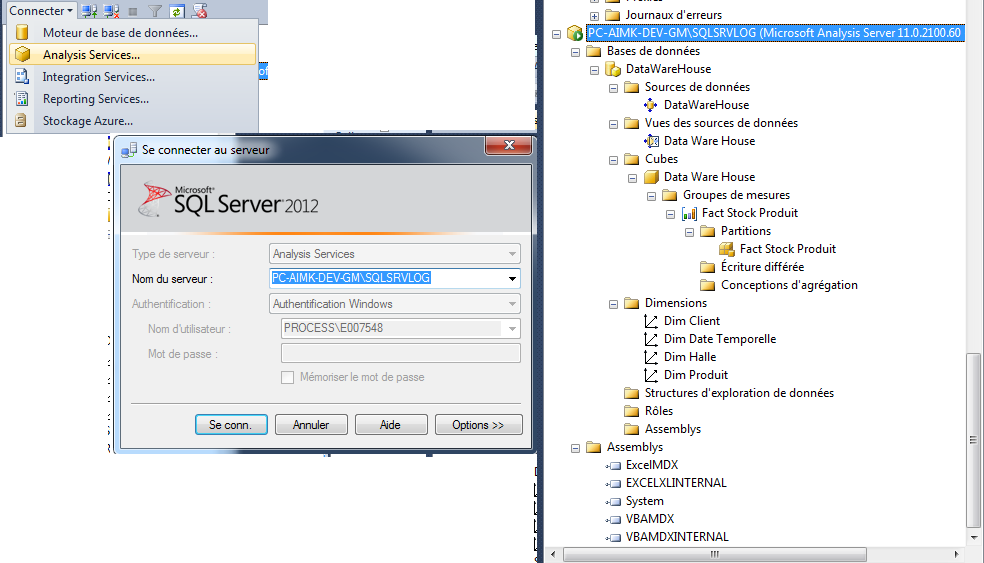
Pour déployer le cube dans le serveur de base de données il suffit de cliquer sur déployer au niveau du projet « projetbobinePDM ».

Une popup apparait pour indiquer que cela s’est bien passé.



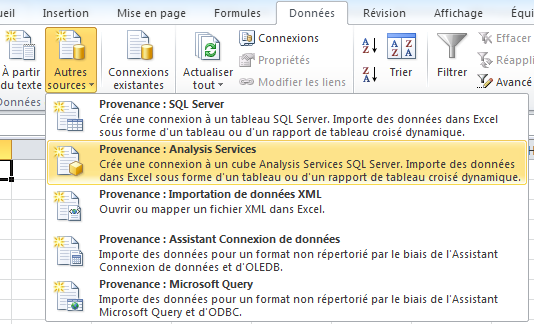
Ensuite dans SQL server management studio il est possible de visualiser les objets déployés en se connectant à la base de données Analysis Services et en sélectionnant le serveur de base de données du projet.

On retrouve donc l’ensemble des objets du cube de nous avons créés précédemment.

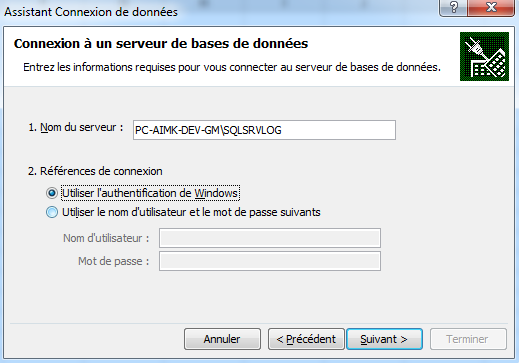


### Exécution du cube avec Excel

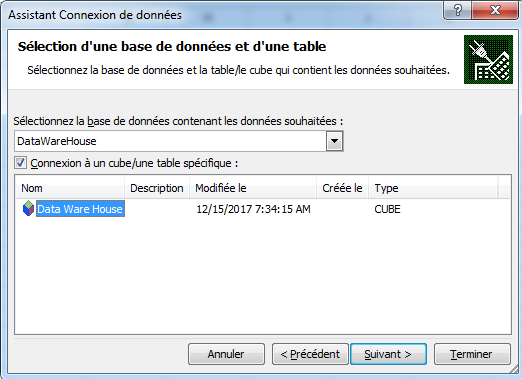
Il est possible d’utiliser directement le cube avec un fichier Excel afin de générer des rapports.



Pour cela il faut créer un nouveau fichier Excel et dans l’onglet « données » sélectionner autres sources puis la provenance : Analysis Services.

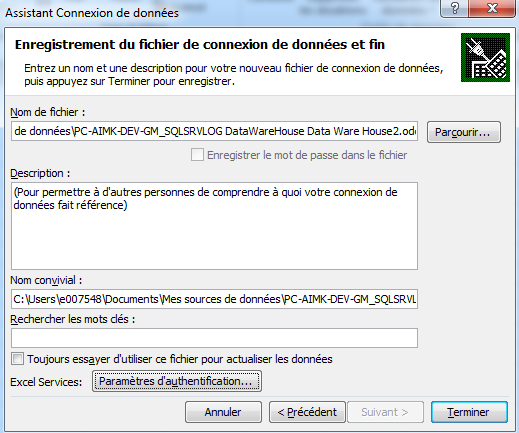


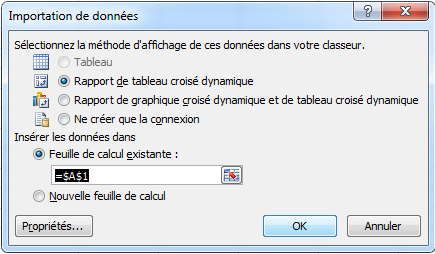
Déclarer la connexion au serveur de la base de données



Enregistrer le fichier de connexion à la base de données. Dans notre cas j’ai pris une identification Windows mais il est possible créer une identification SSS (Secure Store Service) est un service partagé qui fournit le stockage et le mappage des informations d'identification telles que les noms de compte et les mots de passe.

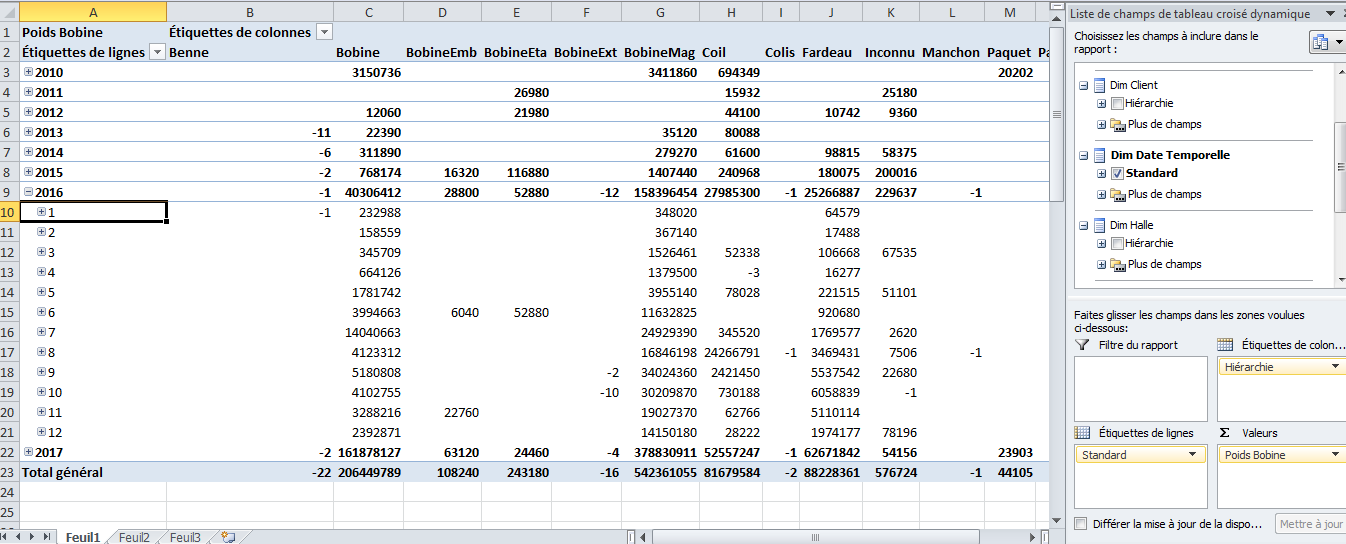
Puis sélectionner la base de données et le cube qui a été précédemment déployé.





On choisit ensuite où on veut importer les données et sous quelle forme. Dans notre nous prenons le tableau croisé dynamique.

La constitution du rapport reste en soit assez simple car on utilise ici les standards des tableaux croisés dynamique. Dans l’exemple ci-dessous j’ai sélectionné pour l’affichage des lignes le champ de date standard créé dans le cube afin d’avoir la hiérarchie des années, des mois puis des jours. Pour les colonnes j’ai pris la hiérarchie de la table de dimension DimProduit afin d’avoir la vision complète de l’ensemble des produits par type.



Enfin en valeur, j’ai pris le poids des bobines. Ainsi en sélectionnant les jours, mois et années on a automatiquement le rapport qui se met à jour.

Il peut ensuite être sauvegardé afin de refaire une visualisation des données en fonction des mises à jour de la base de données.

### Conclusion

Cette partie du projet m’a permis de prendre en main SSAS et d’utiliser un cube OLAP à partir du Data Warehouse.

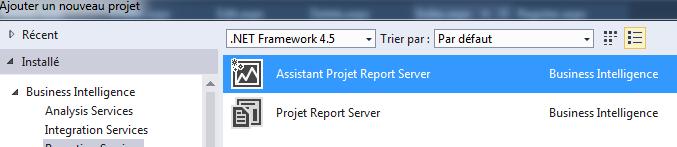
Il est évident que ce que j’ai mis en place n’est qu’une infime partie de ce qui est possible de faire avec SSAS qui permet entre autre de faire des calculs de KPI, d’ajouter des indicateurs de calculs automatiques (sum, moy …).

Cette partie du projet a représenté environ 30 heures de travail. Ceci en comptant le temps passé à me documenter sur internet afin d’appréhender le sujet et les multiples essais de création du cube.

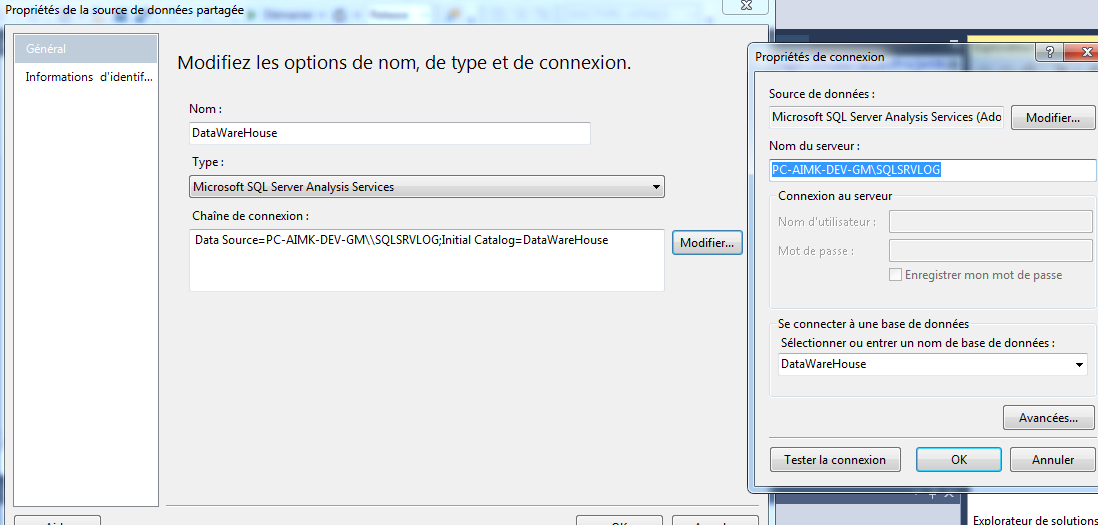
## Création des rapports avec SSRS

### Création des rapports

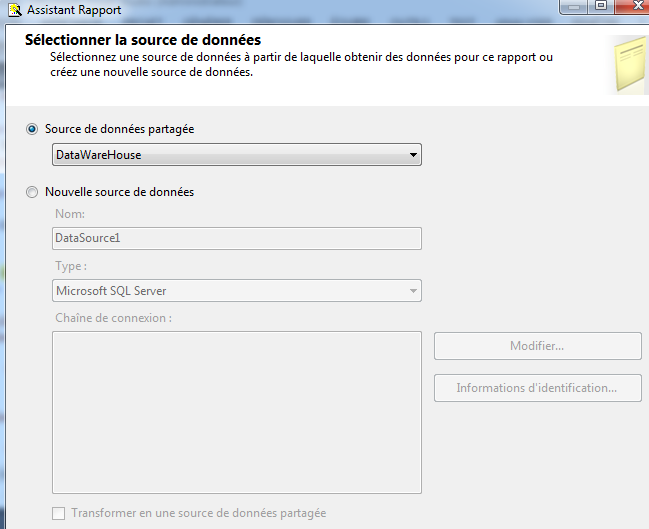
Visual Studio permet de créer des rapports automatique grâce à la couche SSRS (SQL Server Reports Services) qui s’interface directement avec le cube précédemment créé. Pour cela il faut créer un nouveau projet dans Visual Studio.



Il faut ensuite configurer la connexion à la base de données.



Enfin on peut créer un rapport qui nous donne le poids des produits en fonction du parc en utilisant l’assistant et en le configurant comme ci-dessous.



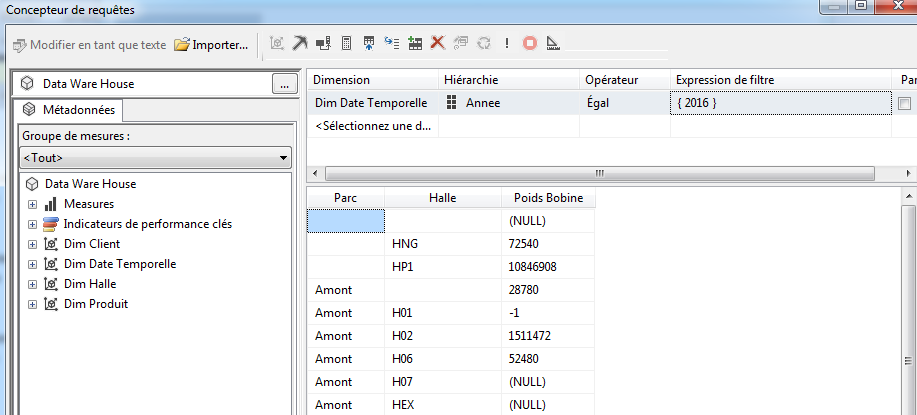
Pour notre rapport des poids bobines en fonction des parcs et des halles on sélectionne les différentes données de façon à les placer à droite de l’interface.

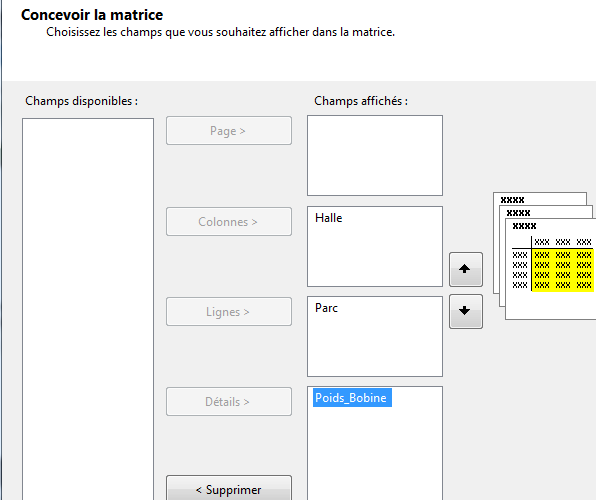
Dans notre nous avons sélectionné l’année 2016.

Puis on sélection le type d’affichage entre tabulaire et matriciel (notre exemple).

Pour la sélection de la source de données on prend DataWareHouse créé précédemment.

Ensuite il faut utiliser le gestionnaire de requête.



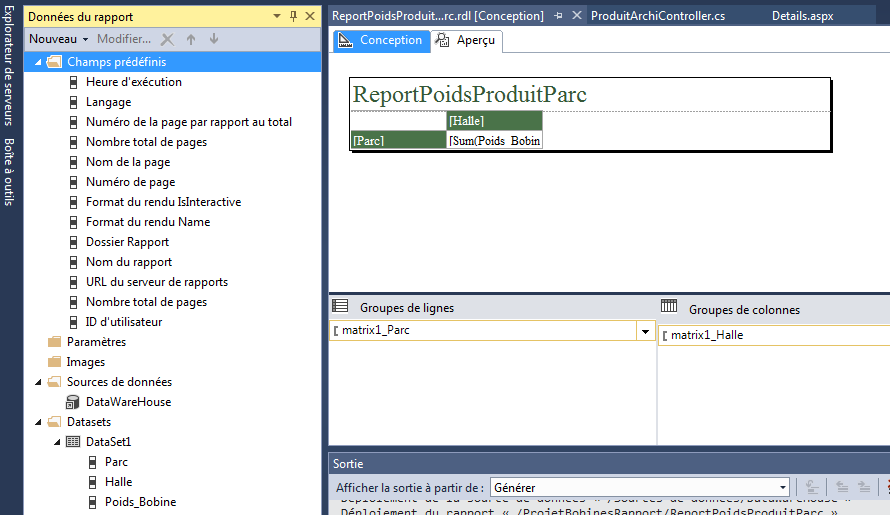


On remplit ensuite les différentes parties de la matrice (sur le même principe que le tableau croisé dynamique sous Excel).

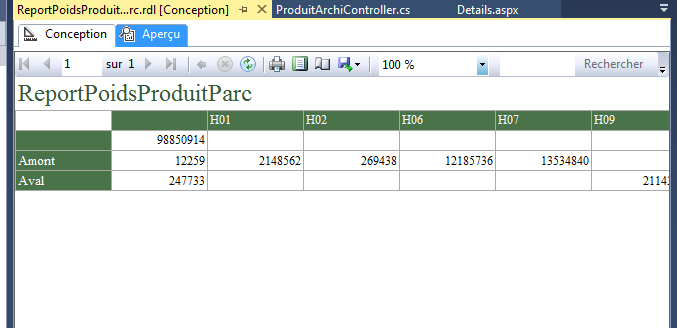
En colonne on met la halle, en ligne le par cet dans le détail on retrouve le poids bobines.

On peut ensuite choisir le style près définie. Dans notre exemple j’ai choisi le style forêt.

Une fois le rapport créé, il est possible d’agrémenter le rapport avec d’autres éléments prédéfinis.

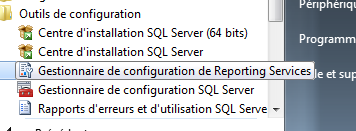


Le mode aperçu permet de visualiser le contenu du rapport et de se donner une idée du résultat final.

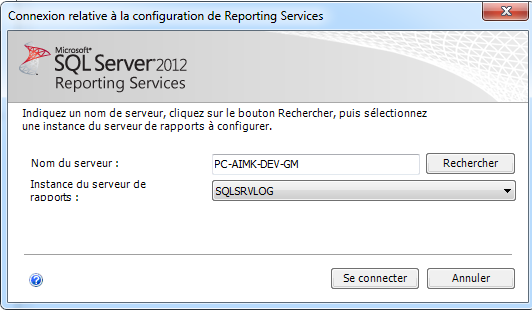


Avant de pouvoir déployer le rapport sur le serveur il faut configurer le Reporting service.

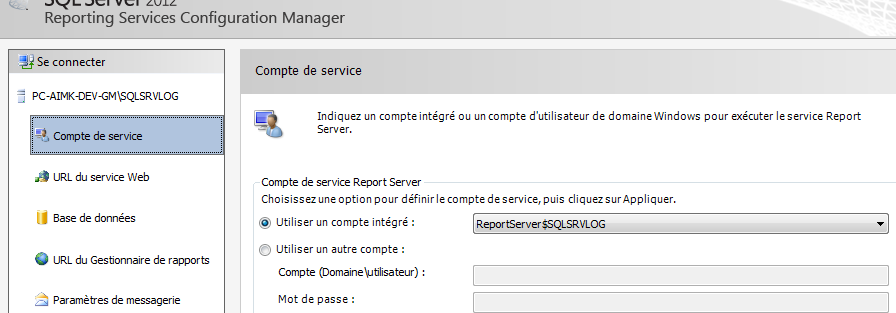
### Configuration de Reporting Service



Avant de pouvoir déployer le rapport sur le serveur il faut configurer le Reporting service en sélectionnant l’utilitaire dans les outils de configuration de SQL Server 2012.



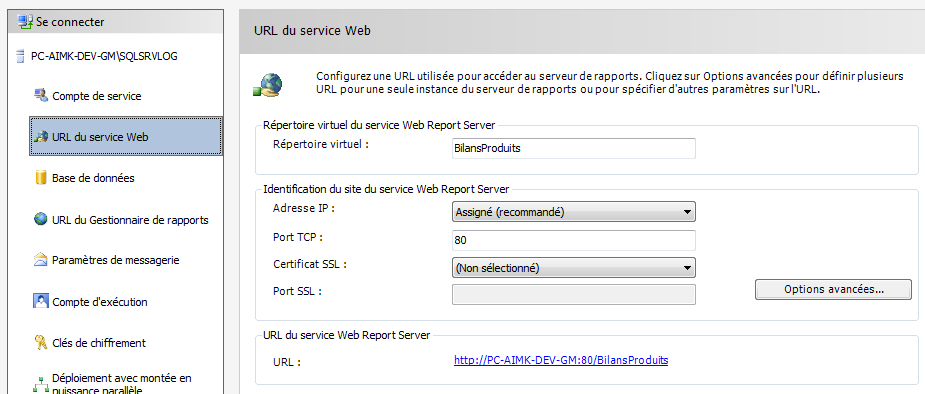
On sélectionne ensuite la configuration à la base de données souhaitée. Dans notre cas nous nous connectons à l’instance SQLSRVLOG.

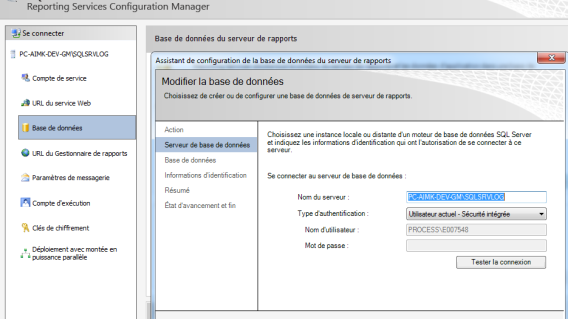


La configuration du compte de service permet de créer le compte qui va exécuter le service report server. Par défaut j’ai pris le compte intégré qui est conseillé.

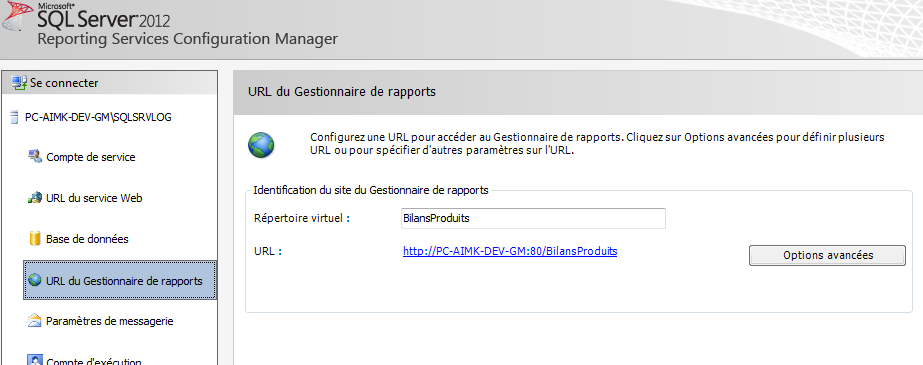
Afin de pouvoir déployer les rapports sur notre serveur web il faut configurer le service web.

Pour cela on renseigne le nom du répertoire virtuel ainsi que le port du service. On récupérer ainsi la future adresse web où l’on pourra consulter les rapports. Les options avancées permettent également de gérer la couche SSL (TODO) pour la sécurisation du site.





J’ai ensuite configuré la base de données sur laquelle le service report server se connectera afin de récupérer les rapports.

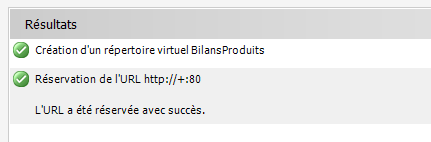


La configuration de l’URL du gestionnaire peut de configurer le site. Par défaut j’ai pris le même répertoire que le site web configuré précédemment.

Pour les autres onglets de paramètre j’ai laissé la configuration par défaut.

Pour la partie sécurité, Reporting Services gère une clé symétrique de chiffrement pour les données d’identification, les chaînes de connexion et les données sensibles de la base de données Report server. J’ai sauvegardé la clé de chiffrement dans un fichier afin de l’archiver. TODO expliquer clé symétrique.

A chaque fois pour chaque onglet, la validation des modifications met à jour le server. Cela permet de savoir au fur et à mesure si les paramètres sont renseignés correctement et ceci grâce aux différents messages affichés en bas des onglets.

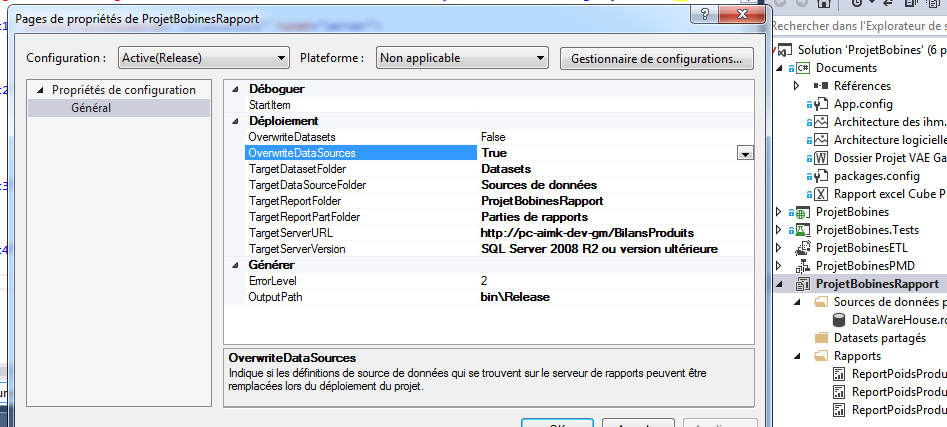


Après avoir appliqué les paramètres le serveur web reporting services est configuré. On peut maintenant passer au déploiement des rapports dans Visual Studio.

### Déploiement des rapports

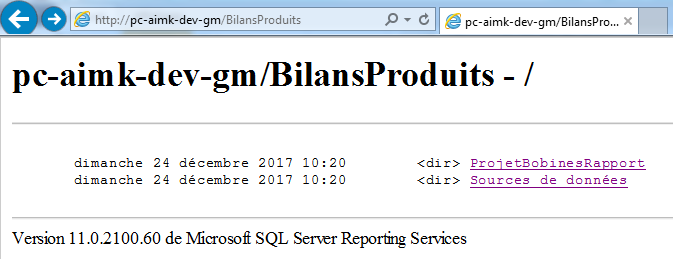
Avant de déployer les rapports il faut configurer les propriétés du projet ProjetBobinesRapport que nous avons créé au début de ce chapitre sur la création de rapports avec SSRS.

Dans ces propriétés j’ai donc modifié le OverwriteDataSources afin d’autoriser le remplacement des rapports présents sur le serveur lors du déploiement et le TargetServerURL afin de spécifier l’adresse et le répertoire de destination des rapports à déployer. Cette adresse web correspond à celle crée lors de la configuration du Reporting Services.

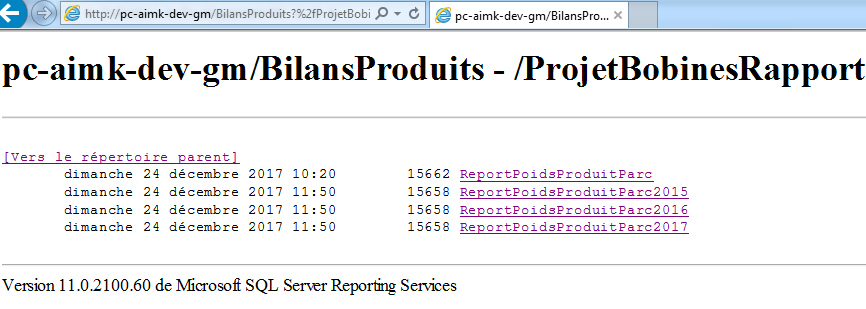


### Consultation des rapports

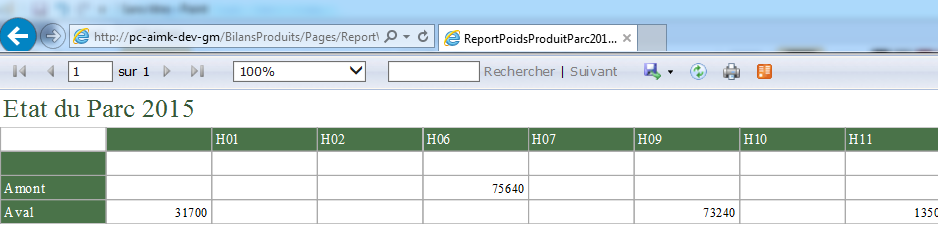
Il est possible de consulter les rapports directement avec internet explorer en mettant l’URL suivante <http://pc-aimk-dev-gm/BilansProduits>.



On peut ainsi accéder aux rapports en cliquant sur le lien ProjetBobinesRappots et à la consultation de la source de données.



Si l’on veut consulter le rapport des poids de produit dans les parcs en 2015 il suffit de cliquer sur le lien en rapport. Cela permet d’afficher le rapport correspondant qui a été créé au tout début du chapitre.



L’interface web permet à l’utilisateur d’imprimer sont rapport mais aussi de le sauvegarder dans différents format. Pour des tableaux de données par exemple, la gestion des boutons des pages suivantes/précédentes par exemple est réalisée automatiquement.

Pour mon application l’accès aux rapports pour les utilisateurs se fera directement via le site web que j’ai créé afin de leurs éviter de changer de page web et d’environnement.

### Conclusion

Cette partie du projet m’a permis de prendre en main SSRS et d’utiliser un générateur de rapport se basant sur le cube OLAP créé dans le chapitre précédent. Je retrouve d’ailleurs de nombreuses similitudes avec SAPBi4 que nous utilisons pour la réalisation de rapports de production.

Il est évident que ce que j’ai mis en place n’est qu’une infime partie de ce qui est possible de faire au niveau des rapports avec SSRS. Des rapports automatiques journaliers, hebdomadaires et mensuels auraient pu être mis en place.

Cette partie du projet a représenté environ 15 heures de travail. Ceci en comptant le temps passé à me documenter sur internet afin d’appréhender le sujet et les multiples essais de création de rapports.

# La sécurité

La sécurité et l'authentification est un problème récurrent des applications web et mobile.

Les API sont utilisées par les différents clients de ces applications comme les navigateurs. Il est souvent nécessaire d'avoir une solution commune et performante.

Pour mon application web cela est assuré par les cookies. Ils sont utilisés et retournés par le serveur au client qui a une authentification réussie. Ces cookies accompagnent chaque requête client afin de l'identifier. Les sessions sont stockées serveur (en mémoire, sur le file-system ou en base de données) avec la possibilité d'avoir des problèmes de charge à gérer.

D'autres solutions existent comme le http basic authentification ou les identifiant s de l'utilisateur transitent à chaque requêtes).

Avec les applications mobiles natives, la gestion des cookies est relativement compliquées et les problèmes de cross-origin ressource sharing (CORS) récurrents. De plus les applications utilisant les API-REST doivent être en stateless, c'est à dire sans session coté serveur).

Ils existent donc plusieurs solutions possibles:

* Les web tokens: Dont le principe consiste après s'être identifié, le serveur génère un hash de plus de 100 caractères qui sert de signature pendant une certaine durée. Ce token transite à chaque requête entre le serveur et le client. Pas cookies et ni de session sur le serveur. Cette solution est applicable à plusieurs types de plateformes comme les applications mobiles natives et les applications web.

Il est possible de se protéger contre les attaques de type CSRF (Cross-site Request Forgery -> usurpation d'identité) en générant un token différent pour chaque requête.

* Le JWT: Le Jason Web Token permet de découper en 3 parties les requêtes.

La première concerne l'entête codé en base 64.

La seconde est le contenu à échanger entre le serveur et le client codé en base 64. Il possède également les infirmations de l'identité de l'utilisateur et l'expiration de token.

Enfin la signature qui correspond à la concaténation de l'entête et du contenu qui est ensuite encodé suivant un algorithme défini dans l'entête.

## Les comptes utilisateurs

Lorsque l’utilisateur créé son compte via la page web, certaines informations sont cryptées en base de données comme son mode de passe.

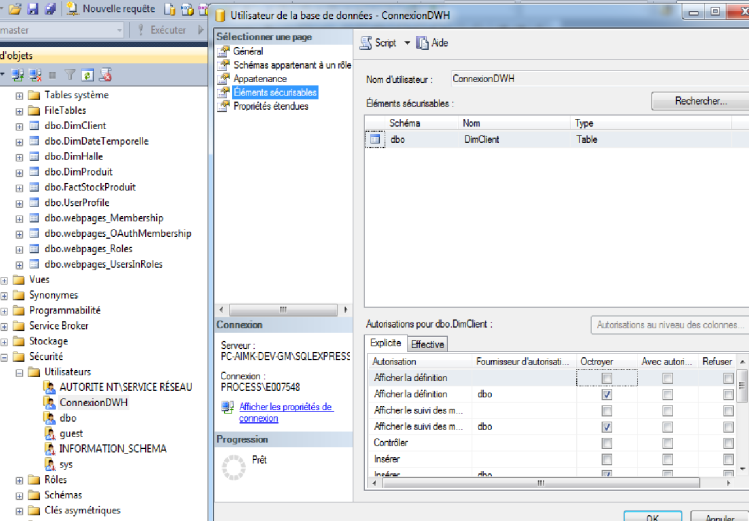
Pour cela il a été utilisé la Classe WebSecurity qui fournit des fonctionnalités de sécurité et d'authentification pour les applications ASP.NET Web Pages, y compris la possibilité de créer des comptes d'utilisateur, de connecter et de déconnecter les utilisateurs, de réinitialiser et de modifier les mots de passe, ainsi que d'exécuter les tâches associées.

### Compte pour l’application Web

L’application web se connecte à la base de données en utilisant un compte spécifique sans utiliser les comptes Windows par défaut.



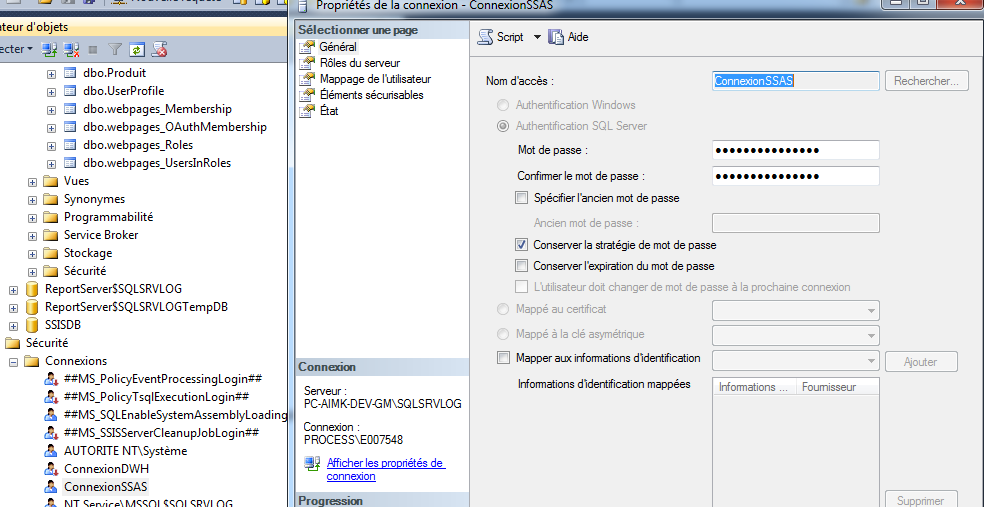
Pour cela il a été créé une connexion au niveau du serveur de base de données qui s’appelle « ConnexionDWH ».



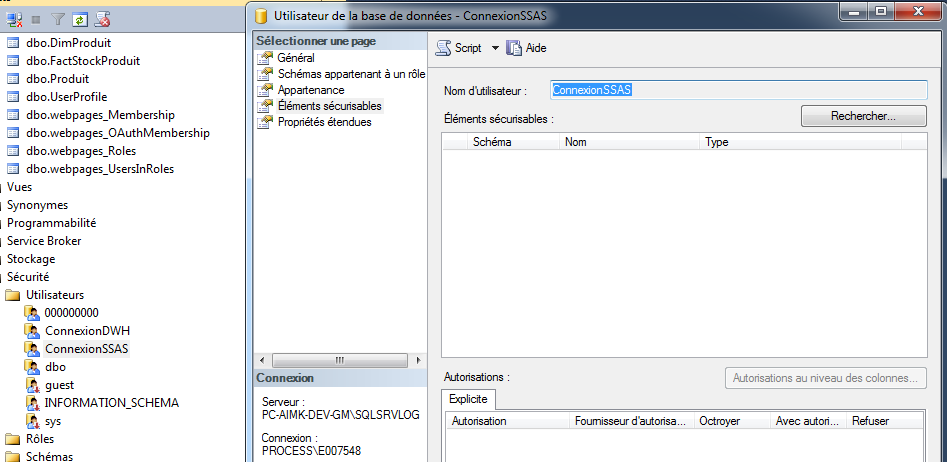
Ensuite un compte utilisateur sur la base de données « ConnexionDWH».

### Compte pour le SSAS

Dans les bonnes pratiques de configuration d’un service SSAS, il est conseillé de configuré un compte de base de données dédié à cela.



Pour cela il a été créé une connexion au niveau du serveur de base de données qui s’appelle « ConnexionSSAS ».



Ensuite un compte utilisateur sur la base de données « ConnexionSSAS».

## Protection des pages web

Lorsque l’utilisateur se signe sur l’application web, cela génère un cookie qui est utilisé pour l’authentification à chaque requête entre le serveur et le client.

Pour cela j’utilise l’API de ASP.NET pour gérer les rôles et qui fournit également un ensemble de fournisseurs pour les stocker (dont un pour SQL et un pour Active Directory), le tout permettant de simplifier l’association entre utilisateurs et rôles.

[Authorize(Roles="admin")]

public ActionResult Create() {

...

}

L’application utilise également le jeton anti-contrefaçon pour protéger l’application contre la falsification de requête intersites. Pour utiliser cette fonctionnalité, la méthode AntiForgeryToken est appelée à partir des contrôleurs.

## Conclusion

Il existe bien d’autres moyens de sécuriser les applications web et mobiles comme par exemple :

* Passport JS : Passeport JS est le module NPM (Node Package Manager) non intrusif d’authentification pour Node.JS. Il est extrêmement flexible et modulaire, il permet de s'authentifier avec les stratégies d'authentification les plus répandues : Basic & Digest, OpenID, OAuth, OAuth 2.0 and JWT. Passport.JS nous permet également d'utiliser des systèmes d'authentifications externes comme Facebook, Google +, Twitter, Github. Très populaire, Passport.JS propose plus de 300 stratégies d'authentifications tierces.
* LetsEncrypt : Let’s Encrypt » (en deux mots) désignera l’[autorité de certification](https://letsencrypt.org/) délivrant des certificats par l’intermédiaire du [protocole ACME](https://datatracker.ietf.org/wg/acme/documents/) (« Automatic Certificate Management Environment »), tandis que « Letsencrypt » (en un seul mot) désignera le [client ACME officiel](https://github.com/letsencrypt/letsencrypt). Letsencrypt s’occupe d’obtenir le/les certificats pour le domaine souhaité et de l’installer. L’intérêt de Letsencrypt pour les applications web, est de pouvoir répartir une application sur plusieurs serveurs et de gérer de façon transparente pour l’utilisateur la connexion à l’application sans devoir se signer à chaque changement de serveur lors de la répartition de charge.

Cette partie du projet a représenté environ 30 heures de travail. Ceci en comptant le temps passé à me documenter sur internet.

*Les applications que je développe et que je maintiens sont basées sur des applications lourdes hébergées par des serveurs TS (Terminal Server) qui n’utilisent pas ces notions de sécurité.*

*Pour la gestion des comptes utilisateurs nous avons un domaine dédié en utilisant active directory. Les comptes sont donc associés à des groupes d’utilisateurs en fonctions de privilèges déterminés. Les machines sont également inscrites dans ce domaine afin de profiter de la centralisation des groupes de fonctionnalités.*

# Qualité du code, gestion des versions et des fiches recettes

Lors de la réalisation du projet, il était important de gérer le versionning du code de l’application et de la documentation (comme ce rapport).

Pour cela il existe différentes méthodes :

* Copier et renommer le ou les répertoires des sources avant développement.
* Utiliser un utilitaire de gestion de code sources.

## Choix de la gestion des versions

### Copie et renommage de répertoires

Cette méthode de versionning obligée chaque développeur à copier les sources soit en local soit sur un serveur, puis de remettre à jour manuellement les fichiers qu’ils avaient modifiés. De plus si plusieurs programmeurs utilisés le même fichier à un moment donné, rien ne leur indiqué s’il était utilisé par un autre membre de l’équipe projet.

Autrefois utilisée elle a montré ces limites depuis l’arrivée des projets en nouvelles technologies où les équipes projets sont relativement importante voir même distante de plusieurs centaines de kilomètres.

Bien évidemment cette solution n’a pas été retenue pour mon projet.

### Utilitaire de gestion de code sources

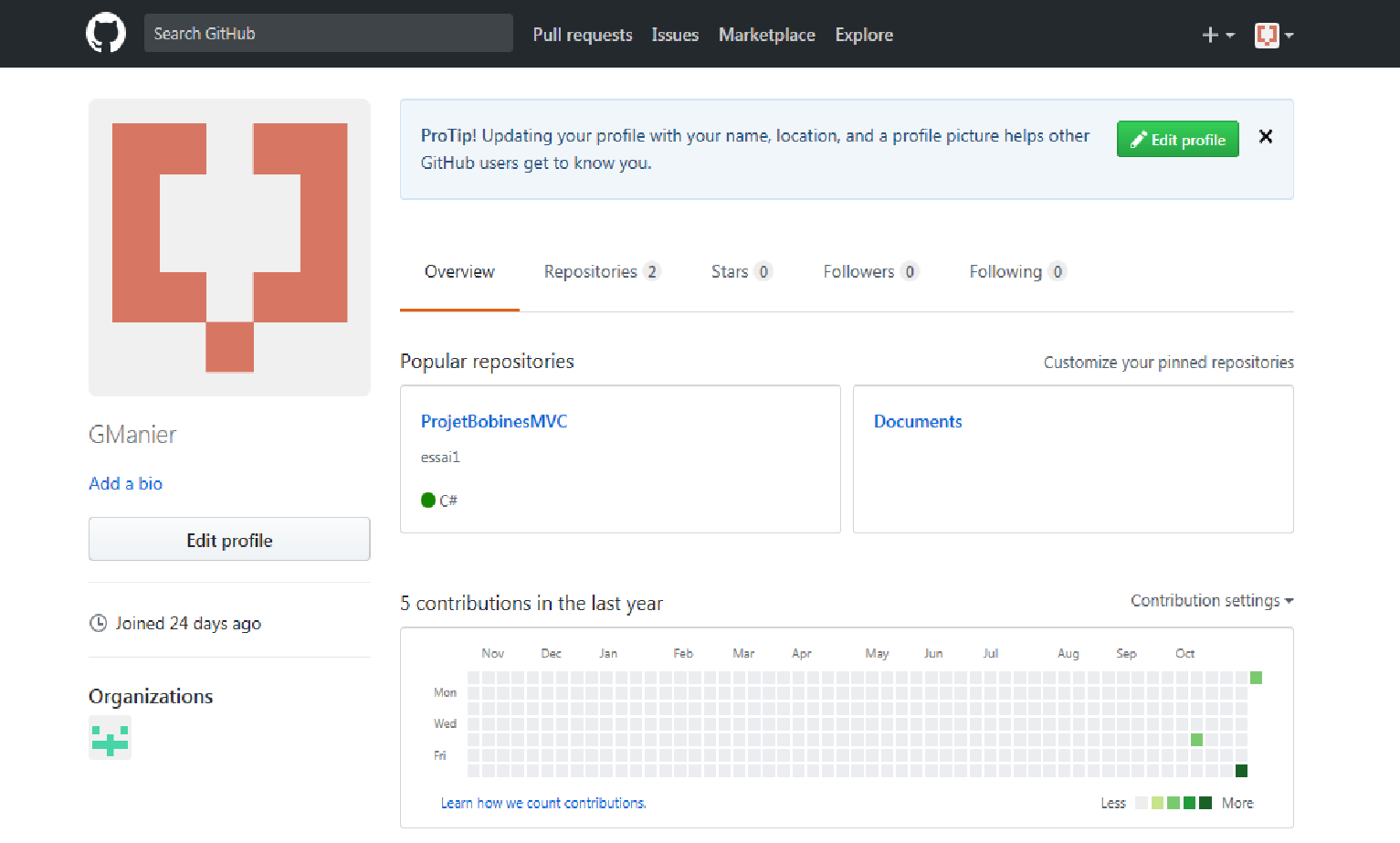
Sur le marché il existe plusieurs utilitaires de gestion de code sources et gestion des versions de fichiers. Pour n’en citer que quelques un, nous avons TFS (Team Foundation Server), Visual sourceSafe, Git, GitHub et BitBuchet.

Dans mon entreprise nous utilisons TFS (Team Foundation Server) qui permet de gérer les différents projets des différentes usines. Ce serveur est géré par nos équipes supports d’infrastructure et de développement logiciel. Pour des raisons d’accès à nos serveurs et à nos réseaux d’entreprise je n’ai pas choisi cet utilitaire pour la gestion des sources de mon projet.

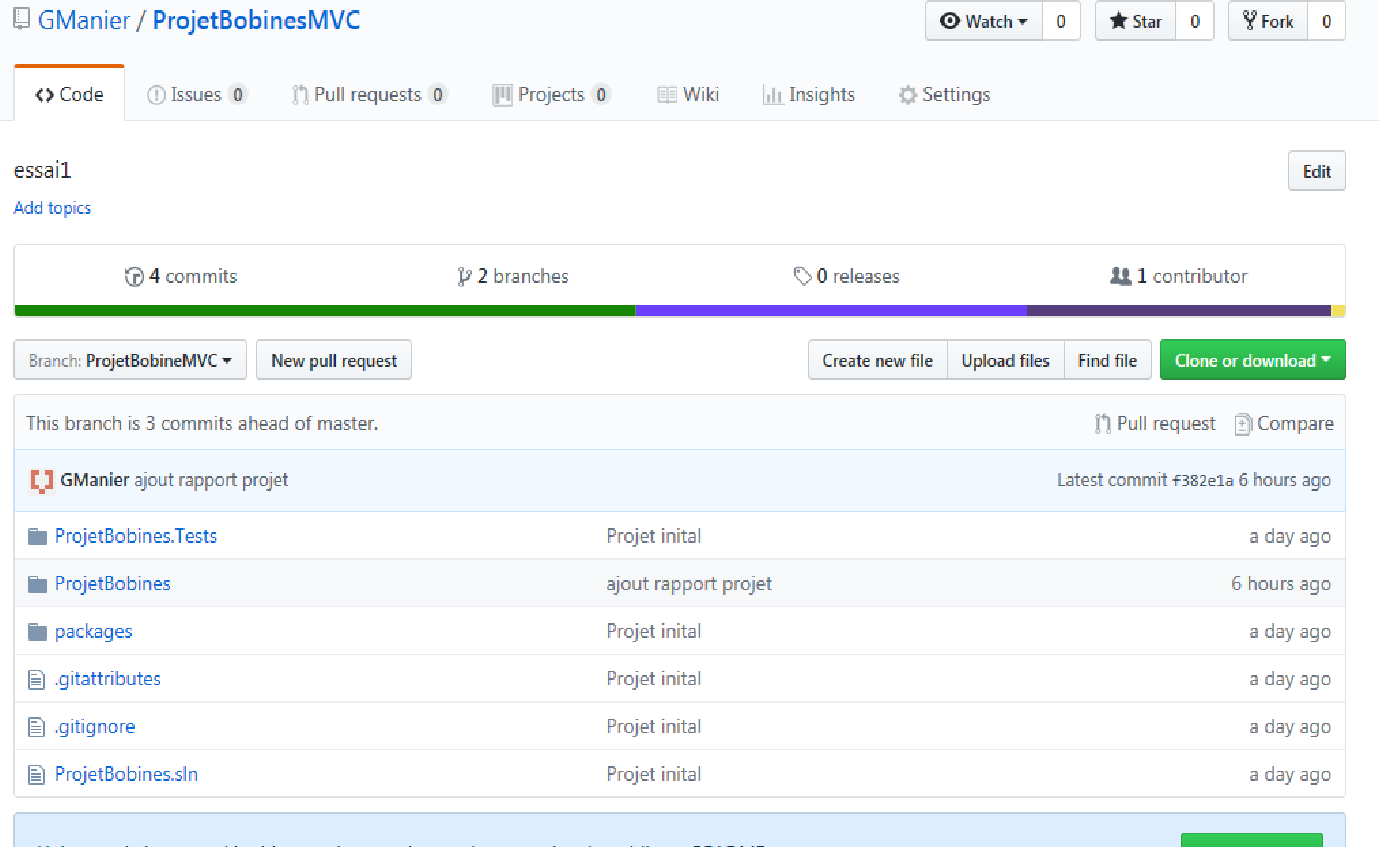
Comme Mon tuteur de projet devait également pouvoir au code source et aux documents que je créé, j’ai opté pour GitHub. L’avantage est qu’il s’interface directement avec Visual Studio 2013 (VS2013) et que l’extraction ou l’archivage des fichiers se fait naturellement avec la même souplesse que TFS.

### GitHub

Le choix s’est donc porté sur ce service web d’hébergement et de gestion pour le développement de logiciel. Pour pouvoir utiliser ces services j’ai donc créé un compte pour lequel j’ai dû spécifier s’il s’agissait d’une utilisation privée (Attention maintenant) ou public.



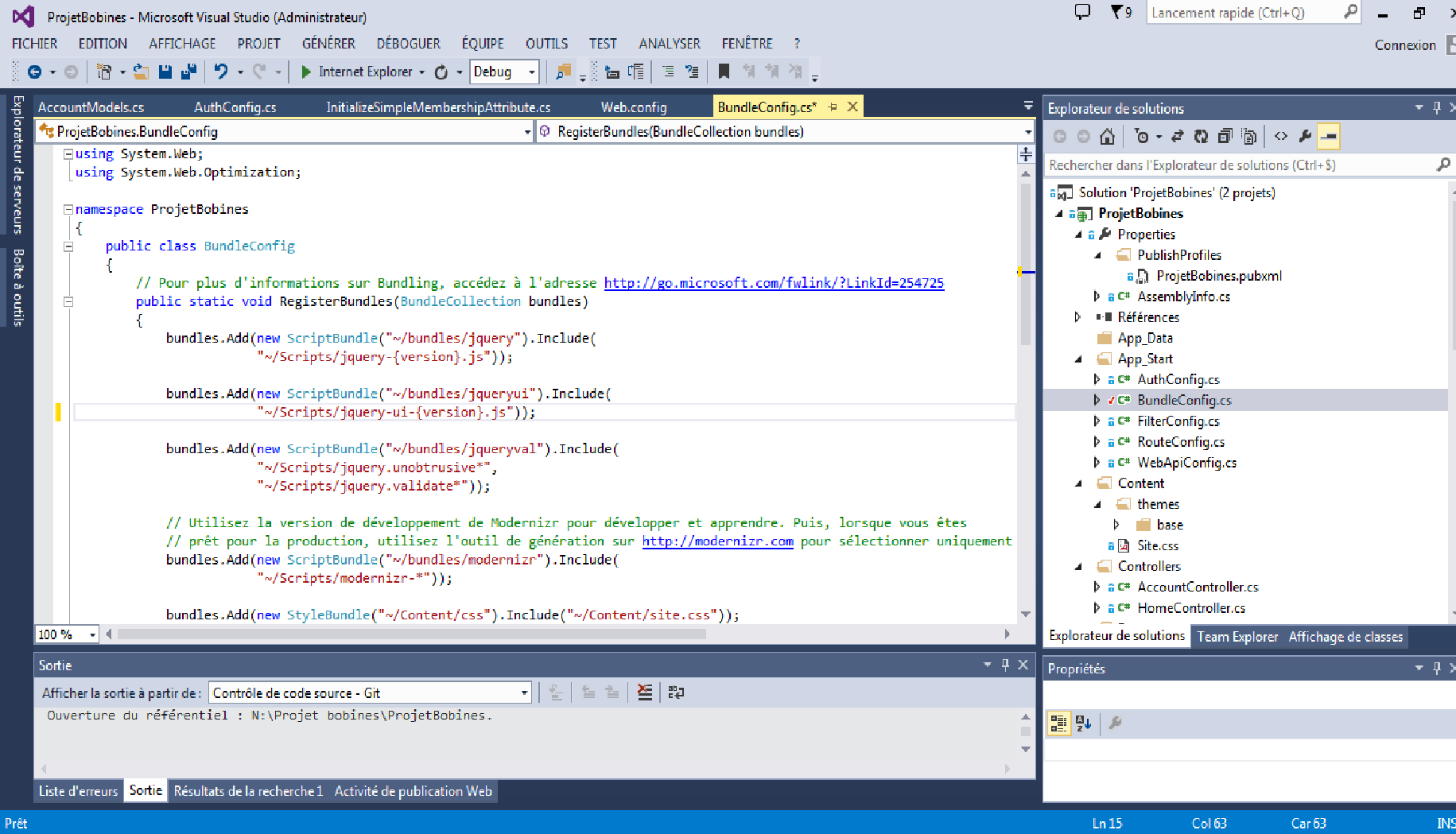
Une fois le compte créé, j’ai dû définir des « Repositories » afin de pouvoir y inclure les différentes solutions Visual Studio du projet. Il est possible ensuite de créer des branches de développements afin de gérer les versions en fonction soit des développements en cours, soit en recette, soit en production.



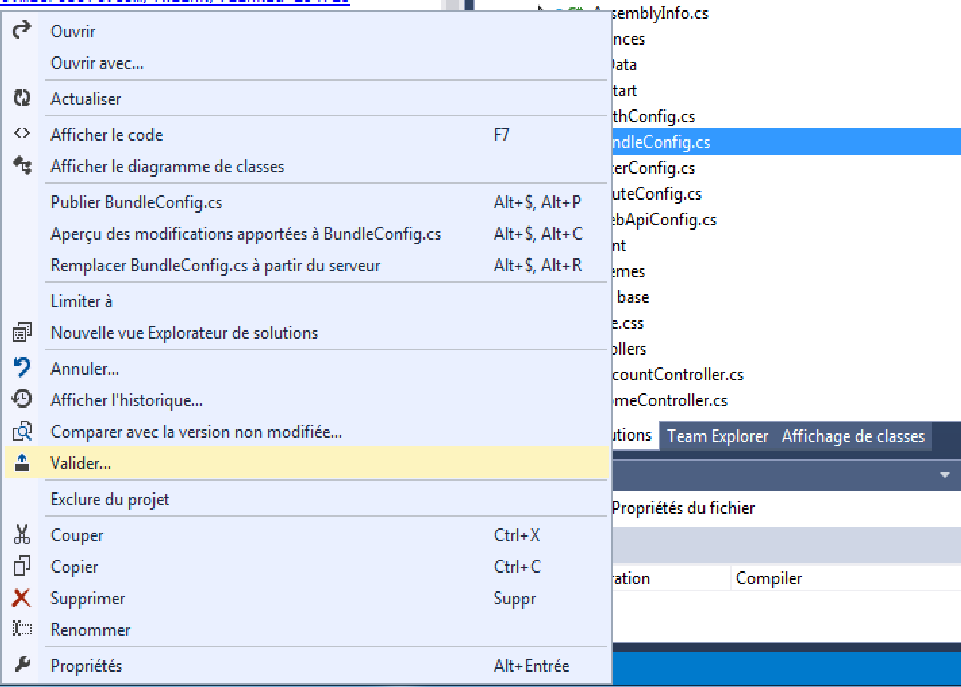
L’utilisation avec Visual Studio 2013 est assez simple car celui-ci propose naturellement 2 gestionnaires de code source, TFS ou Git.

En sélectionnant Git, il faut déclarer son compte GitHub puis le « repositories » et branche sur laquelle on souhaite archiver ses projets.

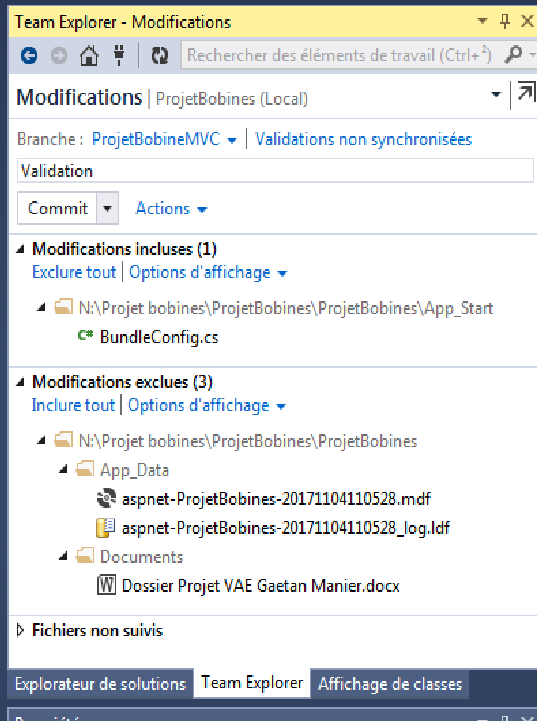
Dans VS2013 les fichiers extraits sont indiqués par une coche en rouge et ceux non extraits sont avec un cadenas bleu.



Pour archiver ses modifications il faut cliquer sur le fichier extrait, ou l’ensemble du projet et sélectionner « valider ».



Cette action nous envoie directement sur l’onglet « Team explorer » où il faut mettre un commentaire et cliquer sur commit.



Cela a pour effet d’archiver localement les modifications. Pour pouvoir les archiver sur le service web GitHub et partager ces modifications il valider la synchronisation.



## Les fiches de recettes

Pour tester les applications informatiques plusieurs méthodes existent comme les fiches de recette, les tests unitaires.

### Les fiches de recette

Elles permettent au fournisseur et au client de valider le bon fonctionnement de l’application en fonction des attendus souhaités et de ce qui a été écrit dans le cahier puis dans l’analyse fonctionnelle du projet. Les fiches de recette sont principalement écrites par le client dans lesquelles plusieurs scénarios sont écris.

Le client passe alors en revue l’ensemble de l’application afin de tester la globalité de son comportement. Cela peut paraît parfois fastidieux surtout si le client reprend l’intégralité des fiches de recettes des qu’il a repéré un ensemble de non conformités dans le souci de s’assurer qu’il n’y ait pas d’effet de bord suite aux dernières corrections.

Dans mon projet j’ai réalisé les recettes suivantes par exemple :

* Consultation des données des produits :
  + Action  : Sélection d’un produit dans la liste
  + Résultat  : Affichage des différentes colonnes du produit
* Création d’un produit :
  + Action  : Accès à l’écran de création en respectant les droits requis

Remplissage des différents champs et validation des données

* + Résultat  : Vérifier que le produit a bien été créé en base de données.

### Les tests unitaires

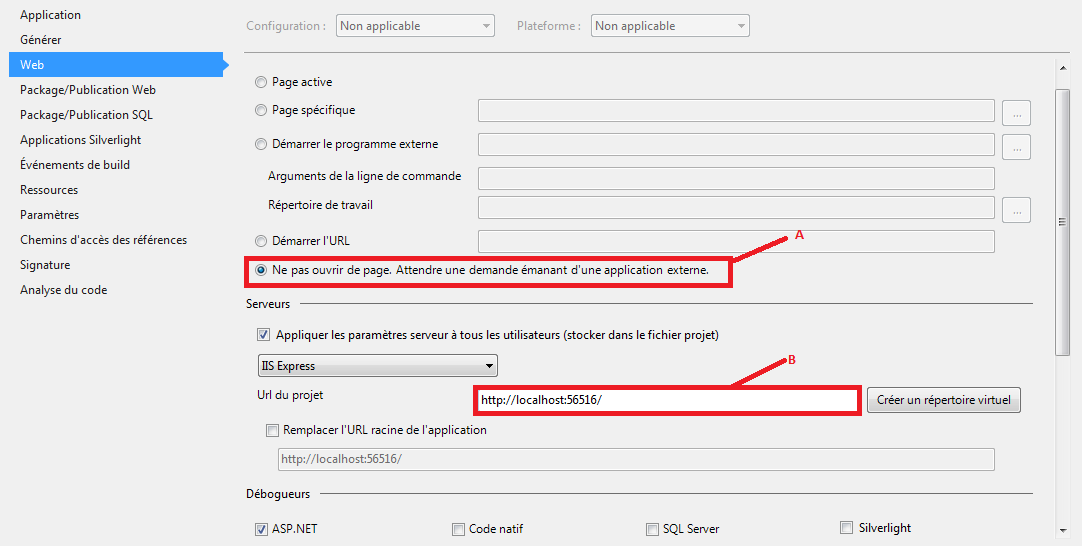
L’utilisation du modèle MVC permet d’implémenter des tests unitaires dans les applications WEB. Lors de la création de mon application j’ai également créé un projet de test unitaire qui permet de tester dans un premier temps le contrôleur de compte et le contrôleur Home.

Les tests unitaires ASP.NET.MVC appellent directement les méthodes des contrôleurs MVC créés dans l’application.

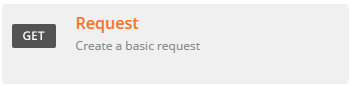
Pour tester l’application Web, j’ai utilisé les bonnes fiches de recettes classiques car elles s’y prêtaient bien du fait qu’il y avait des interfaces de visualisation des données après chaque requête.

Pour les tests unitaires de l’application Web Api j’ai donc utilisé POSTMAN. L’utilisation de POSTMAN est relativement simple et j’ai procédé de la façon suivante.

Il faut dans un premier temps lancer l’application via Visual Studio en ayant déclaré le projet comme projet de démarrage et en ayant paramétré les propriétés de l’application comme suit :

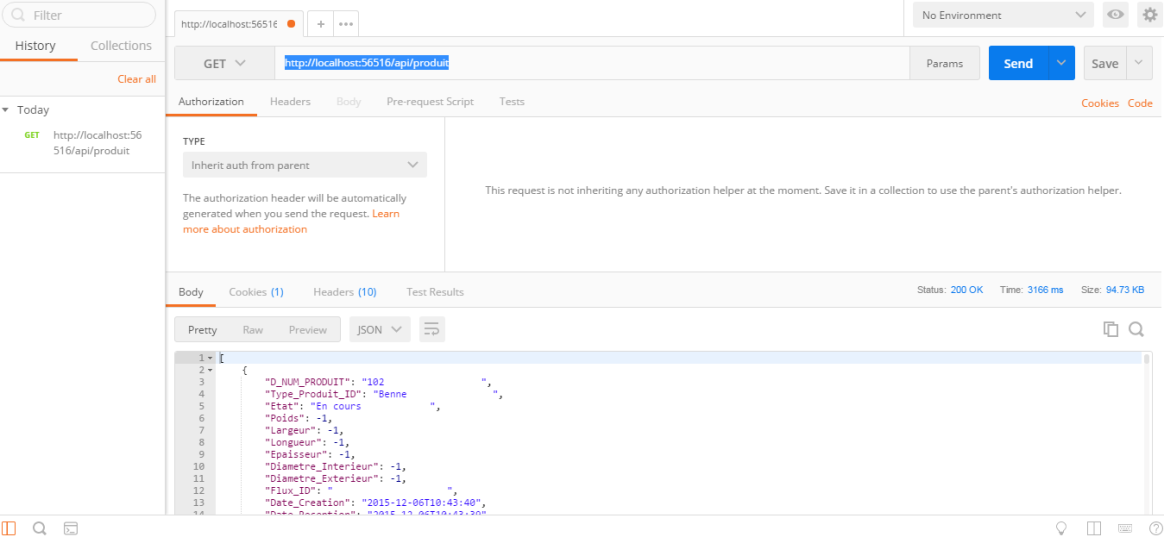


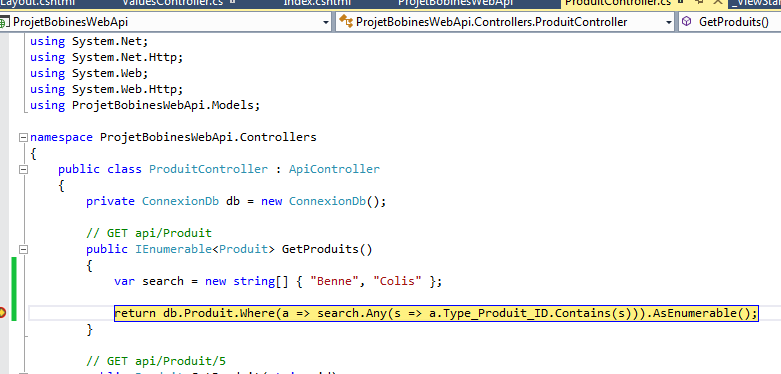
Le repère (B) représente l’URL qui sera utilisée dans POSTMAN pour effectuer le recettage de la page d’accueil de l’application (voir Annexe 4 page TODO).

Puis j’ai lancé POSTMAN (après l’avoir téléchargé du net) en prenant soin de sélectionner au démarrage. 

Une fois POSTMAN ouvert, j’ai renseigné l’URL <http://localhost:56516/api/produit> de l’application pour tester la liste des produits.

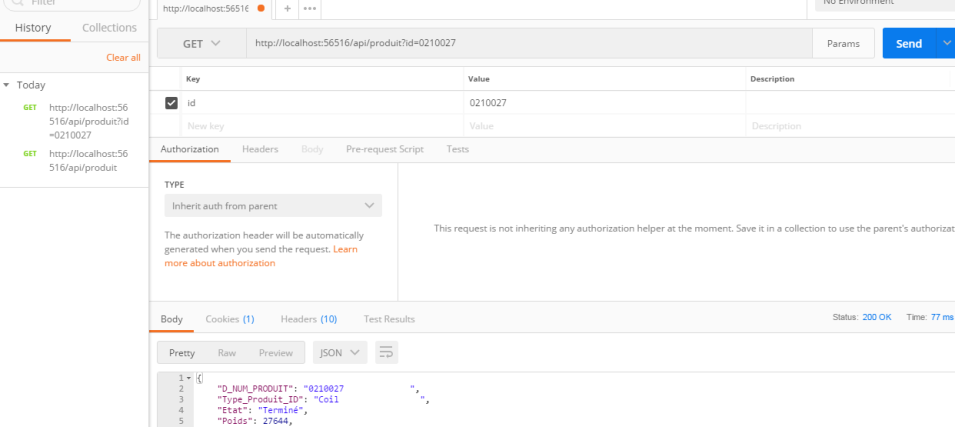
Le choix  et l’appuie sur le bouton  exécute la méthode Get du contrôleur Produit. Le résultat de la requête se retrouve en bas de POSTMAN avec une liste des données présentent dans le message JSON qui sera par la suite envoyé au client (web ou Mobile).





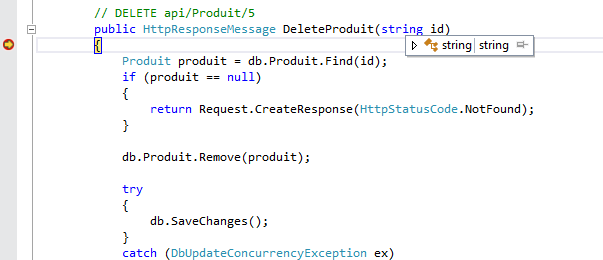
En mettant un point d’arrêt dans la méthode GetProduit du contrôleur, on voit bien que le programme passe par cette méthode lors du test avec POSTMAN.

Pour pouvoir tester la méthode de recherche d’un produit en fonction de son numéro, j’ai rajouté un paramètre nommé « id » et dont la valeur correspond à un numéro de bobine connu dans POSTMAN.



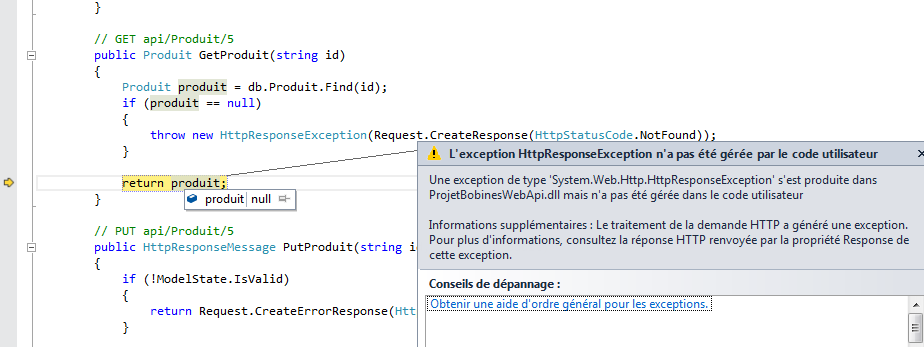
Pour tester la suppression d’un produit dans l’application web API, j’ai sélectionné dans la liste déroulante le  et j’ai rajouté en paramètre nommé « id » avec comme valeur la bobine de notre exemple précédent. L’envoie de la commande par  donne ceci :





En mettant un point d’arrêt dans la méthode DeleteProduit du contrôleur, on voit bien que le programme passe par cette méthode lors du test avec POSTMAN.

La suppression du produit a bien été réalisée et pour le vérifier il suffit de refaire le teste précédent avec le Get. Une exception apparaît dans Visual Studio, indiquant que le produit est nul.



Pour les autres méthodes de l’application Web API, il faut réitérer les tests en prenant le même principe.

POSTMAN permet de conserver un historique des différents tests effectués et ainsi de pouvoir rejouer cet ensemble de tests soit lors de modification du contrôleur, soit en faisiant du « copié /collé » et en jouant sur le lien HTML pour tester d’autres contrôleurs.

### Conclusion

En début de projet j’ai eu des doutes sur la réelle utilité des fonctions de tests car cela demande des temps de développements supplémentaire, surtout quand on utilise les fonctions de tests unitaires fournis avec Visual Studio.

Il faut presque développer une application en double pour intégrer l’ensemble des tests alors que sur l’application web il suffit d’utiliser les fiches de recette classique.

Par contre pour le recettage de l’application Web Api cela a tout son sens car il n’y a pas d’interface utilisateur créée et donc il faut pouvoir tester les différentes méthodes. L’utilisation de POSTMAN a grandement facilité ces recettes grâce a sa prise en main facile et aux différents didacticiels trouvé sur le Net.

Dans ce projet le temps passé pour effectuer les tests et la prise en main des outils de Test de Visual Studio et de POSTMAN représente environ 20 heures.

# Conclusions

Difficultés vue les techno exmployée

Surmonté les pb doc internet et travail

Apport sur la vision que l on peut avoir pour construire nos nouvelles applications. Proposer des solutions technqiues aux equipes supports.

Nos clients on de plus en plsu besoin de rapports divers afin d’améliorer leurs process. Nous utilisons principalement SAPBi4 pour la partie reporting mappée sur une base de données de type serveur de données technique. L’apporche faite dans ce projet donne une autre dimension à ce que je peux proposer en terme de reporting.

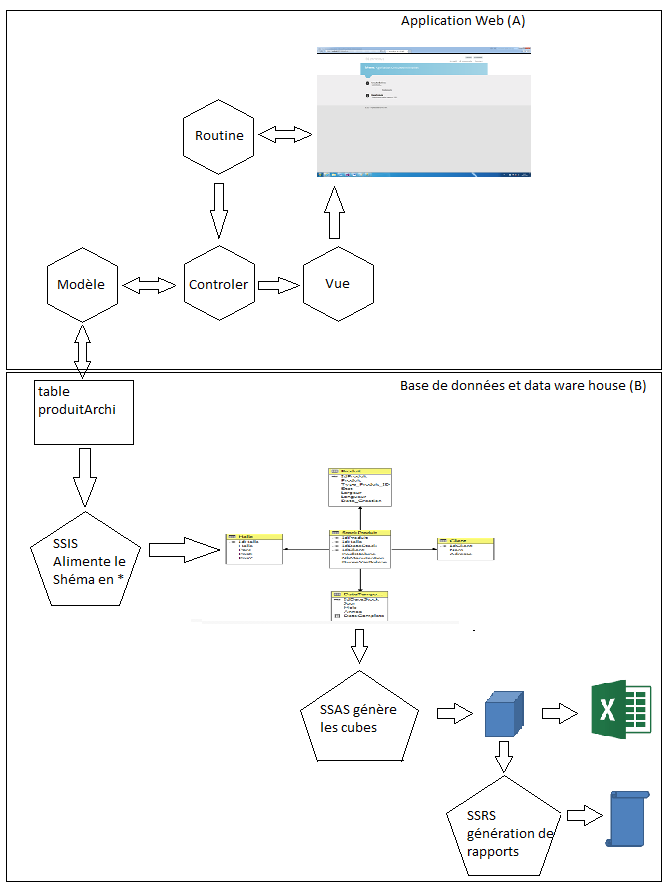
Projet relativement long et complet nécessitant 300h de travail dont 40 h pour la redaction du rapport.

# Annexes

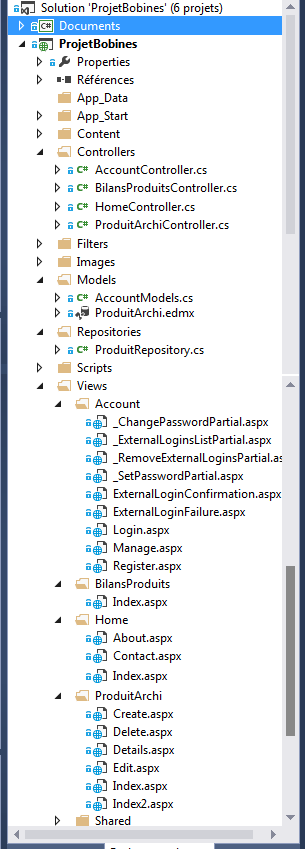
## Annexe 1 : Arborescence des vues



## Annexe 2 : Architecture globale



## Annexe 3 : Architecture de l’application



## Annexe 4 : POSTMAN, test de la page d’accueil



