Thomas **Petillon**

les Cahiers du Programmeur

ASP.NET

Infrastructure Web d'une PME avec ASP.NET

Avec la contribution de Martine Chalmond

EYROLLES

ÉDITIONS EYROLLES
61, Bld Saint-Germain
75240 Paris Cedex 05
www.eyrolles.com

Le code de la propriété intellectuelle du 1er juillet 1992 interdit en effet expressément la photocopie à usage collectif sans autorisation des ayants droit. Or, cette pratique s'est généralisée notamment dans les établissements d'enseignement, provoquant une baisse brutale des achats de livres, au point que la possibilité même pour les auteurs de créer des œuvres nouvelles et de les faire éditer correctement est aujourd'hui menacée.

En application de la loi du 11 mars 1957, il est interdit de reproduire intégralement ou partiellement le présent ouvrage, sur quelque support que ce soit, sans autorisation de l'éditeur ou du Centre Français d'Exploitation du Droit de Copie, 20, rue des Grands-Augustins, 75006 Paris.

© Groupe Eyrolles, 2003, pour la présente édition, ISBN : 2-212-11210-6.

Avant-propos

Quel est l'objectif de cet ouvrage ?

Les développeurs Web ont probablement tous entendu parler d'ASP.NET. Néanmoins, nombreux sont ceux qui hésitent encore à franchir le pas vers cette nouvelle technologie, qui constitue pourtant une évolution majeure pour la réalisation des applications Web :

- les pages Web compilées et implémentées en langage objet apportent des améliorations notables en termes de performance et robustesse ;
- l'utilisation de contrôles graphiques encapsulant la génération du HTML augmente drastiquement la productivité du développement et les possibilités de réutilisation ;
- le mécanisme de gestion des événements permet d'implémenter plus facilement des pages Web interactives, sans avoir recours à des scripts clients ;
- concepteurs graphiques et développeurs peuvent travailler simultanément sur le même projet, grâce à un mécanisme de séparation de la présentation et du code ;
- un ensemble de mécanismes applicatifs fournis en standard (sécurité, gestion des erreurs, internationalisation) facilite la tâche au développeur ;
- le déploiement des applications Web peut désormais s'effectuer par simple copie de fichier, sans nécessiter l'enregistrement de composants sur le serveur ;
- enfin, la bibliothèque du framework .NET offre un nombre imposant de classes utilitaires : accès à des sources de données, gestion de documents XML, mise en œuvre de services Web, etc.

Ce livre a pour objet de présenter de manière pragmatique le champ des nouvelles possibilités offertes par ASP.NET à travers une étude de cas : la mise en place d'une infrastructure d'échanges de données via le Web pour une PME spécialisée dans la vente par correspondance.

À qui s'adresse cet ouvrage ?

Cet ouvrage s'adresse à tous les lecteurs désireux de découvrir la technologie ASP.NET :

- aux développeurs Web, utilisateurs d'ASP, PHP, ColdFusion ou JSP, qui vont voir leurs possibilités décuplées ;
- aux développeurs habitués aux environnements client-serveur (Visual Basic, C++, Java...) qui vont enfin pouvoir effectuer des développements Web avec un niveau de puissance conforme à leurs habitudes ;
- aux chefs de projets désireux d'avoir une vision générale et pragmatique de ce que permet ASP.NET ;
- d'une manière générale, à tous les lecteurs curieux de découvrir les possibilités offertes par cette nouvelle technologie.

Exemples de code

Les exemples de code sont présentés dans deux langages : C# et VB.NET.

Pour profiter au mieux des exemples présentés, il est préférable d'avoir des notions sur le développement Web et les bases de données. En revanche, la connaissance des langages VB.NET et C#, utilisés lors de l'étude de cas, n'est pas préalablement requise : on présente en annexe un résumé rapide des principales règles syntaxiques de ces langages, qui sont facilement assimilables par tout développeur pratiquant déjà un langage de programmation.

Configuration logicielle requise

Pour pouvoir reproduire les exemples présentés dans cet ouvrage, la configuration logicielle suivante est nécessaire :

- Windows 2000 ou Windows XP (toutes versions) ;
- ASP.NET (téléchargement gratuit sur www.asp.net) ;
- environnement de développement Web Matrix (gratuit sur www.asp.net) ;
- moteur de base de données MSDE (gratuit sur www.asp.net).

Choix des outils

Les lecteurs qui le souhaitent pourront, s'ils le préfèrent, utiliser Visual Studio.NET à la place de Web Matrix et Microsoft Access ou SQL Server à la place de MSDE.

Sujets couverts par cet ouvrage

Le fil conducteur de cet ouvrage est la réalisation d'un intranet pour une PME.

La société « Les Savons du Soleil », petite entreprise spécialisée dans la vente par correspondance de produits cosmétiques, est répartie géographiquement sur plusieurs sites et éprouve de ce fait des difficultés dans ses échanges d'informations au quotidien.

Elle a par conséquent décidé de mettre en place une base de données centralisée, accessible via une interface Web et mise à jour automatiquement depuis l'usine marseillaise afin de permettre aux départements logistiques et marketing, situés à Paris, de travailler plus efficacement.

Elle a choisi d'effectuer ces développements avec la technologie ASP.NET qui permet de couvrir tous ses besoins, de la consultation et la mise à jour de la base aux échanges XML avec les fournisseurs, en passant par la sécurisation des accès et l'implémentation d'un service Web de mise à jour de la base.

Cette étude de cas sera l'occasion de faire un très large tour d'horizon des possibilités offertes par la technologie ASP.NET :

- dans le chapitre 1, nous effectuons une présentation détaillée de l'étude de cas et nous exposons les justifications techniques du choix d'ASP.NET, de la base de données MSDE et de l'environnement de développement Web Matrix ;
- le chapitre 2 décrit l'installation des outils de développement et la mise en place de la base de données utilisée par l'étude de cas, en présentant au passage l'interface de gestion des données de Web Matrix ;
- le chapitre 3 est consacré à l'architecture d'une page ASP.NET : à travers la réalisation du squelette de l'application, nous illustrons la séparation du code et de la partie graphique (avec les deux variantes *code behind* ou *code in-line*), le rôle des contrôles serveur et la notion de contrôle utilisateur ;
- dans le chapitre 4, consacré au module de suivi des stocks, nous étudions comment accéder à des données de la base en lecture en utilisant la bibliothèque ADO.NET et le contrôle serveur `DataGrid`, dont on souligne au passage les nombreuses possibilités de paramétrage ; le mécanisme de gestion des événements d'ASP.NET et la notion de conservation de l'état (`ViewState`) sont également décrits ;
- le chapitre 5 est consacré au module de gestion des commandes fournisseur, lequel nécessite non seulement un accès en lecture à la base (qui sera cette fois mis en œuvre à l'aide du contrôle `Repeater`) mais aussi un accès en écriture (modification et ajout de commandes), qui sera implémenté à l'aide des classes `SqlDataAdapter` et `DataSet` ; nous abordons également dans ce chapitre les contrôles de validation et la gestion des transactions ;
- ce module est enrichi dans le chapitre 6 avec l'implémentation d'échanges XML avec les fournisseurs, qui permet d'aborder quelques-unes des nombreuses possiblités d'ASP.NET sur le sujet : génération de fichiers XML, application d'une transformation XSL, envoi de fichiers par messagerie ;
- le chapitre 7 présente la notion de contrôle serveur spécifique à travers l'implémentation du module d'analyse des ventes : la répartition des ventes par régions est représentée graphiquement à l'aide d'un composant du marché ; puis, un contrôle serveur de type graphique composé de secteurs colorés est implémenté pour afficher les ventes par famille de produits ;
- dans le chapitre 8, nous réalisons un service Web permettant la mise à jour des données de stocks depuis un système externe, puis nous implémentons un contrôle utilisateur qui réalise l'affichage des informations météorologiques fournies par un service Web externe ;
- enfin, le chapitre 9 passe en revue les aspects liés à la mise en production de l'application : sécurisation, configuration, internationalisation, analyse des performances, gestion spécifique des erreurs et déploiement.

Étude de cas en ligne

La version en ligne et les sources de l'étude de cas sont disponibles sur les sites d'accompagnement aux adresses :

- http://www.savonsdusoleil.com
- http://www.editions-eyrolles.com

Les lignes de code réparties sur plusieurs lignes en raison de contraintes de mise en pages sont signalées par la flèche ➥.

Les chapitres étant relativement indépendants, le lecteur qui le souhaite peut aborder directement les sujets qui l'intéressent, bien qu'il soit conseillé d'avoir préalablement lu la présentation de l'étude de cas et mis en place les outils de développement et la base de données.

Le chef de projet tirera probablement avantage de la lecture de l'ensemble des chapitres, même s'il ne rentre pas dans le détail des codes proposés ; en outre, un résumé de synthèse est systématiquement inclus à la fin de chaque chapitre.

À l'issue de cette étude de cas, le lecteur aura mis en œuvre une application concrète qui lui aura permis – du moins, nous l'espérons – d'acquérir une bonne vision d'ensemble des possibilités d'ASP.NET et des mécanismes fondamentaux de cette technologie.

Remerciements

Je tiens à remercier toutes les personnes qui ont rendu possible la parution de cet ouvrage et avec qui ce fut un réel plaisir de travailler : Aurélie, pour la pertinence de ses relectures, Sophie et Anne, pour leurs précieux conseils et leur efficacité, Martine et Jean-Marie, pour le formidable travail réalisé et enfin Muriel, qui a réussi à me convaincre de tenter l'aventure !

Thomas PETILLON

petillon@topic.fr

www.topic.fr

Table des matières

L'étude de cas « Les Savons du Soleil »

1

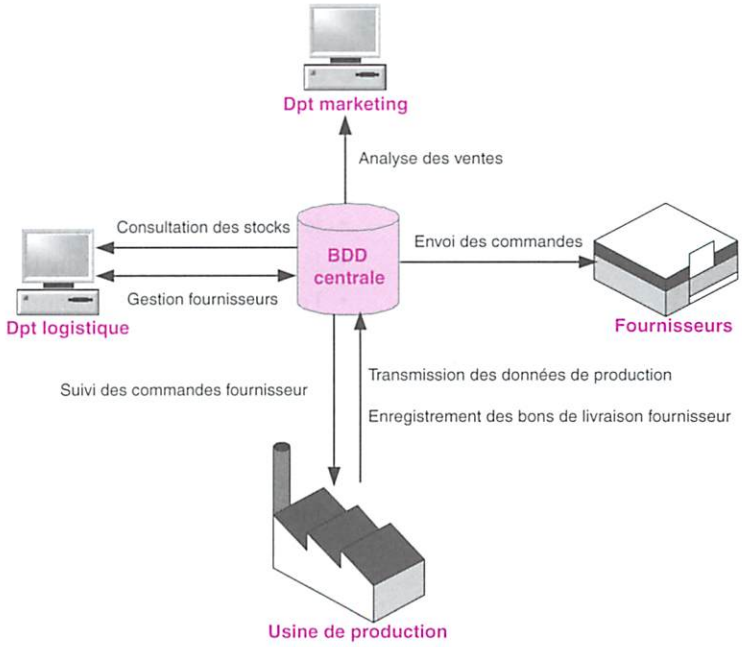
ASP.NET

Base de données | Intranet | Échanges XML | Services Web | ASP.NET | MSDE | Web Matrix

SOMMAIRE

MOTS-CLÉS

- Base de données centralisée
- Intranet
- Échanges XML
- Services Web
- ASP.NET
- MSDE
- Web Matrix



Dans ce chapitre, on présente en détail l'étude de cas qui servira de fil conducteur au reste de cet ouvrage : après avoir exposé les difficultés de circulation d'informations entre les différents sites de notre société exemple, on explique comment la mise en place une base de données centralisée régulièrement mise à jour depuis l'usine et dotée d'une interface de gestion adéquate peut améliorer la situation. Enfin, on détaille et on justifie les choix techniques retenus pour la réalisation de l'étude de cas : ASP.NET, MSDE et Web Matrix.

Une PME géographiquement dispersée

La société des Savons du Soleil est une PME spécialisée dans la vente par correspondance de cosmétiques à base de produits naturels : savons au miel, shampoing au tilleul, bain douche à la lavande…

Les activités de la société sont réparties entre trois entités géographiquement séparées :

- *Marseille* : l'usine de production assure la fabrication et le stockage des produits, ainsi que le traitement des commandes (prise de commande, préparation et envoi des colis) ;
- *Lyon* : le département logistique assure le suivi des stocks et les commandes de matières premières auprès des fournisseurs ;
- *Paris* : le département marketing travaille sur les actions publicitaires, le développement de nouveaux produits et l'analyse des ventes.

Des échanges d'informations fastidieux entre les sites

Les trois sites doivent échanger régulièrement des informations :

- le département logistique a besoin d'être renseigné sur l'évolution des stocks ;
- le département marketing doit analyser les chiffres de vente ;
- l'usine veut suivre les commandes passées par le département logistique.

Malheureusement, la remontée des informations depuis l'usine est complexe à gérer (figure 1–1), chaque unité possédant son propre système de gestion des données :

- l'usine de production utilise un logiciel de gestion commerciale dans lequel sont enregistrées toutes les commandes clients. À chaque fin de semaine, les informations relatives aux nouvelles commandes (coordonnées des clients, détail des produits commandés) sont extraites et transmises aux autres unités ;
- le département logistique suit l'évolution des stocks grâce à un tableur, dont le contenu est actualisé manuellement chaque semaine à partir des données fournies par l'usine ;
- le département marketing effectue ses analyses des ventes grâce à une base de données relationnelle, actualisée chaque semaine, elle aussi, à partir des informations de l'usine.

Quant à la gestion des commandes fournisseur – qui implique trois acteurs distincts – elle est, elle aussi, fastidieuse (figure 1–2) :

- le département logistique passe commande par télécopie auprès du fournisseur ;

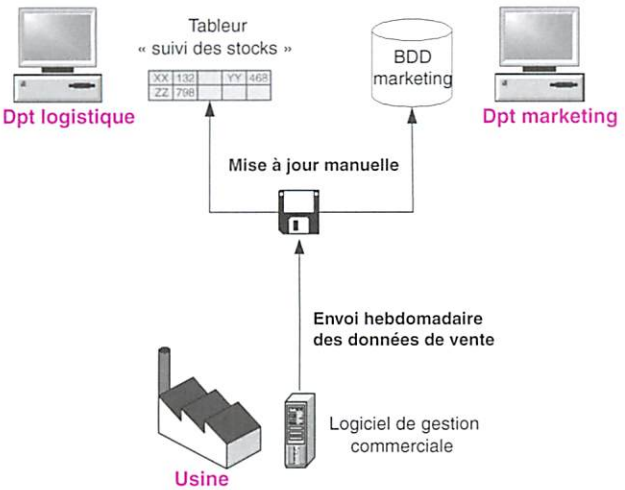


Figure 1–1 Transfert des données commerciales (avant)

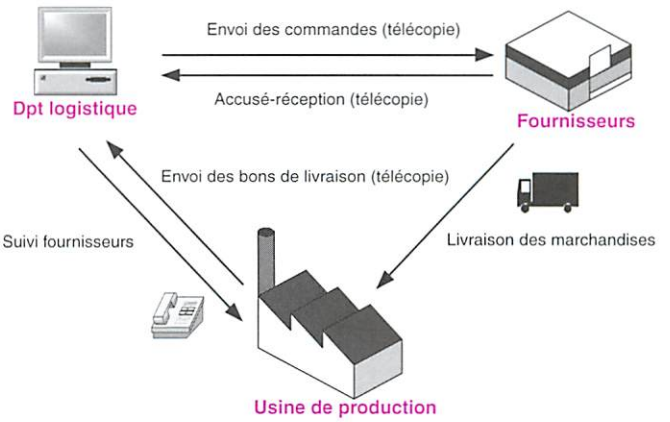


Figure 1–2 Gestion des commandes fournisseur (avant)

- le fournisseur accuse réception de la commande, par télécopie également ;
- le bon de livraison remis par le fournisseur lors de la livraison de la marchandise est envoyé par télécopie, depuis l'usine, au département logistique.

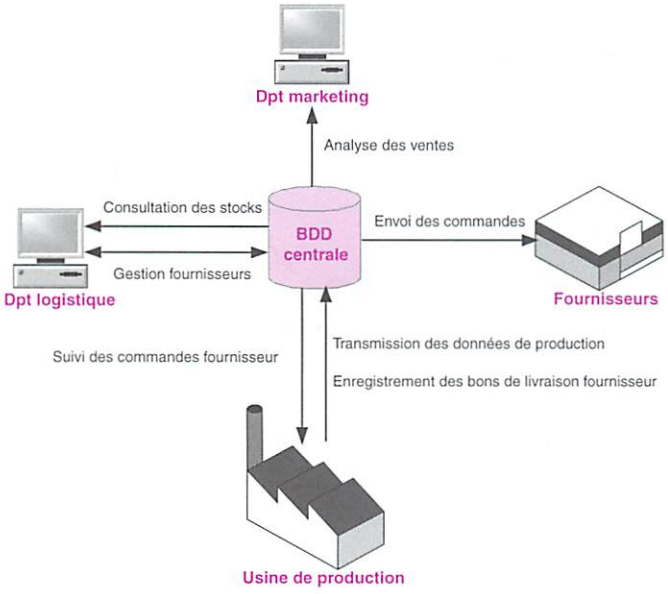
Inconvénients liés à la situation actuelle

La situation actuelle présente un certain nombre d'inconvénients :

- les départements marketing et logistique n'ont pas accès aux informations « en temps réel », ce qui n'est pas dramatique pour le premier, mais nettement plus gênant pour le second ;
- les opérations de mise à jour des données à partir des informations envoyées par l'usine sont longues et fastidieuses ; de plus, elles contraignent le département marketing à employer à mi-temps un administrateur de base de données ;
- le département logistique ne dispose que de moyens limités d'analyse et souhaiterait disposer d'une base de données de suivi des stocks ;
- la gestion des commandes fournisseur est entièrement manuelle et pénible ;
- l'usine de production est contrainte de téléphoner au département logistique chaque fois qu'elle désire obtenir des informations sur les commandes fournisseur en cours.

En résumé, le fait d'avoir des sources d'information décentralisées est pénalisant pour l'activité de l'entreprise.

Le projet de nouvelle infrastructure technique



Pour augmenter sa productivité au quotidien, la société des Savons du Soleil a donc décidé de mettre en place une nouvelle infrastructure technique :

- mise en place d'une base de données centralisée ;
- accès sécurisé à cette base pour toutes les unités via une interface Web permettant d'effectuer le suivi des stocks et des commandes fournisseur, ainsi que l'analyse des ventes ;
- mise à jour automatique et quotidienne de la base, à partir d'informations extraites des fichiers de gestion commerciale utilisés à l'usine ;
- nouveau mécanisme d'échange avec les fournisseurs, fondé sur XML.

Dans les sections qui suivent, nous détaillons les différentes composantes du projet :

- structure de la base de données ;
- interface de gestion de la base ;
- transferts de données automatisés (échanges fournisseur, mise à jour des données depuis l'usine).

Structure de la base de données

Pour simplifier la présentation de l'étude de cas – dont la base n'est pas le sujet principal – on se concentrera uniquement sur le modèle de données. Bien entendu, dans le contexte d'une application réelle, il faudrait aussi aborder d'autres sujets comme les règles d'intégrité référentielles (index, contraintes, déclencheurs), l'optimisation des performances, la gestion des transactions, la sécurité (authentification, rôles), la disponibilité de la base ou encore son administration.

DANS UN CAS RÉEL

Axes d'analyse plus nombreux

Dans un cas réel, les axes d'analyse des ventes seraient beaucoup plus nombreux (par produits, par types de client, etc.). De même, le suivi de la logistique serait probablement plus complexe (gestion des stocks de produits intermédiaires...).

Mise en place d'une base de données centralisée

L'intérêt de mettre en place une base centrale est de disposer d'un référentiel de données unique partagé entre toutes les unités : on évitera ainsi les pertes de temps dues à la mise à jour manuelle des données de suivi des stocks et d'analyse des ventes, et on donnera à tous l'accès à une information à jour. Cette base, accessible via Internet, sera hébergée chez un prestataire externe.

Les données à stocker dans la base peuvent être réparties en deux sous-ensembles :

- données utilisées pour l'analyse des ventes et le suivi des stocks ;
- données utilisées pour la gestion des commandes fournisseur.

Données nécessaires à l'analyse des ventes et au suivi des stocks

Pour éviter une trop grande complexité, on se limitera aux hypothèses suivantes :

- le département marketing souhaite pouvoir consulter la répartition des ventes par famille de produits ou par région, pour un mois donné ;
- le département logistique souhaite pouvoir suivre l'évolution des stocks de produits finis.

Pour effectuer ces opérations, il faut au minimum stocker dans la base :

- la liste des produits ;
- la liste des familles de produits ;

- la liste des mouvements de stocks (arrivée ou sortie du stock) ;
- la liste des ventes mensuelles par famille de produits et par région.

Tableau 1–1 Données nécessaires à l'analyse des ventes et au suivi des stocks

Entité	Caractéristiques, relations
Produit	• Désignation et description du produit • Un produit est rattaché à une famille de produits
Famille de produits	• Nom de la famille de produits • À une famille de produits sont rattachés de 1 à n produits
Mouvement de stocks	• Date, type de mouvement, produit concerné et quantité (positive ou négative) • Un mouvement est rattaché à un produit
Vente	• Ventes par famille de produits et par région • Une donnée de vente est rattachée à une famille de produits et à une région

Le modèle physique correspondant, qui ne comporte aucune difficulté particulière, est présenté figure 1-3.

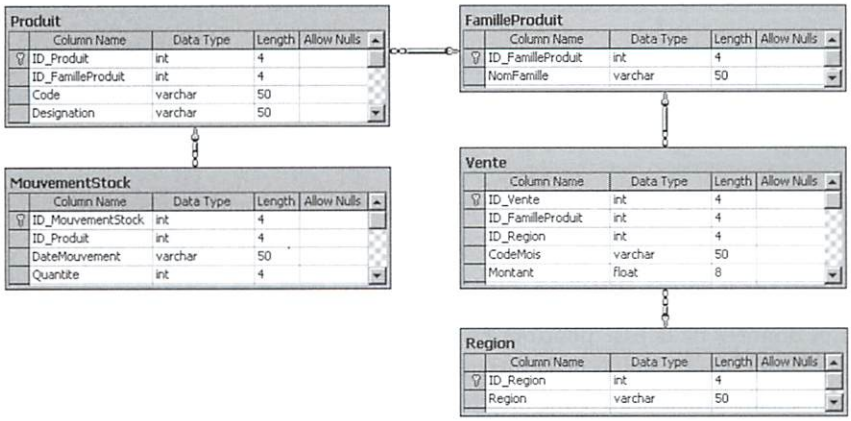


Figure 1–3 Modèle physique de la base : analyse des ventes et suivi des stocks

Notations

Par convention, toutes les clés commencent par « ID_ » comme « IDentifiant ».

Données nécessaires à la gestion des commandes fournisseur

Pour assurer le suivi des commandes fournisseur, le département logistique souhaite pouvoir garder trace des références fournisseur commandées (type et quantité), de la date de livraison prévue et du statut de la livraison (commande livrée ou non).

Par conséquent, il faut stocker dans la base :

- la liste des fournisseurs ;
- la liste des références fournisseur (produits proposés par ces fournisseurs) ;
- la liste des commandes passées auprès de ces fournisseurs (avec les lignes de commandes associées).

Tableau 1–2 Données nécessaires au suivi des commandes fournisseur

Entité	Caractéristiques, relations
Fournisseur	• Code fournisseur, nom et adresse • À un fournisseur sont associées de 1 à n références fournisseur
Commande	• Date création, date livraison prévue, date livraison effective • Une commande est rattachée à un fournisseur • 1 à n lignes de commandes lui sont associées
Ligne de commande	• Référence fournisseur, quantité commandée, quantité livrée • Rattachée à une commande
Référence fournisseur	• Référence et désignation du produit • Associée à un fournisseur

Le modèle physique correspondant est présenté figure 1-4.

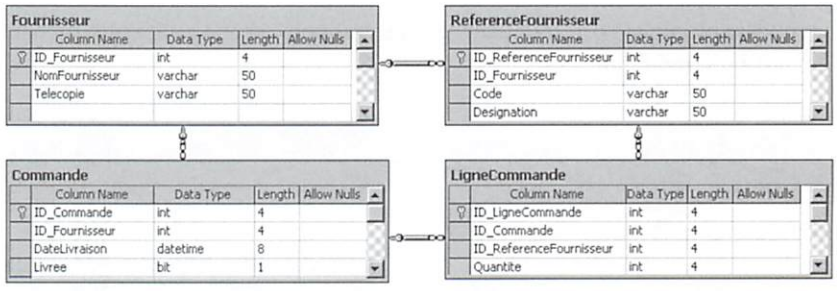


Figure 1–4 Modèle physique de la base : gestion des commandes fournisseur

Interface Web pour la consultation/mise à jour de la base

Les données de la base pourront être consultées et éventuellement mises à jour par les différentes unités de la société, en fonction de leurs besoins. Pour cela, les utilisateurs disposeront d'une interface de gestion permettant :

- le suivi des stocks ;
- la gestion de commandes fournisseur ;
- l'analyse des ventes.

Comme la base est hébergée sur un site externe, et que les utilisateurs se répartissent en trois endroits géographiquement distincts, ces opérations vont s'effectuer par l'intermédiaire d'une interface Web, dont l'accès devra être sécurisé (*intranet*).

Module de suivi des stocks

Le module de suivi des stocks doit proposer :

- l'affichage de la liste des produits avec les quantités en stock correspondantes, filtrée par famille de produits ;
- l'accès à la fiche de détail d'un produit (historique de la variation du stock).

Toutes ces opérations sont présentées figure 1-5.

Figure 1–5 Interface de suivi des stocks

Module de gestion des commandes fournisseur

Le module de gestion des commandes fournisseur doit permettre :

- la création d'une commande fournisseur ;
- l'affichage de la liste des commandes fournisseur ;
- l'enregistrement d'un bon de livraison fournisseur.

La création d'une commande fournisseur se déroule en plusieurs étapes (voir figure 1-6) :

1. Sélection du fournisseur.
2. Sélection des références, des quantités commandées et de la date de livraison souhaitée.
3. Enregistrement de la commande dans la base.
4. Transmission de la commande au fournisseur (sous forme de fichier XML).

La page de consultation des commandes fournisseur doit présenter une liste récapitulative des commandes : en sélectionnant une commande, on doit accéder au détail correspondant (références et quantité commandées, date de livraison prévue). Enfin, lors de la réception d'une commande à l'usine, il doit être possible de modifier le statut de la commande : « livrée » au lieu d' « en attente de livraison » (voir figure 1-7).

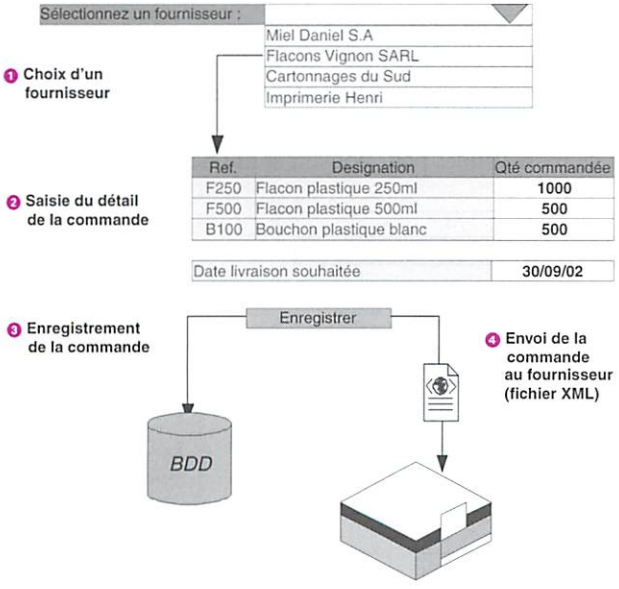


Figure 1–6 Création d'une commande fournisseur

Figure 1–7 Consultation des commandes fournisseur

Module d'analyse des ventes

Ce module doit permettre la consultation du chiffre d'affaires réalisé par famille de produits ou par région sur une période donnée, avec visualisation des résultats sous forme graphique.

La cinématique correspondante est présentée figure 1-8.

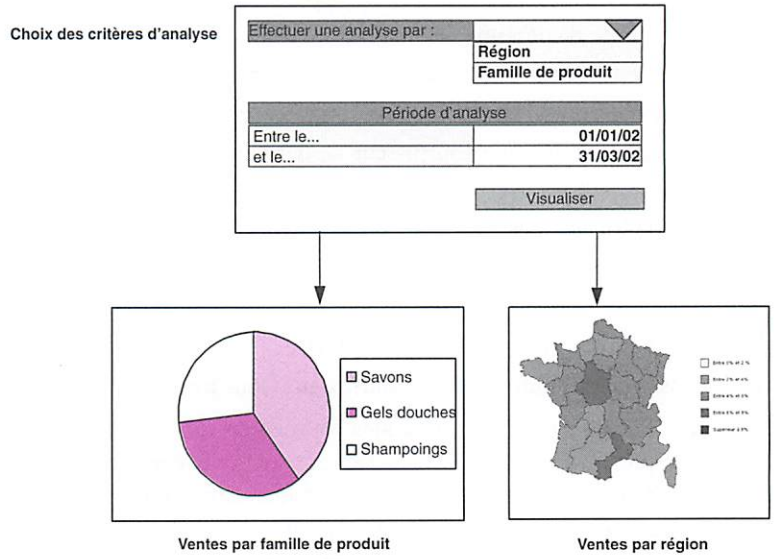


Figure 1–8 Interface d'analyse des ventes

Authentification

Cette base contient des données confidentielles qui ne doivent être consultées et modifiées que par les personnes qui y sont habilitées. Il est donc indispensable de prévoir un mécanisme d'authentification : les utilisateurs devront fournir un identifiant et un mot de passe valides pour pouvoir se connecter à l'application ;

DANS UN CAS RÉEL Sécurisation

Dans le cadre d'une application réelle, il faudrait passer en revue tous les aspects relatifs à la sécurité : protection du serveur Web par un pare-feu et un logiciel anti-virus, chiffrement des communications (SSL), authentification forte des utilisateurs, etc.

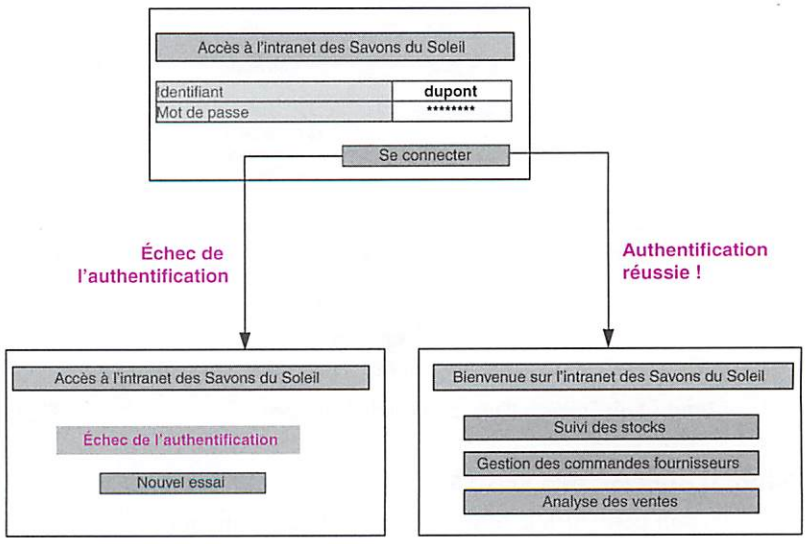


Figure 1–9 Authentification des utilisateurs

de plus, l'accès aux différents modules dépendra des droits de l'utilisateur (gestion des autorisations). Le mécanisme requis pour l'authentification est présenté figure 1-9.

Mise à jour automatique des données de la base

Le suivi des stocks et l'analyse des ventes n'auront de réel intérêt que si les données commerciales sont mises à jour régulièrement depuis l'usine : c'est pourquoi il est prévu de mettre en place un mécanisme de remontée d'informations depuis le logiciel de gestion commerciale vers la base centralisée.

Ce mécanisme reposera sur l'implémentation d'un service Web permettant la mise à jour de la base, qui sera appelé depuis l'usine de production (voir figure 1-10) ; dans un souci de simplification, on se limitera à la mise à jour des données de stocks.

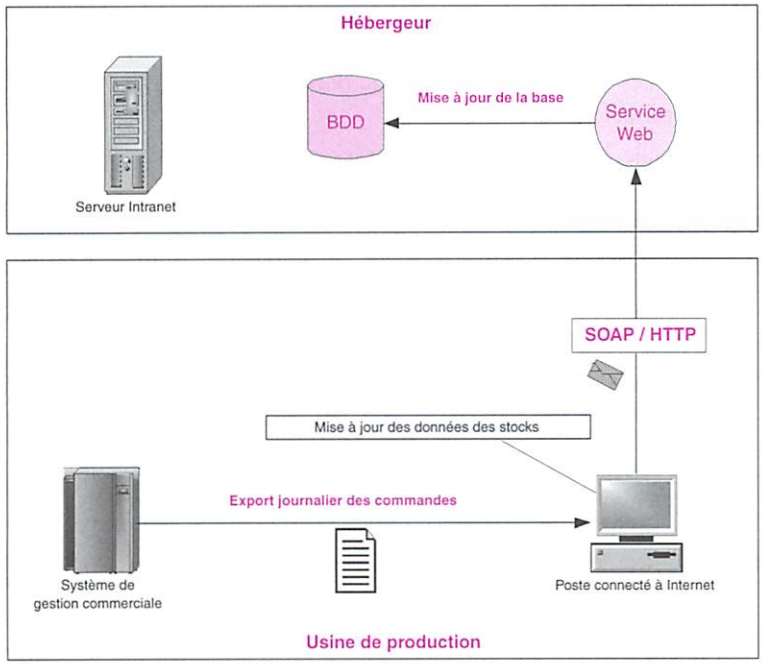


Figure 1–10 Mise à jour des données de stock via un service Web

Le cahier des charges étant établi, nous allons maintenant choisir les outils techniques qui vont permettre sa réalisation.

Utiliser un ERP ?

Pour répondre de manière exhaustive aux besoins actuels et futurs de la société, il pourrait être envisagé de mettre en place un progiciel de gestion intégrée (ERP) : néanmoins, cette option représenterait un coût et un délai de mise en place jugés trop importants pour la société de notre étude de cas.

Choix d'architecture technique : ASP.NET, MSDE et Web Matrix

Les utilisateurs étant situés dans des lieux distincts, équipés d'infrastructures techniques différentes, il est naturel de retenir une architecture de type intranet, reposant un serveur HTTP qui génère des pages dynamiques à partir d'informations contenues dans une base de données centralisée.

Ce type d'architecture nécessite de se déterminer sur les points suivants :

- choix de la technologie de développement Web ;
- choix de la base de données ;
- choix de l'environnement de développement.

Choix de la technologie de développement Web

La technologie de développement constitue le moteur d'une application Web : c'est elle qui détermine le mécanisme de génération dynamique des pages Web ainsi les systèmes d'exploitation, bases de données, langages et outils de développement utilisables.

À l'heure actuelle, les choix possibles sont les suivants (par ordre alphabétique) :

Technologie	Description succincte
ASP	Technologie de génération de sites Web dynamiques introduite par Microsoft en 1996, qui repose sur l'interprétation de scripts et l'activation d'objets ActiveX/COM côté serveur ; fonctionne uniquement sous Windows ; généralement utilisée avec Microsoft SQL Server.
ASP.NET	Nouvelle technologie de développement d'applications et de services Web, introduite par Microsoft en 2001, qui repose sur des pages compilées, des contrôles prédéfinis, un mécanisme de gestion événementielle et une infrastructure technique intégrant des fonctionnalités de sécurité, optimisation et gestion des erreurs ; fonctionne uniquement sous Windows 2000 et XP équipé de l'extension .NET Framework ; généralement utilisée avec Microsoft SQL Server.
ColdFusion	Nom désignant à la fois une technologie de type serveur d'application et un outil de développement, qui repose sur l'utilisation d'un langage spécifique (CFML : Coldfusion Markup Language) permettant d'activer des objets sur le serveur lors de l'interprétation d'une page Web ; fonctionne sous Unix, Linux, Windows, avec la majorité des serveurs HTTP et des bases de données ; historiquement développé par Allaire et récemment acquis par Macromedia (ColdFusion MX).
JSP	Technologie de génération de sites Web dynamiques développée par Sun, qui repose sur l'utilisation de pages compilées contenant du Java et l'activation d'objets serveurs (JavaBeans) ; fonctionne sur toutes les plates-formes équipées d'une machine virtuelle Java.
PHP	Langage de script open source créé en 1994, actuellement dans sa version 4, dont la syntaxe est inspirée de C / Perl et Java, permettant la génération de sites Web dynamiques ; fonctionne sur la majorité des plates-formes Unix, Linux et Windows et est compatible avec un grand nombre de bases de données ; couramment utilisé au sein du trio Linux/Apache/MySQL.

Aujourd'hui, toutes ces technologies ont atteint un niveau de maturité assez avancé et, sur le papier, toutes permettraient de répondre aux besoins exprimés dans notre cahier des charges.

Néanmoins, ASP.NET se distingue de ses concurrents par son architecture novatrice et son nouveau modèle de programmation qui permet la réalisation d'applications Web avec les mêmes méthodes de conception (découpage en objets métier et composants réutilisables), le même degré de robustesse et de puissance (orientation objet, typage fort des variables, compilation) et le même degré d'organisation du code (séparation entre code et contenu graphique, organisation en gestionnaires d'événements) qu'une application client-serveur classique. Pour en savoir plus, nous allons détailler ces différentes caractéristiques de la technologie, en commençant par répondre à une question simple : qu'est-ce qu'ASP.NET ?

Qu'est-ce qu'ASP.NET ?

En pratique, ASP.NET est une extension logicielle qui permet l'exécution d'applications Web (sites Web dynamiques, services Web). Disponible en téléchargement gratuit sur le site www.asp.net et destinée à être installée sur une machine équipée de Windows 2000/XP et d'un serveur HTTP, elle se compose des éléments techniques suivants :

- une extension du serveur HTTP, implémentée sous la forme de filtre ISAPI nommé `aspnet_filter.dll` (voir figure 1-11), qui permet de traiter spécifiquement les requêtes vers les pages comportant une extension particulière (notamment les fichiers `.aspx`, qui correspondent aux pages ASP.NET standards, voir figure 1-12) ;
- un processus principal nommé `aspnet_wp.exe` (pour : ASP.NET worker process) qui tourne en tâche de fond et traite les requêtes correspondant à des pages ASP.NET (figure 1-13), en s'appuyant lui-même sur trois éléments :

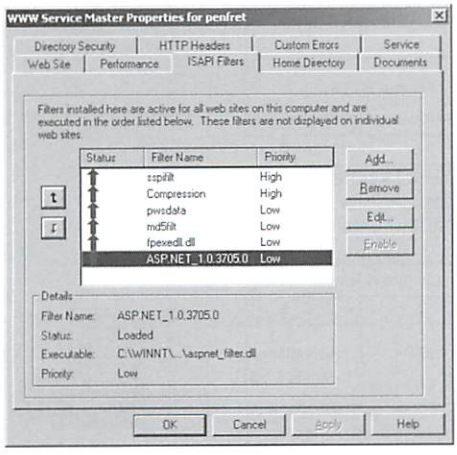


Figure 1–11
Le filtre ISAPI `aspnet_filter.dll`

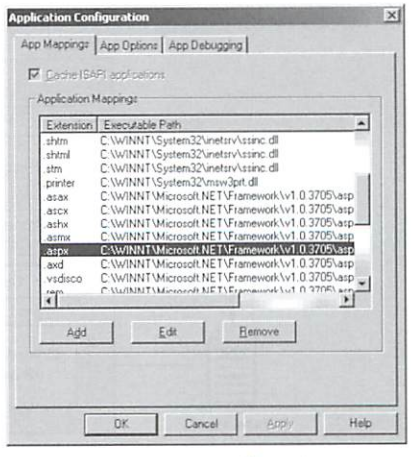


Figure 1–12 Configuration des extensions pour ASP.NET

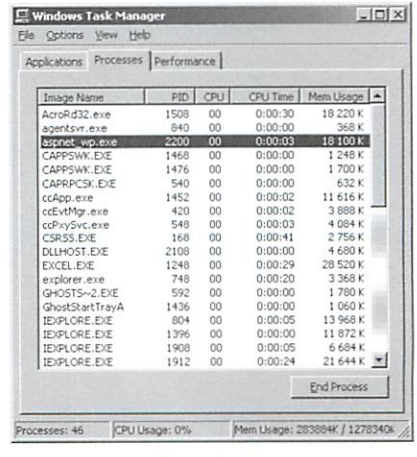


Figure 1–13
Le processus `aspnet_wp.exe`

ASP.NET est indissociable du framework .NET

De par son architecture, ASP.NET est indissociable du *framework .NET*, l'environnement d'exécution des applications .NET constitué du Common Language Runtime (CLR), de la bibliothèque de classes .NET et des compilateurs .NET

Anecdote

Le nom initial de la technologie ASP.NET était ASP+.

Extension .aspx

Les pages ASP.NET portent l'extension `.aspx` (pour : ASP eXtended).

La compilation n'a lieu que lors de la première requête

La compilation d'une page ASP.NET n'a lieu que lors du la première requête effectuée vers cette page ; l'assemblage produit est alors gardé en cache pour les requêtes successives (à moins que le fichier source ne soit modifié). Ceci garantit une performance optimale lors de l'exécution de l'application.

- les compilateurs .NET, qui permettent de compiler le code source des pages ASP.NET vers un format objet intermédiaire dit *Intermediate Language* (ou IL) ;
- une bibliothèque de classes utilitaires qui offre des fonctionnalités allant de l'accès aux données à la génération de documents XML, en passant par la réalisation de services Web ou de génération dynamique d'images, pour ne citer qu'une très brève étendue de ses possibilités ;
- une machine virtuelle nommée *Common Language Runtime* (CLR) prenant en charge l'exécution du code de la page (compilé en IL) et l'édition de liens avec les classes de la bibliothèque auxquelles le code de la page fait référence.

La figure 1-14 illustre le mécanisme d'appel d'une page ASP.NET :

- Lorsque le serveur HTTP reçoit une requête à destination d'une page portant l'extension `.aspx`, celle-ci est automatiquement traitée par le filtre ISAPI (`aspnet_filter.dll`) et transmise au processus principal ASP.NET (`aspnet_wp.exe`).
- La page demandée est alors compilée sous la forme d'un module objet en langage MSIL (Microsoft Intermediate Language) par le compilateur correspondant au langage utilisé dans la page (C#, VB.NET et JScript.NET sont les trois langages utilisables en standard) puis transmises à Common Language Runtime (CLR) qui réalise l'édition de lien avec les modules objet des classes utilisées de la bibliothèque .NET et la production d'un assemblage (composant binaire exécutable) dont l'exécution conduit à la production d'un contenu HTML, transmis en retour au navigateur.

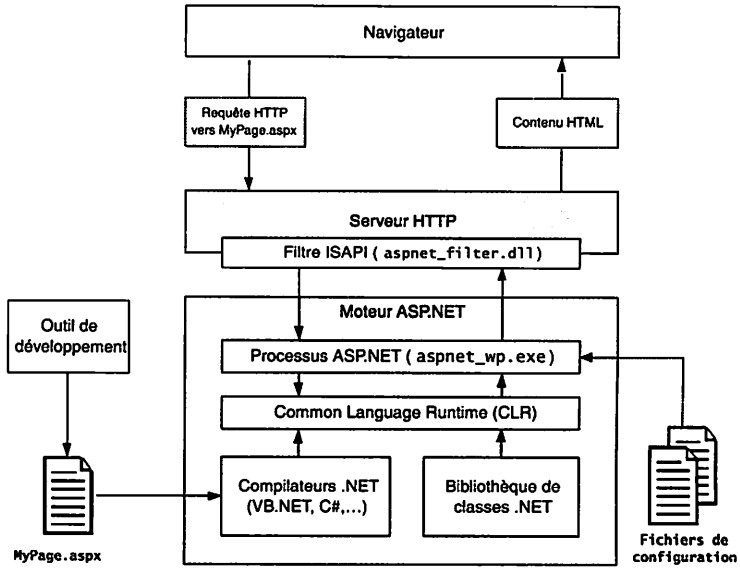


Figure 1–14 Traitement d'une requête vers une page ASP.NET

Notons que la configuration du processus principal ASP.NET (langue, sécurité, gestion des erreurs) est entièrement paramétrable à l'aide de fichiers texte au format XML (`machine.config` et `web.config`), dont les changements sont détectés et pris en compte dynamiquement, sans nécessiter de redémarrage du serveur HTTP.

Nous avons décrit succinctement l'infrastructure d'exécution des pages ASP.NET (nous aurons l'occasion de présenter plus précisément le mécanisme de compilation et la notion d'assemblage dans le chapitre 3 et les possibilités de configuration dans le chapitre 9) : détaillons maintenant les nouvelles possibilités offertes au développeur pour la réalisation de pages Web.

Parallèle avec Java

Le langage MSIL est en quelque sorte équivalent au *byte code* Java, tandis que le Common Language Runtime est proche conceptuellement de la machine virtuelle Java. De plus, le mécanisme de compilation des pages à la volée est proche de celui implémenté par l'architecture JSP.

Les nouveautés apportées par ASP.NET

Contrairement à une page ASP classique qui contient un entremêlement de contenu HTML et de script, les pages ASP.NET sont séparées en deux parties bien distinctes :

- la partie graphique constituée de balises et texte HTML (qui sera restitué tel quel), de contrôles serveur (éléments graphiques paramétrables qui seront, à l'exécution de la page, remplacés par un contenu HTML) et de contrôles utilisateur (fragments de pages HTML encapsulés au sein d'objets graphiques indépendants) ;
- la partie code qui spécifie le paramètrage des contrôles serveur et implémente la cinématique de la page en général, organisée au sein de gestionnaires d'événements et codée dans un des langages objet compatibles .NET : C# (proche de C++ et Java), VB.NET (proche de Visual Basic) ou JScript.NET (proche de JScript)

Formulaires Web (Web Forms)

Par analogie avec les formulaires Visual Basic, eux-mêmes constitués d'un assemblage de contrôle graphique dont on implémente la cinématique dans une partie code associée, les pages ASP.NET sont également appelées Formulaires Web (*Web Forms*).

Langages utilisables dans une page ASP.NET

À l'heure actuelle, le développeur a le choix entre trois langages par défaut pour développer une page ASP.NET :

- **VB.NET** : évolution majeure de Visual Basic auquel ont été principalement ajouté des fonctionnalités objet (constructeurs et destructeurs, surcharge de méthodes, héritage, etc.) ;
- **C#** : langage objet dont la syntaxe et les fonctionnalités sont proches de C++ et Java ;
- **JScript.NET** : évolution de JScript prenant en charge, entre autres, le typage des variables et les fonctionnalités objet (classe, héritage, etc.).

Ainsi, le développeur peut choisir le langage le moins éloigné de son champ de connaissance actuel et concentrer son attention sur les nouveautés de l'architecture ASP.NET plutôt que sur l'apprentissage d'un nouveau langage.

Les exemples de ce livre seront présentés dans les deux langages VB.NET et C#.

Autres langages

En théorie, une application .NET peut être développée avec tout langage conforme à la Common Language Specification (CLS) émise par Microsoft. Bien que le support d'une trentaine de langages ait été annoncé (dont Python, Perl, Eiffel, etc.), la mise en pratique peut s'avérer fastidieuse : nécessité de télécharger et d'installer un compilateur spécifique, absence d'exemples de code dans la documentation... C'est pourquoi les applications ASP.NET sont généralement implémentées avec VB.NET ou C#.

Code disponible en ligne

Dans certains cas, une version unique (VB.NET ou C#) sera proposée, dans un souci d'économie de place ; la version alternative sera alors consultable sur le site de l'étude de cas :

▸ www.savonsdusoleil.com

Aide-mémoire

Un aide-mémoire résumant les principales syntaxes des langages VB.NET et C# est fourni en annexe.

Cette architecture présente un certain nombre d'avantages :

- la séparation entre partie graphique et code augmente la lisibilité du code et facilite par conséquent sa maintenance ; d'autre part, elle permet, le cas échéant, de répartir plus facilement le travail entre des graphistes qui travaillent sur le contenu HTML et les développeurs qui se consacrent sur la logique applicative ;
- l'utilisation de langages orientés objet rend les développements plus robustes (typage fort, gestion des exceptions) et plus simples à réutiliser (héritage, polymorphisme) ;
- l'organisation du code en gestionnaires d'événements (« exécuter tel code lorsque tel événement survient »), rendu possible par un mécanisme capable de réaliser des allers-retours (*round-trip*) entre le navigateur et le serveur HTTP en conservant les valeurs contenues dans les contrôles, facilite l'implémentation de pages interactives (par exemple : mise à jour des produits disponibles lorsque l'utilisateur sélectionne une famille de produits).

Compatibilité ascendante avec ASP

Toute application ASP peut s'exécuter dans un environnement ASP.NET.

Nous développerons tous ces sujets dans le chapitre 3, où nous étudierons l'architecture d'une page ASP.NET (séparation entre contenu graphique et code, technique du *code behind*, notion de contrôle serveur) ; la notion de gestionnaire d'événement sera, quant à elle, décrite en détail à la fin du chapitre 4.

Par ailleurs, les pages ASP.NET bénéficient de la richesse de la bibliothèque .NET qui comporte plusieurs milliers de classes utilitaires permettant entre autres :

- la consultation et la mise à jour de données contenues dans une base, notamment à l'aide de contrôles serveur liés aux données comme `DataGrid`, `DataList` et `Repeater` (voir chapitres 4 et 5) ;
- la création et la manipulation de documents XML ainsi que la réalisation de transformations XSL (voir chapitre 6) ;
- l'envoi de fichiers par messagerie (voir chapitre 6) ;
- la génération dynamique d'images (voir chapitre 7) ;
- l'implémentation et l'appel de services Web (voir chapitre 8).

Choix de la base de données

Pourquoi pas Microsoft Access ?

Il est vrai que Microsoft Access a toujours constitué une alternative facile à utiliser et peu coûteuse à SQL Server. Néanmoins, le fait que le moteur Jet ne soit pas géré nativement par .NET, conjugué à la gratuité de MSDE plaide en défaveur du choix de cet outil pour notre étude de cas.

Dans une application ASP.NET comme dans la grande majorité des sites Web dynamiques, la base de données est une composante fondamentale de l'architecture. L'offre logicielle en la matière est vaste ; parmi les produits les plus connus disponibles sous Windows, citons : Microsoft SQL Server, Oracle, PostgreSQL, Informix, 4D, Microsoft Access…

À l'heure actuelle, la bibliothèque .NET propose les types d'accès suivants :

- accès natif à Microsoft SQL Server (et à sa version allégée : MSDE) ;
- accès natif à Oracle (fournisseur disponible en téléchargement séparé) ;
- pilote OLE-DB ;
- pilote ODBC (fournisseur disponible en téléchargement séparé).

Pour notre étude de cas, nous nous proposons d'utiliser la base Microsoft SQL Server Desktop Engine (MSDE) en raison de sa gratuité.

MSDE : une version allégée et gratuite de SQL Server 2000

Microsoft SQL Server Desktop Engine (MSDE) est une version allégée et gratuite de Microsoft SQL Server, doté d'un support complet de Transact SQL (y compris la gestion des transactions, des déclencheurs et de la sécurité). En revanche, elle n'offre qu'un support partiel de la réplication, ne dispose pas d'outils graphiques d'administration (comme SQL Enterprise Manager ou Analyseur de requête) et son usage reste limité à un nombre réduit d'utilisateurs simultanés.

Cette base constitue néanmoins une solution bien adaptée pour un poste de développement, du fait de sa faible consommation de ressources, d'une part, mais également de sa compatibilité totale avec SQL Server 2000, qui facilite un éventuel déploiement vers un serveur de production doté de ce SGBD. L'installation de MSDE sera décrite dans le chapitre suivant.

À propos de la similarité des moteurs SQL

La version précédente du moteur MSDE (1.0) était différente de SQL Server 7.0 ; désormais, MSDE 2000 et SQL Server 2000 sont dotés du même moteur.

Si vous disposez de SQL Server

Bien entendu, les lecteurs qui le souhaitent pourront utiliser SQL Server au lieu de MSDE : les exemples donnés dans cet ouvrage seront parfaitement applicables à l'un comme à l'autre.

Choix de l'environnement de développement

Dans l'absolu, il est possible de développer une application ASP.NET avec un simple éditeur de texte : fastidieuse, cette approche a néanmoins l'avantage indéniable de permettre au développeur de maîtriser tout le code qu'il écrit, et certains en sont partisans !

À l'opposé, il existe des environnements de développement qui se proposent d'automatiser les tâches laborieuses à l'aide d'assistants, de faciliter le confort de travail du développeur (coloration syntaxique, complétion automatique du code saisi, compilateurs intégrés, aide en ligne), citons :

- **Visual Studio.NET** : très complet, il gère de nombreux types d'applications (ASP.NET, WinForms, MFC, ATL), plusieurs langages (C, C++, VB.NET, C#), est doté de fonctionnalités puissantes (grand nombre d'assistants, aide contextuelle, technologie Intellisense pour compléter le code) et est adapté au travail en groupe (gestion de projets complexes, intégration avec des outils de contrôles de source) ;
- **Web Matrix** : environnement léger dédié au développement d'applications ASP.NET ; moins riche en fonctionnalités que Visual Studio.NET (pas d'aide contextuelle, pas de complétion automatique du code, pas de notion de projet), il n'en est pas moins doté d'un grand nombre d'assistants (modèles de pages, ajout de contrôles, gestion de la base de données) et présente l'indéniable avantage d'être gratuit ;
- **Delphi 7 Studio** : la nouvelle version du célèbre environnement de développement édité par Borland fournit désormais un support pour le développement d'application ASP.NET avec le langage Delphi.

Dans le cadre d'une grosse équipe de développement travaillant sur des projets importants, l'emploi de Visual Studio.NET ou de Delphi serait probablement nécessaire ; pour notre étude de cas, limitée à une application ASP.NET, réalisée a priori par un développeur seul, Web Matrix répond amplement aux besoins.

Utiliser quand même VS.NET ou Delphi

Bien entendu, les lecteurs qui le souhaitent pourront tout de même tirer parti de cet ouvrage en utilisant Visual Studio.NET ou Delphi au lieu de Web Matrix : en effet, l'accent sera plus porté sur la technologie ASP.NET proprement dite que sur les outils ou langages utilisés pour la mettre en œuvre.

Web Matrix : version 0.5 (en anglais)

À l'heure actuelle, Web Matrix n'est disponible qu'en version 0.5 (Technology Preview), en anglais. Si quelques fonctionnalités font encore défaut, cette version est globalement satisfaisante en terme de stabilité (et les quelques commandes de menu en anglais ne devraient pas dérouter le lecteur). Notons que dans sa version finale, le produit sera toujours gratuit.

- www.asp.net/webmatrix

Code in-line vs code behind

Une des différences fondamentales entre Web Matrix et Visual Studio.NET réside dans la gestion de la séparation entre code et contenu HTML : Web Matrix place le code et le contenu HTML d'une page dans le même fichier, tout en offrant une séparation visuelle entre les deux à l'aide d'onglets (*code in-line*), tandis que Visual Studio.NET place le code associé à une page dans un fichier séparé du contenu HTML (*code behind*). Nous aurons l'occasion de détailler ces deux techniques dans le chapitre 3.

Web Matrix : un environnement de développement efficace et gratuit

Web Matrix est un environnement léger dédié au développement ASP.NET. Disponible en téléchargement gratuit sur www.asp.net/webmatrix (son installation sera détaillée dans le chapitre suivant), il démontre la volonté de Microsoft de fédérer une communauté de développeurs autour d'ASP.NET : outil gratuit, nombreux exemples de sources, accès à des forums depuis l'interface.

Le tableau ci-dessous résume les principales différences entre Visual Studio.Net et Web Matrix :

	Visual Studio.NET	Web Matrix
Assistants graphiques	X	X
Gestion base de données	X	X
Complétion du code (Intellisense)	X	
Client FTP intégré		X
Serveur HTTP intégré		X
Gestion de projets	X	
Contenu du code source	X	
Compilation	Intégrée	Séparée (ligne de commande)
Séparation HTML/code	Fichiers séparés (code behind)	Même fichier (code in-line)
Version actuelle	7.0	0.5
Coût licence	Variable selon le type de licence	Gratuit

Bien qu'il soit moins riche en fonctionnalités que Visual Studio.NET ou Delphi, Web Matrix s'avère tout à fait adapté au développement ASP.NET : les nombreux assistants et modèles, le client FTP intégré et le serveur HTTP personnel font de cet outil gratuit un compagnon fidèle et efficace.

En résumé

Dans ce premier chapitre, nous avons posé les bases de notre étude de cas : après avoir présenté les activités des « Savons du Soleil » et les problématiques auxquelles est confrontée cette société, nous avons décrit en détail le projet d'évolution du système d'information.

Puis, nous avons proposé une vue d'ensemble de la technologie ASP.NET et justifié le choix de la base de données MSDE et de l'environnement de développement Web Matrix pour la réalisation de notre étude de cas.

Dans le chapitre suivant, nous allons décrire l'installation de ces outils sur le poste de travail.

Installation des outils et création de la base

2

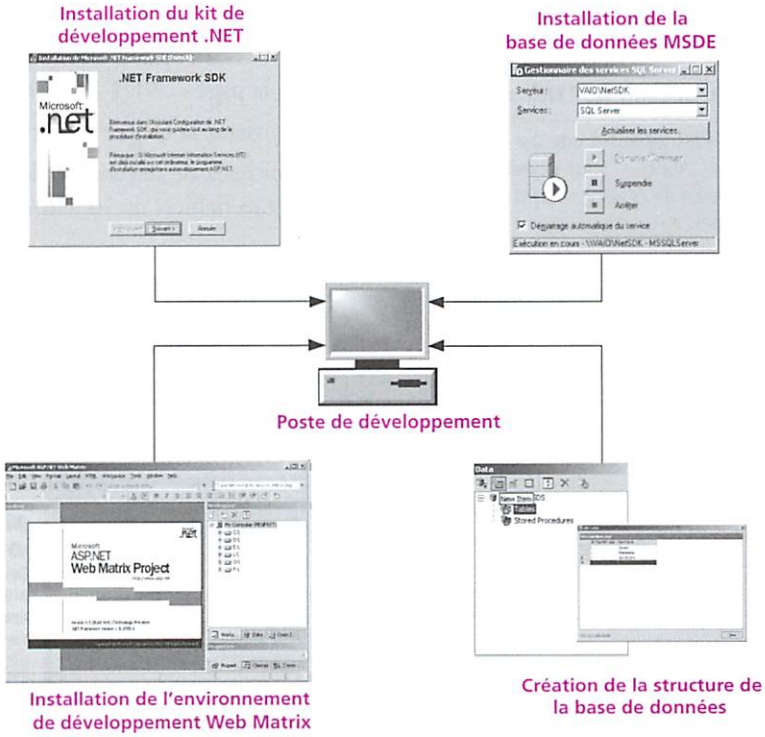
ASP.NET

.NET Framework | ASP.NET | MSDE | Web Matrix | Scripts SQL

SOMMAIRE

MOTS-CLÉS

- .NET Framework
- ASP.NET
- MSDE
- Web Matrix
- Scripts SQL



La réalisation de notre application ASP.NET nécessite l'installation, sur le poste de développement, de plusieurs outils : le kit de développement .NET, le moteur de base de données MSDE et, enfin, un environnement de développement léger et gratuit, Web Matrix. Dans ce chapitre, nous montrons comment installer ces outils et les utiliser pour mettre en place la structure de la base de données.

Installation des outils de développement

Obtention du kit de développement .NET

Le kit de développement .NET (ou .NET SDK en anglais) contient l'ensemble des outils nécessaires au développement d'applications .NET :

- le Common Language Runtime (CLR) ;
- la bibliothèque de classes .NET;
- les compilateurs des langages .NET (VB.NET, C# et JScript.NET) ;
- le moteur de bases de données MSDE (Microsoft Desktop Data Engine) ;
- la documentation complète ;
- de nombreux exemples de code.

Ce kit de développement est téléchargeable gratuitement depuis le site www.asp.net à l'adresse www.asp.net/download.aspx (130 Mo, nécessite une connexion rapide) :

1. Choisissez l'option Download .NET Framework SDK : vous êtes alors redirigé vers la page « .NET SDK » du site de Microsoft (figure 2-1).
2. Choisissez la langue de votre choix (le kit est disponible en français).
3. Choisissez de télécharger le kit en une seule fois (130 Mo) ou en plusieurs fois (13 fichiers de 10 Mo).

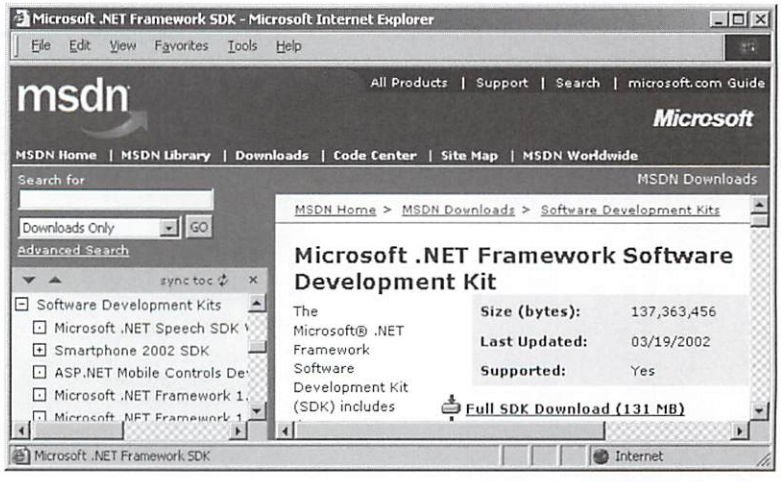


Figure 2–1 Télécharger le kit de développement .NET

Il faut avouer que la taille de ce téléchargement est conséquente mais le jeu en vaut la chandelle !

.NET SDK

Dans la documentation en anglais, ce kit de développement est appelé « .NET SDK » pour *.NET Software Development Kit*.

▸ www.asp.net/download.aspx

ALTERNATIVE Obtention du .NET SDK

Le .NET SDK est aussi livré aux abonnés MSDN (CD « Microsoft .NET Framework SDK ») et fourni en standard avec Visual Studio.NET sur le CD-Rom Windows Component Update.

Ne peut-on obtenir ASP.NET seul ?

Même si vous n'êtes intéressé que par la réalisation d'applications ASP.NET, vous devrez télécharger l'ensemble du kit de développement .NET !

.NET Redistributable vs .NET SDK

Le kit *.NET Redistributable* ne contient que le minimum vital pour pouvoir faire fonctionner une application .NET, à savoir le Common Language Runtime, les bibliothèques de classes et les compilateurs – il est généralement installé sur les serveurs de production. Le *.NET SDK* contient en plus la documentation, des exemples de code et la base de données MSDE.

ASTUCE Optimiser le téléchargement

Étant donné la taille des fichiers à télécharger, il est recommandé d'utiliser un logiciel FTP capable de reprendre un téléchargement interrompu ; par exemple, Fresh Download de :

▸ http://www.freshdevices.com

Prérequis à l'installation du kit de développement .NET

Il est indispensable de disposer d'une machine équipée d'un système d'exploitation compatible avec ASP.NET :

- Windows 2000 – toutes versions (Service Pack 2 recommandé).
- Windows XP Professional.

D'autre part, il est recommandé d'installer préalablement le serveur Web IIS 5.0 ainsi que la version 2.7 de MDAC (figure 2-2).

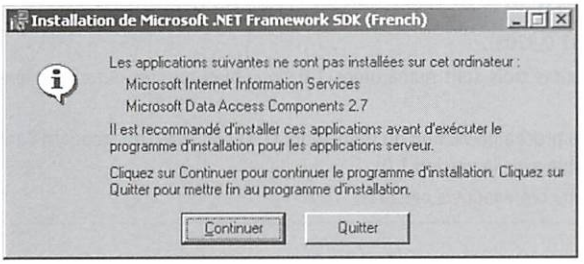


Figure 2–2 Installation d'IIS et MDAC recommandée

ATTENTION ASP.NET ne fonctionne pas avec NT4

Bien que le framework .NET puisse être installé sur Windows NT4 (Service Pack 6a requis), la technologie ASP.NET ne fonctionne qu'avec Windows 2000 et XP.

Installation du serveur Web Internet Information Server (IIS)

L'installation d'Internet Information Server (IIS), serveur HTTP fourni en standard dans Windows, est recommandée dans la mesure où elle permet de profiter des exemples fournis avec le kit de développement NET.

Pour installer Internet Information Server :

1. Choisissez Ajout/Suppression de programmes dans le panneau de configuration.
2. Choisissez Ajouter/Supprimer des composants Windows.
3. Cochez la case Internet Information Server et cliquez sur OK. Le CD-Rom d'installation de Windows sera requis au cours de l'installation.

ALTERNATIVES Serveur HTTP de Web Matrix ou Apache

Si vous ne souhaitez pas installer IIS sur votre machine, vous pouvez utiliser le serveur HTTP léger fourni avec Web Matrix pour visualiser les pages ASP.NET. D'autre part, notez qu'il existe une version d'Apache compatible avec ASP.NET.

- www.covalent.com

Installation du kit de pilotes de bases de données MDAC

Microsoft Data Access Components (MDAC) est un kit logiciel gratuit contenant tous les pilotes de bases de données nécessaires pour Windows. L'installation de la dernière version (2.7 et au-delà) est indispensable pour utiliser la bibliothèque d'accès aux données ADO.NET.

Pour installer MDAC 2.7 :

1. Téléchargez MDAC 2.7 sur www.microsoft.com/data (environ 5 Mo).
2. Exécutez le programme d'installation (aucune option à choisir).

VERSION Quelle est votre version de MDAC ?

Pour connaître le numéro de version du kit MDAC installé sur votre poste, téléchargez l'outil « Component Checker » à l'adresse :

- www.microsoft.com/data

ATTENTION Mise à jour de MDAC

Dans certains cas, l'installation de MDAC 2.7 peut poser des problèmes de compatibilité avec des applications antérieures.

- www.microsoft.com/data/MDAC27Info/en/readmerefresh.htm

Désinstallation des versions beta antérieures

Enfin, avant d'installer le kit de développement .NET, il est nécessaire d'avoir préalablement désinstallé les éventuelles versions beta de .NET présentes sur la machine.

Versions du framework .NET

À ce jour, trois versions du framework .NET se sont succédées, qui sont incompatibles entre elles :

- la *beta1* (1.0.2204),
- la *beta2* (1.0.2914)
- la version 1.0 (1.0.3705).

Curieusement, toutes trois sont numérotées 1.0.xxxx, bien que les deux premières soient des versions beta !

Il est prévu que la prochaine version du framework (1.1) soit livrée en standard dans .NET Server 2003 (et compatible avec la version 1.0).

▸ www.microsoft.com/windows.netserver

Ces installations préliminaires étant effectuées, votre poste est maintenant prêt pour l'installation du kit de développement .NET.

Installation du kit de développement

Utilisateurs de Visual Studio.NET

Si vous installez le .NET Framework à partir de Visual Studio, insérez le disque 1 et laissez-vous guider.

Une fois l'exécutable d'installation téléchargé, la procédure est très simple:

1. Lancez `setup.exe`.
2. Un écran de choix apparaît : gardez les options par défaut (figure 2-3).
3. Cliquez sur Suivant et patientez quelques dizaines de secondes… c'est fini !

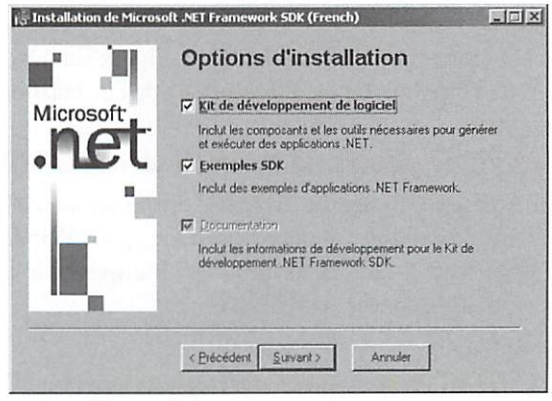


Figure 2–3 Options d'installation du kit de développement .NET

L'installation du framework .NET est maintenant terminée :

- la bibliothèque de classes et les programmes nécessaires au fonctionnement de l'architecture .NET (CLR, compilateurs, processus ASP.NET, fichiers de

configuration) ont été installés sur votre poste de travail, dans `\WINNT\Microsoft.NET\Framework\v.1.0.3705` ;

- la documentation et les fichiers exemples ont été copiés sur le disque dur (une étape de configuration supplémentaire devra être effectuée après l'installation de la base de données) dans le répertoire d'installation `\Program Files\Microsoft.NET\FrameworkSDK` ;
- des raccourcis ont été ajoutés dans le menu Démarrer (figure 2-4).

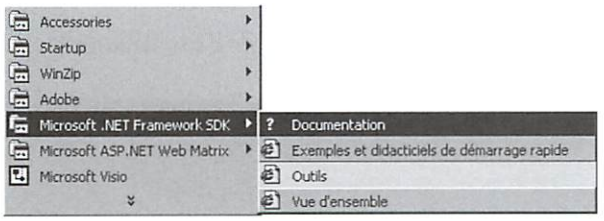


Figure 2–4 Les raccourcis ajoutés dans le menu Démarrer

Passons maintenant à l'étape suivante : la mise en place de la base de données MSDE.

Installation de Microsoft SQL Server Desktop Engine (MSDE)

L'installation de MSDE est très simple :

1. Localisez l'exécutable d'installation `instmsde.exe` situé dans le répertoire `\Program Files\Microsoft.NET\Framework SDK\Samples\setup\msde`.
2. Lancez l'installation (elle est rapide et aucune option n'est à choisir) : un écran de progression apparaît (figure 2-5).
3. L'installation est terminée lorsque cet écran disparaît : redémarrez alors la machine.

Au redémarrage, la seule trace visible de l'installation de MSDE est l'icône du gestionnaire des services SQL Server, qui permet de faire apparaître la boîte de dialogue présentée figure 2-6 : ne changez rien aux options si vous souhaitez que MSDE démarre automatiquement lors du chargement de Windows.

MSDE ne comprend aucune interface graphique de gestion de la base (contrairement à SQL Server qui est fourni avec la console d'administration Entreprise Manager et divers utilitaires dont un analyseur de requêtes, des outils de configuration réseau).

Heureusement, il est possible de configurer MSDE via l'environnement de développement Web Matrix, que nous allons maintenant installer, juste après avoir configuré les exemples du kit de développement .NET.

Installation du service pack le plus récent

Après l'installation, il est recommandé de télécharger et d'appliquer le service pack le plus récent relatif au framework .NET.

▸ http://msdn.microsoft.com/netframework/

Utilisateurs de SQL Server

Si SQL Server est déjà installé sur votre poste, il n'est évidemment pas nécessaire d'installer MSDE.

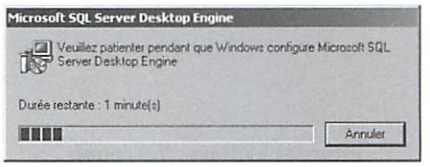


Figure 2–5 Installation de MSDE

Curieusement, aucun message n'annonce la fin de l'installation de MSDE !

Figure 2–6 Gestionnaire des services SQL Server

Installation des exemples du kit de développement .NET

Maintenant que la base MSDE est installée, il est possible d'installer les exemples livrés avec le kit de développement .NET :

1. Localisez l'exécutable d'installation `ConfigSamples.exe` situé dans le répertoire `\Program Files\Microsoft.NET\Framework SDK\Samples\setup`.
2. Lancez l'installation (aucune option n'est à choisir) : l'utilitaire crée un certain nombre de répertoires virtuels dans IIS et des bases de données exemples dans MSDE.
3. Un message apparaît pour signaler la fin de l'installation (figure 2-7).

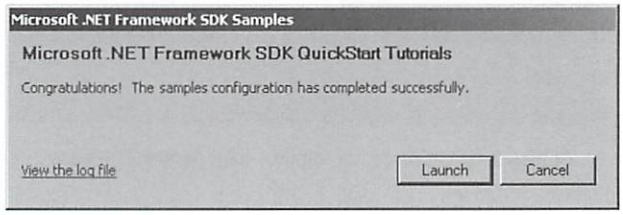


Figure 2–7 Installation des exemples livrés avec .NET

On peut alors cliquer sur Launch pour faire apparaître les exemples, qui illustrent les principales fonctionnalités d'ASP.NET et auxquels vous pouvez d'ores et déjà jeter un coup d'œil !

Téléchargement et installation de Web Matrix

L'installation de Web Matrix est très rapide :

1. Téléchargez l'exécutable d'installation sur http://www.asp.net/webmatrix (gratuit).
2. Lancez l'exécutable d'installation : un écran apparaît (figure 2-8).
3. Cliquez sur Next (vous pouvez changer le répertoire de destination).
4. Patientez quelques dizaines de secondes… c'est terminé !

L'installation a ajouté un groupe ASP.NET Web Matrix dans le menu Programmes : nous allons tout de suite utiliser Web Matrix pour créer la structure de la base de données.

Internet Explorer 5.5 requis

L'installation de Web Matrix requiert Internet Explorer 5.5 ou supérieur. Téléchargement gratuit à l'adresse :

- www.microsoft.com/windows/ie

Création de la structure de la base de données

Le lecteur trouvera en annexe tous les détails relatifs à la structure de la base à mettre en place (schéma des tables, listes des procédures stockées…) et pourra télécharger directement un script de création complet de la base à l'adresse :

- www.savonsdusoleil.com

Plusieurs options sont possibles pour créer la structure de la base de données : utiliser Web Matrix, utiliser la console SQL Server Enterprise Manager, utiliser le mode « projets de données » de Microsoft Access, ou enfin avoir recours à des scripts SQL.

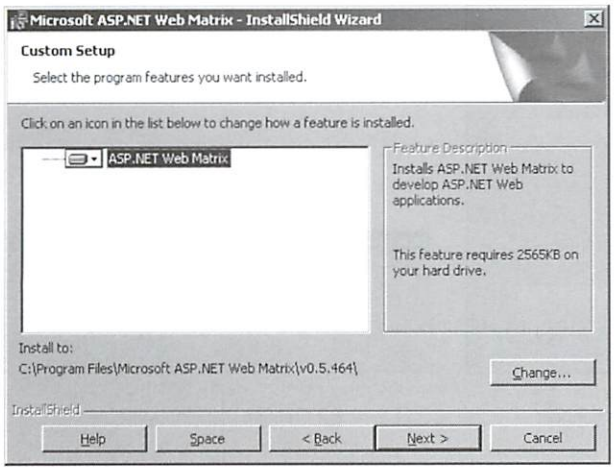


Figure 2–8 Installation de Web Matrix

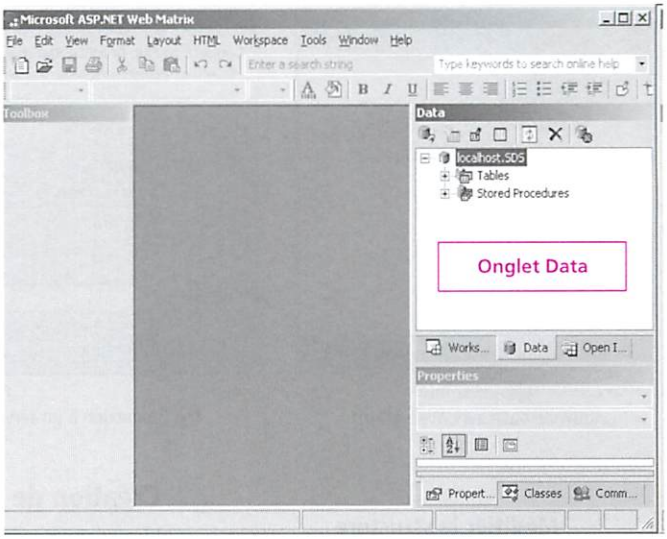


Figure 2–9 L'onglet Data de Web Matrix

Dans cette section, nous ne faisons que décrire le fonctionnement général de ces outils, en détaillant plus particulièrement les fonctionnalités proposées par Web Matrix.

Création de la structure de la base avec Web Matrix

L'environnement Web Matrix dispose d'une interface graphique permettant d'effectuer des opérations simples sur une base de données SQL Server ou MSDE :

- créer une base de données (ou se connecter à une base existante) ;
- créer ou modifier des tables (structure et données) ;
- créer ou modifier des procédures stockées.

Toutes les opérations de manipulation de données sont effectuées depuis l'onglet Data de la fenêtre Workspace, situé en haut à droite de l'environnement de développement (figure 2-9).

Création d'une nouvelle base

Pour créer une nouvelle base ou se connecter à une base existante avec Web Matrix :

1. Cliquez sur l'icône New Connection dans l'onglet Data ❶. Une fenêtre de connexion à un serveur de données apparaît ❷.
2. Spécifiez le nom du serveur et le mode d'authentification souhaité.
3. Cliquez sur le lien Create a new database et saisissez un nom de base ❸.

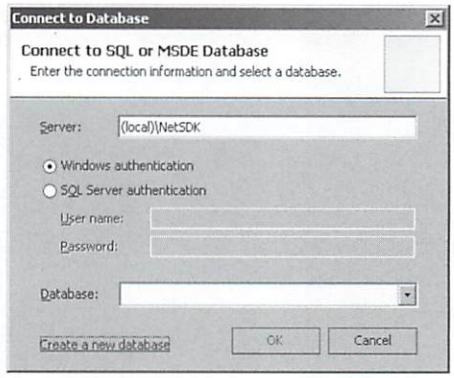
IMPORTANT Nom du serveur de données

Lors de l'installation de MSDE, une instance nommée (*named instance*) de SQL Server est créée : pour accéder au serveur, il faut utiliser l'adresse `(local)\NetSDK`.

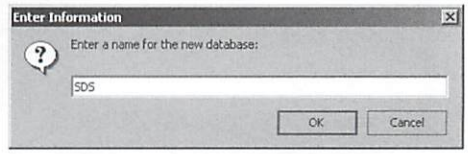
Localisation des fichiers de la base

Les fichiers de la base sont situés, par défaut, dans le répertoire `\Program Files\Microsoft SQL Server\MSSQL$NetSDK\Data`.

1 Créer une nouvelle base avec Web Matrix



2 Connexion à un serveur de données



3 Création d'une nouvelle base

Modifier la structure d'une table existante

Un lien *Edit/View Table Design* situé en bas de la fenêtre de saisie de données dans une table permet de consulter ou modifier la structure de cette table. Malheureusement, il n'est possible de modifier la structure que si la table est vide !

Création de tables

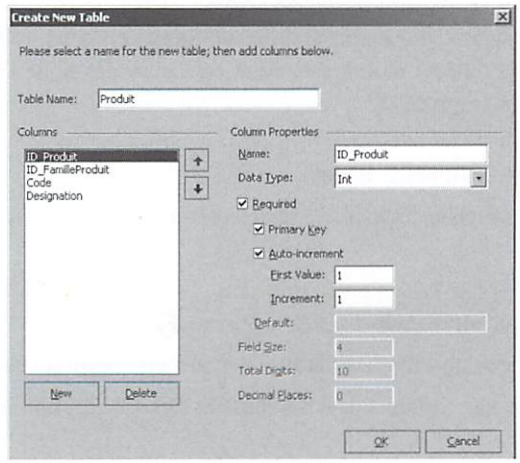
Une fois connecté à une base de données, voici comment créer une table avec Web Matrix :

1. Sélectionnez l'élément Tables dans la fenêtre Data.
2. Cliquez sur l'icône New Item 4. Un écran permettant de spécifier les caractéristiques de la table apparaît 5
3. Spécifiez le nom de la table ainsi que le nom et les caractéristiques de chaque colonne (nom, type, attributs : champ obligatoire, clé primaire, numéro automatique, etc.).
4. Cliquez sur OK pour enregistrer les modifications.

Pour mettre à jour le contenu d'une table existante :

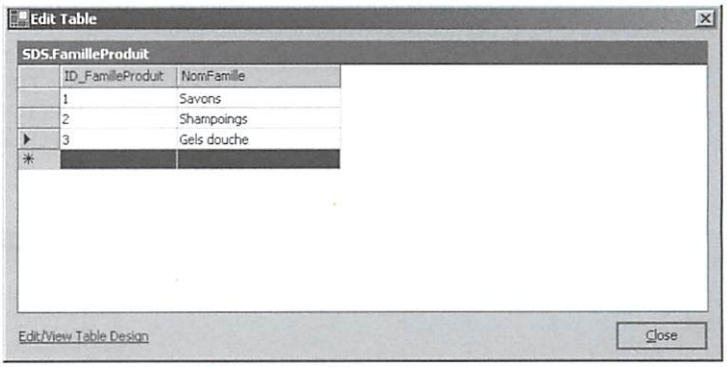
1. Sélectionnez la table souhaitée dans la fenêtre Data et cliquez sur l'icône Edit 6
2. Saisissez les données dans la fenêtre d'édition qui apparaît 7

4 Création d'une nouvelle table



5 Définition de la structure d'une table

6 Édition des données d'une table



7 Saisie de données dans une table

Création de procédures stockées

Web Matrix permet de créer des procédures stockées, en saisissant directement le code SQL dans une fenêtre de saisie :

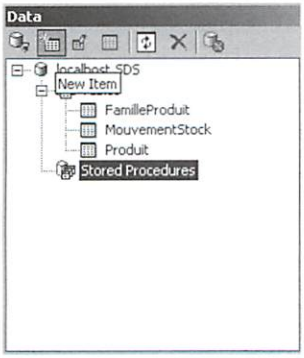
1. Sélectionnez l'élément Stored Procedures dans la fenêtre Data.
2. Cliquez sur l'icône New Item 8 : une fenêtre destinée à l'édition du code SQL de la procédure apparaît 9.
3. Saisissez le code de la procédure.
4. Cliquez sur OK pour enregistrer les modifications.

WEB MATRIX Vues SQL non gérées !

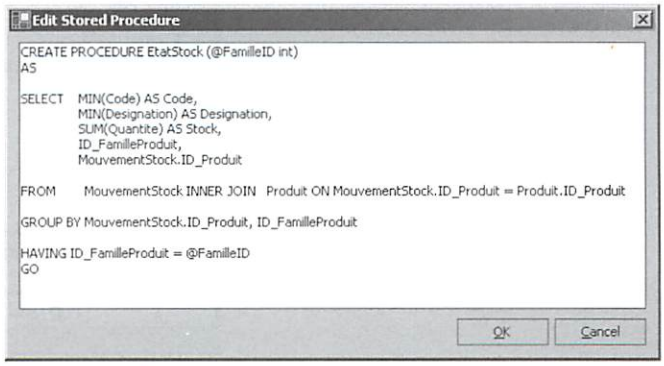
À l'heure actuelle, Web Matrix ne permet malheureusement pas de créer des vues SQL : nous nous limiterons donc à l'utilisation de procédures stockées dans notre étude de cas.

Conclusion sur l'interface de gestion de données de Web Matrix

Comme vous avez pu le constater, les possibilités de Web Matrix en matière de gestion des données sont relativement réduites : il n'est en particulier pas possible de créer des relations, des vues, ni de modifier la structure d'une table lorsqu'elle est remplie, encore moins de gérer les permissions, les index ou les déclencheurs… Néanmoins, Web Matrix permet aux utilisateurs ne disposant pas de Microsoft Access ni de SQL Server Entreprise Manager d'effectuer les opérations de base sans avoir recours à l'utilisation de scripts.



8 Création d'une nouvelle procédure stockée



9 Saisie du code d'une procédure stockée

Création de relations entre tables

Dans une base correctement structurée, on établit des relations entre les tables pour s'assurer de l'intégrité des données. Malheureusement, Web Matrix ne propose *pas* d'interface graphique pour la création de relations entre tables : à moins de disposer d'une version complète de SQL Server ou de Microsoft Access, le seul moyen de créer des relations est donc d'utiliser des scripts SQL en mode ligne de commande !

Pour cela, on utilise l'outil `osql.exe`, fourni en standard avec MSDE et installé par défaut dans le répertoire `\Program Files\Microsoft SQL Server\80\Tools\Binn`, qui permet d'exécuter des scripts SQL sur une base de données (voir figure 2-10).

L'outil `osql` se démarre en mode ligne de commande DOS avec la syntaxe suivante :

```
osql -S<NomServeur> -d<NomBase> -E
```

où :

- `<NomServeur>` désigne le nom du serveur de données ;
- `<NomBase>` désigne le nom de la base de données ;
- `-E` indique qu'il faut la sécurité intégrée de Windows pour authentifier la connexion.

Voici ensuite la syntaxe SQL à utiliser pour la définition d'une relation :

```
> ALTER TABLE <TableEtrangere>
  ADD CONSTRAINT <NomContrainte>
  FOREIGN KEY (<CleEtrangere>)
  REFERENCES <TablePrimaire> (<ClePrimaire>)
> GO
```

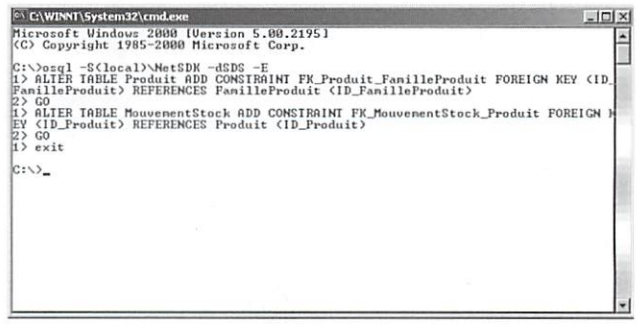


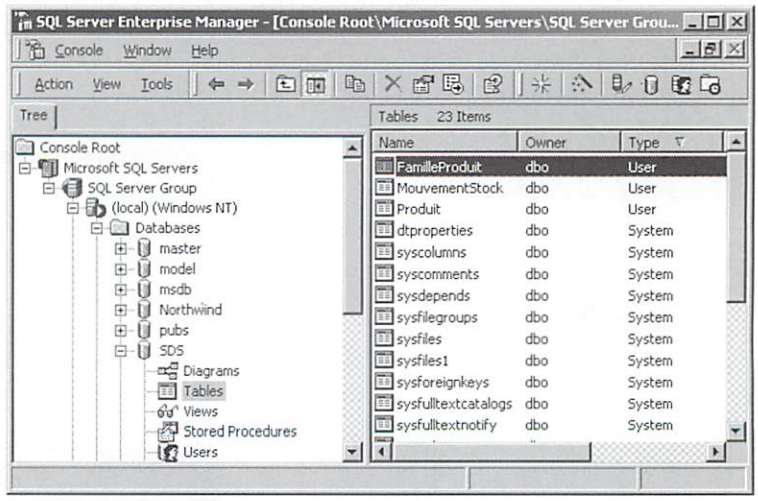
Figure 2–10 Créer les relations en mode ligne de commande

Alternative 1 - Utilisation de la console SQL Server Enterprise Manager

Si vous possédez une édition de SQL Server, vous pouvez effectuer toutes les opérations de création de la structure de la base à l'aide de la console SQL Server Enterprise Manager (1).

On ne décrira pas ici l'ensemble des possibilités de cette interface très complète, pour laquelle Microsoft fournit une aide en ligne à l'adresse :

› www.microsoft.com/sql



(1) L'interface SQL Server Enterprise Manager

Alternative 2 - Utilisation de Microsoft Access

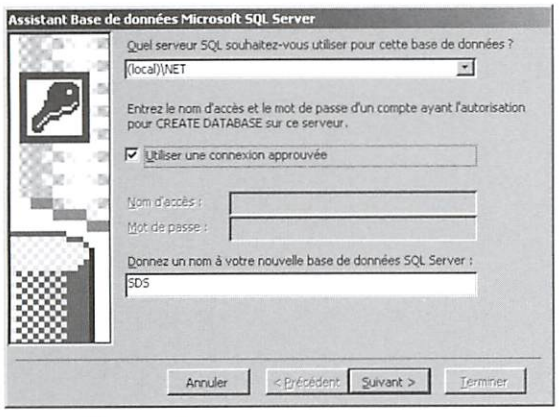
Microsoft Access dispose d'une fonctionnalité peu connue mais néanmoins très riche en possibilités : les *projets de données*, qui permettent de gérer des bases de données MSDE ou SQL Server en utilisant Access comme interface graphique de gestion (et non pas également en tant que moteur de base de données, comme c'est généralement le cas).

Pour créer une base SQL avec Microsoft Access :

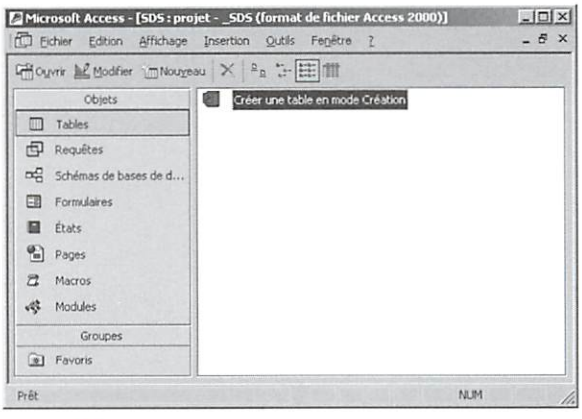
1. Créez un nouveau projet de données (fichier `.adp`).
2. Paramétrez les informations relatives à la base (2).
3. Créez la structure de la base avec l'interface graphique (3).

Un article complet sur ce mode d'utilisation d'Access est disponible sur le site de Microsoft :

- http://office.microsoft.com/france/assistance/2002/articles/WaysToWorkWithSQLData.aspx



(2) Création d'une base SQL avec Microsoft Access



(3) Création d'une base SQL avec Microsoft Access

Alternative 3 - Utilisation de scripts SQL

Enfin, il est possible d'utiliser un script SQL pour créer la structure de la base et peupler des tables, en utilisant l'exécutable en ligne de commande `osql`.

La syntaxe de la ligne de commande est la suivante :

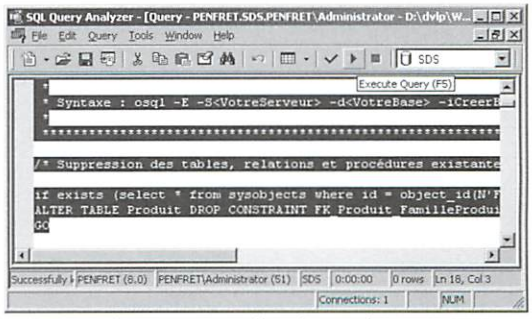
```
osql -S<NomServeur> -d<NomBase> -i<NomFichierScript> -E
```

où :

- `<NomServeur>` désigne le nom du serveur de données ;
- `<NomBase>` désigne le nom de la base de données ;
- `<NomFichierScript>` désigne le nom du fichier de script à appliquer ;
- `-E` indique qu'il faut la sécurité intégrée de Windows pour authentifier la connexion.

ALTERNATIVE Utiliser l'analyseur de requête SQL Server

Pour ceux qui possèdent SQL Server, il suffit d'ouvrir le fichier de script avec l'analyseur de requêtes SQL, de sélectionner tout le texte et de cliquer sur le bouton Execute Query



Exécution d'un script avec l'analyseur de requête SQL

RAPPEL Mise en place de la base pour l'étude de cas

Avant de passer au chapitre suivant, le lecteur est invité à mettre en place la structure de la base de données de l'étude de cas à l'aide des scripts SQL disponibles sur le site :

- www.savonsdusoleil.com

En résumé...

Après avoir installé le kit de développement .NET, la base de données MSDE et l'outil de développement Web Matrix, nous avons décrit les différentes alternatives possibles pour la mise en place de la base de données : l'interface de gestion des données de Web Matrix, le mode « projets de données » de Microsoft Access, la console SQL Server Entreprise Manager et l'utilisation de scripts SQL.

Dans le chapitre suivant, nous allons voir concrètement à quoi ressemble une page ASP.NET, comment est réalisée la séparation entre code et présentation et ce que sont les contrôles serveur et les contrôles utilisateur.

Architecture d'une page ASP.NET

3

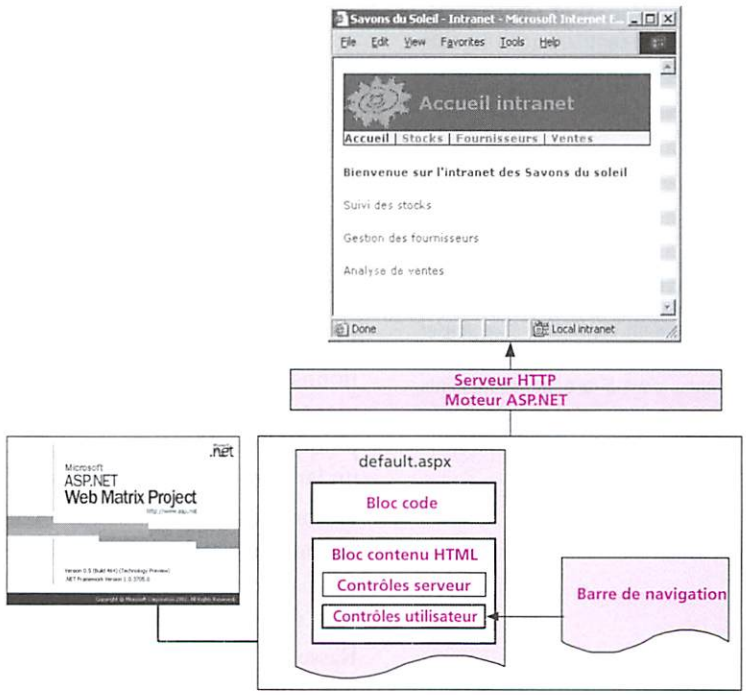
ASP.NET

Contrôle serveur | contrôle utilisateur | Code behind | Web Matrix | attribut CssClass

SOMMAIRE

MOTS-CLÉS

- Contrôle serveur
- Contrôle utilisateur
- Code behind
- Web Matrix
- Directive <%@Page ...%>
- Directive <%@Register...%>
- Directive <%@Control...%>
- Attribut CssClass



Dans ce chapitre, nous présentons l'architecture générale d'une page ASP.NET : quels sont les concepts principaux du nouveau modèle de programmation qui leur est associé, comment s'effectue en pratique la séparation entre le code et le contenu graphique, comment le développement devient plus simple et plus productif grâce à l'utilisation de *contrôles serveur* et *contrôles utilisateur*. Puis nous mettons en pratique ces connaissances pour réaliser la barre de navigation de notre étude de cas.

La page Web vue comme une interface classique

Le principe fondamental d'ASP.NET est de considérer une page Web non plus comme un document HTML mais comme une interface graphique classique d'application client-serveur.

Tout l'intérêt d'ASP.NET est de permettre au développeur d'implémenter une page Web comme un assemblage de *contrôles* graphiques (listes, boutons…) réagissant à des *événements* (changement de sélection dans une liste, clic sur un bouton…), tout en assurant la traduction de cette vision conceptuelle en HTML standard.

Prenons un exemple inspiré de notre étude de cas, l'interface de consultation de la liste des produits des Savons du Soleil, classés par famille de produits :

- Dans le cadre d'une application client-serveur classique, cette interface aurait été implémentée avec un langage objet, sous la forme d'une boîte de dialogue contenant des contrôles – une liste déroulante affichant la liste des familles et une grille de données affichant la liste des produits – avec une cinématique associée à des événements : remplissage des contrôles lors du chargement de l'interface, mise à jour de la liste des produits dans le cas d'un changement de sélection de famille.
- Dans le cadre d'une application Web classique, cette interface aurait été implémentée sous la forme d'une page Web dynamique, contenant du HTML mêlé à du langage de script, sans possibilité de gérer de manière transparente un changement de sélection de famille de produits.

Dans le passé, il fallait donc choisir entre une interface classique, riche en fonctionnalités implémentées avec un langage puissant, mais imposant un déploiement sur le poste client, et une interface Web (plus limitée en possibilités graphiques) n'intégrant pas ou peu de gestion événementielle et implémentée avec un langage de script, même si cela éliminait le problème du déploiement.

Désormais, la technologie ASP.NET permet de profiter des avantages de l'approche classique tout en bénéficiant de l'architecture Web : implémentation des pages Web comme un assemblage de contrôles, utilisation de langages évolués, gestion événementielle, performance liée à la compilation (figure 3-1).

Rassurez-vous, rien de magique dans tout cela, si ce n'est une restructuration complète du modèle de programmation de pages Web dynamiques, initialement introduit par ASP et désormais principalement articulé autour des mécanismes suivants :

- les contrôles serveur ;
- le séparation entre code et contenu graphique ;
- les contrôles utilisateur ;
- la gestion des événements « côté serveur ».

Contrôle

Un contrôle est un élément graphique paramétrable ; l'utilisation de contrôles rend le code plus modulaire et accroît la productivité du développement.

Programmation événementielle

Modèle de programmation consistant à organiser le code d'une application sous la forme de gestionnaires d'événements, c'est-à-dire de traitements exécutés lorsque tel ou tel événement survient. Technique couramment utilisée lors du développement d'interfaces graphiques.

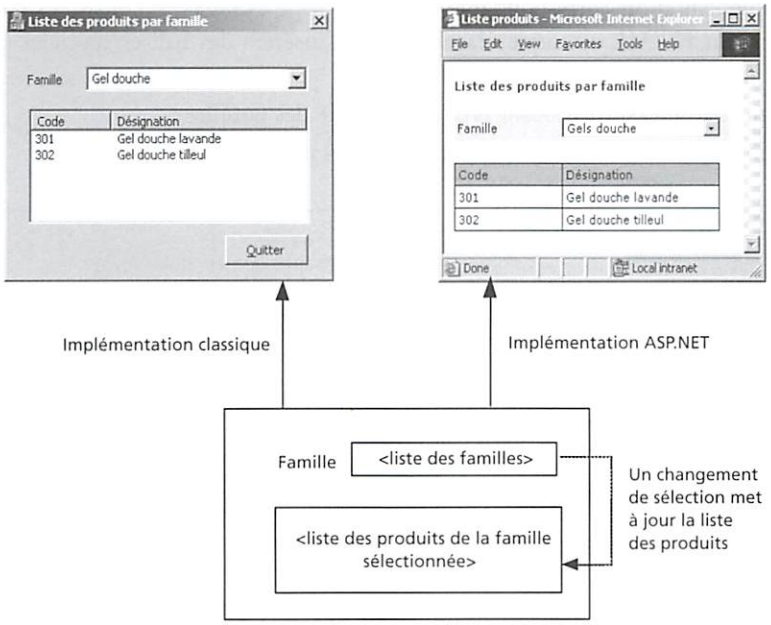


Figure 3–1 Deux implémentations d'une même vision conceptuelle

Nous allons détailler et mettre en pratique les trois premiers mécanismes dans la suite de ce chapitre, tandis que nous traiterons la gestion des événements dans le chapitre suivant.

À retenir

ASP.NET permet d'appréhender une page Web comme une interface graphique d'application client-serveur lors de la phase de développement, tout en produisant des pages HTML standards lors de la phase d'exécution.

Un contenu HTML mieux organisé grâce aux contrôles serveur

Comme pour une interface client classique, le développement d'une page ASP.NET se déroule en deux phases distinctes :

- la *réalisation de la maquette*, qui consiste à positionner les éléments graphiques sur la page en isolant, en particulier, les composants paramétrables ;
- l'*implémentation de la cinématique*, qui consiste à spécifier les actions à réaliser en fonction des événements qui surviennent.

Poursuivons avec notre exemple de page Web qui affiche la liste des produits ; cette page contient deux éléments paramétrables :

- une liste déroulante contenant les familles de produits, à remplir lors du chargement de la page ;
- un tableau de données contenant les produits de la famille sélectionnée, à remplir lors du chargement de la page et à actualiser lors de chaque changement de sélection dans la liste des familles.

APARTÉ

Un mode de programmation familier ?

Les développeurs d'applications client-serveur classiques (Visual Basic, Visual C++, Delphi…) sont familiers de ce modèle de programmation : ils découvriront dans ce chapitre comment ASP.NET l'applique dans le développement Web.
Les développeurs plus habitués à la réalisation de pages Web fondées sur des langages de scripts (VBScript, JScript…) découvriront la puissance de cette nouvelle approche, notamment en termes de productivité et de structuration du code.

L'implémentation correspondante dans une page ASP.NET consistera en un document HTML classique, dans lequel on insérera des balises spéciales pour chacun des composants paramétrables :

- une balise `<asp:DropDownList>` pour la liste des familles ;
- une balise `<asp:DataGrid>` pour la liste des produits.

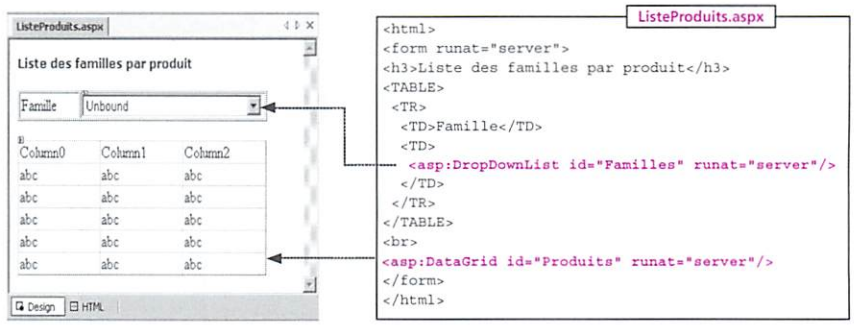


Figure 3–2 Conception de la maquette d'une page ASP.NET

Dans le vocabulaire ASP.NET, ces balises correspondent à des *contrôles serveur* : elles seront remplacées par de l'HTML généré dynamiquement lors de l'exécution de la page (figure 3-2).

Contrôle serveur

Composant graphique paramétrable, que l'on peut insérer dans une page ASP.NET et qui encapsule la génération de HTML. Les contrôles serveur participent à la séparation entre code et contenu graphique et accroissent la productivité du développement en facilitant la réutilisation.

Plus d'informations sur les contrôles serveur

Les contrôles serveur sont des éléments graphiques déclarés, positionnés dans la partie HTML de la page et accessibles sous forme d'objets dans la partie code.

La déclaration d'un contrôle serveur s'effectue de la manière suivante :

```
<asp:TypeControle runat = "server"
 id="NomDuControle"></asp:TypeControle>
```

On peut aussi employer la syntaxe abrégée :

```
<asp:TypeControle runat = "server"
 id="NomDuControle" />
```

Chaque contrôle serveur propose des propriétés permettant de contrôler son apparence et son comportement (taille, couleur, format, source de données liées, etc.). Lors de l'exécution de la page, les contrôles serveur génèrent un contenu HTML, en fonction de leur paramétrage et du navigateur utilisé par le client (adaptation du HTML généré en fonction des possibilités du navigateur).

Lors d'un « aller-retour » de la page suite à un événement donné, les contrôles serveur conservent leur valeur (grâce au mécanisme du **VIEWSTATE**). Enfin, il est possible d'associer des gestionnaires d'événements à un contrôle serveur.

Rassemblés dans l'espace de nommage **System.Web.UI**, les contrôles serveur fournis avec le framework .NET peuvent être répartis en plusieurs groupes :

- Les **contrôles HTML** : correspondant aux éléments HTML standards (boutons, zones de texte, liens...) , ils offrent l'avantage d'exposer leurs attributs sous forme de propriétés. Ils sont employés lorsqu'on souhaite contrôler l'apparence des éléments HTML depuis le code.
- Les **contrôles Web** : équivalents aux contrôles HTML standards, ils proposent des propriétés personnalisées qui aident au développement. Ils sont très fréquemment employés dans les pages ASP.NET.
- Les **contrôles de validation** : ils facilitent les opérations de validation de formulaires en vérifiant le contenu des champs (diverses règles de validation possibles) et en indiquant les champs erronés. De plus, le code de validation généré est adapté au navigateur du client.
- Les **contrôles liés à des données** : très utilisés dans les applications Web qui manipulent des données, ils permettent la présentation de listes de données et l'édition d'enregistrements.
- Les **contrôles complexes** : on regroupe dans cette catégorie les contrôles évolués tels que le calendrier ou le contrôle AdRotator, qui permet d'afficher des publicités en alternance.
- Les **contrôles mobiles** : ensemble de contrôles générant du contenu adapté à des terminaux mobiles (téléphone, Pocket PC...).

Enfin, notons qu'il est possible d'implémenter des contrôles serveur spécifiques ; il existe d'ailleurs un marché important de contrôles serveur proposés par des vendeurs indépendants.

Par exemple, le contrôle `DataGrid` (grille de données) sera remplacé à l'exécution par un tableau HTML :

```
<table>
   <tr>
   <td >Code</td><td>Désignation</td>
   </tr><tr>
      <td>101</td><td>Savon miel</td>
   </tr><tr>
      <td>102</td><td>Savon miel & amandes</td>
   </tr><tr>
      <td>103</td><td>Savon lavande</td>
   </tr>
</table>
```

Nous aurons l'occasion, au cours de cet ouvrage, de détailler les différents types de contrôles serveur ainsi que leurs possibilités.

Pour l'instant, nous n'avons parlé que de la partie graphique de la page… Mais où est le code qui régit la cinématique de la page ? En l'occurrence, où et comment indique-t-on que les listes de familles et de produits doivent être remplies à partir des tables correspondantes de la base de données, et qu'un changement de sélection de la famille doit provoquer le rechargement de la liste des produits ? Nous allons répondre à cette question sans plus attendre, dans le paragraphe qui suit.

Un code plus structuré grâce à la séparation du contenu HTML et de la cinématique

Avec les techniques de développement Web classique, le code qui régit le comportement de la page est *mélangé* au contenu HTML ; dans le cas d'une page ASP.NET, ce code est *séparé physiquement* du contenu HTML.

Le développeur dispose de deux techniques pour la localisation du code :

1. placer le code au sein du même fichier `.aspx` que le contenu HTML (*code in-line*) ;
2. placer le code dans un fichier séparé (*code behind*).

Alternative 1 - Placer le code et le HTML de la page dans un même fichier

Intéressons-nous dans un premier temps à la première technique, illustrée à la figure 3-3, qui est celle que nous emploierons le plus souvent dans cet ouvrage. Elle correspond à un découpage du fichier `.aspx` en deux blocs distincts :

- le bloc « HTML », composé d'un mélange de balises HTML classiques et de balises spécifiques ASP.NET correspondant généralement à des contrôles serveur ;
- le bloc « code », encadré par deux balises `<script>`, qui contient l'implémentation du paramétrage des contrôles et de la cinématique de la page ;

WEB MATRIX FACE À VS.NET

Gestion du code

Une des différences principales entre Web Matrix et Visual Studio.NET se situe dans la technique de gestion du code : Web Matrix a choisi de placer le code et le contenu HTML dans un fichier unique, tout en offrant des vues logiques separées, grâce à un système d'onglet, alors que VS.NET a adopté la technique du code behind, probablement à cause de son analogie avec Visual Basic.

SYNTAXE ASP.NET

Spécifier le langage utilisé

La directive <%@ Page Language...%> permet, entre autres, de spécifier le langage utilisé dans la page (VB.NET, C# ou JScript.NET). En l'absence de cette directive, le langage par défaut est VB.NET, sauf si les paramètres par défaut ont été changés dans le fichier `web.config` de l'application.

SYNTAXE ASP.NET

Position du bloc <script>

Le bloc de code est généralement placé en haut du fichier .aspx, avant le bloc de contenu HTML. Néanmoins, il est techniquement possible de le placer où on le souhaite : par exemple en dessous, voire au milieu du bloc HTML. On peut même répartir le code entre plusieurs blocs, à condition que chacun soit encadré par des balises <script>.

Notons que ces deux blocs sont, en général, précédés d'une directive <% Page ...%>, facultative mais utilisée dans la majorité des cas, qui permet de spécifier un certain nombre d'attributs de la page.

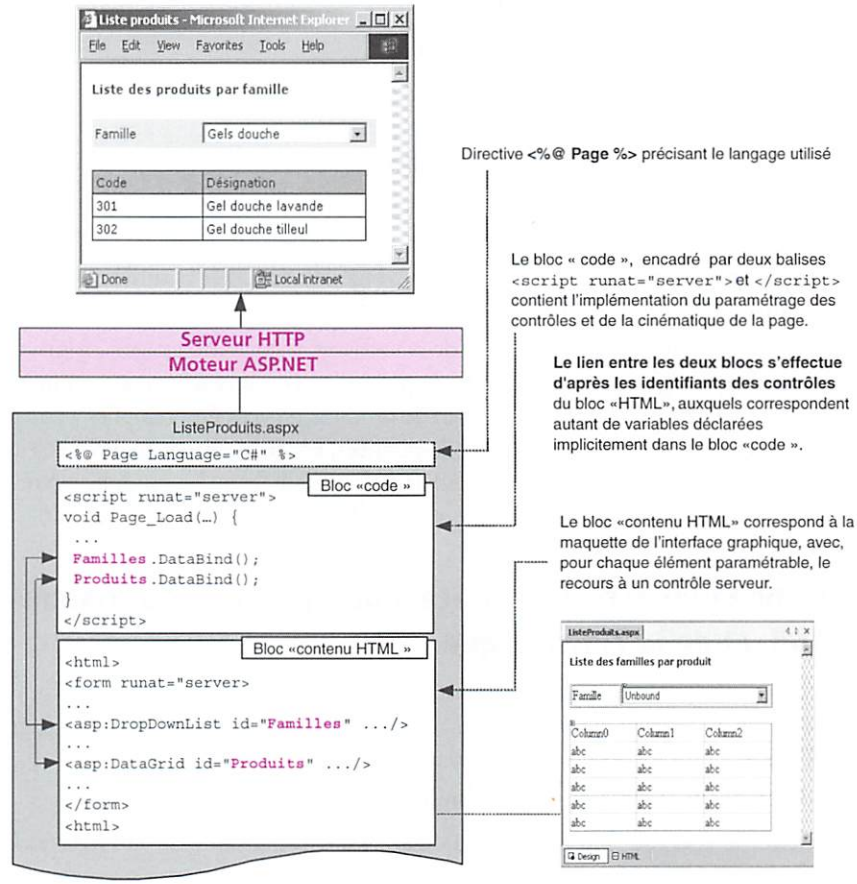


Figure 3–3 Architecture d'une page ASP.NET

ASP.NET

Compatibilité avec ASP

S'il est recommandé, pour une meilleure organisation, de placer l'intégralité du code de la page dans le bloc <script>, il est toutefois possible de placer du code au sein du contenu HTML, à l'intérieur d'un bloc <% ...%>, du fait de la compatibilité ascendante avec ASP.

Sans entrer dans le détail du code, soulignons dès à présent deux points d'architecture fondamentaux :

- **Le lien entre entre les contrôles serveur et le code s'effectue par l'intermédiaire des identifiants** : à chaque contrôle serveur contenu dans le bloc « HTML » correspond une variable déclarée *implicitement* dans le bloc « code », le nom et le type de cette variable correspondant respectivement au nom et au type du contrôle.
- **Le code de la page est organisé en gestionnaires d'événements** : chaque portion de code s'exécute en réaction à un événement donné (ici, en l'occurrence, l'événement `Page_Load`, correspondant au chargement de la page).

Ce dernier principe nous permettra, par exemple, d'implémenter le rechargement de la liste des produits lorsque la sélection de la famille change, comme nous le verrons dans le prochain chapitre.

Alternative 2 - Placer le code de la page dans un fichier séparé

Comme on l'a indiqué plus haut, une technique alternative (dite *code behind*) consiste à placer le code associé à la page dans un fichier distinct du fichier `.aspx` principal (figure 3-4).

Le principal avantage de cette séparation du contenu graphique et du code en deux fichiers distincts est de pouvoir faire travailler en parallèle un concepteur HTML et un développeur à la production d'une même page – à condition qu'ils se soient préalablement mis d'accord sur les types et les noms des contrôles.

À quoi ça ressemble ?

Sans entrer dans le détail de la syntaxe, nous présentons ici l'extrait du code correspondant à l'affichage du contenu d'une table dans un tableau HTML – en l'occurrence, notre liste de produits – en comparant la version ASP et la version ASP.NET, afin que le lecteur puisse dès maintenant voir « à quoi ça ressemble ».

Dans la version ASP, les éléments HTML et les instructions en langage de script sont entremêlées :

Version ASP

```
<%
Set conn = Server.CreateObject("ADODB.Connection")
conn.Open "<Chaîne de connexion>"
Set rs = conn.Execute("SELECT * FROM Produit")
%>

<table border="1">
<%
Do While Not rs.EOF
   %>
   <tr>
      <td><%= rs.Fields("Code").Value %></td>
      <td><%= rs.Fields("Designation").Value %>
      </td>
   </tr>
   <%
   rs.MoveNext
Loop
%>
</table>
<%
   rs.Close
   conn.Close
%>
```

Dans la version ASP.NET, on note la séparation claire entre le code (entre les balises `<script>`) et la partie graphique (entre les balises `<html>`), ainsi que l'organisation du code en gestionnaires d'événements (ici, un seul gestionnaire : Page_Load) :

Version ASP.NET (C#)

```
<script runat="server">

void Page_Load(Object sender, EventArgs e)
{
   SqlConnection conn ;
   SqlCommand cmd;
   SqlDataReader reader;

   string SQL = "SELECT * FROM Produit";

   conn = new SqlConnection(<Chaîne de connexion>);
   conn.Open();

   cmd = new SqlCommand(SQL,conn);
   reader = cmd.ExecuteReader();

   Produits.DataSource = reader
   Produits.DataBind();
   reader.Close();
   conn.Close();
}

</script>

<html>
<asp:DataGrid id="Produits" runat="server"/>
</html>
```

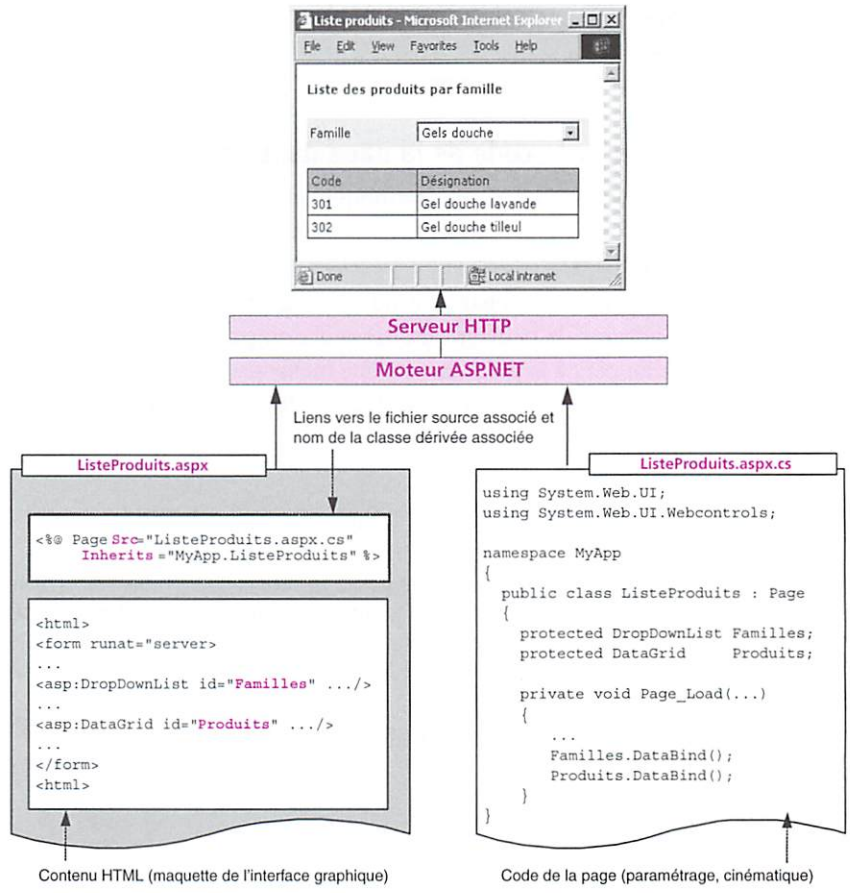


Figure 3–4 Architecture d'une page ASP.NET, avec code dans un fichier à part

Un second avantage, plus théorique : cette option fait apparaître clairement la nature technique des pages ASP.NET car le fichier code associé doit contenir la définition d'une classe dérivée de `System.Web.UI.Page` ; il faut déclarer, *explicitement* cette fois, des variables membres correspondant aux contrôles serveur placés dans le fichier `.aspx`.

Par convention, on nomme généralement le fichier code associé à une page d'après le nom du fichier `.aspx`, auquel on ajoute l'extension propre au langage (`.vb` pour VB.NET, `.cs` pour C#) : par exemple, `MyPage.aspx.vb` (ou `.cs`) pour le code associé à `MyPage.aspx`.

Le lien entre le fichier `.aspx` et le code associé s'effectue par l'intermédiaire de l'attribut `Inherits` de la directive `<%@ Page %>`, qui doit obligatoirement spécifier le nom de la classe correspondant à la page. L'attribut `Src`, facultatif, permet d'indiquer au compilateur *Just-In-Time* le nom du fichier source, qui est alors compilé à la volée lors de l'exécution de la page ; en l'absence de directive `Src`, le moteur ASP.NET recherche l'implémentation de la classe associée à la page parmi les assemblages disponibles.

Assemblage

Un assemblage (*assembly* en anglais) désigne une brique binaire élémentaire d'une application .NET. C'est en quelque sorte l'équivalent d'un composant COM dans le monde « Windows DNA » mais avec des fonctionnalités supplémentaires (regroupement logique de plusieurs fichiers DLL ou EXE dans un assemblage, gestion des versions, description de l'assemblage par des métadonnées…).

ASP.NET Différences entre Inherits, Src, et Codebehind

Il existe parfois une confusion entre ces trois attributs, tous utilisés dans la directive <%Page...%> et relatifs au placement du code dans une page à part, mais dont les significations sont différentes :

- `Inherits` est le seul attribut qui soit obligatoire dans le cas de placement du code dans un fichier séparé. Il doit spécifier le nom de la classe associée à la page (dérivée de `System.Web.UI.Page`).
- `Src` est un attribut facultatif spécifiant le chemin du fichier source associé à la classe. Si cet attribut est spécifié, le moteur ASP.NET effectuera une compilation à la volée (*JIT compilation*) de la page indiquée. Dans le cas contraire, le moteur recherchera la classe associée à la page parmi les assemblages disponibles.
- `Codebehind` est le plus trompeur des trois : bien qu'il porte le nom dont on qualifie la technique de séparation du code et du contenu, ce n'est pas un attribut ASP.NET : c'est en réalité un attribut spécifique de Visual Studio.NET utilisé par l'éditeur HTML pour savoir quel est le fichier source associé au fichier graphique.

Cette technique a néanmoins pour inconvénient de rendre le déploiement plus complexe : nombre de fichiers plus important, nécessité de compiler à l'avance les sources si l'on n'utilise pas l'attribut `Src`. En outre, notons qu'il est difficile d'échanger des pages ASP.NET entre deux outils de développement n'ayant pas adopté la même technique de gestion du code – en l'occurrence, entre Visual Studio.NET et Web Matrix.

Faciliter la réutilisation avec les contrôles utilisateur

La question de la réutilisation de « morceaux de pages » se pose fréquemment en développement Web : par exemple, on souhaite souvent placer une même barre de navigation sur toutes les pages d'un site ou utiliser un même composant (boîte de dialogue Login ou Recherche) dans plusieurs sites.

Pour faire cela avec une technologie classique comme ASP, il fallait utiliser une instruction de type `<!--#include file... -->` permettant d'inclure un fichier à un endroit donné. Ce mécanisme permettait, dans une certaine mesure, d'organiser le code mais présentait un certain nombre d'inconvénients : en particulier, il y avait risque de *collisions de noms* entre les variables du fichier principal et du fichier inclus.

Pour pallier ce type d'inconvénients, ASP.NET a introduit la notion de *contrôle utilisateur*, autrement dit de *morceau de page Web réutilisable* : du point de vue du développeur, un contrôle utilisateur s'implémente, à quelques détails près, comme une page ASP.NET normale ; du point de vue de la page utilisatrice, ce contrôle est vu comme un objet externe.

La figure 3-5 illustre l'utilisation d'un contrôle utilisateur – une barre de navigation :

- le contrôle doit être implémenté dans un fichier d'extension `.ascx` et doit comporter une directive `<% Control...%>` à la place de la directive `<% Page...%>` ;
- la page utilisatrice doit enregistrer le contrôle avec une directive `<%Register %>`, qui précise le nom du contrôle (ici : `Header`) et son espace de nommage (ici : `SDS` pour « Savons du Soleil ») ;
- enfin, il peut être fait référence au contrôle, à l'endroit où l'on souhaite l'insérer, comme pour un contrôle serveur normal : `<SDS:Header...runat="server">`.

Voici les principaux avantages offerts par les contrôles utilisateur :

- les noms utilisés par le contrôle sont *encapsulés* et ne risquent pas d'entrer en collision avec ceux de la page utilisatrice ;
- le contrôle est un objet pouvant exposer des propriétés : par exemple, l'index de la rubrique active d'une barre de navigation peut être paramétré depuis l'extérieur ;
- les contrôles sont organisés dans des espaces de nommage : plusieurs contrôles peuvent porter le même nom s'ils sont dans des espaces de nommage différents.

Contrôle utilisateur

Morceau de page Web encapsulé dans un composant graphique réutilisable. Très faciles à implémenter, les contrôles utilisateur permettent la réutilisation d'éléments graphiques constitutifs d'une application (barre de navigation, menu, etc.) et remplacent en quelque sorte les `<!-- #Include -->` des pages ASP).

Pagelets

Les contrôles utilisateur (*user controls*) sont parfois dénommés *pagelets*, que l'on peut approximativement traduire par « morceaux de page ».

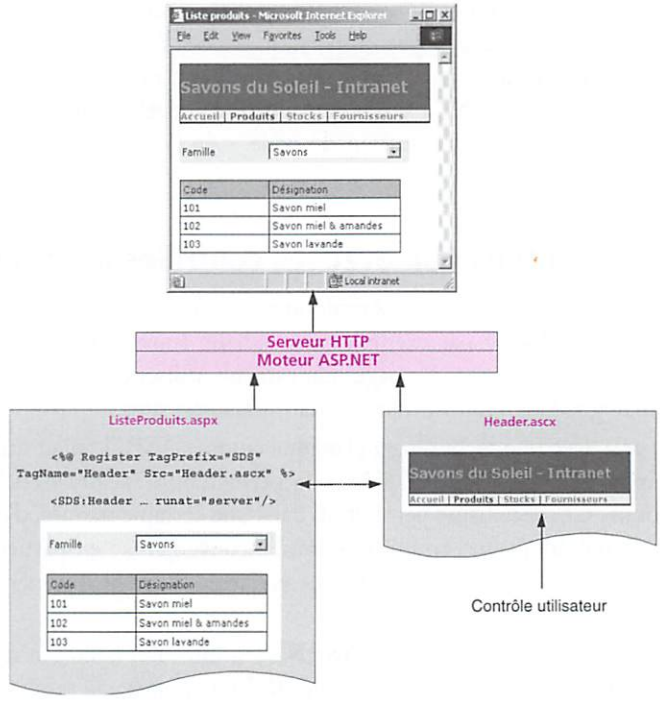


Figure 3–5 Utilisation d'un contrôle utilisateur

Il est à présent temps de mettre en pratique ces notions de contrôle serveur et contrôle utilisateur, en réalisant la barre de navigation qui sera utilisée dans notre étude de cas.

Mise en pratique : réalisation d'une barre de navigation

Dans cette seconde partie de chapitre, nous allons réaliser un contrôle utilisateur de type « barre de navigation », qui nous servira pour la suite de l'étude de cas : après une rapide prise en main de l'environnement de développement Web Matrix, nous réaliserons en deux temps la barre de navigation – partie graphique et implémentation - puis nous verrons comme l'intégrer au sein d'une page existante, en mettant en place le squelette de notre intranet.

Création de la barre de navigation

Notre projet est donc de réaliser une barre de navigation semblable à celle illustrée à la figure 3-6, qui sera présente dans toutes les pages de l'intranet et permettra à l'utilisateur de naviguer d'une rubrique à l'autre ; cette barre de navigation devra afficher le titre de la rubrique en cours et mettre en surbrillance la rubrique active.

Figure 3–6 Le projet de barre de navigation

Voyons maintenant comment utiliser Web Matrix pour créer ce contrôle utilisateur.

Création d'un contrôle utilisateur avec Web Matrix

Démarrez Web Matrix (les instructions d'installation se trouvent dans le chapitre précédent) :

- une boîte de dialogue apparaît, proposant différents types de fichiers (figure 3-7) ;
- choisissez ASP.NET User Control ;
- indiquez votre répertoire de travail dans le champ Location ;
- spécifiez un nom de fichier, par exemple : `NavBar.ascx` ;
- enfin, indiquez le langage que vous souhaitez utiliser (C# ou VB.NET).

Une fois la page créée, l'interface principale de Web Matrix apparaît (figure 3-8).

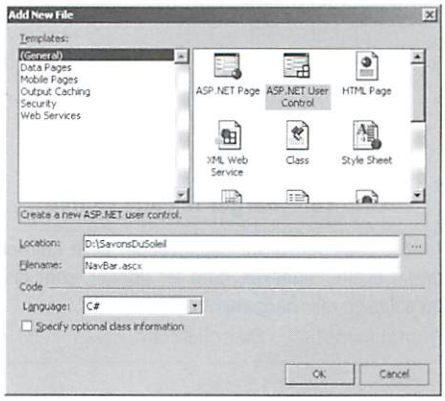


Figure 3–7 Choix du type de fichier dans Web Matrix

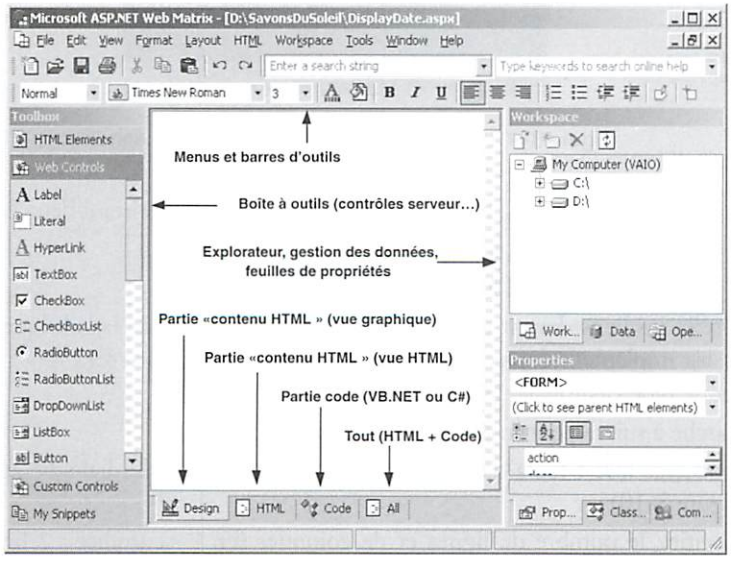


Figure 3–8 L'interface principale de Web Matrix

WEB MATRIX

Pas de Undo en mode design

À l'heure actuelle, la fonctionnalité Undo est inactive dans l'onglet Design de Web Matrix. Peut-être cette fonctionnalité sera-t-elle implémentée dans la version finale ?

La partie centrale de l'interface est occupée par un éditeur de texte, qui fait apparaître clairement la séparation entre contenu graphique et code grâce à quatre onglets :

- l'onglet Design permet une édition du contenu HTML en mode graphique ;
- l'onglet HTML contient le code HTML correspondant ;
- l'onglet Code contient le code ASP.NET de la page (VB.NET ou C#) ;
- l'onglet All affiche le fichier complet, tel qu'il est stocké sur le disque.

De part et d'autre de cette zone centrale sont situées un certain nombre de fenêtres utilitaires :

- boîtes à outils « contrôles » sur la gauche ;
- explorateurs (fichiers, données) et feuilles de propriétés sur la droite.

Pour l'instant, tous les onglets sont vides, à l'exclusion de l'onglet All qui commence par une directive destinée à indiquer au moteur ASP.NET que le contenu du fichier correspond à un contrôle utilisateur :

```
<%@ Control Language="<VotreLangage>" %>
```

Passons maintenant à la réalisation de notre barre de navigation, qui s'effectuera, comme il se doit, en deux temps : réalisation de la partie graphique puis implémentation du code.

Réalisation de la partie graphique de la barre de navigation

Pour commencer, nous allons réaliser la partie graphique de notre barre de navigation, en nous plaçant dans l'onglet Design de Web Matrix.

Figure 3–9 Partie graphique de la barre de navigation

RAPPEL **Télécharger le code source**

Tout le code source de l'application, ainsi que les divers fichiers annexes, dont les images, sont disponibles en téléchargement à l'adresse :

› http://www.savonsdusoleil.com

D'un point de vue HTML, notre contrôle est une table comportant deux lignes :

- la première contiendra le logo de la société et le titre ;
- la seconde les liens HTML vers les rubriques.

Le titre et les liens devant être paramétrables en fonction de la rubrique, nous allons les implémenter avec des *contrôles serveur*, de type `<asp:Label>` pour le titre et de type `<asp:Hyperlink>` pour les liens.

La marche à suivre pour insérer une table est très simple :

1. choisissez Insert Table dans le menu HTML : une boîte de dialogue apparaît (figure 3-10) ;
2. spécifiez le nombre de lignes et de colonnes (en l'occurrence : 2 lignes, 1 colonne).

Figure 3–10 Insérer une table

Malheureusement, les possibilités de paramétrage de la taille du tableau sont relativement réduites : la boîte de dialogue n'accepte que des valeurs en pixels. Dans notre cas, nous souhaitons que la table occupe 100% de la largeur de la page et que la hauteur ne soit pas spécifiée.

Pour cela, deux solutions sont possibles :

- utiliser la boîte de dialogue Propriétés pour modifier le style de la table (figure 3-11) ;
- modifier directement le contenu HTML (`<table style="WIDTH:100%">`).

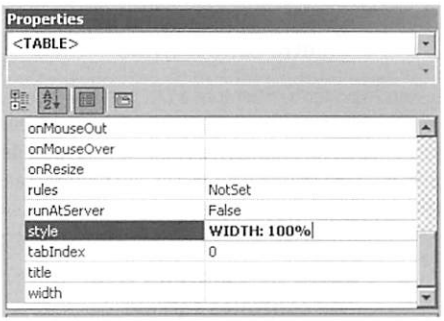


Figure 3–11 Modifier les propriétés de la table

Nous allons maintenant placer une image dans la première ligne de la table. Pour cela :

1. sélectionnez l'élément Image dans la boîte à outils Eléments HTML (figure 3-12) et le faire glisser vers la première ligne de la table ;
2. utilisez la boîte de dialogue Propriétés pour spécifier la source de l'image (`soleil.gif`) ;
3. spécifiez également la valeur `center` pour l'attribut `align`.

Enfin, spécifiez deux couleurs différentes pour les deux lignes de la table, par exemple :

- couleur verte pour la ligne supérieure (logo et titre) : `bgcolor="#00c000"` ;
- couleur jaune pour la ligne inférieure (rubriques) : `bgcolor="#ffff80"`.

Nous en avons fini avec la partie purement HTML de l'interface graphique ; nous allons maintenant passer à la mise en place des contrôles serveur.

Notre barre de navigation fera appel à deux types de contrôles serveur :

- un contrôle de type `Label` pour le titre ;
- quatre contrôles de type `HyperLink` pour les liens vers les rubriques.

Pour créer ces contrôles, on utilisera la boîte à outils Contrôles Web (figure 3-13) qui, lorsqu'on sélectionne puis place un contrôle, génère automatiquement la balise `<asp :TypeControle...>` correspondante dans le contenu HTML.

Commençons par le titre :

1. sélectionnez un contrôle serveur de type `Label` dans la barre d'outils ;
2. faites-le glisser vers la première ligne de la table, à droite du soleil (voir figure 3-9) ;
3. ajustez les caractéristiques du contrôle à l'aide des barres d'outils : gras, taille de police « 5 » et couleur jaune.

Plaçons maintenant les liens hypertexte :

1. sélectionnez un contrôle serveur de type `Hyperlink` dans la barre d'outils ;
2. faites-le glisser vers la deuxième ligne de la table (voir figure 3-9) ;
3. répétez trois fois l'opération, en séparant les contrôles par une barre verticale (« | »).

Servez-vous de l'onglet HTML de Web Matrix pour voir le code qui a été généré : une balise `<asp:TypeControle id="NomControle" runat="server">` a été créée pour chaque contrôle serveur.

Figure 3–12 La boîte à outils Eléments HTML

Figure 3–13 La boîte à outils Contrôles Web

ALTERNATIVE Utilisation d'un autre éditeur HTML

Si vous le préférez, il est tout à fait possible de réaliser le contenu HTML à l'aide d'un autre éditeur plus complet (Dreamweaver, FrontPage...), puis de revenir à Web Matrix pour placer les contrôles serveur et implémenter le code.

TERMINOLOGIE Contrôles Web ou contrôles serveur ?

Web Matrix regroupe sous le vocable de « contrôle Web » l'ensemble des contrôles serveur, à l'exclusion des contrôles mobiles, tandis que, dans la documentation .NET, le terme « contrôles Web » se réfère parfois uniquement aux équivalents des contrôles HTML standards (boutons, liens, zones de texte...).

Changer le nom des contrôles avec Web Matrix

Il existe deux moyens pour changer le nom d'un contrôle avec Web Matrix : modifier la rubrique (ID) dans la feuille de propriétés du contrôle, ou modifier directement la balise correspondante dans l'onglet HTML.

Les noms de contrôles générés par défaut sont de type `HyperLink1`, `HyperLink2`, etc. Pour plus de clarté, renommez-les conformément au tableau ci-dessous :

Nom du contrôle	Type	Rôle
Titre	<asp:Label>	Titre de la rubrique
Rubrique1	<asp:HyperLink>	Lien vers la rubrique « Accueil »
Rubrique2	<asp:HyperLink>	Lien vers la rubrique « Stocks »
Rubrique3	<asp:HyperLink>	Lien vers la rubrique « Fournisseurs »
Rubrique4	<asp:HyperLink>	Lien vers la rubrique « Ventes »

Voici, après renommage des contrôles, le contenu HTML de notre barre de navigation.

NavBar.ascx (partie graphique)

```
<table style="WIDTH: 100%">
  <tbody>
    <tr>
      <td bgcolor="#00c000">
        <img src="../img/soleil.gif" align="center" />
        <asp:Label id="Titre" runat="server" Font-Bold="True"
         Font-Size="Large" ForeColor="#ffff80"/>
      </td>
    </tr>
    <tr>
      <td bgcolor="#ffff80">
        <asp:HyperLink id="Rubrique1" runat="server"/> |
        <asp:HyperLink id="Rubrique2" runat="server"/> |
        <asp:HyperLink id="Rubrique3" runat="server"/> |
        <asp:HyperLink id="Rubrique4" runat="server"/> |
      </td>
    </tr>
  </tbody>
</table>
```

Et voilà, la partie graphique de notre contrôle utilisateur est terminée !

Nous allons maintenant passer dans l'onglet Code pour implémenter le paramétrage et la cinématique de notre barre de navigation.

Programmation de la barre de navigation

Lors du chargement de la barre de navigation, il faut effectuer les opérations suivantes :

- paramétrage des liens HTML vers les rubriques ;
- mise à jour du titre en fonction de la rubrique sélectionnée ;
- mise en surbrillance de la rubrique sélectionnée.

L'utilisateur de la barre de navigation doit avoir un moyen de spécifier la rubrique sélectionnée : ceci se fera par l'intermédiaire d'une *propriété* exposée par le contrôle utilisateur.

Pour réaliser la mise en surbrillance de la rubrique active, nous allons faire appel à une feuille de style, définissant une classe `active-rubrique` :

SDS.css (HTML)

```
body        { font-family: Verdana }
p           { font-size: 8pt  }
table       { font-size: 8pt }
h4          { font-size: 8pt; font-weight:bold; color: #000080 ; }
a           { font-size: 8pt; text-decoration:none; color: #000080 }
a.active-rubrique    { font-weight:bold }
```

Cette feuille de style (SDS comme « Savons du Soleil ») permet de définir quelques caractéristiques de la page : utilisation de la police Verdana (taille 8 points), liens en bleu non soulignés, lien en bleu gras pour la rubrique active. Avant de continuer, créez cette feuille de style avec un éditeur de texte ou avec Web Matrix et sauvegardez-la dans votre répertoire de travail.

Passons maintenant au code du contrôle, dont nous présentons ici deux versions, VB.NET et C# :

NavBar.ascx (code) – Version C#

```
public int SelectedIndex;
void Page_Load(Object sender, EventArgs e)
{
    Rubrique1.Text = "Accueil";
    Rubrique1.NavigateUrl = "default.aspx";
    Rubrique2.Text = "Stocks";
    Rubrique2.NavigateUrl = "stocks.aspx";
    Rubrique3.Text = "Fournisseurs";
    Rubrique3.NavigateUrl = "fournisseurs.aspx";
    Rubrique4.Text = "Ventes";
    Rubrique4.NavigateUrl = "ventes.aspx";
    switch (SelectedIndex)
    {
        case 0 :
            Titre.Text = "Accueil Intranet";
            Rubrique1.CssClass = "active-rubrique";
        break;
        case 1 :
            Titre.Text = "Suivi des stocks";
            Rubrique2.CssClass = "active-rubrique";
        break;
        case 2 :
            Titre.Text = "Gestion des fournisseurs";
            Rubrique3.CssClass = "active-rubrique";
        break;
        case 3 :
            Titre.Text = "Analyse des ventes";
            Rubrique4.CssClass = "active-rubrique";
        break;
    }
}
```

Propriété publique spécifiant la rubrique active

Mise à jour des liens

Mise à jour du titre

La propriété CssClass

Cette propriété permet de paramétrer la classe de style (CSS : *Cascading Style Sheet*) d'un contrôle serveur et correspond à l'attribut `class` d'une balise HTML. En l'occurrence, dans notre exemple, le contenu HTML généré pour le lien actif sera le suivant : `<a class="active-rubrique"...>`

Pour plus d'information sur les feuilles de style :

- http://www.w3.org/style/css

NavBar.ascx (code) – Version VB.NET

Propriété publique spécifiant la rubrique active

Mise à jour des liens

Mise à jour du titre

```
Public SelectedIndex As Int32
Sub Page_Load(sender As Object,e As EventArgs)
    Rubrique1.Text = "Accueil"
    Rubrique1.NavigateUrl = "default.aspx"
    Rubrique2.Text = "Stocks"
    Rubrique2.NavigateUrl = "stocks.aspx"
    Rubrique3.Text = "Fournisseurs"
    Rubrique3.NavigateUrl = "fournisseurs.aspx"
    Rubrique4.Text = "Ventes"
    Rubrique4.NavigateUrl = "ventes.aspx"
    Select Case SelectedIndex
        Case 0
            Titre.Text = "Accueil Intranet"
            Rubrique1.CssClass = "active-rubrique"
        case 1
            Titre.Text = "Suivi des stocks"
            Rubrique2.CssClass = "active-rubrique"
        case 2
            Titre.Text = "Gestion des fournisseurs"
            Rubrique3.CssClass = "active-rubrique"
        case 3
            Titre.Text = "Analyse des ventes"
            Rubrique4.CssClass = "active-rubrique"
    End Select
End Sub
```

L'examen de ce code est relativement rapide.

On commence par déclarer une propriété publique `SelectedIndex`, qui sera accessible depuis l'extérieur du contrôle et permettra à l'utilisateur de spécifier l'index de la rubrique active (entre 0 et 3) depuis la page qui incluera le contrôle.

Puis, dans le gestionnaire `Page_Load` exécuté lors du chargement de la page, on paramètre les liens hypertextes – mise à jour du texte et de l'adresse de lien via les propriétés `Text` et `NavigateUrl` – on spécifie le titre – mise à jour de la propriété `Text` du contrôle `Titre` – puis on attribue à la rubrique active le style `active-rubrique`, défini dans la feuille de style grâce la propriété `CssClass`.

Notre barre de navigation est maintenant terminée. Néanmoins, il n'est pas possible de la tester individuellement : il nous faut obligatoirement l'intégrer dans une page ASP.NET. C'est ce que nous allons faire dans le paragraphe qui suit en créant le squelette de notre intranet.

À propos de Page_Load

Dans le chapitre suivant, nous aurons l'occasion de parler plus amplement des événements liés à une page : `Page_Init`, `Page_Load`, etc. Dans un premier temps, retenons uniquement que `Page_Load` est appelé lors du chargement de la page.

Aparté pour les programmeurs objet

L'objet exécutable qui produit le code HTML correspondant à cette page est une instance d'une classe dérivée de `System.Web.UI.Page`. `Page_Load` est une fonction virtuelle redéfinie dans cette classe dérivée. Enfin, les contrôles serveur sont implicitement déclarés comme des variables membres (propriétés) privées .

Création du squelette de l'intranet

Notre intranet sera composé de quatre rubriques principales :

- page d'accueil ;
- suivi des stocks ;

- gestion des fournisseurs ;
- analyse des ventes.

La barre de navigation devra apparaître en haut de chacune des pages, la rubrique active étant sélectionnée (voir figure 3-14).

Commençons par réaliser la page d'accueil : pour cela, créer dans Web Matrix un nouveau fichier nommé `default.aspx`, en choisissant cette fois le type `ASP.NET Page` (c'est le choix par défaut).

L'intégration de la barre de navigation en haut de la page se fait en deux étapes :

1. enregistrement du contrôle utilisateur avec l'instruction `<% Register...%>` ;
2. positionnement du contrôle dans la page.

L'enregistrement du contrôle, indispensable à son utilisation dans la page, s'effectue à l'aide de l'instruction suivante, placée en haut du fichier (juste après la directive `<%@ Page ...%>`) :

```
<%@ Register TagPrefix="SDS" TagName="NavBar" Src="NavBar.ascx" %>
```

Les significations des différents attributs, tous obligatoires, sont les suivantes :

- `TagPrefix` indique le préfixe qui sera utilisé pour référencer le contrôle, autrement dit l'espace de nommage auquel il appartient ;
- `TagName` spécifie le nom du contrôle ;
- `Src` indique le nom du fichier source correspondant au contrôle.

L'enregistrement effectué, on peut maintenant inclure la balise correspondant au contrôle, à l'endroit où l'on souhaite que la barre de navigation soit insérée :

```
<SDS:NavBar id="MyNavBar" SelectedIndex="0" runat="server">
</SDS:NavBar>
```

On note évidemment la similitude avec une balise de contrôle serveur, la différence étant qu'on a remplacé `<asp:TypeControle...>` par `<TagPrefix:TagName...>`.

Autre point important, *la propriété publique* `SelectedIndex`, déclarée dans le code du contrôle, *est accessible sous forme d'attribut de la balise du contrôle* ; dans notre cas, l'index correspondant à la page d'accueil est 0.

Terminons la réalisation de notre page d'accueil en insérant un lien vers la feuille de style externe `SDS.css` définie plus haut, indispensable pour la mise à jour de la rubrique active :

```
<link href="SDS.css" type="text/css" rel="stylesheet" />
```

Enfin, pour remplir la page, on se propose d'ajouter des liens vers les autres rubriques dans le corps de la page.

Figure 3–14 Page d'accueil de l'intranet

À propos de TagPrefix

`TagPrefix` contient généralement le nom de l'application ou le nom de la société (ici : SDS pour « Savons du Soleil »). Le principal intérêt de cet attribut est de permettre l'existence de plusieurs contrôles portant le même nom, à condition qu'ils soient dans des espaces de nommage distincts.

Le nom du contrôle est fixé depuis la page cliente

Notons que le nom de la classe du contrôle est fixé par la page utilisatrice, via les attributs `TagPrefix` et `TagName`, contrairement à ce qui se passe dans une interface classique, de type Visual Basic, où seul le nom de la variable correspondant au contrôle est paramétrable par l'utilisateur.

Voici le code complet de la partie graphique de la page d'accueil :

default.aspx (partie graphique)

```
<%@ Page Language="<VotreLangage>" Debug="true" %>
<%@ Register TagPrefix="SDS" TagName="NavBar" Src="NavBar.ascx" %>
<html>
<head>
    <title>Savons du Soleil - Intranet</title>
    <link href="SDS.css" type="text/css" rel="stylesheet" />
</head>
<body>
    <SDS:NavBar id="MyNavBar" SelectedIndex="0" runat="server">
    </SDS:NavBar>
    <p></p><p></p>
    <form runat="server">
        <h4>Bienvenue sur l'intranet des Savons du soleil
        </h4>
        <p><a href="./Stocks.aspx">Suivi des stocks</a> </p>
        <p><a href="./Fournisseurs.aspx">Gestion des fournisseurs</a>
        </p>
        <p><a href="./Ventes.aspx">Analyse des ventes</a></p>
    </form>
</body>
</html>
```

Il reste à répéter trois fois l'opération pour les autres rubriques, en modifiant uniquement le titre de la page, le nom du fichier et la valeur de `SelectedIndex`.

Rubrique	Nom du fichier	Valeur de SelectedIndex
Suivi des stocks	stocks.aspx	1
Gestion des fournisseurs	fournisseurs.aspx	2
Analyse des ventes	ventes.aspx	3

À titre d'exemple, voici le code de la partie graphique du fichier `stocks.aspx` :

stocks.aspx (partie graphique)

```
<%@ Page Language="<VotreLangage>" Debug="true" %>
<%@ Register TagPrefix="SDS" TagName="NavBar" Src="NavBar.ascx" %>
<html>
<head>
    <title>Suivi des stocks</title>
    <link href="SDS.css" type="text/css" rel="stylesheet" />
</head>
<body>
    <SDS:NavBar id="MyNavBar" runat="server" SelectedIndex="1">
    </SDS:NavBar>
    <p></p> <p></p>
    <form runat="server">
        <h4>Bienvenue dans le module de suivi des stocks </h4>
    </form>
</body>
</html>
```

Le développement du squelette de notre application est terminé : il ne reste plus qu'à le tester !

Tester l'application

Pour tester notre application, deux options sont possibles :

- utiliser le serveur HTTP léger fourni avec Web Matrix ;
- utiliser le serveur IIS.

À propos de la compilation des pages ASP.NET

Si vous avez fait une erreur dans la syntaxe du code (par exemple, vous avez omis le « ; » final d'une instruction en C#), une erreur de compilation se produira : vous verrez alors apparaître un écran similaire à celui représenté à la figure 3-15.

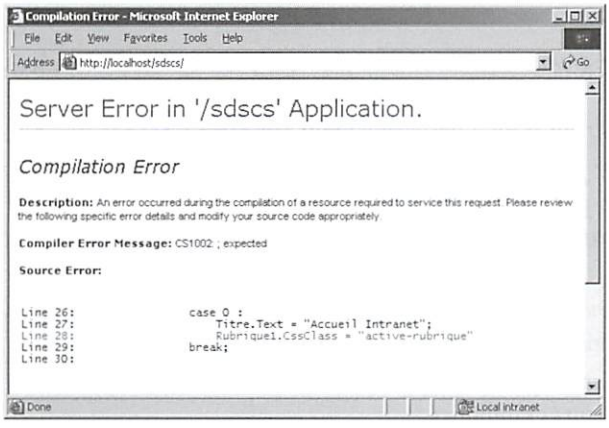


Figure 3–15 Erreur de compilation

Ceci permet au passage de souligner un des avantages du caractère *compilé* des pages ASP.NET : l'*intégralité* du code est passée en revue avant la première exécution de la page et *toutes* les erreurs de syntaxe sont détectées lors de cette phase de compilation, alors qu'avec une page interprétée (de type ASP), seules les erreurs dans les éléments interprétés étaient détectées (à l'exclusion, par exemple, des erreurs dans les branches non exécutées du code).

Néanmoins, la compilation n'élimine pas totalement le risque d'erreur à l'exécution, des *exceptions* pouvant toujours se produire.

Le résultat de la compilation de la page est un fichier exécutable – un assemblage dans le langage .NET – stocké sous la forme d'une DLL dans le répertoire Temporary ASP.NET Files situé sous le répertoire Microsoft.NET dans le dossier système (figure 3-16).

Si vous jetez un œil à ce dossier – ou plus précisément dans celui des ses sous-dossiers qui porte le nom de votre application – vous y trouverez les fichiers `.dll` correspondant aux résultats de la compilation de

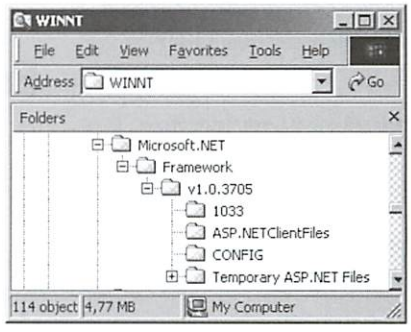


Figure 3–16 Le dossier Temporary ASP.NET Files

vos pages : chacun de ces fichiers est, en quelque sorte, un exécutable capable de « cracher » le HTML correspondant à la page ou au contrôle utilisateur.

Ces fichiers contiennent du langage IL (*Intermediate Language*) qu'il est possible de désassembler en utilisant l'utilitaire `ildasm.exe` situé dans le répertoire FrameworkSDK (figure 3-17).

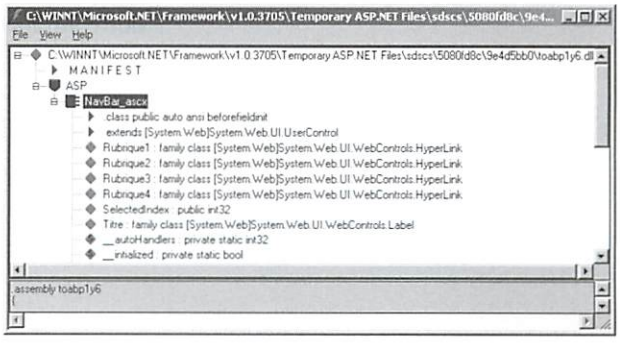


Figure 3–17 Désassemblage de la barre de navigation

Rappelons pour finir que la compilation n'intervient que lors de la première requête vers la page ; lors de requêtes ultérieures, l'assemblage en cache est réutilisé, sauf si le code source a été modifié entre temps.

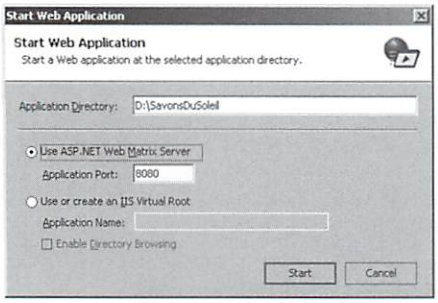


Figure 3–18 Choix du serveur HTTP

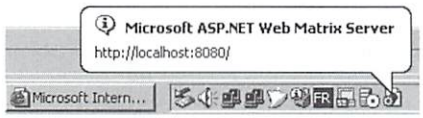


Figure 3–19 Démarrage du serveur Web Matrix

Positionnez-vous sur la page d'accueil et sélectionnez Start dans le menu View, ou utilisez la touche de raccourci F5, pour faire apparaître la boîte de dialogue permettant d'effectuer le choix du serveur HTTP (figure 3-18).

La première option permet de sélectionner le serveur Web Matrix (figure 3-19) qui a pour principal avantage d'éviter l'installation d'IIS, bien qu'il soit plus limité fonctionnellement (pas de racines virtuelles, de pages d'erreurs personnalisées, etc.).

La seconde option permet d'utiliser le serveur IIS : dans la case Application Name, saisissez le nom de la racine virtuelle de votre application. Si elle n'existe pas, Web Matrix la créera.

En résumé...

Dans ce long chapitre, nous avons présenté les principes fondamentaux de la technologie ASP.NET :

- le *modèle de programmation orienté interface* qui permet d'appréhender une page Web comme une interface graphique classique : un assemblage de contrôles graphiques réagissant à des événements ;
- la *séparation entre partie graphique et code*, qui rend le contenu HTML plus clair car il n'est plus entremêlé avec des scripts et peut faciliter, le cas échéant, la répartition du travail entre développeurs et concepteurs (designers) ;
- les *contrôles serveur*, qui augmentent la productivité du développement en encapsulant la génération de HTML et peuvent réagir à des événements ;
- les *contrôles utilisateur* qui constituent un moyen efficace et simple de réutilisation de composants graphiques.

Puis nous avons mis en œuvre ces mécanismes pour réaliser la maquette de notre intranet, en effectuant au passage un prise en main de l'environnement de développement Web Matrix.

Dans le chapitre suivant, nous allons implémenter le module de suivi des stocks de notre application, qui sera l'occasion d'illustrer deux mécanismes importants :

- l'accès à une base de données depuis une application ASP.NET ;
- la gestion des événements côté serveur.

Consulter une base de données : l'interface de suivi des stocks

4

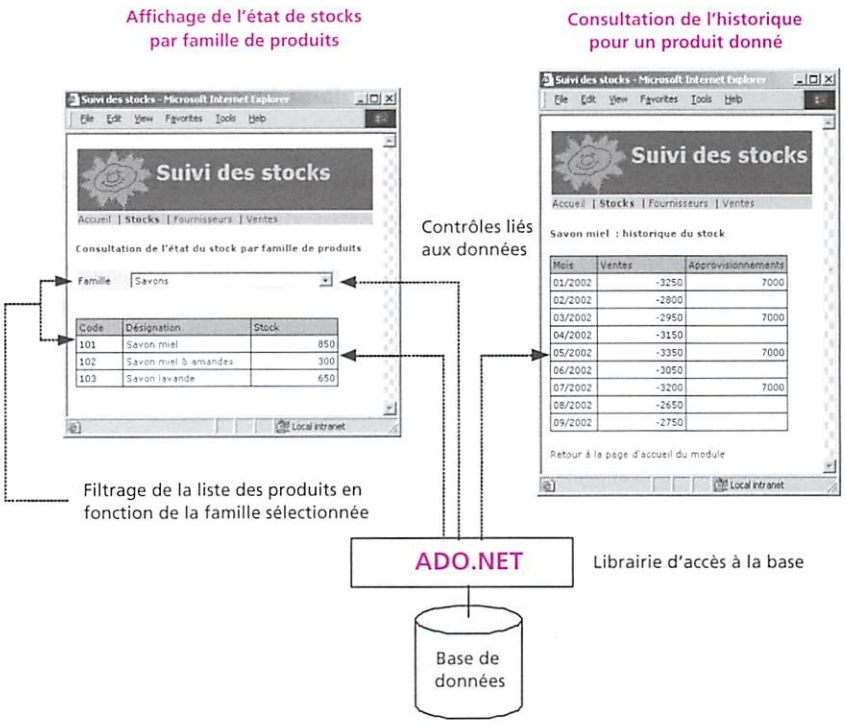
ASP.NET

ADO.NET | DataGrid | DropDownList | ViewState | IsPostBack | global.asax | Session

SOMMAIRE

MOTS-CLÉS

- ADO.NET
- DataGrid
- DropDownList
- SqlConnection
- SqlCommand
- SqlDataReader
- SqlDataAdapter
- ViewState
- IsPostBack
- global.asax
- Session



Dans ce chapitre, nous réalisons l'interface de suivi des stocks, qui repose principalement sur l'utilisation de contrôles serveur (`DataGrid` et `DropDownList`) liés à la source de données par l'intermédiaire de la bibliothèque ADO.NET, que nous présentons au passage. Nous abordons également le mécanisme des événements serveur, lequel nous permettra d'ajouter de l'interactivité à notre interface.

Réalisation de la maquette de l'interface

Le module de suivi des stocks doit permettre, d'une part, l'affichage de la synthèse des quantités en stock, filtrées par famille de produits et, d'autre part, la consultation de l'historique des variations de stocks pour un produit donné.

Notre interface sera, par conséquent, constituée de deux pages distinctes :

- La page `stocks.aspx`, écran d'accueil du module, présentera la liste des stocks disponibles pour une famille de produits donnée. Il devra être possible de sélectionner la famille désirée dans une liste déroulante, la liste étant alors mise à jour en conséquence ; et il devra également être possible de cliquer sur un produit donné pour accéder à l'historique du stock de ce produit.
- La page `detailstock.aspx` présentera l'historique des stocks d'un produit donné (détail des ventes et approvisionnement par mois).

Comme pour toute page ASP.NET, le développement se déroulera en deux temps :

- réalisation de la maquette de la page ;
- implémentation de la cinématique de la page.

Nous allons donc commencer par réaliser la maquette de ces deux pages.

Maquette de la page de consultation des stocks par famille de produits

À la fin du chapitre précédent, nous avons réalisé le squelette de la page d'accueil du module de suivi des stocks (fichier `stocks.aspx`), dans laquelle nous avons déjà intégré la barre de navigation.

Nous allons repartir de cette ébauche, à laquelle nous allons ajouter deux contrôles serveur :

- un contrôle de type `DropDownList` (liste déroulante) pour afficher la liste des familles ;
- un contrôle de type `DataGrid` (grille de données) pour afficher l'état des stocks.

Le résultat auquel nous parviendrons lorsque la page sera réalisée est représenté figure 4-1.

Avant de modifier notre page, nous allons en sauvegarder une copie, qui nous servira également de point de départ pour la page de consultation détaillée de l'historique :

1. Démarrez Web Matrix.
2. Ouvrez la version actuelle du fichier `stocks.aspx`.
3. Sauvegardez-la sous le nom `detailstocks.aspx`, dans le même répertoire.

Ceci étant fait, nous pouvons passer à la réalisation de la maquette de notre première page :

1. Ouvrez à nouveau la version actuelle du fichier `stocks.aspx`.

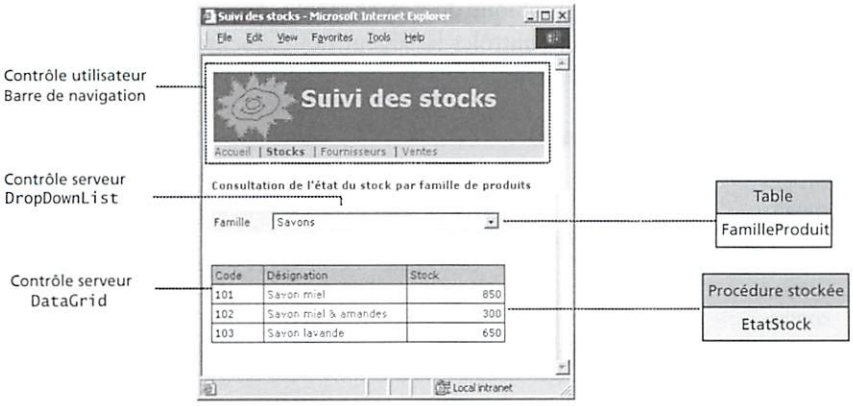


Figure 4–1 La page de consultation des stocks par famille de produits

2. Placez-vous dans l'onglet Design de Web Matrix.
3. Insérez un titre (par exemple : « Consultation de l'état du stock par famille de produits »).
4. Créez un tableau HTML (une ligne ; deux colonnes, arrière-plan jaune).
5. Insérez un contrôle serveur `DropDownList` dans la partie droite de ce tableau et renommez-le `ListeFamilles`.
6. Insérez un contrôle serveur `DataGrid` sous le tableau et renommez-le `EtatStock`.

Le résultat de la maquette doit ressembler à celle montrée figure 4-2.

Figure 4–2 Maquette de la page de suivi des stocks

RAPPEL Insérer un contrôle serveur

Pour insérer un contrôle serveur, il faut utiliser l'onglet Web Controls de Web Matrix et faire glisser les éléments voulus aux endroits souhaités.

ATTENTION Ne pas confondre DataGrid et MxDataGrid

Nous utiliserons dans notre étude de cas le contrôle serveur standard `DataGrid`, à ne pas confondre avec le contrôle serveur spécifique nommé `MxDataGrid`, fourni avec Web Matrix, qui offre également des fonctionnalités d'affichage de grilles de données, mais que nous n'étudierons pas dans cet ouvrage.

RAPPEL Renommer un contrôle

Il existe deux moyens pour changer le nom d'un contrôle avec Web Matrix : modifier la rubrique (ID) dans la feuille de propriétés du contrôle, ou modifier directement la balise correspondante dans l'onglet HTML.

Voici le code de la partie graphique du fichier `stocks.aspx` après ajout de quelques attributs de style pour contrôler la largeur des éléments (les contrôles serveur sont indiqués en couleurs) :

stocks.aspx (partie graphique)

```
<html>
<head>
   <title>Suivi des stocks</title>
   <link href="SDS.css" type="text/css" rel="stylesheet" />
</head>
<body>
   <SDS:NavBar id="MyNavBar" runat="server" SelectedIndex="1">
   </SDS:NavBar>
   <p></p>
   <p></p>
   <form runat="server">
      <h4>Consultation de l'état du stock par famille de produits</h4>
      <table style="width: 325px" bgcolor="#ffffc0"><tbody>
         <tr>
            <td width="60">Famille</td>
            <td>
               <asp:DropDownList id="ListeFamilles" runat="server"
                width="250px">
               </asp:DropDownList>
            </td>
         </tr>
         </tbody></table>
      <br />
      <asp:DataGrid id="EtatStock" runat="server"></asp:DataGrid>
   </form>
</body>
</html>
```

Nous avons fait la moitié du travail de réalisation de la maquette ; passons maintenant à la page de consultation de l'historique du stock d'un produit donné.

Maquette de la page de consultation de l'historique du stock d'un produit

Pour réaliser cette page, nous allons partir de l'ébauche de page, enregistrée au paragraphe précédent sous le nom `detailstocks.aspx`, dans laquelle nous allons ajouter :

- un contrôle de type `Label` (étiquette de texte) pour afficher le nom du produit ;
- un contrôle de type `DataGrid` (grille de données) pour afficher l'historique du stock.

Le résultat auquel nous parviendrons lorsque la page sera réalisée est représenté figure 4-3.

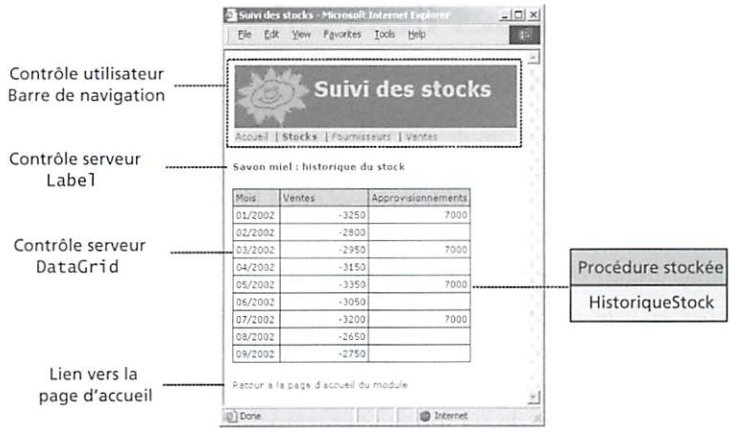


Figure 4–3 La page de consultation de l'historique du stock pour un produit

Voici les étapes nécessaires à la réalisation de cette page :

1. Ouvrez le fichier `detailstocks.aspx` créé précédemment.
2. Placez-vous dans l'onglet Design de Web Matrix.
3. Insérez un contrôle serveur de type `Label`, renommez-le `NomProduit`, puis appliquez-lui le style Heading 4 et faites-le suivre du texte : « historique du stock ».
4. Insérez un contrôle serveur de type `DataGrid` et renommez-le `HistoriqueStock`.
5. Insérez un lien hypertexte « Retour à la page d'accueil du module » pointant vers la page `stocks.aspx`.

Le résultat de la maquette doit ressembler à l'illustration de la figure 4-4.

Voici le code de la partie graphique du fichier `detailstock.aspx` :

```
<html>
<head>
   <title>Suivi des stocks</title>
   <link href="SDS.css" type="text/css" rel="stylesheet" />
</head>
<body>
   <SDS:NavBar id="NavBar" SelectedIndex="1" runat="server">
   </SDS:NavBar>
   <p></p>
   <form runat="server">
      <h4><asp:Label id="NomProduit" runat="server">Label</asp:Label>
      </h4>
      <asp:DataGrid id="HistoriqueStock" runat="server"></asp:DataGrid>
      <p></p>
      <a href="stocks.aspx">Retour à la page d'accueil du module</a>
   </form>
</body>
</html>
```

Nous en avons terminé avec la phase de maquettage… passons maintenant au cœur du problème : l'implémentation du lien entre l'interface et la base de données.

WEB MATRIX Insérer un lien hypertexte

Pour insérer un lien hypertexte, il faut choisir Insert HyperLink dans le menu HTML ou, au choix, sélectionner le texte à transformer en lien et utiliser le raccourci clavier Ctrl + K.

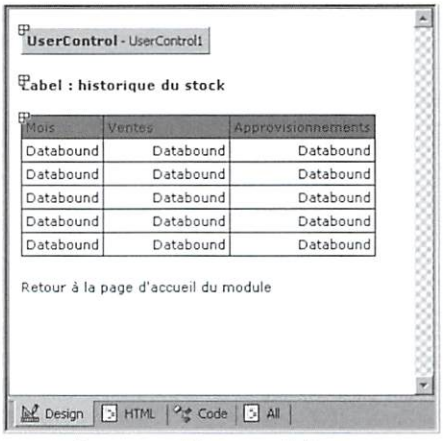


Figure 4–4 Maquette de la page de consultation de l'historique

Mise en place des liens entre l'interface et la base de données

À l'image d'un grand nombre de contrôles serveur ASP.NET, `DropDownList` et `DataGrid` offrent la possibilité d'être liés à une source de données : autrement dit, leur contenu peut être automatiquement mis à jour à partir de valeurs contenues dans une base de données, un fichier XML, voire un tableau en mémoire.

Nous allons utiliser cette fonctionnalité pour :

- afficher la liste des familles dans le contrôle `ListeFamilles` à partir des valeurs contenues dans la table `FamilleProduit` ;
- afficher l'état du stock correspondant dans le contrôle `EtatStock` à partir du résultat de l'exécution de la procédure `EtatStock`.

La communication avec la source de données s'effectuera par l'intermédiaire de la bibliothèque ADO.NET, dont nous allons présenter les principales classes.

Différences entre ADO.NET et ADO

Même si elles portent des noms proches, les librairies ADO et ADO.NET n'ont que peu de choses en commun : ADO.NET intègre le support natif de SQL Server (ADO utilisait systématiquement OLE-DB), un `DataSet` qui représente un magasin de données en mémoire, pouvant contenir plusieurs tables, être utilisé en mode déconnecté puis transmettre le changement vers la base (alors que le `DataSet` d'ADO ne représentait qu'un seul jeu d'enregistrement et était connecté en permanence à la base de données).

Présentation de la librairie ADO.NET

ADO.NET est un nom commercial qui désigne un ensemble de classes utilisées pour communiquer avec des sources de données et manipuler des données (figure 4-5) :

- L'espace de nommage `System.Data` fournit des classes utilitaires qui permettent de manipuler des données ; ces classes sont toutes indépendantes de la source de données.
- L'espace de nommage `System.Data.OleDb` fournit les classes nécessaires à la communication avec une source de données OLE-DB.
- L'espace de nommage `System.Data.SqlClient` fournit les classes nécessaires à la communication avec une base de données SQL Server ou MSDE.

Espace de nommage

Un espace de nommage (*namespace*, parfois traduit par espace de noms) est un regroupement de classes dans une enveloppe conceptuelle, qui les isole des conflits de noms avec l'extérieur : en pratique, deux classes peuvent porter le même nom si elles sont situées dans deux espaces de nommage différents. Les classes étant généralement regroupées par fonctionnalités, les espaces de nommage ont également un rôle dans l'organisation des bibliothèques de classes.

OleDb et SqlClient « jumeaux »

Ces deux espaces de nommage sont en quelque sorte jumeaux, dans la mesure où ils contiennent les mêmes classes préfixées soit par `OleDb`, soit par `Sql` : par exemple, `OleDbConnection` et `SqlConnection`.

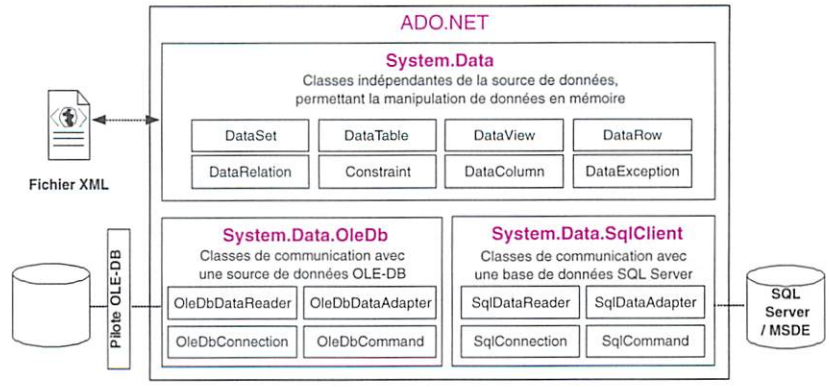


Figure 4–5 Organisation de la bibliothèque ADO.NET

La première caractéristique notable de l'architecture de cette bibliothèque est la séparation nette entre les classes de communication, dépendantes de la source de données, et les classes de manipulation, indépendantes de la source de données : le but étant de disposer d'une séparation entre la couche présentation et la couche accès aux données, afin de rendre le code plus modulaire et plus facile à faire évoluer.

Le second point notable est la richesse de la bibliothèque `System.Data`, qui dispose de toutes les classes nécessaires à la création d'une véritable petite base de données en mémoire, incluant un schéma relationnel, des contraintes et même des déclencheurs ! L'intérêt principal d'une telle architecture est d'adapter l'accès aux données au caractère déconnecté des applications Web : les classes de manipulation permettent en effet de conserver les données en cache et de transmettre les changements effectués à la base, lors de la connexion suivante (nous aurons l'occasion d'illustrer ce mécanisme dans le chapitre suivant).

À l'heure actuelle, ADO.NET propose deux modes d'accès aux données :

- Utilisation du fournisseur OLE-DB, qui permet d'accéder à toute base de données dotée d'un pilote OLE-DB (classes correspondantes implémentées dans l'espace de nommage `System.Data.OleDb`).
- Utilisation du fournisseur SQL natif, qui permet d'accéder de manière native (autrement dit, plus rapidement et avec accès à l'ensemble des fonctionnalités disponibles) aux bases de données SQL Server ou MSDE (classes correspondantes implémentées dans l'espace de nommage `System.Data.SqlClient`).

Si vous disposez d'une base de données SQL Server ou MSDE, il est fortement recommandé d'utiliser le fournisseur SQL natif, plus performant : c'est ce que nous allons faire dans la section suivante, en implémentant la connexion à la base avec `SqlConnection`.

À propos d'OLE-DB

OLE-DB est une spécification d'interface publique pour l'implémentation de pilotes d'accès à des sources de données : c'est un successeur d'ODBC qui ne se limite pas uniquement aux sources de données relationnelles (possibilité d'accéder à un fichier Excel, par exemple).
Pour plus d'informations, voir :

- www.microsoft.com/data

Autres fournisseurs : OBDC, Oracle...

Il est prévu que la liste des fournisseurs de données compatibles .NET s'étoffe progressivement. Actuellement, deux extensions sont disponibles en téléchargement : un fournisseur ODBC et un fournisseur natif Oracle. Il faut aller sur :

- www.microsoft.com/downloads

et chercher « provider ».

Établissement de la connexion à la base de données

Dans cette section, nous allons voir comment utiliser la bibliothèque ADO.NET pour se connecter à une base de données, à l'aide de la classe `SqlConnection`.

Notre connexion étant destinée à être utilisée plusieurs fois au sein de notre application, nous allons également voir comment la partager, autrement dit, l'implémenter dans un endroit central accessible à tous les éléments de l'application, grâce à l'objet `Session` et au fichier `global.asax`.

Consulter la documentation des bibliothèques .NET

Pour consulter la liste exhaustive et les caractéristiques des classes contenues dans ADO.NET ou dans les autres bibliothèques .NET, plusieurs moyens sont à notre disposition.

1 - Utilisation de Web Matrix

Web Matrix dispose d'un onglet Classes (dans le coin inférieur droit de l'interface) qui permet de consulter la liste des classes disponibles dans les bibliothèques .NET (figure 4-6).

En double-cliquant sur une classe, on fait apparaître le volet de documentation correspondant (figure 4-7).

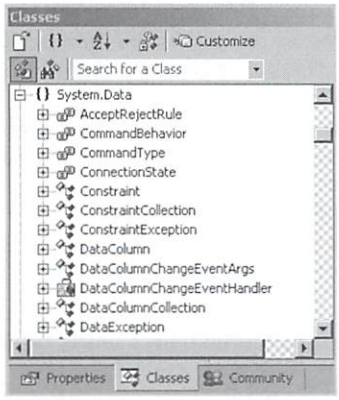


Figure 4–6 L'onglet Classes de Web Matrix

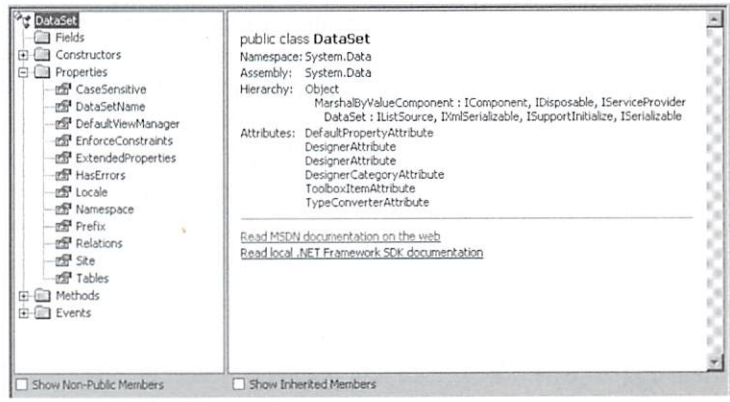


Figure 4–7 Volet Documentation de Web Matrix

2 – Documentation .NET installée sur le poste de travail

Lors de l'installation du .NET Framework SDK sur votre poste de travail (voir chapitre 2), une documentation complète a été installée : elle est accessible via le menu Démarrer (figure 4-8) et détaille de manière exhaustive les classes disponibles (figure 4-9).

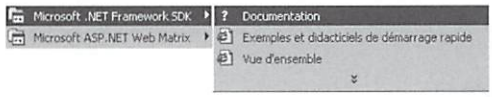


Figure 4–8 Démarrer la documentation installée en local

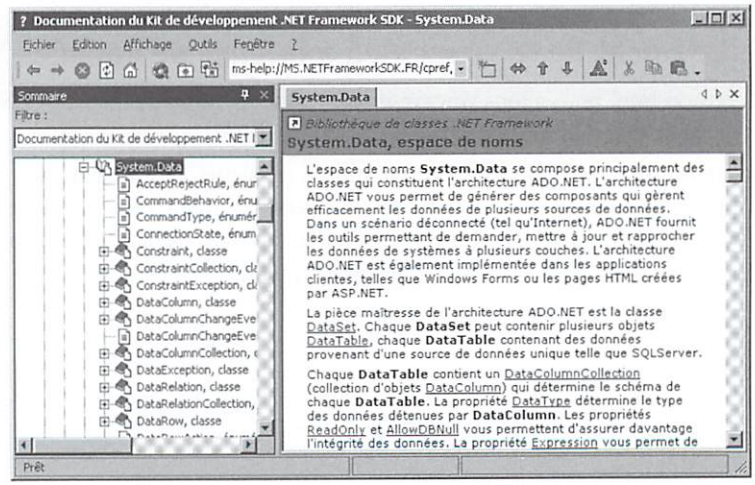


Figure 4–9 La documentation locale du kit de développement .NET

3 – Documentation MSDN en ligne

L'intégralité de la documentation, tenue à jour, est disponible sur le site MSDN (Microsoft Developer Network) à l'adresse :

- http://msdn.microsoft.com

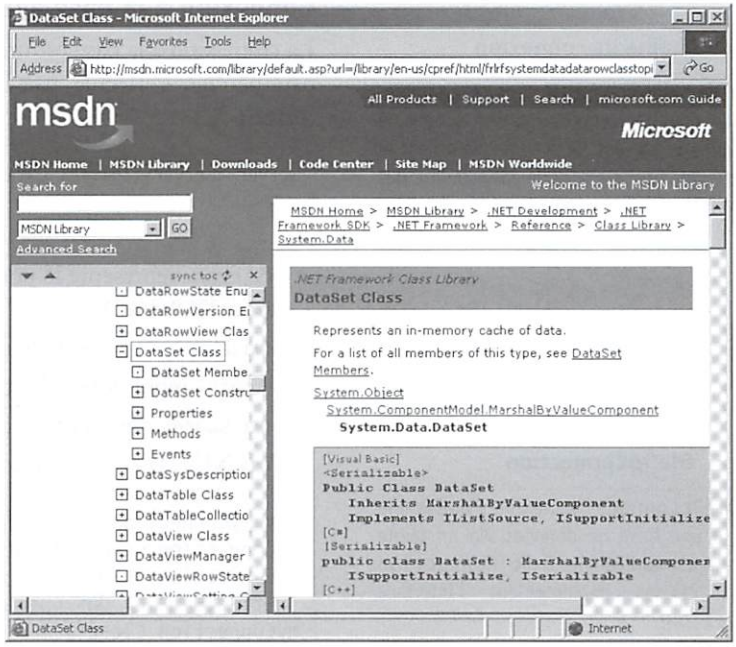


Figure 4–10 Documentation MSDN en ligne

4 –Reflector

Reflector est un utilitaire gratuit qui permet d'examiner, par introspection (*réflection* en anglais, d'où le nom du programme), l'ensemble des assemblages .NET installés sur la machine.

Il dispose d'une interface très ergonomique permettant d'explorer facilement les bibliothèques .NET (avec, en particulier, les liens d'héritage entre classes), d'un moteur de recherche puissant et d'un lien automatique vers les rubriques correspondantes de la documentation MSDN.

Reflector peut être téléchargé gratuitement sur le site de son auteur, Lutz Roeder :

- http://www.aisto.com/roeder/dotnet

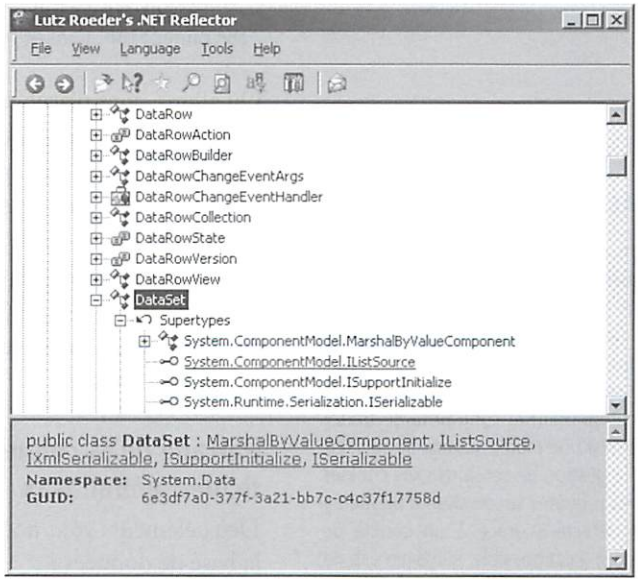


Figure 4–11 L'utilitaire Reflector

Chaîne de connexion

Une chaîne de connexion contient tous les paramètres nécessaires à la connexion à une source de données : nom du serveur, de la base, identifiant et mot de passe de l'utilisateur, et, le cas échéant, des options diverses.

OleDbConnection

La classe `OleDbConnection` permet d'établir une connexion à une base de données via un pilote OLE-DB ; elle est équivalente à `SqlConnection` (ces deux classes implémentent l'interface `IDbConnection`).

Session ASP.NET vs Session ASP ?

L'objet `Session` d'ASP.NET est une version évoluée de l'équivalent ASP, qui introduit notamment deux innovations appréciables : le support des serveurs multiples (les objets `Session` peuvent être stockés dans une base partagée entre plusieurs serveurs Web, ce qui autorise l'équilibrage de charge tout en supprimant la contrainte de traiter toutes les requêtes d'une même session sur un serveur unique) et la gestion de sessions sans cookies (normalement, les requêtes successives d'un même utilisateur sont détectées grâce à un cookie de session : désormais, il est possible, si besoin est, de gérer les sessions sans ce type de cookie grâce à l'ajout automatique du numéro de session dans chaque chaîne de requête).

Connexion à la base avec SqlConnection

La connexion à la base est implémentée par une instance `SqlConnection` dont le paramétrage s'effectue par l'intermédiaire d'une chaîne de connexion de la forme :

```
Server=<Server>;Initial Catalog=<Database>;uid=<Login>;password=<Password>
```

où :

- `<Server>` représente le nom du serveur de données (exemple : `(local)\NetSDK`) ;
- `<Database>` représente le nom de la base de données (exemple : `SDS`) ;
- `<Login>` représente l'identifiant utilisé pour se connecter à la base ;
- `<Password>` représente le mot de passe utilisé pour se connecter à la base.

En VB.NET, la syntaxe de connexion à la base est la suivante :

```
Dim myConnection As New SqlConnection
myConnection.ConnectionString="<VotreChaineDeConnexion>"
myConnection.Open()
' Effectuer ici des opérations sur la base
myConnection.Close()
```

Et voici l'équivalent en C# :

```
SqlConnection myConnection = new SqlConnection();
myConnection.ConnectionString="<VotreChaineDeConnexion>"
myConnection.Open();
// Effectuer ici des opérations sur la base
myConnection.Close();
```

Où allons-nous placer ce code de connexion/déconnexion à la base de données ?

Une première option est de le répéter au sein de chaque page : cette solution n'est pas optimale puisqu'elle alourdit le code de l'application, dégrade la performance (nombreuses opérations de connexion/déconnexion) et rend la maintenance difficile en cas de modification des options de connexion.

Une seconde option, préférable, consiste à se connecter au début de la session de l'utilisateur et à conserver la connexion dans un endroit central, accessible à tous les éléments de l'application : c'est cette technique que nous allons exposer dans la section suivante.

Partage de la connexion à la base avec l'objet Session et le fichier global.asax

Deux éléments vont nous permettre d'implémenter le partage de la connexion à la base de données :

- l'objet `Session`, qui permet de partager des données entre tous les éléments d'une application Web pendant la durée d'une session utilisateur (une instance différente est créée pour chaque session utilisateur) ;

- le fichier `global.asax`, qui permet de stocker des variables globales et d'implémenter des gestionnaires devant être exécutés au début et à la fin d'une session utilisateur, ainsi qu'au démarrage et à l'arrêt d'une application Web.

Voici le processus que nous souhaitons implémenter :

1. Ouverture d'une connexion au démarrage de chaque session utilisateur.
2. Stockage de cette connexion dans le dictionnaire de l'objet `Session` correspondant.
3. Fermeture de la connexion lors de la fin de la session.

Le code correspondant, à implémenter dans le fichier `global.asax` (lequel aura été préalablement créé et placé dans le répertoire racine de l'application), est présenté et commenté ci-après.

global.asax (Version VB.NET)

```
<%@Import Namespace="System.Data.SqlClient"%>
<script language="VB" runat="server">



Sub Session_Start(Sender As Object,E As EventArgs )
   Dim myConnection As SqlConnection
   myConnection = new SqlConnection()
   myConnection.ConnectionString="<VotreChaineDeConnexion>"
   myConnection.Open()
   Session("myConnection") = myConnection
End Sub

Sub Session_End(Sender As Object,E As EventArgs )
   Dim myConnection As SqlConnection
   myConnection = CType(Session("myConnection"),SqlConnection)
   myConnection.Close()
End Sub
</script>
```

L'instruction `<%@ Import...>` permet d'inclure une référence à un espace de nommage, ce qui permet ensuite de référencer les classes de cet espace par leur nom court : par exemple `SqlConnection`, au lieu de `System.Data.SqlClient.SqlConnection`.

Dans ce gestionnaire exécuté au début de chaque session, on crée et on ouvre la connexion à la base, puis on stocke l'objet correspondant dans le dictionnaire de l'objet `Session` (remplacer `<VotreChaineDeConnexion>` par votre véritable chaîne de connexion).

Dans ce gestionnaire exécuté à la fin de chaque session, on ferme la connexion à la base.

Voici le code correspondant en C# :

global.asax (Version C#)

```
<%@Import Namespace = "System.Data.SqlClient"%>
<script language="C#" runat="server">

void Session_Start(Object sender, EventArgs E)
{
   SqlConnection myConnection = new SqlConnection();
   myConnection.ConnectionString="<VotreChaineDeConnexion>"
   myConnection.Open();
   Session["myConnection"]=myConnection;
}
```

global.asax vs global.asa ?

Le fichier `global.asax` est l'équivalent ASP.NET du fichier `global.asa`. À la différence de son prédécesseur, le code qu'il contient peut être rédigé dans n'importe quel langage compatible .NET (VB.NET, C#, etc.). L'extension `.asax` est utilisée afin de ne pas interférer avec les anciens fichiers `.asa` (autrement dit, les fichiers `global.asax` et `global.asa` peuvent cohabiter dans la même application). À ce sujet, il faut veiller à bien placer le fichier `global.asax` à la racine de l'application pour qu'il soit correctement pris en compte.

ALTERNATIVE Utiliser le fichier web.config

Une alternative courante consiste à stocker la chaîne de connexion dans la section `<appSettings>` du fichier `web.config`, ce qui la rend accessible via les membres statiques de la classe `ConfigurationSettings`.

DANS UN CAS RÉEL **Protection du mot de passe**

Dans le cas d'une application réelle, le mot de passe ne serait vraisemblablement pas laissé en clair dans un fichier texte : on utiliserait par exemple le mot de passe fourni par un utilisateur lors de la connexion à l'application.

```
void Session_End(Object sender, EventArgs E)
{
   SqlConnection myConnection = (SqlConnection)Session["myConnection"];
   myConnection.Close();
}
</script>
```

Performance et consommation de ressources

En ce qui concerne le partage de connexion, on peut se demander s'il vaut mieux conserver une connexion ouverte pendant toute la durée de la session (option choisie dans notre cas, plus rapide à l'exécution mais plus consommatrice de ressources serveur) ou partager uniquement la chaîne de connexion et ouvrir/fermer une connexion à chaque exécution de page (option plus lente mais moins consommatrice de ressources) : en pratique, les fournisseurs .NET étant dotés d'un mécanisme de mise en cache de la connexion (*connection pooling*), les deux options sont équivalentes !

Maintenant que nous sommes connectés à la base de données, passons à l'établissement du lien entre cette connexion et les contrôles.

Exécution en mode Debug

Si le caractère compilé des pages ASP.NET élimine un grand nombre d'erreurs dès la phase de développement, il est néanmoins toujours possible qu'il se produise des erreurs à l'exécution : on parle alors d'exceptions.

Par exemple, si la page ASP.NET n'est pas capable de se connecter à la base de données SQL Server (serveur indisponible, nom d'utilisateur incorrect, etc.), une exception de type `SqlException` surviendra (voir figure 4-12).

Pour savoir quelle est la ligne de code qui a provoqué l'exception, il est nécessaire que la page soit exécutée en mode Debug, lequel peut être activé de deux manières différentes :

- par l'ajout d'une directive `Debug="true"` dans la directive `<% Page %>` ;
- par l'ajout d'une section `<compilation debug="true">` dans le fichier `web.config`.

Nous avons déjà eu l'occasion de parler, au chapitre précédent, de la directive `<%@ Page ...%>` qui, placée en haut du fichier `.aspx`, permet notamment de spécifier le langage utilisé dans la page. Cette même directive permet d'activer le mode Debug pour la page :

```
<% @ Page Language="C#" Debug="true" %>
```

Si l'on souhaite activer le mode Debug pour l'ensemble des pages de l'application, il est plus simple de créer un fichier de configuration de l'application nommé `web.config` et placé dans le répertoire racine de l'application : ce fichier permet de spécifier les paramètres d'exécution de l'application (nous aurons l'occasion d'en reparler au chapitre 9).

Fichier web.config

```
<configuration>
    <system.web>
       <compilation debug="true"/>
    </system.web>
</configuration>
```

Notez que le mode Debug est déconseillé pour des applications en production : il ralentit considérablement l'exécution et, de plus, rend le code source en partie visible par les utilisateurs en cas d'erreur.

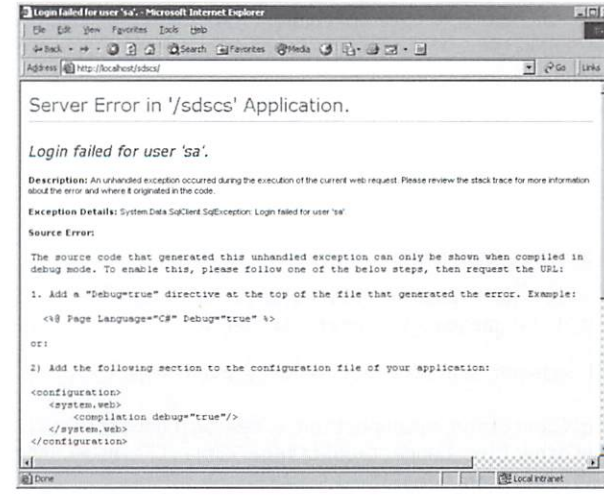


Figure 4–12 Écran indiquant qu'une exception est survenue

Liaison du contrôle DropDownList à la table FamilleProduit

Comme nous l'avons indiqué plus haut, le contrôle serveur `DropDownList` offre la possibilité d'être automatiquement rempli à partir de valeurs contenues dans une base de données.

En pratique, ce lien s'effectue à l'aide de trois propriétés du contrôle :

- la propriété `DataSource`, utilisée pour spécifier la source de données (dans notre cas, la table `FamilleProduit`) ;
- la propriété `DataTextField`, utilisée pour spécifier le nom du champ à afficher dans la liste (dans notre cas, le champ `NomFamille`) ;
- la propriété `DataValueField`, utilisée pour spécifier la valeur à associer aux éléments de la liste (dans notre cas, le champ `ID_FamilleProduit`).

Pour établir le lien entre la table de données et la liste déroulante, deux options sont possibles :

- utiliser des classes `SqlCommand` et `SqlDataReader` ;
- utiliser des classes `SqlDataAdapter`, `DataTable` et `DataView`.

L'option avec utilisation de `SqlDataReader` est, a priori, la plus simple et la plus performante ; quant à la seconde option, elle est fréquemment présentée dans la documentation .NET et permet de présenter des classes importantes comme `SqlDataAdapter` et `DataTable`.

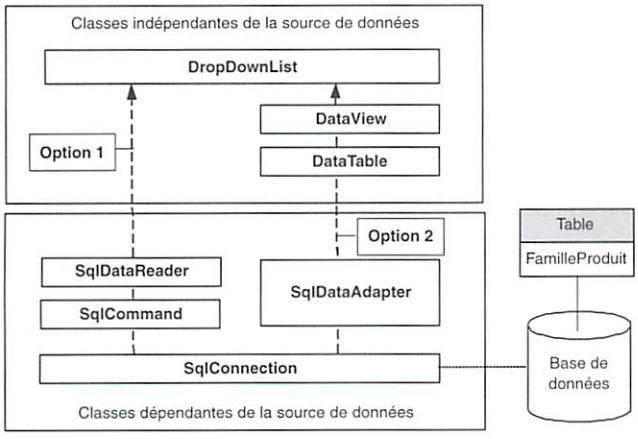


Figure 4–13 Options possibles pour le lien avec la base de données

Nous allons donc implémenter deux versions d'une fonction permettant d'effectuer le chargement de la liste des familles, que nous nommerons `ChargerListeFamilles`. Puis, nous intégrerons cette fonction dans notre page `stocks.aspx`.

Jeu de résultat

Désigne un ensemble de données organisées en lignes et en colonnes (contenu d'une table, d'une vue, résultat de l'exécution d'une procédure stockée...).

À propos de SqlDataReader

Un objet de type `SqlDataReader` permet de balayer un jeu de résultat, en lecture seule, de la première vers la dernière ligne : l'avantage principal est la performance (seule la ligne active du jeu de résultat est conservée en mémoire). En revanche, les fonctionnalités de manipulations de données sont beaucoup plus réduites qu'avec un objet de type `DataTable` (décrit plus loin).

Alternative 1 - Utilisation de SqlCommand et SqlDataReader

Cette première option utilise trois classes extraites de l'espace de nommage `System.Data.SqlClient` :

- La classe `SqlConnection`, déjà décrite précédemment, implémente la connexion à la base de données.
- La classe `SqlCommand` permet de spécifier à quel objet de la base de données on s'intéresse, en l'occurrence la table `FamilleProduit`.
- La classe `SqlDataReader` permet de manipuler les données, en l'occurrence les parcourir en lecture seule.

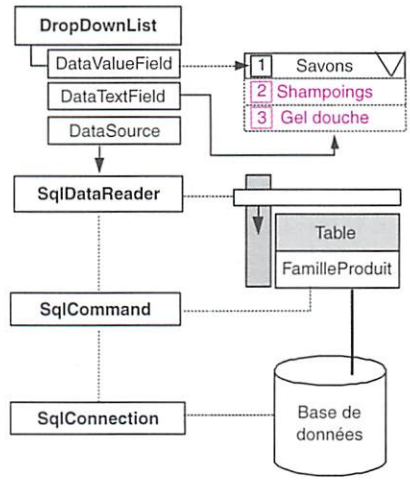


Figure 4–14 Accès à la base en utilisant SqlDataReader

Voici la version correspondante de la fonction `ChargerListeFamilles` :

Stocks.aspx - Fonction ChargerListeFamilles (version VB.NET/SqlDataReader)

```
Sub ChargerListeFamilles()
    Dim myConnection As SqlConnection
    Dim myCommand As SqlCommand
    Dim myReader  As SqlDataReader
    Dim SQL As String
    myConnection = CType(Session("myConnection"),SqlConnection)

    SQL = "SELECT * FROM FamilleProduit"
    myCommand = new SqlCommand(SQL,myConnection)

    myReader = myCommand.ExecuteReader()
```

Déclaration des variables

Récupération de la connexion (stockée dans l'objet `Session`, grâce au fichier `global.asax`)

Allocation de l'objet `SqlCommand` (« connecte-toi à telle base et intéresse-toi à telles données »)

Allocation de l'objet `SqlDataReader` par l'intermédiaire de l'objet `SqlCommand` (« fournis-moi un objet capable de parcourir les données »)

```
    ListeFamilles.DataSource      = myReader
    ListeFamilles.DataValueField = "ID_FamilleProduit"
    ListeFamilles.DataTextField  = "NomFamille"
    ListeFamilles.DataBind()

    myReader.Close()

End Sub
```

Paramétrage de l'objet `ListeFamilles`

Chargement effectif de la liste (sans cette ligne, la liste resterait vide !)

Fermeture de l'objet `SqlDataReader` (indispensable, car `myReader` a été ouvert par `ExecuteReader...`)

Et voici la version équivalente en C# :

Stocks.aspx – Fonction ChargerListeFamilles (version C#/SqlDataReader)

```
void ChargerFamillesProduits()
{
    SqlCommand    myCommand;
    SqlDataReader myReader;
    SqlConnection myConnection = (SqlConnection)Session["myConnection"];

    string SQL = "SELECT * FROM FamilleProduit";

    myCommand = new SqlCommand(SQL,myConnection);
    myReader = myCommand.ExecuteReader();

    ListeFamilles.DataSource = myReader;
    ListeFamilles.DataValueField = "ID_FamilleProduit";
    ListeFamilles.DataTextField = "NomFamille";
    ListeFamilles.DataBind();

    myReader.Close();
}
```

Avant de tester le résultat du chargement de la liste des familles de produits, nous présentons la deuxième option possible pour accéder aux données.

Alternative 2 - Utilisation de SqlDataAdapter, DataTable et DataView

Cette deuxième option utilise quatre classes :

- La classe `SqlConnection`, déjà décrite précédemment, implémente la connexion à la base de données.
- La classe `SqlDataAdapter` permet de gérer la communication avec un objet de la base de données de manière bidirectionnnelle (lecture, mise à jour, suppression de données).
- La classe `DataTable` représente une table en mémoire dont il est possible de manipuler les données, en lecture et en écriture.
- La classe `DataView` représente une vue d'une table, autrement dit, un sous-ensemble énumérable des lignes de la table.

Les deux premières classes font partie de l'espace de nommage `System.Data.SqlClient` (elles sont donc dépendantes de la base de données) tandis que les deux dernières sont extraites de `System.Data` (elles sont donc indépendantes de la base de données).

ASP.NET Vers quoi peut pointer la propriété DataSource ?

La propriété `DataSource` d'un contrôle lié aux données ne doit pas nécessairement pointer vers une base de données relationnelle (via `DataView` ou `DataReader`) : il est également possible de lier un contrôle serveur à un tableau de données en mémoire (`ArrayList`, `HashTable`) ou au contenu d'un fichier XML (via `XmlNodeList`). La seule contrainte est que l'objet pointé permette d'énumérer des données – en termes techniques, qu'il implémente une des interfaces suivantes : `ICollection`, `IEnumerable` ou `IListSource`. Dans le cas d'un contrôle lié à une base de données, le choix le plus courant est l'utilisation d'un objet de type `DataView`, qui représente un jeu de données issues d'une table.

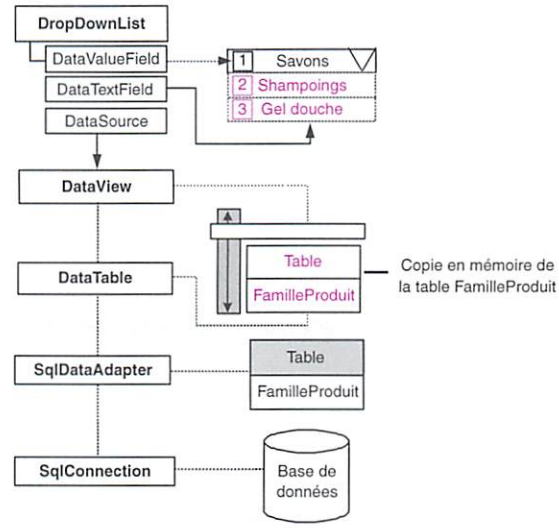


Figure 4–15 Accès à la base en utilisant SqlDataAdapter

Comme on le voit sur la figure 4-15, cette solution constitue en quelque sorte « la grosse artillerie » pour remplir une liste déroulante :

- Une copie de la table `FamilleProduit` est conservée en mémoire (gérée par `DataTable`), ce qui consomme inutilement des ressources.
- La classe `SqlDataAdapter` est utilisée pour remplir cette table en mémoire ; néanmoins, dans notre cas, elle est sous-utilisée car elle pourrait également servir à synchroniser les changements effectués sur la version « en mémoire » vers la base de données.

L'intérêt de présenter cette solution est donc plus didactique que technique !

Voici la version correspondante de la fonction `ChargerListeFamilles` :

Stocks.aspx – Fonction ChargerListeFamilles (version VB.NET/SqlDataAdapter)

```
Sub ChargerListeFfamilles()
   Dim myConnection As SqlConnection
   Dim myAdapter    As SqlDataAdapter
   Dim myDataTable  As DataTable
   Dim SQL As String
   myConnection = CType(Session("myConnection"),SqlConnection)

   SQL = "SELECT * FROM FamilleProduit"
   myAdapter  = new SqlDataAdapter(SQL,myConnection)

   myDataTable = new DataTable()

   myAdapter.Fill(myDataTable)

   ListeFamilles.DataSource = myDataTable.DefaultView
   ListeFamilles.DataValueField = "ID_FamilleProduit"
   ListeFamilles.DataTextField = "NomFamille"
   ListeFamilles.DataBind()
End Sub
```

Déclaration des variables utilisées

Récupération de la connexion (stockée dans l'objet `Session`)

Allocation de l'objet `SqlDataAdapter` (lié à la table `FamilleProduit`)

Allocation de l'objet `DataTable`

Remplissage de l'objet `DataTable`

Initialisation du contrôle `ListeFamilles` (l'appel à `DataBind` réalise le chargement de la liste)

Voici le code correspondant en C# :

Stocks.aspx – Fonction ChargerListeFamilles (version C#/SqlDataAdapter)

```
void ChargerListeFamilles()
{
   SqlConnection myConnection = (SqlConnection)Session["myConnection"];
   SqlDataAdapter   myAdapter;
   DataTable        myDataTable;

   string strSQL = "SELECT * FROM FamilleProduit";

   myAdapter  = new SqlDataAdapter(strSQL,myConnection);
   myDataTable = new DataTable();
   myAdapter.Fill(myDataTable);

   ListeFamilles.DataSource = myDataTable.DefaultView;
   ListeFamilles.DataValueField = "ID_FamilleProduit";
   ListeFamilles.DataTextField = "NomFamille";
   ListeFamilles.DataBind();
}
```

Tester le remplissage de la liste des familles de produits

Avant de tester notre page, il faut réaliser deux opérations supplémentaires :

- ajout des directives `<%@ Import %>` ;
- implémentation de l'appel de la fonction `ChargerListeFamilles` depuis `Page_Load`.

Dans notre cas, voici les directives `<%@ Import %>` à spécifier :

Stocks.aspx (début du fichier)

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
```

Enfin, il ne reste plus qu'à rajouter un appel à la fonction `ChargerListeFamilles` dans le gestionnaire `Page_Load` de notre page :

Stocks.aspx – Fonction Page_Load (version VB.NET)

```
Sub Page_Load(sender As Object, e As EventArgs)
   ChargerListeFamilles()
End Sub
```

Stocks.aspx – Fonction Page_Load (version C#)

```
void Page_Load(Object sender, EventArgs e)
{
   ChargerListeFamilles();
}
```

RAPPEL Les directives <%@ Import %>

Les directives `<%@ Import ...%>` permettent de spécifier les espaces de nommage pour lesquels on souhaite pouvoir faire référence aux classes contenues par leur nom court.

Si vous avez choisi l'alternative 1 pour le chargement de la liste des familles (`SqlDataReader`), seule la référence à `System.Data.SqlClient` est nécessaire.

ATTENTION Position des directives Import

Les directives `<%@ Import ...%>` doivent être placées hors du bloc `<script runat="server">`; en général, elles sont placées en haut du fichier. Avec Web Matrix, positionnez-vous dans l'onglet All pour saisir ces directives (si vous êtes dans l'onglet Code, les directives seront insérées dans le bloc `<script>` !).

À propos de Page_Load

Page_Load est une fonction appelée au chargement de la page. Nous aurons l'occasion de présenter plus en détail le mécanisme des événements au sein d'une page ASP.NET à la fin de ce chapitre.

Espaces de nommage importés implicitement

Les espaces de nommage suivants sont implicitement importés dans les pages ASP.NET :

- System
- System.Collections
- System.Collections.Specialized
- System.Web
- System.Web.UI
- System.Web.UI.WebControls
- System.Web.UI.HtmlControls
- System.Web.Caching
- System.Web.Security
- System.Web.SessionState
- System.Text
- System.Text.Configuration
- System.Text.RegularExpressions

Voilà ! Si tout se passe bien, vous devez maintenant voir apparaître la liste des familles de produits remplie lorsque vous exécutez la page (voir figure 4-16).

Figure 4–16 Remplissage de la liste des familles de produits

Nous avons réalisé la première moitié de la page. Passons maintenant à la seconde : l'implémentation du contrôle `DataGrid` destiné à afficher l'état des stocks.

Utilisation de DataGrid pour afficher l'état des stocks

La liaison entre le contrôle `EtatStock` de type `DataGrid` et la procédure stockée `EtatStock` va s'effectuer de manière similaire à ce que nous avons réalisé dans la section précédente.

Rappelons que deux techniques sont à notre disposition :

- utiliser `SqlCommand` et `SqlDataReader` ;
- utiliser `SqlDataAdapter`, `DataTable` et `DataView`.

Nous allons ici opter pour l'utilisation de `SqlDataReader`, la technique la plus performante et la mieux adaptée à nos besoins. Encore une fois, nous allons implémenter le paramétrage dans une fonction distincte, que nous nommerons `AfficherStockProduits`, profitant ainsi des possibilités de structuration du code offertes par ASP.NET.

La seule particularité supplémentaire est ici l'emploi d'une procédure stockée (ce que nous spécifions grâce à la propriété `CommandType` de `SqlCommand`) qui accepte en entrée un paramètre de type entier égal au numéro de la famille sélectionnée dans `ListeFamilles`.

Stocks.aspx – Fonction AfficherStocksProduits (version VB.NET)

Déclaration des variables

```
Sub AfficherStocksProduits()
   Dim myCommand As SqlCommand
   Dim myReader  As SqlDataReader
   Dim myConnection As SqlConnection
   Dim FamilleID As String
```

```
    myConnection = CType(Session("myConnection"),SqlConnection)
    FamilleID    = ListeFamilles.SelectedItem.Value

    myCommand = new SqlCommand("EtatStock",myConnection)
    myCommand.CommandType = CommandType.StoredProcedure
    myCommand.Parameters.Add("@FamilleID",SqlDbType.Int).Value = FamilleID
    myReader = myCommand.ExecuteReader()

    EtatStock.DataSource = myReader
    EtatStock.DataBind()

    myReader.Close()

End Sub
```

- Récupération de la connexion ainsi que du numéro de la famille sélectionnée
- Construction de l'objet `SqlCommand` associé à la procédure `EtatStock` et spécification du paramètre d'entrée, égal au numéro de la famille sélectionnée
- Paramétrage de la grille de données
- Fermeture de l'objet `SqlDataReader` (à ne pas oublier !)

Stocks.aspx – Fonction AfficherStocksProduits (version C#)

```
void AfficherStocksProduits()
{
    SqlCommand      myCommand;
    SqlDataReader   myReader;
    SqlConnection   myConnection = (SqlConnection)Session["myConnection"];
    string FamilleID = ListeFamilles.SelectedItem.Value;

    myCommand = new SqlCommand("EtatStock",myConnection);
    myCommand.CommandType = CommandType.StoredProcedure;
    myCommand.Parameters.Add("@FamilleID",SqlDbType.Int).Value = FamilleID;
    myReader = myCommand.ExecuteReader();

    EtatStock.DataSource = myReader;
    EtatStock.DataBind();

    myReader.Close();
}
```

ALTERNATIVE Syntaxe in-line pour la procédure stockée

Au lieu de recourir aux paramètres explicites, il est possible d'utiliser la syntaxe dite en ligne (*in-line*) pour l'appel de la procédure stockée :

```
string SQL = "EtatStock '"+FamilleID+"'"
myCommand = new SqlCommand(SQL,
            ➥ myConnection)
```

Avant de tester notre page, il ne reste plus qu'à ajouter un appel à la fonction `AfficherStocksProduits` dans le gestionnaire `Page_Load` :

Stocks.aspx – Fonction Page_Load (version VB.NET)

```
Sub Page_Load(sender As Object, e As EventArgs)
    ChargerListeFamilles()
    AfficherStocksProduits()
End Sub
```

Stocks.aspx – Fonction Page_Load (version C#)

```
void Page_Load(Object sender, EventArgs e)
{
    ChargerListeFamilles();
    AfficherStocksProduits();
}
```

Figure 4–17 La page de suivi des stocks avec le contrôle `DataGrid`

Le résultat de l'exécution de la page est présenté sur la figure 4-17 : on peut apprécier le peu de lignes de code qui ont été nécessaires pour générer un tableau HTML complet !

Néanmoins, l'aspect du tableau laisse encore un peu à désirer ; de plus, il ne semble pas indispensable d'afficher les colonnes ID_Produit et ID_FamilleProduit…

Nous allons voir dans la section suivante comment personnaliser le contrôle `DataGrid` pour qu'il soit mieux adapté à nos besoins.

Personnalisation du contrôle DataGrid

Le contrôle `DataGrid` dispose de nombreuses possibilités de paramétrage, notamment :

- le contrôle des colonnes à afficher et celui du format de ces colonnes (alignement, police) ;
- le contrôle des couleurs (en-tête, pied de tableau, lignes) ;
- la gestion de la pagination (navigation entre les pages).

Tous ces paramétrages peuvent être effectués via une interface graphique :

1. Sélectionnez le contrôle `EtatStock` dans l'onglet Design de Web Matrix.
2. Dans la feuille de propriétés associée, cliquez sur le lien PropertyBuilder (figure 4-18).
3. Une boîte de dialogue permettant le paramétrage de `EtatStock` apparaît (figure 4-19).

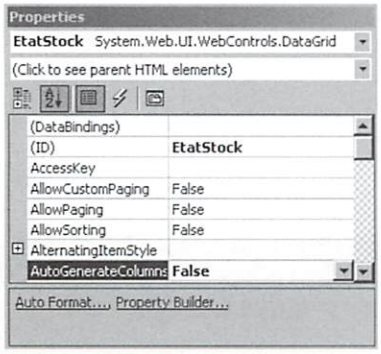


Figure 4–18 Feuille de propriétés EtatStock

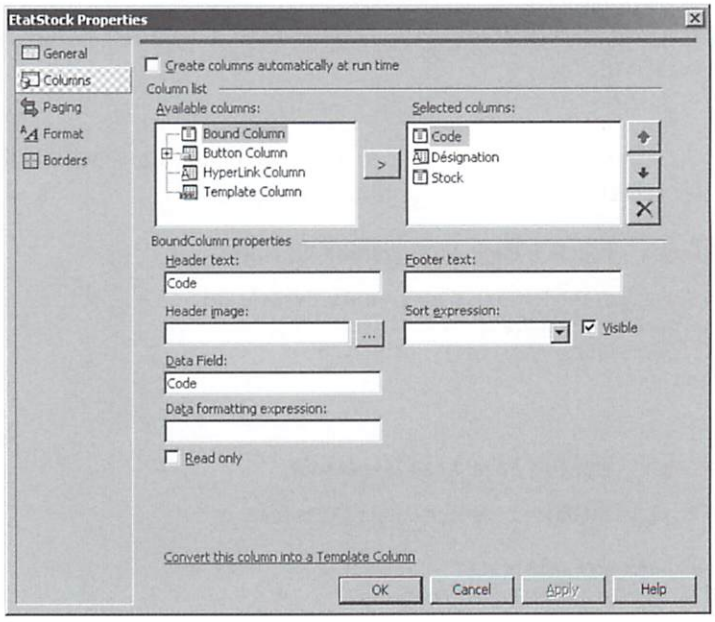


Figure 4–19 Paramétrage du contrôle EtatStock

Par défaut, les colonnes du `DataGrid` sont générées dynamiquement à partir des colonnes présentes dans la source de données. Dans notre cas, nous souhaitons n'afficher que les colonnes Code, Désignation et Stocks, à l'exclusion des colonnes clés (ID_Produit et ID_FamilleProduit). En outre, les éléments de la colonne Désignation doivent agir comme des liens hypertextes qui permettent d'accéder à l'historique du stock.

Pour cela, nous allons désactiver la fonctionnalité de génération automatique des colonnes (propriété `AutoGenerateColumns`) et spécifier une liste personnalisée de colonnes :

- colonne Code de type `BoundColumn` ;
- colonne Désignation de type `HyperLinkColumn` ;
- colonne Stocks de type `BoundColumn`.

Voici la marche à suivre pour spécifier notre propre liste de colonnes :

1. Faites apparaître la boîte de dialogue de paramétrage de `EtatStock` (voir précédemment).
2. Cliquez sur Columns afin de faire apparaître la feuille de propriétés relative aux colonnes.
3. Désactivez l'option Create columns automatically at run time.
4. Créez une nouvelle colonne de type `BoundColumn` ayant les caractéristiques suivantes :

Header Text	Code	Le texte de l'en-tête de la colonne
Data Field	Code	Le champ correspondant de la base

5. Créez une nouvelle colonne de type `HyperLinkColumn` ayant les caractéristiques suivantes :

Header Text	Désignation	Le texte de l'en-tête de la colonne
Text Field	Code	Le champ correspondant de la base
Url Field	ID_Produit	Le champ à rendre disponible dans l'URL
Url Format String	DetailsStocks.aspx?id={0}	L'URL associée ({0} désignant Url Field)

6. Créez une nouvelle colonne de type `BoundColumn` ayant les caractéristiques suivantes :

Header Text	Stocks	Le texte de l'en-tête de la colonne
Data Field	Stocks	Le champ correspondant de la base

7. Cliquez sur OK pour enregistrer les modifications.

Les types de colonnes gérés par DataGrid

Une grille de données (contrôle `DataGrid`) peut contenir des colonnes de type `BoundColumn` (colonne simple liée à un champ de la base), `HyperLinkColumn` (colonne contenant des liens hypertextes), `ButtonColumn` (colonne contenant des boutons d'actions), `TemplateColumn` (colonne dont le format est fixé par l'utilisateur) et `EditCommandColumn` (colonne permettant d'éditer les valeurs de la ligne).

Enfin, nous allons personnaliser l'aspect graphique de notre grille de données (couleur de l'en-tête, espacement entre les cellules, largeur des colonnes) :

1. Faites apparaître la boîte de dialogue de paramétrage de `EtatStock` (voir plus haut).
2. Cliquez sur `Format` afin de faire apparaître la feuille de propriétés relative au formatage.
3. Sélectionnez l'élément `Header` dans la liste `Objects` et spécifiez :

Back Color	#AAAADD

4. Sélectionnez l'élément `Columns` dans la liste `Objets` et spécifiez les largeurs suivantes :

Colonne 0 (Code)	50
Colonne 1 (Désignation)	100
Colonne 2 (Stock)	50

5. Pour la colonne 2 (Stocks), sélectionnez l'élément fils `Items` et spécifiez :

Horizontal alignement	Right

6. Cliquez sur `Borders` afin de faire apparaître la feuille de propriétés relative aux bordures.
7. Spécifiez les caractéristiques suivantes pour les bordures :

Cell padding	3
Grid lines	Both
Border color	Black
Border width	1 px

8. Cliquez sur OK pour enregistrer les modifications.

Jetez un coup d'œil à l'onglet HTML : vous constaterez que la partie relative au contrôle `EtatStock` a été modifiée en conséquence.

Figure 4–20 Page de suivi des stocks après paramétrage

stocks.aspx (contenu HTML) – Après paramétrage du DataGrid

```
<asp:DataGrid id="EtatStock" runat="server" BorderColor="black"
   CellPadding="3" BorderWidth="1" AutoGenerateColumns="false"
   HeaderStyle-BackColor="#aaaadd" CellSpacing="0" GridLines="Both">

  <Columns>
    <asp:BoundColumn HeaderText="Code" DataField="Code">
        <HeaderStyle width="50px"></HeaderStyle>
    </asp:BoundColumn>
    <asp:HyperLinkColumn
          HeaderText="Désignation"
          DataNavigateUrlField="ID_Produit"
          DataNavigateUrlFormatString="DetailStocks.aspx?id={0}"
          DataTextField="Designation">
```

```
        <HeaderStyle width="150px"></HeaderStyle>
    </asp:HyperLinkColumn>
    <asp:BoundColumn HeaderText="Stock"
                     ItemStyle-HorizontalAlign="right"
                     DataField="Stock">
        <HeaderStyle width="100px"></HeaderStyle>
    </asp:BoundColumn>
  </Columns>
</asp:DataGrid>
```

Exécutez la page pour admirer le résultat (voir figure 4-20) !

Il est maintenant temps de réaliser la page `detailstocks.aspx` vers laquelle pointent les liens hypertextes de la colonne Désignation.

Implémentation de la page de consultation de l'historique

La page `detailstocks.aspx` affiche l'historique des variations de stocks (approvisionnements, ventes) d'un produit donné.

Nous avons déjà réalisé la maquette de cette page au début du chapitre ; l'implémentation doit réaliser les étapes suivantes :

1. Récupération du numéro de produit sur la ligne de requête.
2. Affichage du nom du produit concerné dans le contrôle `Label`.
3. Paramétrage du contrôle `DataGrid`, lié à la procédure `HistoriqueStock`.

La récupération du paramètre sur la ligne de commande s'effectuera par l'intermédiaire de l'objet global `Request`, qui contient toutes les informations relatives à la requête HTTP correspondant à la page : paramètres de la chaîne de requête, informations sur l'utilisateur (adresse IP, type de navigateur, cookies...) et valeurs des champs de formulaire.

Pour la récupération des informations dans la base, nous allons utiliser la procédure `HistoriqueStock`, qui permet à la fois de récupérer le nom du produit et l'historique du stock pour ce produit, à condition de passer le numéro du produit en paramètre.

Voici le code correspondant :

DetailStock.aspx (Version VB.NET)

```
Sub Page_Load(sender As Object,e As EventArgs)
    Dim myConnection As SqlConnection
    Dim myReader  As SqlDataReader
    Dim myCommand As SqlCommand
    myConnection = CType(Session("myConnection"),SqlConnection)

    myCommand = new SqlCommand("HistoriqueStock",myConnection)
    myCommand.CommandType = CommandType.StoredProcedure
    myCommand.Parameters.Add("@ProduitID",SqlDbType.Int).
    myCommand.Parameters("@ProduitID").Value = Request.Params("id")
```

Déclaration des variables

Récupération de la connexion (stockée dans l'objet `Session`)

Définition de la commande associée à la procédure stockée (le numéro de produit, récupéré sur la ligne de commande, est passé en paramètre de la procédure)

Objet Request : version ASP.NET vs ASP

L'objet `Request` d'ASP.NET est une version améliorée de son correspondant ASP : notons, entre autres, que tous les paramètres sont accessibles via une collection nommée `Params`, qui regroupe les informations de la chaîne de requête, les variables serveur, les cookies, etc.

Objets globaux en ASP.NET

Plusieurs objets globaux sont accessibles depuis une page ASP.NET : aux objets `Request`, `Response`, `Application`, `Server` et `Session`, qui correspondent à leurs prédécesseurs ASP, s'ajoutent deux nouveaux objets : `Page` et `Cache`.

Récupération du nom du produit (un premier SqlDataReader est mis en œuvre ; la méthode Read permet de se positionner sur le premier enregistrement ; 4 est l'indice de la colonne contenant le nom du produit)

```
    myReader = myCommand.ExecuteReader()
    myReader.Read()
    NomProduit.Text = myReader.GetString(4)
    myReader.Close()
```

Paramétrage de la grille de données (on réutilise la variable myReader pour la faire pointer vers un nouveau SqlDataReader ; l'ancien objet sera « désalloué » automatiquement par le garbage collector du CLR)

```
    myReader = myCommand.ExecuteReader()
    HistoriqueStocks.DataSource = myReader
    HistoriqueStocks.DataBind()
    myReader.Close()
End Sub
```

DetailStock.aspx (Version C#)

```
void Page_Load(Object sender, EventArgs e)
{
    SqlDataReader myReader;
    SqlCommand    myCommand;
    SqlConnection myConnection;

    myConnection = (SqlConnection)Session["myConnection"];

    myCommand = new SqlCommand("HistoriqueStock",myConnection);
    myCommand.CommandType = CommandType.StoredProcedure;
    myCommand.Parameters.Add("@ProduitID",SqlDbType.Int);
    myCommand.Parameters["@ProduitID"].Value = Request.Params["id"];

    myReader = myCommand.ExecuteReader();
    myReader.Read();
    NomProduit.Text = myReader.GetString(4);
    myReader.Close();

    myReader = myCommand.ExecuteReader();
    HistoriqueStocks.DataSource = myReader;
    HistoriqueStocks.DataBind();
    myReader.Close();
}
```

Pour finir, paramétrons l'aspect graphique du contrôle DataGrid, comme nous l'avons fait déjà fait précédemment :

1. Faites apparaître la boîte de dialogue de paramétrage associée à HistoriqueStocks.
2. Cliquez sur Columns afin de faire apparaître la feuille de propriétés relative aux colonnes.
3. Désactivez l'option Create columns automatically at run time.
4. Créez trois colonnes de type Bound Column, ayant les caractéristiques suivantes :

Header Text	Data Field
Mois	DateMouvement
Ventes	Ventes
Approvisionnements	Approvisionnements

5 Spécifiez les mêmes paramétrages de couleurs et de bordure que pour le contrôle `EtatStock` de la page `stocks.aspx`.

6 Cliquez sur OK pour enregistrer les changements.

Le résultat de l'exécution de la page est représenté à la figure 4-21.

Notre module de suivi de stocks est maintenant pratiquement terminé… à un détail près : le rechargement de la liste des produits en cas de changement de famille. Nous allons traiter ce point dans la section suivante.

Figure 4–21 La page « historique du stock »

Rendre la page interactive grâce à la gestion des événements

Une des grandes nouveautés apportées par ASP.NET est la gestion des événements : ce mécanisme, qui permet d'associer facilement un gestionnaire à un événement donné, améliore l'organisation du code et rend très simple la création de pages Web interactives.

Il existe deux types d'événements :

- les événements liés à la page ;
- les événements déclenchés par les contrôles.

Après une description du mécanisme des événements liés à la page (dont nous avons déjà vu des exemples : en particulier, le fameux `Page_Load`), nous allons voir comment il est possible d'implémenter le rechargement de la page `stocks.aspx` lorsque la sélection de la famille de produits est modifiée.

Mieux structurer le code au sein d'une page grâce aux événements prédéfinis

Au sein d'une page ASP.NET, le code est organisé en gestionnaires d'événements, autrement dit en fonctions qui s'exécutent lorsqu'un événement donné survient.

Lors du chargement d'une page, un certain nombre d'événements se produisent :

Événement	Description
`Page_Init`	Déclenché après l'initialisation de la page (à ce stade, les contrôles ne sont pas encore chargés)
`Page_Load`	Déclenché après le chargement de la page (à ce stade, les valeurs des contrôles ont été mises à jour à partir des valeurs contenues dans `__VIEWSTATE`)
`Page_Unload`	Déclenché lors de la désallocation de la page

__VIEWSTATE

__VIEWSTATE est un contrôle HTML caché géré par ASP.NET, qui permet aux contrôles contenus dans la page de « garder leurs valeurs » lors d'un aller-retour de la page : nous décrivons son fonctionnement dans les pages qui suivent.

Pour associer un gestionnaire à un événement, le développeur doit placer dans le code une fonction de la forme suivante :

```
void TheObject_TheEvent (Object sender, EventArgs e)
```

où :

- `TheObject` désigne le nom de l'objet ayant déclenché l'événement ;
- `TheEvent` désigne le nom de l'événement ;
- `sender` est une variable de type `Object` qui pointe vers l'objet ayant déclenché l'événement ;
- `e` est une variable de type `EventArgs` qui contient des informations sur l'événement.

Il n'est pas nécessaire d'associer un gestionnaire à tous les événements de la page, mais uniquement à ceux qui sont utiles au développement (dans notre cas, `Page_Load` a pour l'instant suffi). Inversement, il est indispensable que tout code soit placé au sein d'un gestionnaire d'événement (ou au sein d'une fonction appelée directement ou indirectement depuis un gestionnaire).

Mais il y a mieux : des événements déclenchés par l'utilisateur (clic sur un bouton, changement de sélection dans une liste) peuvent provoquer un nouveau chargement de la page, avec au passage l'appel du gestionnaire d'événement associé. C'est le mécanisme des événements côté serveur que nous allons détailler maintenant.

Comparaison avec ASP

Dans le cas d'une page ASP, l'intégralité du code était exécutée lors du chargement de la page et il était nécessaire d'employer des `<%If...Then%>` pour exécuter spécifiquement un bloc donné en fonction des circonstances. De plus, un événement ne pouvait pas provoquer le rechargement de la page, comme c'est désormais le cas avec ASP.NET

Gérer le changement de famille grâce aux événements déclenchés par DropDownList

Nous souhaitons ajouter de l'interactivité à la page qui affiche l'état des stocks classés par famille de produits : si l'utilisateur modifie la sélection de la famille, nous souhaitons que la liste des produits soit rechargée en conséquence.

Pour cela, pas de miracle : il faut détecter le changement de sélection dans la liste et, lorsqu'il survient, faire une nouvelle requête vers le serveur en demandant le rechargement de la page, actualisée en fonction de la nouvelle famille de produits sélectionnée.

Pour accomplir ceci avec une technologie classique comme ASP, il aurait fallu implémenter, côté client, un gestionnaire en langage de script (JavaScript) associé à l'événement `onChange` de la liste des familles, qui provoque le rechargement de la page.

Dans cet exemple, on fait l'hypothèse que la page associée à l'attribut `action` du formulaire est la page qui contient le formulaire (le formulaire « s'auto-poste »).

```
<select onChange="document.forms[0].submit()"></select>
```

Puis, il aurait fallu implémenter le remplissage de la liste des familles, la sélection de la nouvelle famille (obtenue à partir des données postées) et le chargement de l'état des stocks correspondants.

Désormais, tout ce travail est effectué automatiquement par ASP.NET grâce aux gestionnaires d'événements serveur.

Pour associer un gestionnaire au changement de sélection dans la liste des familles, il suffit de modifier comme suit le contenu HTML de la page `stocks.aspx` :

Stocks.aspx (contenu HTML)

```
<asp:DropDownList id="ListeFamilles" runat="server"
   AutoPostBack="true"
   OnSelectedIndexChanged="OnFamilleChanged"/>
```

Détaillons les modifications effectuées :

- L'instruction `OnSelectedIndexChanged="OnFamilleChanged"` associe le gestionnaire `OnFamilleChanged` (nom choisi par le développeur) à l'événement « changement de sélection ».
- L'instruction `AutoPostBack="True"` indique au moteur ASP.NET qu'il doit générer automatiquement le script client qui provoquera le rechargement de la page lors d'un changement de famille (notons que ce script sera adapté aux possibilités du navigateur en fonction de sa version).

Dans la page `stocks.aspx`, implémentez le gestionnaire d'événement `OnFamille Changed`, qui effectue la mise à jour du tableau `EtatStock` (on présente seulement la version VB.NET) .

Stocks.aspx – Fonction OnFamilleChanged (version VB.NET)

```
Sub OnFamilleChanged(sender As Object, e As EventArgs)
   AfficherStocksProduits()
End Sub
```

Enfin, modifiez la fonction `Page_Load`, de manière à ne pas effectuer d'opération inutile lors du rechargement de la page (la propriété `IsPostBack` indique que la page a été postée vers le serveur, suite à un événement) :

Stocks.aspx – Fonction Page_Load (version VB.NET)

```
Sub Page_Load(sender As Object, e As EventArgs)
   If Not IsPostBack Then
        ChargerListeFamilles()
        AfficherStocksProduits()
   End If
End Sub
```

Stocks.aspx – Fonction Page_Load (version C#)

```
void Page_Load(Object sender, EventArgs e)
{
  if(IsPostBack==false)
  {
      ChargerListeFamilles();
      AfficherStocksProduits();
   }
}
```

Testez la page… un changement de sélection dans la liste des familles de produits doit désormais provoquer la mise à jour du tableau d'état des stocks !

Comprendre le mécanisme de gestion événementielle d'ASP.NET

Pour mieux comprendre le mécanisme de gestion événementielle, détaillons ce qui se passe lors de cet aller-retour (*round trip*) vers le serveur, illustré à la figure 4-22.

Si vous regardez le fichier source de la page affichée dans votre navigateur, vous y trouverez un script provoquant l'envoi du formulaire au serveur lors de la sélection de la famille de produits :

```
<select id="ListeFamilles"
 onchange="__doPostBack('ListeFamilles','')" >
```

La fonction __doPostBack réalise l'envoi du formulaire (qui conduit au rechargement de la même page ; voir l'attribut action de la balise <form>)

```
function __doPostBack(eventTarget, eventArgument) {
   var theform = document._ctl0;
   theform.__EVENTTARGET.value = eventTarget;
   theform.__EVENTARGUMENT.value = eventArgument;
   theform.submit();
}
```

Grâce à deux contrôles (__EVENTTARGET et __EVENTARGUMENT), la source de l'événement et les éventuels arguments sont transmis au serveur, qui peut ainsi noter qu'il s'agit d'une page postée (mise à jour d'IsPostBack), déterminer le gestionnaire d'événement à appeler et lui passer les arguments nécessaires.

Sans AutoPostBack

Si on ne spécifie pas AutoPostBack="True", aucun script client ne sera généré : un changement de sélection ne provoquera donc pas le rechargement « automatique » de la page ; néanmoins, il provoquera l'appel du gestionnaire OnFamilleChanged lors du prochain rechargement de la page, grâce à la comparaison entre les valeurs des champs de formulaire et celles stockées dans ViewState.

La page contient un troisième contrôle caché nommé __VIEWSTATE qui contient une copie (codée sous une forme synthétique) des valeurs des contrôles serveur (dans notre cas : la liste des familles et le contenu du tableau d'état du stock) :

```
<input type="hidden" name="__VIEWSTATE"
 value="dDwtNTc5NDY2NTIyO3Q8O2w8aT… " />
```

Le rôle de __VIEWSTATE est de permettre l'initialisation des contrôles dans la page nouvellement chargée à partir des valeurs existantes des contrôles : arrivée sur le serveur, la requête HTTP provoque une nouvelle exécution du programme « cracheur de HTML » associé à la page (déjà compilé et maintenu en cache), lequel décode les valeurs contenues dans __VIEWSTATE et initialise en conséquence les contrôles serveur. En d'autres termes, on a l'impression de charger la même page, alors qu'en réalité, on charge une nouvelle instance de la page initialisée avec les mêmes valeurs !

Après cette phase d'initialisation, le gestionnaire Page_Load est appelé. Grâce à la propriété IsPostBack, il est possible de contrôler spécifiquement les données à recharger : en l'occurrence, le contenu de la liste des familles ayant été initialisé à partir des valeurs contenues dans __VIEWSTATE, il n'est plus nécessaire de se connecter à la base pour récupérer la liste des familles.

Ensuite, l'examen de la valeur du contrôle caché __EVENTTARGET conduit à l'appel du gestionnaire OnFamilleChanged, qui réalise, dans notre cas, le chargement de l'état de stocks associé à la nouvelle famille sélectionnée.

Remarquons au passage qu'ici, il n'est pas nécessaire de transporter la valeur du DataGrid dans __VIEWSTATE, puisqu'il sera systématiquement rechargé à partir de la base :

```
<asp:DataGrid id="EtatStock" runat="server" …
 EnableViewState="false">
```

Enfin, la page est renvoyée vers le navigateur !

À RETENIR EnableViewState

La propriété EnableViewState permet d'exclure spécifiquement un contrôle serveur du mécanisme de ViewState. Ceci permet d'optimiser la taille des pages HTML générées par le moteur ASP.NET.

En résumé, ce mécanisme de gestion des événements facilite l'implémentation de pages Web interactives et conduit à une meilleure structuration du code, tout en restant relativement peu coûteux en termes de performances, à condition de tirer parti des possibilités d'optimisation offertes par IsPostBack et EnableViewState.

Enfin, notons que cette architecture ne stocke aucune donnée associée au client sur le serveur (l'information est stockée dans les contrôles cachés de la page HTML) : par conséquent, plusieurs requêtes successives d'un même client peuvent être traitées par des serveurs différents, ce qui est appréciable dans le cas de grappes de serveurs.

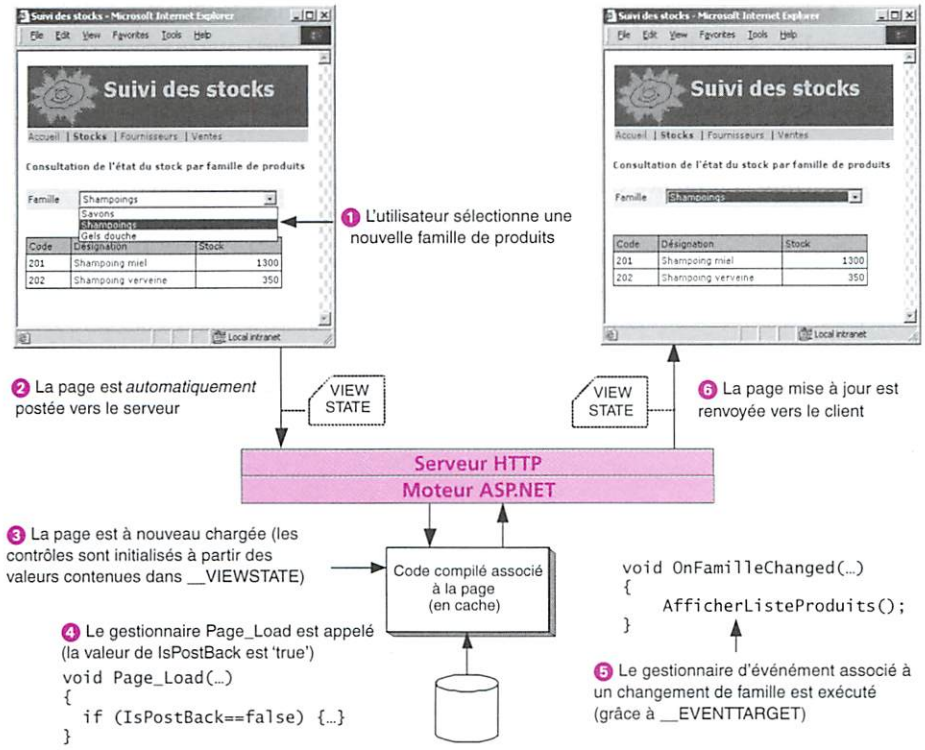


Figure 4–22 Le mécanisme de gestion événementielle

En résumé…

Dans ce chapitre, nous avons présenté plusieurs éléments fondamentaux d'ASP.NET, que nous avons mis en œuvre pour réaliser le module de suivi des stocks de notre étude de cas :

- la bibliothèque ADO.NET, qui permet de gérer la communication avec des sources de données et qui a été conçue pour obtenir un code mieux organisé (séparation nette entre les classes de manipulation et les classes de communication) et mieux adapté aux architectures Web (conservation de données en cache du fait du caractère déconnecté des applications, gestion de sources de données XML) ;
- la technique permettant de partager la connexion à la base de données entre tous les éléments de l'application, via le fichier `global.asax` et l'objet `Session` ;
- le mécanisme de liaison d'un contrôle serveur à une source de données via la propriété `DataSource` avec l'aide des classes `SqlConnection`, `SqlCommand` et `SqlDataReader` (variante possible avec `SqlDataAdapter` et `DataTable`) ;

- les nombreuses possibilités de paramétrage du contrôle `DataGrid` (colonnes liées à un champ, colonnes hypertextes, contrôle des couleurs et de la largeur des colonnes, contrôle des bordures du tableau) ;
- le mécanisme de gestion événementielle d'ASP.NET, fondé sur la technique de l'aller-retour (*round trip*), détecté grâce à la propriété `IsPostBack` et la conservation des valeurs des contrôles assurée par `__VIEWSTATE`.

Dans le chapitre suivant, nous allons implémenter le module de gestion des fournisseurs de notre étude de cas : nous verrons au passage comment mettre à jour une base de données et valider les informations saisies par un utilisateur.