Variations autour d'une architecture web MVC à trois couches
- Plate-forme .NET, langage VB.net -

Partie 1

Les idées exprimées dans ce document ont pour origine un livre lu au cours de l'été 2004, le magnifique travail de **Rod Johnson : J2EE Development without EJB** aux éditions **Wrox**.

serge.tahe@istia.univ-angers.fr, avril 2005

1 Introduction

Objectifs de l'article :

- écrire une application web à 3 couches [interface utilisateur, métier, accès aux données]
- configurer l'application avec Spring IOC
- écrire différentes versions en changeant l'implémentation de l'une ou l'autre des trois couches.

Outils utilisés :

- **Visual Studio.net** pour le développement - voir en annexe le paragraphe 7.1, page 52.
- **Serveur web Cassini** pour l'exécution - voir annexe paragraphe 7.2, page 55
- **Nunit** pour les tests unitaires - voir annexe paragraphe 7.4, page 62
- **Spring** pour l'intégration et la configuration des couches de l'application web - voir annexe paragraphe 7.3, page 61

Dans une échelle débutant-intermédiaire-avancé, ce document est dans la partie [intermédiaire-avancé]. Sa compréhension nécessite divers pré-requis. Certains d'entre-eux peuvent être acquis dans des documents que j'ai écrits. Dans ce cas, je les cite. Il est bien évident que ce n'est qu'une suggestion et que le lecteur peut utiliser ses documents favoris.

- langage VB.net : [http://tahe.developpez.com/dotnet/vbnet/]
- programmation web en VB.net : [http://tahe.developpez.com/dotnet/aspnet/vol1 et http://.../vol2]
- utilisation de l'aspect IoC de Spring : [http://tahe.developpez.com/dotnet/springioc]
- documentation Spring.net : [http://www.springframework.net/documentation.html]

Ce document reprend le fil conducteur d'un document écrit pour Java [http://tahe.developpez.com/java/web3tier]. Nous construisons en VB.NET l'application web MVC à trois couches écrite en Java. L'idée que l'on veut faire passer ici est que les plate-formes de développement Java et .Net sont suffisamment proches l'une de l'autre pour que les compétences acquises dans l'un de ces deux domaines puissent être réutilisées dans l'autre.

Il ne semble pas exister de solution de développement MVC ASP.NET largement reconnue. La solution qui suit reprend la méthode introduite dans le document [http://tahe.developpez.com/dotnet/aspnet/vol1]. Si celle-ci a le mérite d'utiliser des concepts répandus dans le développement J2EE, il ne faut cependant la prendre que pour ce qu'elle est, c.a.d. une méthode parmi d'autres de développement MVC. Dès qu'une méthode de développement MVC en ASP.NET sera largement acceptée, il faudra alors adopter cette dernière. La version .Net de Spring, en cours de développement, pourrait bien être une première solution.

2 L'application webarticles

Nous présentons ici les éléments d'une application web simplifiée de commerce électronique. Celle-ci permettra à des clients du web :

- de consulter une liste d'articles provenant d'une base de données
- d'en mettre certains dans un panier électronique
- de valider celui-ci. Cette validation aura pour seul effet de mettre à jour, dans la base, les stocks des articles achetés.

Les différentes vues présentées à l'utilisateur seront les suivantes :

- la vue "LISTE" qui présente une liste des articles en vente

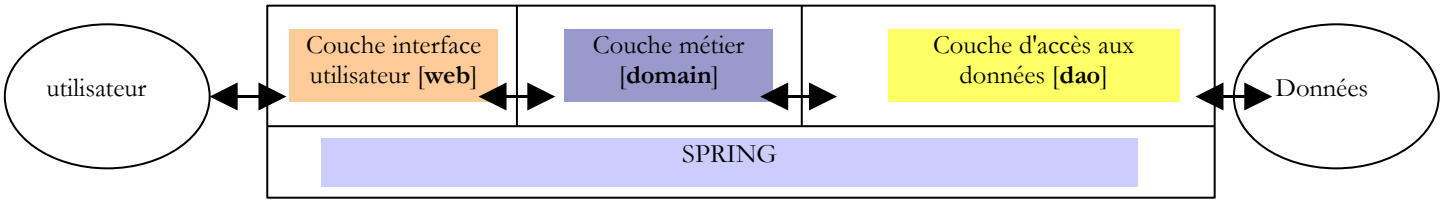
- la vue [INFOS] qui donne des informations supplémentaires sur un produit :

- les vues [PANIER] et [PANIERVIDE] qui donnent le contenu du panier du client

- la vue [ERREURS] qui signale toute erreur de l'application

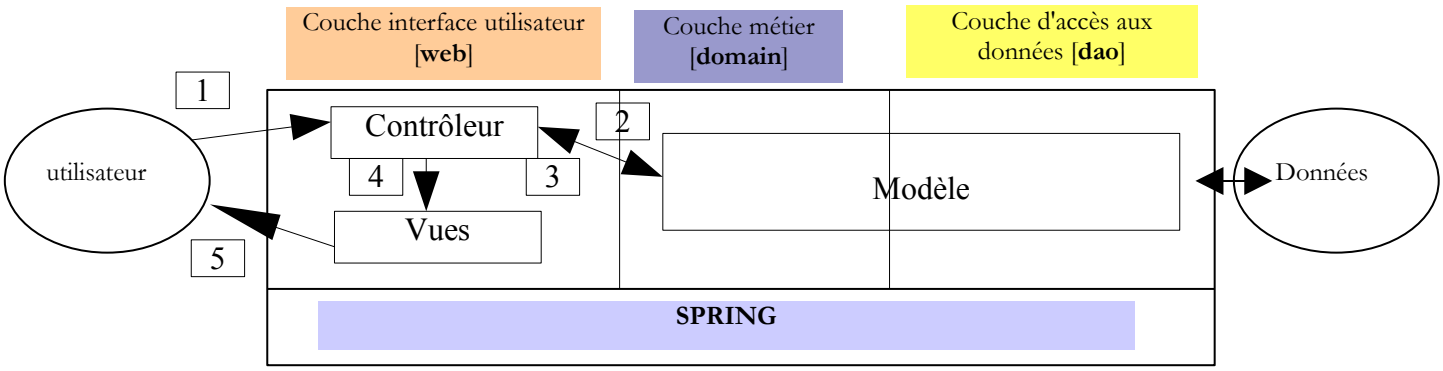
3 Architecture générale de l'application

On souhaite construire une application ayant la structure à trois couches suivante :



- les trois couches sont rendues indépendantes grâce à l'utilisation d'interfaces
- l'intégration des différentes couches est réalisée par **Spring**
- chaque couche fait l'objet d'espaces de noms séparés : **web** (couche UI), **domain** (couche métier) et **dao** (couche d'accès aux données).

L'application respectera une architecture MVC (Modèle - Vue - Contrôleur). Si nous reprenons le schéma en couches ci-dessus, l'architecture MVC s'y intègre de la façon suivante :



Le traitement d'une demande d'un client se déroule selon les étapes suivantes :

1. le client fait une demande au contrôleur. Ce contrôleur sera ici une page .aspx à laquelle on fera jouer un rôle particulier. Elle voit passer toutes les demandes des clients. C'est la porte d'entrée de l'application. C'est le C de MVC.
2. le contrôleur traite cette demande. Pour ce faire, il peut avoir besoin de l'aide de la couche métier, ce qu'on appelle le modèle M dans la structure MVC.
3. le contrôleur reçoit une réponse de la couche métier. La demande du client a été traitée. Celle-ci peut appeler plusieurs réponses possibles. Un exemple classique est
   - une page d'erreurs si la demande n'a pu être traitée correctement
   - une page de confirmation sinon
4. le contrôleur choisit la réponse (= vue) à envoyer au client. Celle-ci est le plus souvent une page contenant des éléments dynamiques. Le contrôleur fournit ceux-ci à la vue.
5. la vue est envoyée au client. C'est le V de MVC.

4 Le modèle

Nous étudions ici le M de MVC. Le modèle est ici constitué des éléments suivants :

1. les classes métier
2. les classes d'accès aux données
3. la base de données

4.1 La base de données

La base de données ne contient qu'une table appelée ARTICLES. Celle-ci a été générée avec les commandes SQL suivantes :

```
CREATE TABLE ARTICLES (
    ID            INTEGER NOT NULL,
    NOM           VARCHAR(20) NOT NULL,
    PRIX          NUMERIC(15,2) NOT NULL,
    STOCKACTUEL   INTEGER NOT NULL,
    STOCKMINIMUM  INTEGER NOT NULL
);
/* contraintes */
ALTER TABLE ARTICLES ADD CONSTRAINT CHK_ID check (ID>0);
ALTER TABLE ARTICLES ADD CONSTRAINT CHK_PRIX check (PRIX>=0);
ALTER TABLE ARTICLES ADD CONSTRAINT CHK_STOCKACTUEL check (STOCKACTUEL>=0);
ALTER TABLE ARTICLES ADD CONSTRAINT CHK_STOCKMINIMUM check (STOCKMINIMUM>=0);
ALTER TABLE ARTICLES ADD CONSTRAINT CHK_NOM check (NOM<>'');
ALTER TABLE ARTICLES ADD CONSTRAINT UNQ_NOM UNIQUE (NOM);
/* clé primaire */
ALTER TABLE ARTICLES ADD CONSTRAINT PK_ARTICLES PRIMARY KEY (ID);
```

`id`	clé primaire identifiant un article de façon unique
`nom`	nom de l'article
`prix`	son prix
`stockactuel`	son stock actuel
`stockminimum`	le stock au-dessous duquel une commande de réapprovisionnement doit être faite

4.2 Les espaces de noms du modèle

Le modèle M est ici fourni sous la forme de deux espaces de noms :

- **istia.st.articles.dao** : contient les classes d'accès aux données de la couche [dao]
- **istia.st.articles.domain** : contient les classes métier de la couche [domain]

Chacun de ces espaces de noms sera généré au sein d'un fichier " assembly " qui lui sera propre :

assembly	*contenu*	*rôle*
webarticles-dao	- [IArticlesDao]: l'interface d'accès à la couche [dao] C'est la seule interface que voit la couche [domain]. Elle n'en voit pas d'autre. - [Article] : classe définissant un article - [ArticlesDaoArrayList] : classe d'implémentation de l'interface [IArticlesDao] avec une classe [ArrayList]	couche d'accès aux données - se trouve entièrement dans la couche [dao] de l'architecture 3-tier de l'application web
webarticles-domain	- [IArticlesDomain]: l'interface d'accès à la couche [domain]. C'est la seule interface que voit la couche web. Elle n'en voit pas d'autre. - [AchatsArticles] : une classe implémentant [IArticlesDomain] - [Achat] : classe représentant l'achat d'un client - [Panier] : classe représentant l'ensemble des achats d'un client	représente le modèle des achats sur le web - se trouve entièrement dans la couche [domain] de l'architecture 3-tier de l'application web

4.3 La couche [dao]

4.3.1 Structure de la couche

La couche [dao] contient les éléments suivants :

- [**IArticlesDao**]: l'interface d'accès à la couche [dao]
- [**Article**] : classe définissant un article
- [**ArticlesDaoArrayList**] : classe d'implémentation de l'interface [IArticlesDao] avec une classe [ArrayList]

La structure du projet [Visual Studio] de la couche [dao] est la suivante :

Commentaires :

- la projet [dao] est de type [bibliothèque de classes]
- les classes ont été mises dans une arborescence de racine le dossier [istia]. Elles sont toutes dans l'espace de noms [istia.st.articles.dao].

4.3.2 La classe [Article]

La classe définissant un article est la suivante :

```
Imports System

Namespace istia.st.articles.dao

  Public Class Article

    ' champs privés
    Private _id As Integer
    Private _nom As String
    Private _prix As Double
    Private _stockactuel As Integer
    Private _stockminimum As Integer

    ' id article
    Public Property id() As Integer
      Get
        Return _id
      End Get
      Set(ByVal Value As Integer)
        If Value <= 0 Then
          Throw New Exception("Le champ id [" + Value.ToString + "] est invalide")
        End If
        Me._id = Value
      End Set
    End Property

    ' nom article
    Public Property nom() As String
      Get
        Return _nom
      End Get
      Set(ByVal Value As String)
        If Value Is Nothing OrElse Value.Trim.Equals("") Then
          Throw New Exception("Le champ nom [" + Value + "] est invalide")
        End If
        Me._nom = Value
      End Set
    End Property

    ' prix article
    Public Property prix() As Double
      Get
        Return _prix
      End Get
      Set(ByVal Value As Double)
        If Value < 0 Then
          Throw New Exception("Le champ prix [" + Value.ToString + "] est invalide")
        End If
        Me._prix = Value
      End Set
    End Property

    ' stock actuel article
    Public Property stockactuel() As Integer
      Get
        Return _stockactuel
      End Get
      Set(ByVal Value As Integer)
        If Value < 0 Then
          Throw New Exception("Le champ stockActuel [" + Value.ToString + "] est invalide")
        End If
        Me._stockactuel = Value
      End Set
    End Property

    ' stock minimum article
    Public Property stockminimum() As Integer
      Get
        Return _stockminimum
      End Get
      Set(ByVal Value As Integer)
        If Value < 0 Then
          Throw New Exception("Le champ stockMinimum [" + Value.ToString + "] est invalide")
        End If
        Me._stockminimum = Value
      End Set
    End Property

    ' constructeur par défaut
    Public Sub New()
    End Sub
```

```
    ' constructeur avec propriétés
    Public Sub New(ByVal id As Integer, ByVal nom As String, ByVal prix As Double, ByVal stockactuel As
Integer, ByVal stockminimum As Integer)
      Me.id = id
      Me.nom = nom
      Me.prix = prix
      Me.stockactuel = stockactuel
      Me.stockminimum = stockminimum
    End Sub

    ' méthode d'identification de l'article
    Public Overrides Function ToString() As String
      Return "[" + id.ToString + "," + nom + "," + prix.ToString + "," + stockactuel.ToString + "," +
stockminimum.ToString + "]"
    End Function
  End Class
End Namespace
```

Cette classe offre :

1. un constructeur permettant de fixer les 5 informations d'un article : [id, nom, prix, stockactuel, stockminimum]
2. des propriétés publiques permettant de lire et écrire les 5 informations.
3. une vérification des données insérées dans l'article. En cas de données erronées, une exception est lancée.
4. une méthode **toString** qui permet d'obtenir la valeur d'un article sous forme de chaîne de caractères. C'est souvent utile pour le débogage d'une application.

4.3.3 L'interface [IArticlesDao]

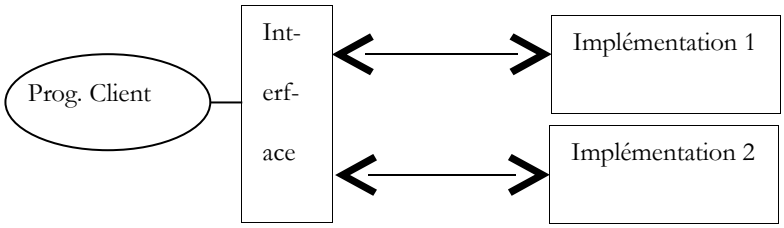
L'interface [IArticlesDao] est définie comme suit :

```
Imports System
Imports System.Collections

Namespace istia.st.articles.dao

  Public Interface IArticlesDao
    ' liste de tous les articles
    Function getAllArticles() As IList
    ' ajoute un article
    Function ajouteArticle(ByVal unArticle As Article) As Integer
    ' supprime un article
    Function supprimeArticle(ByVal idArticle As Integer) As Integer
    ' modifie un article
    Function modifieArticle(ByVal unArticle As Article) As Integer
    ' recherche un article
    Function getArticleById(ByVal idArticle As Integer) As Article
    ' supprime tous les articles
    Sub clearAllArticles()
    ' change le stock d'u article
    Function changerStockArticle(ByVal idArticle As Integer, ByVal mouvement As Integer) As Integer
  End Interface
End Namespace
```

Le rôle des différentes méthodes de l'interface est le suivant :

`getAllArticles`	rend tous les articles de la source de données
`clearAllArticles`	vide la source de données
`getArticleById`	rend l'objet [Article] identifié par sa clé primaire
`ajouteArticle`	permet d'ajouter un article à la source de données
`modifieArticle`	permet de modidier un article de la source de données
`supprimerArticle`	permet de supprimer un article de la source de données
`changerStockArticle`	permet de modifier le stock d'un article de la source de données

L'interface met à disposition des programmes clients un certain nombre de méthodes définies uniquement par leurs signatures. Elle ne s'occupe pas de la façon dont ces méthodes seront réellement implémentées. Cela amène de la souplesse dans une application. Le programme client fait ses appels sur une interface et non pas sur une implémentation précise de celle-ci.



Le choix d'une implémentation précise se fera au moyen d'un fichier de configuration **Spring**. Afin de montrer que pour tester l'application web, seule l'interface d'accès aux données compte et non sa classe d'implémentation, nous allons tout d'abord implémenter la source de données par un simple objet [ArrayList]. Ultérieurement, nous présenterons une solution à base de SGBD.

4.3.4 La classe d'implémentation [ArticlesDaoArrayList]

La classe d'implémentation [ArticlesDaoArrayList] est définie comme suit :

```
Imports System
Imports System.Collections

Namespace istia.st.articles.dao

  Public Class ArticlesDaoArrayList
    Implements istia.st.articles.dao.IArticlesDao

    Private articles As New ArrayList
    Private Const nbArticles As Integer = 4

    ' constructeur par défaut
    Public Sub New()
      ' on construit qqs articles
      For i As Integer = 1 To nbArticles
        articles.Add(New Article(i, "article" + i.ToString, i * 10, i * 10, i * 10))
      Next
    End Sub

    ' liste de tous les articles
    Public Function getAllArticles() As IList Implements IArticlesDao.getAllArticles
      ' on retourne la liste des articles
      SyncLock Me
        Return articles
      End SyncLock
    End Function

    ' suppression de tous les articles
    Public Sub clearAllArticles() Implements IArticlesDao.clearAllArticles
      ' on vide la liste des articles
      SyncLock Me
        articles.Clear()
      End SyncLock
    End Sub

    ' obtenir un article identifié par sa clé
    Public Function getArticleById(ByVal idArticle As Integer) As Article Implements
IArticlesDao.getArticleById
      ' on recherche l'article dans la liste
      SyncLock Me
        Dim ipos As Integer = posArticle(articles, idArticle)
        If ipos <> -1 Then
          Return CType(articles(ipos), Article)
        Else
          Return Nothing
        End If
      End SyncLock
    End Function

    ' ajouter un article à la liste des articles
    Public Function ajouteArticle(ByVal unArticle As Article) As Integer Implements
IArticlesDao.ajouteArticle
      ' on ajoute l'article à la liste des articles
      SyncLock Me
        ' on vérifie qu'il n'existe pas déjà
        Dim ipos As Integer = posArticle(articles, unArticle.id)
        If ipos <> -1 Then
          Throw New Exception("L'article d'id [" + unArticle.id.ToString + "] existe déjà")
        End If
        ' on ajoute l'article
        articles.Add(unArticle)
        ' on rend le résultat
        Return 1
```

```
    End SyncLock
  End Function

  ' modifier un article
  Public Function modifieArticle(ByVal articleNouveau As Article) As Integer Implements
IArticlesDao.modifieArticle
    ' on modifie un article
    SyncLock Me
      ' on vérifie qu'il existe
      Dim ipos As Integer = posArticle(articles, articleNouveau.id)
      ' le cas où il n'existe pas
      If ipos = -1 Then Return 0
      ' il existe - on le modifie
      articles(ipos) = articleNouveau
      ' on rend le résultat
      Return 1
    End SyncLock
  End Function

  ' supprimer un article identifié par sa clé
  Public Function supprimeArticle(ByVal idArticle As Integer) As Integer Implements
IArticlesDao.supprimeArticle
    ' suppression d'un article
    SyncLock Me
      ' on vérifie qu'il existe
      Dim ipos As Integer = posArticle(articles, idArticle)
      ' le cas où il n'existe pas
      If ipos = -1 Then Return 0
      ' il existe - on le supprime
      articles.RemoveAt(ipos)
      ' on rend le résultat
      Return 1
    End SyncLock
  End Function

  ' changer le stock d'un article identifié par sa clé
  Public Function changerStockArticle(ByVal idArticle As Integer, ByVal mouvement As Integer) As Integer
Implements IArticlesDao.changerStockArticle
    ' changer le stock d'un article
    SyncLock Me
      ' on vérifie qu'il existe
      Dim ipos As Integer = posArticle(articles, idArticle)
      ' le cas où il n'existe pas
      If ipos = -1 Then Return 0
      ' il existe - on modifie son stock si on peut
      Dim unArticle As Article = CType(articles(ipos), Article)
      ' on ne change le stock que s'il est suffisant
      If unArticle.stockactuel + mouvement >= 0 Then
        unArticle.stockactuel += mouvement
        Return 1
      Else
        Return 0
      End If
    End SyncLock
  End Function

  ' chercher un article identifié par sa clé
  Private Function posArticle(ByVal listArticles As ArrayList, ByVal idArticle As Integer) As Integer
    ' rend la position de l'article [idArticle] dans la liste ou -1 si pas trouvé
    Dim unArticle As Article
    For i As Integer = 0 To listArticles.Count - 1
      unArticle = CType(listArticles(i), Article)
      If unArticle.id = idArticle Then
        Return i
      End If
    Next
    ' pas trouvé
    Return -1
  End Function
 End Class

End Namespace
```

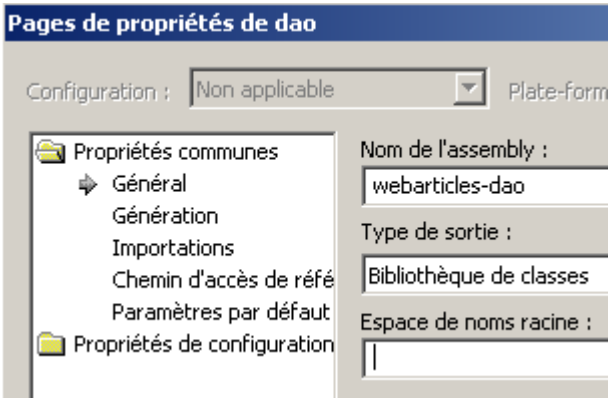
Commentaires :

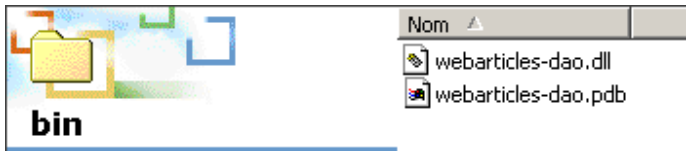
- la source de données est simulée par le champ privé [articles] de type [ArrayList]
- le constructeur de la classe crée par défaut 4 articles dans la source de données.
- toutes les méthodes d'accès aux données ont été synchronisées afin d'éviter les problèmes d'accès concurrents à la source de données. A un moment donné, un seul thread a accès à une méthode donnée.
- la méthode [posArticle] permet de connaître la position [0..N] dans la source [ArrayList] d'un article identifié par son n°. Si l'article n'existe pas, la méthode rend la position -1. Cette méthode est utillisée de façon répétée par les autres méthodes.
- la méthode [ajouteArticle] permet d'ajouter un article dans la liste des articles. Elle rend le nombre d'articles insérés : 1. Si l'article existait déjà, une exception est lancée.

- la méthode [modifieArticle] permet de modifier un article existant. Elle rend le nombre d'articles modifiés : 1 si l'article existait, 0 sinon.
- la méthode [supprimeArticle] permet de supprimer un article existant. Elle rend le nombre d'articles supprimés : 1 si l'article existait, 0 sinon.
- la méthode [getAllArticles] donne la liste de tous les articles
- la méthode [getArticleById] permet d'obtenir un article identifié par son n°. On obtient la valeur [nothing] si l'article n'xiste pas.
- le code ne présente pas de réelle difficulté. Nous laissons le lecteur le parcourir et le comprendre.

4.3.5 Génération de l'assembly de la couche [dao]

Le projet Visual Studio est configuré pour générer l'assembly [webarticles-dao.dll]. Celui-ci est généré dans le dossier [bin] du projet :

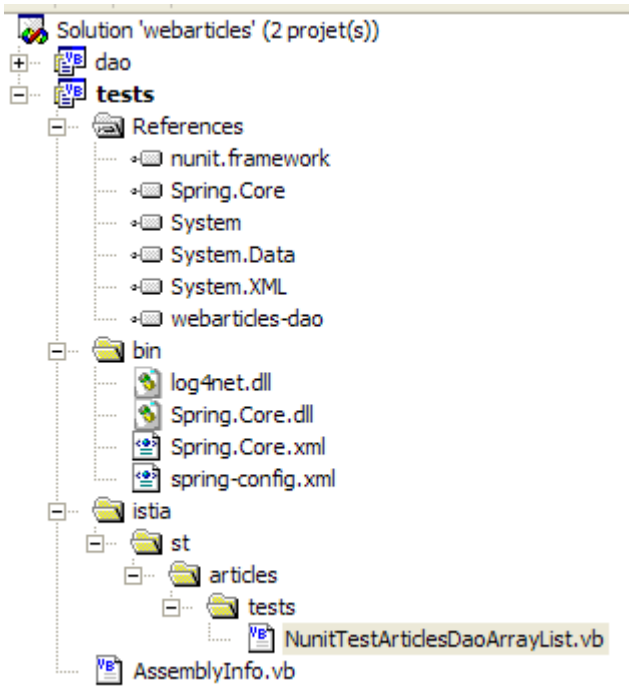




4.3.6 Tests Nunit de la couche [dao]

En Java, les classes sont testées avec le framework [Junit]. En .Net, le framework **Nunit** offre les mêmes possibilités de tests unitaires :

La structure du projet Visual Studio de test est la suivante :



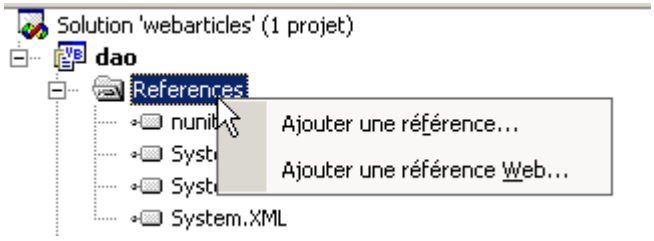
Commentaires :

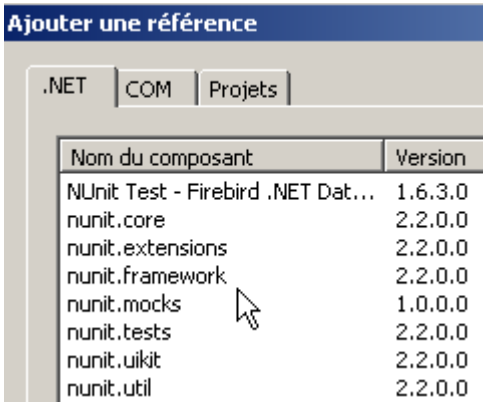
- le projet [tests] est de type [bibliothèque de classes]
- les tests [NUnit] nécessitent une référence sur l'assembly [nunit.framework.dll]
- la classe de test [NUnit] récupère une instance de l'objet à tester via Spring. Aussi trouve-t-on
  - dans le dossier [bin], les fichiers de classes de Spring
  - dans [References], une référence à l'assembly [Spring-Core.dll] du dossier [bin]
  - dans [bin], un fichier de configuration pour Spring
- la classe de test a besoin de l'assembly [webarticles-dao.dll] de la couche [dao]. Celui-ci a été placé dans le dossier [bin] et sa référence ajoutée aux références du projet.

Une classe de test [NUnit] nécessite l'accès aux classes de l'espace de noms [NUnit.Framework]. On y trouve donc l'instruction d'import suivante :

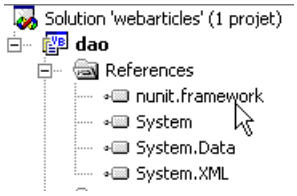
```
Imports NUnit.Framework
```

L'espace de noms [NUnit.Framework] se trouve dans " l'assembly " [nunit.framework.dll] qu'on doit ajouter aux références du projet :





L'assembly [nunit.framework.dll] doit être dans la liste proposée si l'installation de [Nunit] a été faite. Il suffit de double-cliquer sur l'assembly pour l'ajouter au projet :



La classe de test [NUnit] de la couche [dao] pourrait être la suivante :

```
Imports System
Imports System.Collections
Imports NUnit.Framework
Imports istia.st.articles.dao
Imports System.Threading
Imports Spring.Objects.Factory.Xml
Imports System.IO

Namespace istia.st.articles.tests

  <TestFixture()> _
  Public Class NunitTestArticlesArrayList
    ' l'objet à tester
    Private articlesDao As IArticlesDao

    <SetUp()> _
    Public Sub init()
      ' on récupère une instance du fabricant d'objets Spring
      Dim factory As XmlObjectFactory = New XmlObjectFactory(New FileStream("spring-config.xml",
FileMode.Open))
      ' on demande l'instanciation de l'objet articlesdao
      articlesDao = CType(factory.GetObject("articlesdao"), IArticlesDao)
    End Sub

    <Test()> _
    Public Sub testGetAllArticles()
      ' vérification visuelle
      listArticles()
    End Sub

    <Test()> _
    Public Sub testClearAllArticles()
      ' on supprime tous les articles
      articlesDao.clearAllArticles()
      ' on demande tous les articles
      Dim articles As IList = articlesDao.getAllArticles
      ' vérification : il doit y en avoir 0
      Assert.AreEqual(0, articles.Count)
    End Sub

    <Test()> _
    Public Sub testAjouteArticle()
      ' suppression de tous les articles
      articlesDao.clearAllArticles()
      ' vérification : la table des articles doit être vide
      Dim articles As IList = articlesDao.getAllArticles
      Assert.AreEqual(0, articles.Count)
      ' on ajoute deux articles
      articlesDao.ajouteArticle(New Article(3, "article3", 30, 30, 3))
      articlesDao.ajouteArticle(New Article(4, "article4", 40, 40, 4))
      ' vérification : il doit y avoir deux articles
      articles = articlesDao.getAllArticles
      Assert.AreEqual(2, articles.Count)
      ' vérification visuelle
      listArticles()
    End Sub

    <Test()> _
    Public Sub testSupprimeArticle()
      ' suppression de tous les articles
      articlesDao.clearAllArticles()
      ' vérification : la table des articles doit être vide
      Dim articles As IList = articlesDao.getAllArticles
      Assert.AreEqual(0, articles.Count)
      ' on ajoute deux articles
      articlesDao.ajouteArticle(New Article(3, "article3", 30, 30, 3))
      articlesDao.ajouteArticle(New Article(4, "article4", 40, 40, 4))
      ' vérification : il doit y avoir 2 articles
      articles = articlesDao.getAllArticles
      Assert.AreEqual(2, articles.Count)
      ' on supprime l'article 4
      articlesDao.supprimeArticle(4)
      ' vérification : il doit rester 1 article
      articles = articlesDao.getAllArticles
      Assert.AreEqual(1, articles.Count)
```

```
    ' vérification visuelle
    listArticles()
  End Sub

  <Test()> _
  Public Sub testModifieArticle()
    ' suppression de tous les articles
    articlesDao.clearAllArticles()
    ' vérification
    Dim articles As IList = articlesDao.getAllArticles
    Assert.AreEqual(0, articles.Count)
    ' ajout de 2 articles
    articlesDao.ajouteArticle(New Article(3, "article3", 30, 30, 3))
    articlesDao.ajouteArticle(New Article(4, "article4", 40, 40, 4))
    ' vérification
    articles = articlesDao.getAllArticles
    Assert.AreEqual(2, articles.Count)
    ' recherche de l'article 3
    Dim unArticle As Article = articlesDao.getArticleById(3)
    ' vérification
    Assert.AreEqual(unArticle.nom, "article3")
    ' recherche article 4
    unArticle = articlesDao.getArticleById(4)
    ' vérification
    Assert.AreEqual(unArticle.nom, "article4")
    ' modification article 4
    articlesDao.modifieArticle(New Article(4, "article4", 44, 44, 44))
    ' vérification
    unArticle = articlesDao.getArticleById(4)
    Assert.AreEqual(unArticle.prix, 44, 0.000001)
    ' vérification visuelle
    listArticles()
  End Sub

  <Test()> _
  Public Sub testGetArticleById()
    ' suppression des articles
    articlesDao.clearAllArticles()
    ' vérification
    Dim articles As IList = articlesDao.getAllArticles
    Assert.AreEqual(0, articles.Count)
    ' ajout de 2 articles
    articlesDao.ajouteArticle(New Article(3, "article3", 30, 30, 3))
    articlesDao.ajouteArticle(New Article(4, "article4", 40, 40, 4))
    ' vérification
    articles = articlesDao.getAllArticles
    Assert.AreEqual(2, articles.Count)
    ' recherche article 3
    Dim unArticle As Article = articlesDao.getArticleById(3)
    ' vérification
    Assert.AreEqual(unArticle.nom, "article3")
    ' recherche article 4
    unArticle = articlesDao.getArticleById(4)
    ' vérification
    Assert.AreEqual(unArticle.nom, "article4")
  End Sub

  ' listing écran
  Private Sub listArticles()
    Dim articles As IList = articlesDao.getAllArticles
    For i As Integer = 0 To articles.Count - 1
      Console.WriteLine(CType(articles(i), Article).ToString)
    Next
  End Sub

  <Test()> _
  Public Sub testArticleAbsent()
    ' suppression de tous les articles
    articlesDao.clearAllArticles()
    ' recherche article 1
    Dim article As article = articlesDao.getArticleById(1)
    ' vérification
    Assert.IsNull(article)
    ' modification d'un article inexistant
    Dim i As Integer = articlesDao.modifieArticle(New article(1, "1", 1, 1, 1))
    ' a du modifier aucune ligne
    Assert.AreEqual(i, 0)
    ' suppression article inexistant
    i = articlesDao.supprimeArticle(1)
    ' a du supprimer aucune ligne
    Assert.AreEqual(0, i)
  End Sub

  <Test()> _
  Public Sub testChangerStockArticle()
    ' suppression tous les articles
    articlesDao.clearAllArticles()
```

```
    ' ajout d'un article
    Dim nbArticles As Integer = articlesDao.ajouteArticle(New Article(3, "article3", 30, 101, 3))
    Assert.AreEqual(nbArticles, 1)
    ' ajout d'un article
    nbArticles = articlesDao.ajouteArticle(New Article(4, "article4", 40, 40, 4))
    Assert.AreEqual(nbArticles, 1)
    ' création de 100 threads
    Dim taches(99) As Thread
    For i As Integer = 0 To taches.Length - 1
      ' on crée le thread i
      taches(i) = New Thread(New ThreadStart(AddressOf décrémente))
      ' on fixe le nom du thread
      taches(i).Name = "tache_" & i
      ' on lance l'exécution du thread i
      taches(i).Start()
    Next
    ' on attend la fin de tous les threads
    For i As Integer = 0 To taches.Length - 1
      taches(i).Join()
    Next
    ' vérifications - article 3 doit avoir un stock de 1
    Dim unArticle As Article = articlesDao.getArticleById(3)
    Assert.AreEqual(unArticle.nom, "article3")
    Assert.AreEqual(1, unArticle.stockactuel)
    ' on décrémente le stock de l'article 4
    Dim erreur As Boolean = False
    Dim nbLignes As Integer = articlesDao.changerStockArticle(4, -100)
    ' vérification : son stock n'a pas du changer
    Assert.AreEqual(0, nbLignes)
    ' vérification visuelle
    listArticles()
  End Sub

  Public Sub décrémente()
    ' thread lancé
    System.Console.Out.WriteLine(Thread.CurrentThread.Name + " lancé")
    ' thread décrémente le stock
    articlesDao.changerStockArticle(3, -1)
    ' thread terminé
    System.Console.Out.WriteLine(Thread.CurrentThread.Name + " terminé")
  End Sub

 End Class
End Namespace
```

Commentaires :

- on a voulu écrire un programme de test de l'interface [IArticlesDao] qui soit indépendant de la classe d'implémentation de celle-ci. Aussi avons-nous utilisé **Spring** pour cacher au programme de test le nom de la classe d'implémentation.
- la méthode d'attribut <Setup()> récupère auprès de Spring une référence sur l'objet [articlesdao] à tester. Celui-ci est défini dans le fichier [spring-config.xml] suivant :

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<!DOCTYPE objects PUBLIC "-//SPRING//DTD OBJECT//EN"
"http://www.springframework.net/dtd/spring-objects.dtd">

<objects>
  <object id="articlesdao" type="istia.st.articles.dao.ArticlesDaoArrayList, webarticles-dao"/>
</objects>
```

Ce fichier indique le nom de la classe d'implémentation [istia.st.articles.dao.ArticlesDaoArrayList] de l'interface [IArticlesDao] et où la trouver [webarticles-dao.dll]. L'instanciation ne nécessitant pas de paramètres, aucun n'est défini ici.

- la plupart des tests sont simples à comprendre. Le lecteur est invité à lire les commentaires.

- La méthode [testChangerStockArticle] nécessite quelques explications. Elle crée 100 threads chargés de décrémenter le stock d'un article donné.

```
  <Test()>
  Public Sub testChangerStockArticle()
    ' suppression tous les articles
    articlesDao.clearAllArticles()
    ' ajout d'un article
    Dim nbArticles As Integer = articlesDao.ajouteArticle(New Article(3, "article3", 30, 101, 3))
    Assert.AreEqual(nbArticles, 1)
    ' ajout d'un article
    nbArticles = articlesDao.ajouteArticle(New Article(4, "article4", 40, 40, 4))
    Assert.AreEqual(nbArticles, 1)
    ' création de 100 threads
```

```
    Dim taches(99) As Thread
    For i As Integer = 0 To taches.Length - 1
      ' on crée le thread i
      taches(i) = New Thread(New ThreadStart(AddressOf décrémente))
      ' on fixe le nom du thread
      taches(i).Name = "tache_" & i
      ' on lance l'exécution du thread i
      taches(i).Start()
    Next
    ' on attend la fin de tous les threads
    For i As Integer = 0 To taches.Length - 1
      taches(i).Join()
    Next
    ' vérifications - article 3 doit avoir un stock de 1
    Dim unArticle As Article = articlesDao.getArticleById(3)
    Assert.AreEqual(unArticle.nom, "article3")
    Assert.AreEqual(1, unArticle.stockactuel)
    ' on décrémente le stock de l'article 4
    Dim erreur As Boolean = False
    Dim nbLignes As Integer = articlesDao.changerStockArticle(4, -100)
    ' vérification : son stock n'a pas du changer
    Assert.AreEqual(0, nbLignes)
    ' vérification visuelle
    listArticles()
  End Sub
```

Il s'agit ici de tester les accès concurrents à la source de données. La méthode chargée de mettre à jour le stock est la suivante :

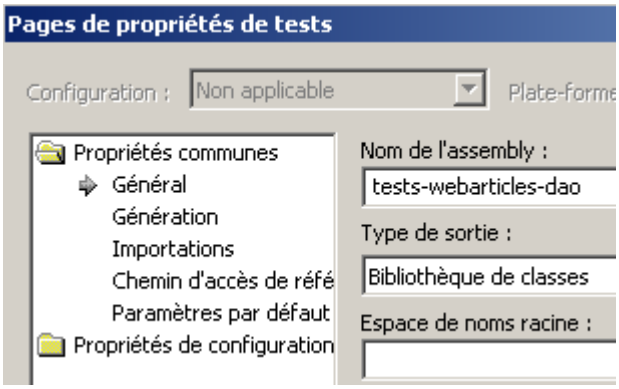
```
  Public Sub décrémente()
    ' thread lancé
    System.Console.Out.WriteLine(Thread.CurrentThread.Name + " lancé")
    ' thread décrémente le stock
    articlesDao.changerStockArticle(3, -1)
    ' thread terminé
    System.Console.Out.WriteLine(Thread.CurrentThread.Name + " terminé")
  End Sub
```

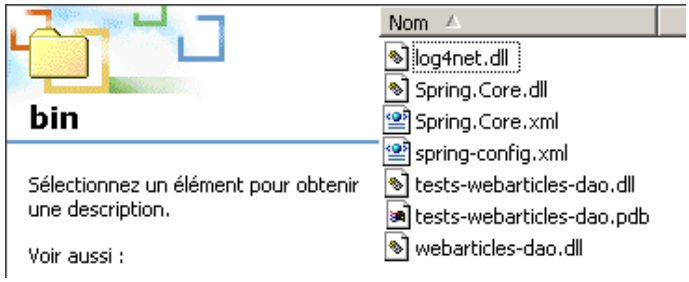
Elle décrémente d'une unité le stock de l'article n° 3. Si on se réfère au code de la méthode [testChangerStockArticle], on voit que :

- le stock de l'article n° 3 est initialisé à 101
- les 100 threads vont décrémenter ce stock d'une unité chacun
- on doit donc avoir un stock de 1 à la fin de l'exécution de tous les threads

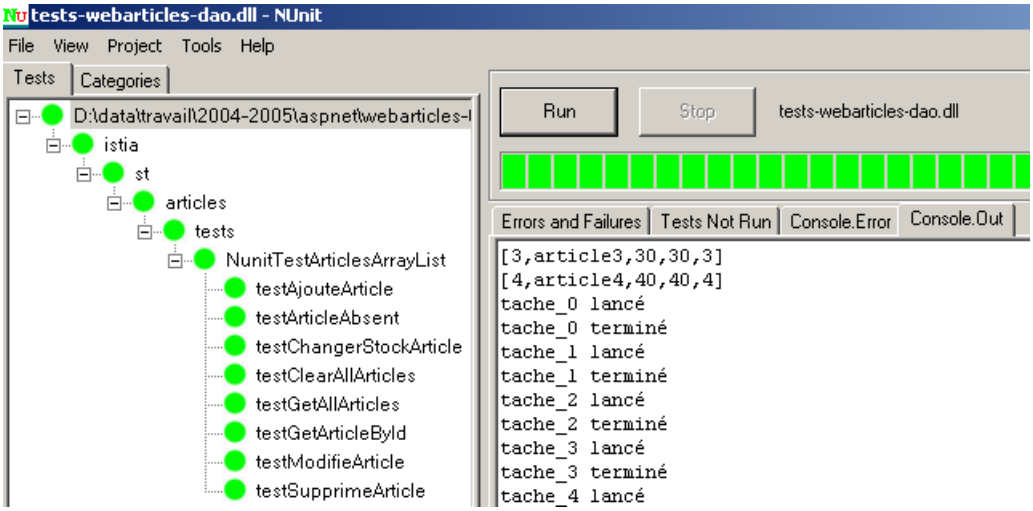
Par ailleurs, toujours dans cette méthode, on essaie de faire passer le stock de l'article n° 4 a une valeur négative. On doit échouer.

Pour tester la couche [dao], nous générons la DLL [tests-webarticles-dao.dll] dans le dossier [bin] du projet [tests] :





Puis, à l'aide de l'application [Nunit-Gui], nous chargeons cette DLL et exécutons les tests :



Dans la fenêtre de gauche, on voit la liste des méthodes testées. La couleur du point qui précède le nom de chaque méthode indique la réussite (vert) ou l'échec (rouge) de la méthode. Le lecteur qui visualise ce document sur écran pourra voir que tous les tests ont été réussis. Par la suite, nous considèrerons que nous avons une couche [dao] opérationnelle.

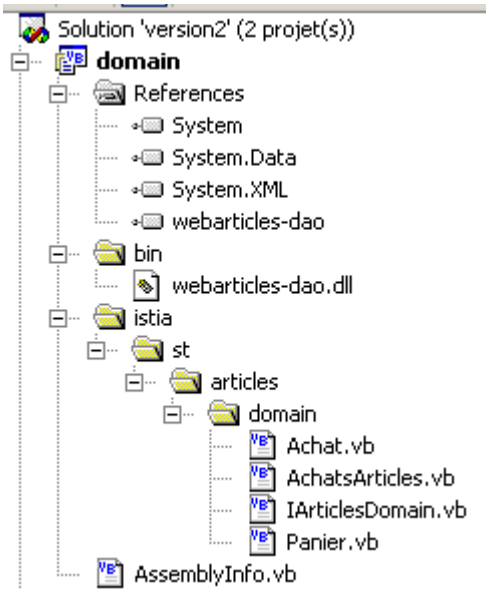
4.4 La couche [domain]

4.4.1 Structure de la couche

La couche [domain] contient les éléments suivants :

- [**IArticlesDomain**]: l'interface d'accès à la couche [domain]
- [**Achat**] : classe définissant un achat
- [**Panier**] : classe définissant un panier d'achats
- [**AchatsArticles**] : classe d'implémentation de l'interface [IArticlesDomain]

La structure de la solution [Visual Studio] de la couche [domain] est la suivante :



Commentaires :

- le projet [domain] est de type [bibliothèque de classes]
- les classes ont été mises dans une arborescence de racine le dossier [istia]. Elles sont toutes dans l'espace de noms [istia.st.articles.domain].
- la DLL de la couche [dao] a été placée dans le dossier [bin] du nouveau projet. Par ailleurs, cette DLL a été ajoutée comme référence au projet.

4.4.2 L'interface [IArticlesDomain]

L'interface [IArticlesDomain] découple la couche [métier] de la couche [web]. Cette dernière accède à la couche [métier/domain] via cette interface sans se préoccuper de la classe qui l'implémente réellement. L'interface définit les actions suivantes pour l'accès à la couche métier :

```
Imports Article = istia.st.articles.dao.Article

Namespace istia.st.articles.domain
  Public Interface IArticlesDomain
    ' méthodes
    Sub acheter(ByVal panier As Panier)
    Function getAllArticles() As IList
    Function getArticleById(ByVal idArticle As Integer) As Article
    ReadOnly Property erreurs() As ArrayList
  End Interface
End Namespace
```

`Function getAllArticles() As IList`	rend la liste d'objets [Article] de la source de données associée
`Function getArticleById(ByVal idArticle As Integer) As Article`	rend l'objet [Article] identifié par [idArticle]
`acheter(ByVal panier As Panier)`	valide le panier du client en décrémentant les stocks des articles achetés de la quantité achetée - peut échouer si le stock est insuffisant
`ReadOnly Property erreurs() As ArrayList`	rend la liste des erreurs qui se sont produites - vide si pas d'erreurs

4.4.3 La classe [Achat]

La classe [Achat] représente un achat du client :

```
Imports istia.st.articles.dao

Namespace istia.st.articles.domain

  Public Class Achat

    ' champs privés
    Private _article As article
    Private _qte As Integer

    ' constructeur par défaut
    Public Sub New()
    End Sub

    ' constructeur avec paramètres
    Public Sub New(ByVal unArticle As article, ByVal qte As Integer)
      ' on passe par les propriétés
      Me.article = unArticle
      Me.qte = qte
    End Sub

    ' article acheté
    Public Property article() As article
      Get
        Return _article
      End Get
      Set(ByVal Value As article)
        _article = Value
      End Set
    End Property

    ' qte achetée
    Public Property qte() As Integer
      Get
        Return _qte
      End Get
      Set(ByVal Value As Integer)
        If Value < 0 Then
          Throw New Exception("Quantité [" + Value.ToString + "] invalide")
        End If
        _qte = Value
      End Set
    End Property

    ' total achat
    Public ReadOnly Property totalAchat() As Double
      Get
```

```
      Return _qte * _article.prix
    End Get
  End Property

  ' identité
  Public Overrides Function ToString() As String
    Return "[" + _article.ToString + "," + _qte.ToString + "]"
  End Function
End Class

End Namespace
```

Commentaires :

- la classe [Achat] a les propriétés et méthodes suivantes :

`Public Property article() As article`	l'article acheté
`Public Property qte() As Integer`	la quantité achetée
`Public ReadOnly Property totalAchat() As Double`	le montant de l'achat
`Public Overrides Function ToString() As String`	chaîne d'identité de l'objet

- elle a un constructeur permettant d'initialiser les propriétés [article, qte] qui définissent un achat.

4.4.4 La classe [Panier]

La classe [Panier] représente l'ensemble des achats du client :

```
Namespace istia.st.articles.domain

Public Class Panier

  ' champs privés
  Private _achats As New ArrayList
    Private _totalPanier As Double = 0

  ' constructeur par défaut
  Public Sub New()
  End Sub

  ' liste des achats
  Public ReadOnly Property achats() As ArrayList
    Get
      Return _achats
    End Get
  End Property

  ' total des achats
  Public ReadOnly Property totalPanier() As Double
    Get
      Return _totalPanier
    End Get
  End Property

  ' méthodes
  Public Sub ajouter(ByVal unAchat As Achat)
    ' on cherche si l'achat existe déjà
    Dim iAchat As Integer = posAchat(unAchat.article.id)
    If iAchat <> -1 Then
      ' on a trouvé
      Dim achatCourant As Achat = CType(_achats(iAchat), Achat)
      achatCourant.qte += unAchat.qte
    Else
      ' on n'a pas trouvé
      _achats.Add(unAchat)
    End If
    ' on incrémente le total du panier
    _totalPanier += unAchat.totalAchat
  End Sub

  ' enlever un achat
  Public Sub enlever(ByVal idAchat As Integer)
    ' on cherche l'achat
    Dim iachat As Integer = posAchat(idAchat)
    ' si on a trouvé, on enlève
    If iachat <> -1 Then
       Dim achatCourant As Achat = CType(_achats(iachat), Achat)
      ' on enlève du panier
      _achats.RemoveAt(iachat)
      ' on décrémente le total du panier
      _totalPanier -= achatCourant.totalAchat
    End If
```

```
  End Sub

  Private Function posAchat(ByVal idArticle As Integer) As Integer
    ' recherche un achat dans la liste des achats
    ' rend sa position dans la liste ou -1 si pas trouvé
    Dim achatCourant As Achat
    Dim trouvé As Boolean = False
    Dim i As Integer = 0
      While Not trouvé AndAlso i < _achats.Count
        ' achat courant
        achatCourant = CType(_achats(i), Achat)
        ' comparaison avec l'article cherché
        If achatCourant.article.id = idArticle Then
          Return i
        End If
        'achat suivant
        i += 1
      End While
      ' pas trouvé
      Return -1
  End Function

  ' function identité
  Public Overrides Function ToString() As String
    Return _achats.ToString
  End Function
End Class

End Namespace
```

Commentaires :

- la classe [Panier] a les propriétés et méthodes suivantes :

`ReadOnly Property achats() As ArrayList`	la liste des achats du client - liste d'objets de type [Achat]
`ajouter(ByVal unAchat As Achat)`	ajoute un achat à la liste des achats
`enlever(ByVal idAchat As Integer)`	enlève l'achat de l'article idAchat
`ReadOnly Property totalPanier() As Double`	le montant total des achats du panier
`Function ToString() As String`	rend la chaîne d'identité du panier

- la méthode [posAchat] est une méthode utilitaire qui permet d'obtenir la position dans la liste des achats, d'un achat identifié par le n° de l'article acheté. La liste des achats est gérée de telle façon qu'un article acheté plusieurs fois n'occupe qu'une position dans la liste. Ainsi un achat peut-il être identifié par le n° de l'article acheté. La méthode [posAchat] rend -1 si l'achat recherché n'existe pas.
- la méthode [ajouter] ajoute un nouvel achat à la liste des achats. Cela revient soit à ajouter une nouvelle entrée à la liste des achats si l'article acheté n'existait pas déjà dans la liste, soit à incrémenter la quantité achetée s'il existait déjà.
- la méthode [enlever] permet d'enlever un achat identifié par un n° de la liste des achats. Si l'achat n'existe pas, la méthode est silencieuse et ne fait rien.
- le montant des achats [totalPanier] est maintenu au fil des ajouts et retraits d'achats.

4.4.5 La classe [AchatsArticles]

L'interface [IArticlesDomain] sera implémentée par la classe [AchatsArticles] suivante :

```
Imports istia.st.articles.dao

Namespace istia.st.articles.domain
  Public Class AchatsArticles
    Implements IArticlesDomain

    'champs privés
    Private _articlesDao As IArticlesDao
    Private _erreurs As ArrayList

    ' constructeur
    Public Sub New(ByVal articlesDao As IArticlesDao)
      _articlesDao = articlesDao
    End Sub

    ' liste des erreurs
    Public ReadOnly Property erreurs() As ArrayList Implements IArticlesDomain.erreurs
      Get
        Return _erreurs
      End Get
    End Property

    ' liste des articles
    Public Function getAllArticles() As IList Implements IArticlesDomain.getAllArticles
```

```
    ' liste de tous les articles
    Try
      Return _articlesDao.getAllArticles
    Catch ex As Exception
      _erreurs = New ArrayList
      _erreurs.Add("Erreur d'accès aux données : " + ex.Message)
    End Try
  End Function

  ' obtenir un article identifié par son n°
  Public Function getArticleById(ByVal idArticle As Integer) As Article Implements
IArticlesDomain.getArticleById
    ' un article particulier
    Try
      Return _articlesDao.getArticleById(idArticle)
    Catch ex As Exception
      _erreurs = New ArrayList
      _erreurs.Add("Erreur d'accès aux données : " + ex.Message)
    End Try
  End Function


  ' acheter un panier
  Public Sub acheter(ByVal panier As Panier) Implements IArticlesDomain.acheter
    ' achat d'un panier - les stocks des articles achetés doivent être décrémentés
    _erreurs = New ArrayList
    Dim achat As achat
    Dim achats As ArrayList = panier.achats
    For i As Integer = achats.Count - 1 To 0 Step -1
      ' décrémenter stock article i
      achat = CType(achats(i), achat)
      Try
        If _articlesDao.changerStockArticle(achat.article.id, -achat.qte) = 0 Then
          ' on n'a pas pu faire l'opération
          _erreurs.Add("L'achat " + achat.ToString + " n'a pu se faire - Vérifiez les stocks")
        Else
          ' l'opération s'est faite - on enlève l'achat du panier
          panier.enlever(achat.article.id)
        End If
      Catch ex As Exception
        _erreurs = New ArrayList
        _erreurs.Add("Erreur d'accès aux données : " + ex.Message)
      End Try
    Next
  End Sub
 End Class

End Namespace
```

Commentaires :

- cette classe implémente les quatre méthodes de l'interface [IArticlesDomain]. Elle a deux champs privés :

`_articlesDao As IArticlesDao`	l'objet d'accès aux données
`_erreurs As ArrayList`	la liste des erreurs éventuelles. Elle est accessible via la propriété publique [erreurs]

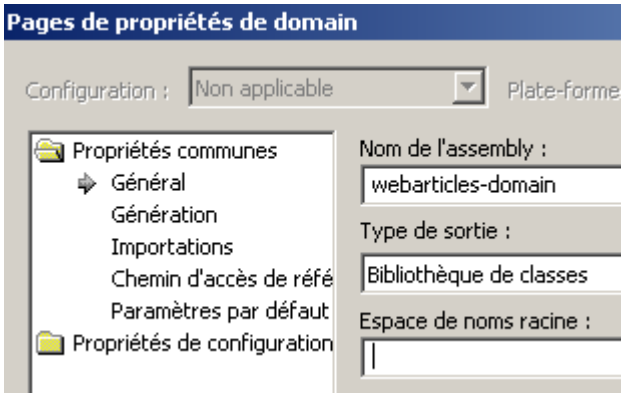
- pour construire une instance de la classe, il faut fournir l'objet permettant l'accès aux données :

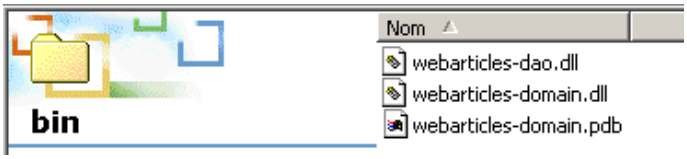
```
Sub New(ByVal articlesDao As IArticlesDao)
```

- les méthodes [getAllArticles] et [getArticleById] s'appuient sur les méthodes de même nom de la couche [dao]
- la méthode [acheter] valide l'achat d'un panier. Cette validation consiste simplement à décrémenter les stocks des articles achetés. L'achat d'un article n'est possible que si son stock le permet. Si ce n'est pas le cas, l'achat est refusé : il reste dans le panier et une erreur est signalée dans la liste [erreurs]. Un achat validé est retiré du panier et le stock de l'article correspondant décrémenté de la quantité achetée.

4.4.6 Génération de l'assembly de la couche [domain]

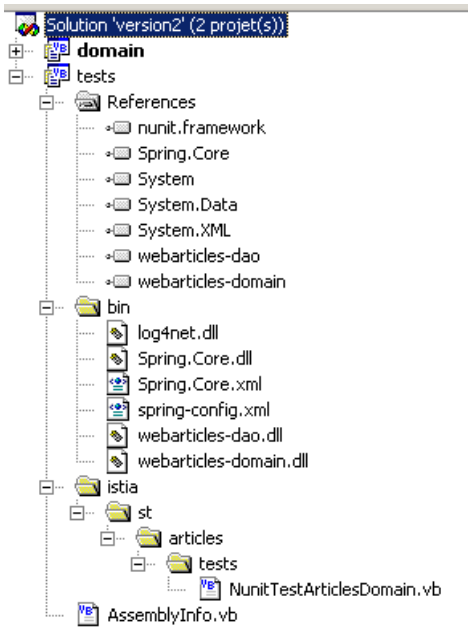
Le projet Visual Studio est configuré pour générer l'assembly [webarticles-domain.dll]. Celui-ci est généré dans le dossier [bin] du projet :





4.4.7 Tests NUnit de la couche [domain]

La structure du projet Visual Studio de test est la suivante :



Commentaires :

- le projet [tests] est de type [bibliothèque de classes]
- les tests [NUnit] nécessitent une référence sur l'assembly [nunit.framework.dll]
- la classe de test [NUnit] récupère une instance de l'objet à tester via Spring. Aussi trouve-t-on
  - dans le dossier [bin], les fichiers de classes de Spring
  - dans [References], une référence à l'assembly [Spring-Core.dll] du dossier [bin]
  - dans [bin], un fichier de configuration pour Spring
- la classe de test a besoin de l'assembly [webarticles-dao.dll] de la couche [dao] et de l'assembly [webarticles-domain.dll] de la couche [domain]. Ceux-ci ont été placés dans le dossier [bin] et leurs références ajoutées aux références du projet.

Une classe de test NUnit de la couche [domain] pourrait être la suivante :

```
Imports NUnit.Framework
Imports istia.st.articles.dao
Imports istia.st.articles.domain
Imports Spring.Objects.Factory.Xml
Imports System.IO

Namespace istia.st.articles.tests

  <TestFixture()> _
  Public Class NunitTestArticlesDomain

    ' l'objet à tester
```

```
    Private articlesDomain As IArticlesDomain
    Private articlesDao As IArticlesDao

    <SetUp()> _
    Public Sub init()
      ' on récupère une instance du fabricant d'objets Spring
      Dim factory As XmlObjectFactory = New XmlObjectFactory(New FileStream("spring-config.xml",
FileMode.Open))
      ' on demande l'instanciation de l'objet articles dao
      articlesDao = CType(factory.GetObject("articlesdao"), IArticlesDao)
      ' puis celui de l'aobjet articlesdomain
      articlesDomain = CType(factory.GetObject("articlesdomain"), IArticlesDomain)
    End Sub

    <Test()> _
    Public Sub getAllArticles()
      ' vérification visuelle
      listArticles()
    End Sub

    <Test()> _
    Public Sub getArticleById()
      ' suppression des articles
      articlesDao.clearAllArticles()
      ' vérification
      Dim articles As IList = articlesDomain.getAllArticles
      Assert.AreEqual(0, articles.Count)
      ' ajout de 2 articles
      articlesDao.ajouteArticle(New Article(3, "article3", 30, 30, 3))
      articlesDao.ajouteArticle(New Article(4, "article4", 40, 40, 4))
      ' vérification
      articles = articlesDomain.getAllArticles
      Assert.AreEqual(2, articles.Count)
      ' recherche article 3
      Dim unArticle As Article = articlesDomain.getArticleById(3)
      ' vérification
      Assert.AreEqual(unArticle.nom, "article3")
      ' recherche article 4
      unArticle = articlesDao.getArticleById(4)
      ' vérification
      Assert.AreEqual(unArticle.nom, "article4")
    End Sub

    <Test()> _
    Public Sub acheterPanier()
      ' suppression des articles
      articlesDao.clearAllArticles()
      ' vérification
      Dim articles As IList = articlesDomain.getAllArticles
      Assert.AreEqual(0, articles.Count)
      ' ajout de 2 articles
      articlesDao.ajouteArticle(New Article(3, "article3", 30, 30, 3))
      articlesDao.ajouteArticle(New Article(4, "article4", 40, 40, 4))
      ' vérification
      articles = articlesDomain.getAllArticles
      Assert.AreEqual(2, articles.Count)
      ' création d'un panier avec deux achats
      Dim panier As New panier
      panier.ajouter(New Achat(New Article(3, "article3", 30, 30, 3), 10))
      panier.ajouter(New Achat(New Article(4, "article4", 40, 40, 4), 10))
      ' vérifications
      Assert.AreEqual(700, panier.totalPanier, 0.000001)
      Assert.AreEqual(2, panier.achats.Count)
      ' validation panier
      articlesDomain.acheter(panier)
      ' vérifications
      Assert.AreEqual(0, articlesDomain.erreurs.Count)
      Assert.AreEqual(0, panier.achats.Count)
      ' recherche article 3
      Dim unArticle As Article = articlesDomain.getArticleById(3)
      ' vérification
      Assert.AreEqual(unArticle.stockactuel, 20)
      ' recherche article 4
      unArticle = articlesDao.getArticleById(4)
      ' vérification
      Assert.AreEqual(unArticle.stockactuel, 30)
      ' nouveau panier
      panier.ajouter(New Achat(New Article(3, "article3", 30, 30, 3), 100))
      ' validation panier
      articlesDomain.acheter(panier)
      ' vérifications
      Assert.AreEqual(1, articlesDomain.erreurs.Count)
      ' recherche article 3
      unArticle = articlesDomain.getArticleById(3)
      ' vérification
      Assert.AreEqual(unArticle.stockactuel, 20)
    End Sub
```

```
    <Test()> _
    Public Sub testRetirerAchats()
      ' suppression du contenu de ARTICLES
      articlesDao.clearAllArticles()
      ' lit la table ARTICLES
      Dim articles As IList = articlesDao.getAllArticles()
      Assert.AreEqual(0, articles.Count)
      ' insertion
      Dim article3 As New Article(3, "article3", 30, 30, 3)
      articlesDao.ajouteArticle(article3)
      Dim article4 As New Article(4, "article4", 40, 40, 4)
      articlesDao.ajouteArticle(article4)
      ' lit la table ARTICLES
      articles = articlesDomain.getAllArticles()
      Assert.AreEqual(2, articles.Count)
      ' création d'un panier avec deux achats
      Dim monPanier As New Panier
      monPanier.ajouter(New Achat(article3, 10))
      monPanier.ajouter(New Achat(article4, 10))
      ' vérifications
      Assert.AreEqual(700.0, monPanier.totalPanier, 0.000001)
      Assert.AreEqual(2, monPanier.achats.Count)
      ' ajouter un article déjà acheté
      monPanier.ajouter(New Achat(article3, 10))
      ' vérifications
      ' le total doit être passé à 1000
      Assert.AreEqual(1000.0, monPanier.totalPanier, 0.000001)
      ' toujours 2 articles dans le panier
      Assert.AreEqual(2, monPanier.achats.Count)
      ' qté article 3 a du passer à 20
      Dim unAchat As Achat = CType(monPanier.achats(0), Achat)
      Assert.AreEqual(20, unAchat.qte)
      ' on retire l'article 3 du panier
      monPanier.enlever(3)
      ' vérifications
      ' le total doit être passé à 400
      Assert.AreEqual(400.0, monPanier.totalPanier, 0.000001)
      ' 1 seul article dans le panier
      Assert.AreEqual(1, monPanier.achats.Count)
      ' ce doit être l'article n° 4
      Assert.AreEqual(4, CType(monPanier.achats(0), Achat).article.id)
    End Sub

    ' listing écran
    Private Sub listArticles()
      Dim articles As IList = articlesDomain.getAllArticles
      For i As Integer = 0 To articles.Count - 1
        Console.WriteLine(CType(articles(i), Article).ToString)
      Next
    End Sub

  End Class


End Namespace
```

Commentaires :

- on a voulu écrire un programme de test de l'interface [IArticlesDomain] qui soit indépendant de la classe d'implémentation de celle-ci. Aussi avons-nous utilisé Spring pour cacher au programme de test le nom de la classe d'implémentation.
- la méthode d'attribut <Setup()> récupère auprès de Spring une référence sur les objets [articlesdomain] et [articlesdao] à tester. Ceux-ci sont définis dans le fichier [spring-config.xml] suivant :

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<!DOCTYPE objects PUBLIC "-//SPRING//DTD OBJECT//EN"
"http://www.springframework.net/dtd/spring-objects.dtd">
<objects>
  <object id="articlesdao" type="istia.st.articles.dao.ArticlesDaoArrayList, webarticles-dao" />
  <object id="articlesdomain" type="istia.st.articles.domain.AchatsArticles, webarticles-domain">
    <constructor-arg index="0">
      <ref object="articlesdao" />
    </constructor-arg>
  </object>
</objects>
```

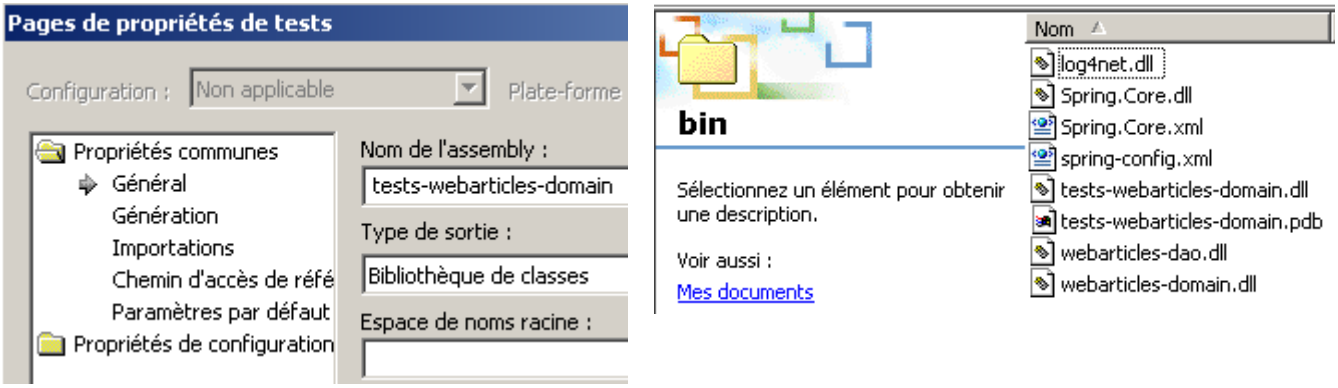
Ce fichier indique

- pour le singleton [articlesdao], le nom de la classe d'implémentation [istia.st.articles.dao.ArticlesDaoArrayList] et où la trouver [webarticles-dao.dll]. L'instanciation ne nécessitant pas de paramètres, auncun n'est défini ici.

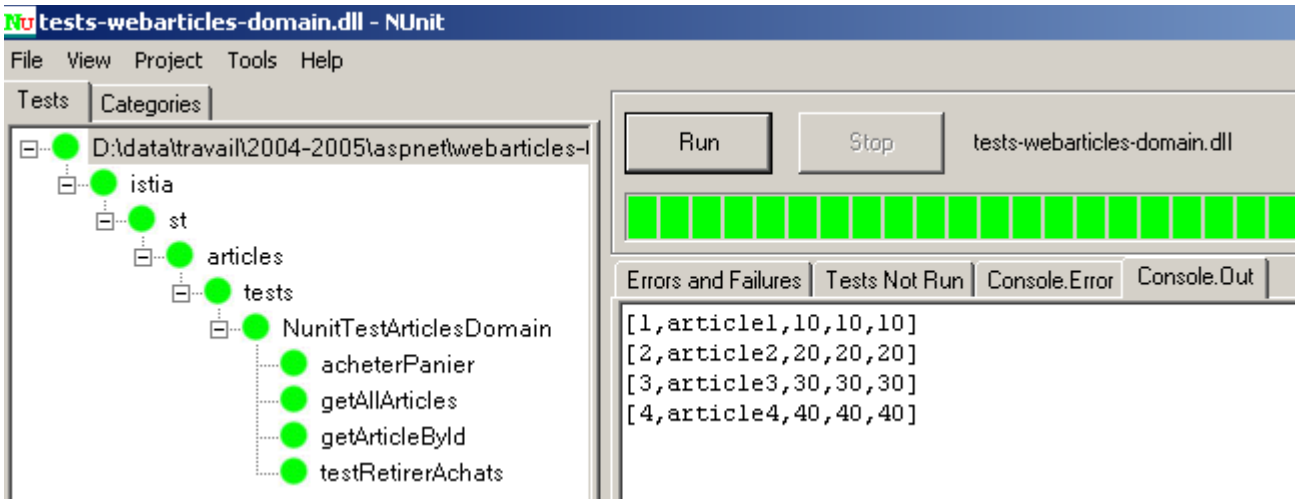
- pour le singleton [articlesdomain], le nom de la classe d'implémentation [istia.st.articles.domain.AchatsArticles] et où la trouver [webarticles-domain.dll]. La classe [AchatsArticles] a un constructeur à un paramètre : le singleton gérant l'accès à la couche [dao]. Ici, celui-ci est défini comme étant le singleton [articlesdao] défini précédemment.

- la classe de test obtient une instance de la classe à tester [articlesdomain] ainsi qu'une instance de la classe d'accès aux données [articlesdao]. Ce dernier point est litigieux. La classe de test ne devrait théoriquement pas avoir besoin d'avoir accès à la couche [dao] qu'elle n'est même pas censée connaître. Ici, nous sommes passés outre cette " éthique " qui, pour être respectée, nous aurait obliger à créer de nouvelles méthodes dans notre interface [IArticlesDomain].

Pour tester la couche [domain], nous générons la DLL [tests-webarticles-domain.dll] dans le dossier [bin] du projet [tests] :



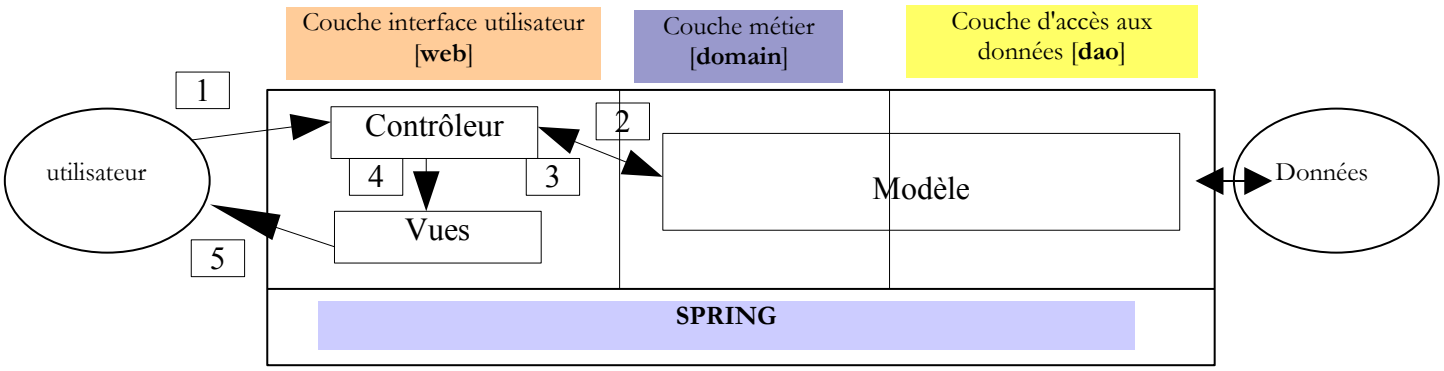
Puis, à l'aide de l'application [Nunit-Gui], nous chargeons cette DLL et exécutons les tests :



Le lecteur qui visualise ce document sur écran pourra voir que tous les tests ont été réussis. Par la suite, nous considèrerons que nous avons une couche [domain] opérationnelle.

4.5 Conclusion

Rappelons que nous voulons construire l'application web à trois couches suivante :

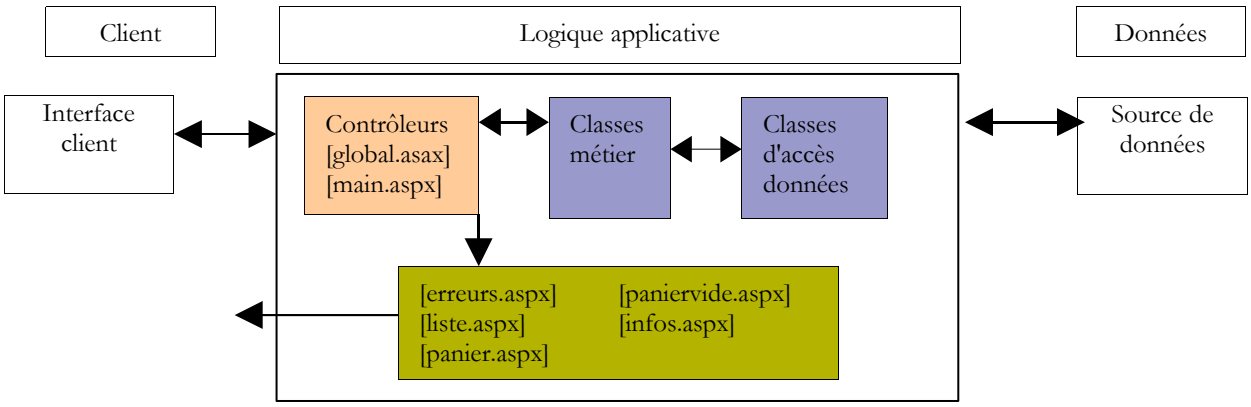


Le modèle M de notre application MVC est désormais écrit et testé. Il nous est fourni dans deux DLL [webarticles-dao.dll, webarticles-domain.dll]. Nous pouvons passer à la dernière couche, la couche [web] qui contient le contrôleur C et les vues V. Nous considèrerons tout d'abord, une méthode présentée dans le document [http://tahe.developpez.com/dotnet/aspnet/vol1/]

- le contrôleur C est assuré par deux fichiers [global.asax, main.aspx]
- les vues V sont assurées par des pages **aspx**

5 La couche [web]

L'architecture MVC de l'application web sera la suivante :



M=modèle	les classes métier [domain], les classes d'accès aux données [dao] et la source de données
V=vues	les pages ASPX
C=contrôleur	toutes les requêtes des clients HTTP transitent par les deux contrôleurs suivants : **global.asax** : gère les évts liés au lancement initial de l'application **main.aspx** : traite individuellement la requête de chaque client

5.1 Le modèle

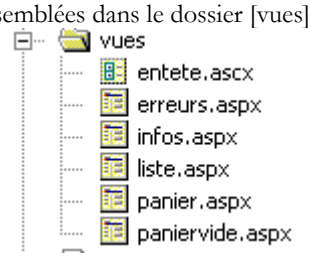
Il a été présenté précédemment. Il est constitué des DLL [webarticles-dao.dll, webarticles-domain.dll].

5.2 Les vues

Les vues correspondent à celles qui ont été présentées en début de document :

LISTE	liste.aspx
INFOS	infos.aspx
PANIER	panier.aspx
PANIERVIDE	paniervide.aspx
ERREURS	erreurs.aspx

Les vues sont rassemblées dans le dossier [vues] de l'application



5.3 Les contrôleurs

Comme il a été indiqué, le contrôleur sera formé de deux éléments:

1. [global.asax,global.asax.vb] : utilisé principalement pour initialiser l'application et mettre dans le contexte de celle-ci toutes les données à partager entre les différents clients
2. [main.aspx, main.aspx.vb] : le véritable contrôleur, celui qui traite les requêtes HTTP des clients.

Les différentes requêtes des clients seront adressées au contrôleur [main.aspx] et contiendront un paramètre appelé [action] précisant l'action demandée par le client :

requête	signification	action du contrôleur	réponses possibles
action=liste	le client veut la liste des articles	- demande la liste des articles à la couche métier	- [LISTE] - [ERREURS]
action=infos	le client demande des informations sur l'un des articles affichés dans la vue [LISTE]	- demande l'article à la couche métier	- [INFOS] - [ERREURS]
action=achat	le client achète un article	- demande l'article à la couche métier et l'intègre dans le panier du client	- [INFOS] si erreur de qté - [LISTE] si pas d'erreur
action=retirerachat	le client veut supprimer un achat de son panier	- récupère le panier dans la session et le modifie	- [PANIER] - [PANIERVIDE] - [ERREURS]
action=panier	le client veut visualiser son panier	- récupère le panier dans la session	- [PANIER] - [PANIERVIDE] - [ERREURS]
action=validationpanier	le client a terminé ses achats et passe à la phase paiement	- met à jour dans la base les stocks des articles achetés - vide le panier du client des articles dont l'achat a été validé	- [LISTE] - [ERREURS]

5.4 Configuration de l'application

Nous chercherons à configurer l'application de façon à la rendre la plus souple possible vis à vis de changements tels que :

1. le changement des url des différentes vues
2. le changement des classes implémentant les interfaces [IArticlesDao] et [IArticlesDomain]
3. le changement du SGBD, de la base, de la table des articles

5.4.1 Les changements d'url

Les noms des url des vues seront placés dans le fichier [web.config] de configuration de l'application avec quelques autres paramètres :

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
..
```

```
  <appSettings>
    <add key="urlMain" value="/webarticles/main.aspx"/>
    <add key="urlInfos" value="vues/infos.aspx"/>
    <add key="urlErreurs" value="vues/erreurs.aspx"/>
    <add key="urlListe" value="vues/liste.aspx"/>
    <add key="urlPanier" value="vues/panier.aspx"/>
    <add key="urlPanierVide" value="vues/paniervide.aspx"/>
  </appSettings>
</configuration>
```

5.4.2 Le changement des classes d'implémentation des interfaces

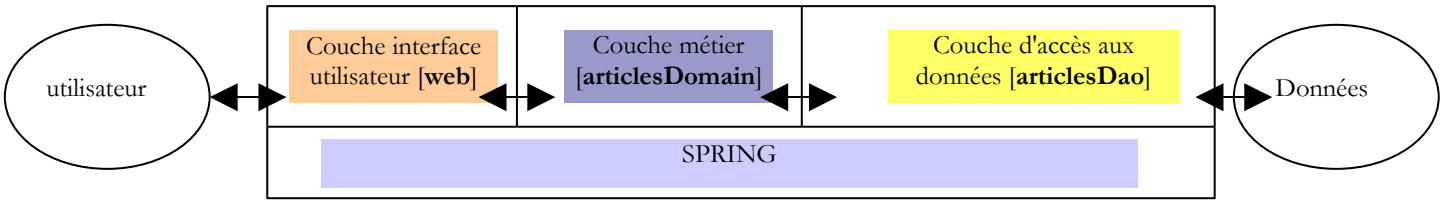
Dans l'esprit des architectures à trois couches, les couches doivent être étanches les unes par rapport aux autres. Cette étanchéité est obtenue de la façon suivante :

- les couches communiquent entre-elles par des interfaces et non par des classes concrètes
- le code d'une couche n'instancie jamais elle-même la classe d'une autre couche afin de l'utiliser. Elle demande simplement à un outil externe, ici **Spring**, une instance d'implémentation de l'interface de la couche qu'elle veut utiliser. Pour cela, nous savons qu'elle n'a pas besoin de connaître le nom de la classe d'implémentation mais seulement le nom du singleton Spring dont elle veut une référence.

Dans notre application, Spring sera configuré dans le fichier [web.config] de l'application web de la façon suivante :

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<configuration>
  <configSections>
    <sectionGroup name="spring">
      <section name="context" type="Spring.Context.Support.ContextHandler, Spring.Core" />
      <section name="objects" type="Spring.Context.Support.DefaultSectionHandler, Spring.Core" />
    </sectionGroup>
  </configSections>
  <spring>
    <context type="Spring.Context.Support.XmlApplicationContext, Spring.Core">
      <resource uri="config://spring/objects" />
    </context>
    <objects>
  <object id="articlesDao" type="istia.st.articles.dao.ArticlesDaoArrayList, webarticles-dao" />
  <object id="articlesDomain" type="istia.st.articles.domain.AchatsArticles, webarticles-domain">
    <constructor-arg index="0">
      <ref object="articlesDao" />
    </constructor-arg>
  </object>
    </objects>
  </spring>
    <appSettings>
    <add key="urlMain" value="/webarticles/main.aspx"/>
    <add key="urlInfos" value="vues/infos.aspx"/>
    <add key="urlErreurs" value="vues/erreurs.aspx"/>
    <add key="urlListe" value="vues/liste.aspx"/>
    <add key="urlPanier" value="vues/panier.aspx"/>
    <add key="urlPanierVide" value="vues/paniervide.aspx"/>
  </appSettings>
</configuration>
```

Pour avoir accès à la couche [métier], une classe de la couche [web] pourra demander le singleton [articlesDomain]. Spring instanciera alors un objet de type [istia.st.articles.domain.AchatsArticles]. Pour cette instanciation, il a besoin d'un objet de type [articlesDao], c'est à dire d'un objet de type [istia.st.articles.dao.ArticlesDaoArrayList]. Spring instanciera alors un tel objet. A la fin de l'opération, la couche [web] qui a demandé le singleton [articlesDomain] a toute la chaîne qui la relie à la source de données :



5.4.3 Les changements liés au SGBD ou à la base des données

Ce point sera ignoré ici puisque nous nous situons dans une application de test sans SGBD. Nous aborderons dans un deuxième temps l'implémentation d'une couche [dao] s'appuyant sur un SGBD.

5.5 La bibliothèque de balises <asp:>

Considérons la vue [ERREURS] qui affiche une liste d'erreurs :

La vue [ERREURS] est chargée d'afficher une liste d'erreurs que le contrôleur [main.aspx] a placée dans le contexte de la requête sous le nom [context.Items("erreurs")]. Il y a plusieurs façons d'écrire une telle page. Nous ne nous intéressons ici qu'à la partie affichage des erreurs.

Rappelons qu'une page ASPX a une partie présentation HTML et une partie code .NET qui prépare les données que la partie présentation doit afficher. Ces deux parties peuvent être dans un même fichier [aspx] (solution WebMatrx) ou dans deux fichiers : [aspx] pour la présentation, [aspx.vb] pour le code. Cette dernière solution est celle de Visual Studio. Pour compliquer les choses, la partie présentation HTML peut comporter, elle aussi, du code .NET, ce qui tend à brouiller la séparation [contrôleur] et [présentation] de la vue. Cette solution est généralement vivement déconseillée. Retirer tout code de la partie [présentation] a nécessité la création de bibliothèques de balises. Celles-ci "cachent" le code sous l'apparence de balises analogues aux balises HTML. Nous présentons deux solutions possibles pour la page [ERREURS].

Notre première solution utilise du code .NET dans la partie [présentation] de la page. La page ASPX récupère la liste des erreurs présente dans la requête dans sa partie contrôleur [erreurs.aspx.vb] :

```
   Protected erreurs As ArrayList

   Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
...
     ' on récupère les erreurs
     erreurs = CType(context.Items("erreurs"), ArrayList)
   End Sub
```

puis l'affiche dans la partie [présentation, erreurs.aspx] :

```
      <h2>Les erreurs suivantes se sont produites :</h2>
      <ul>
      <%
        for i as integer=0 to erreurs.count-1
          response.write("<li>" & erreurs(i).ToString & "</li>")
        next
      %>
      </ul>
```

La seconde solution utilise la balise <asp:repeater> de la bibliothèque de balises <asp:> d'ASP.NET. Si on construit une page ASPX graphiquement, cette balise est disponible sous la forme d'un composant serveur qu'on dépose sur le formulaire de conception. Si on construit le code ASPX à la main, on peut parler de bibliothèque de balises.

Avec la bibliothèque de balises <asp:>, le code ASPX de la vue [ERREURS] précédente devient le suivant :

```
   <asp:Repeater id="rptErreurs" runat="server">
     <HeaderTemplate>
       <h3>Les erreurs suivantes se sont produites :
       </h3>
       <ul>
     </HeaderTemplate>
     <ItemTemplate>
       <li>
         <%# Container.DataItem %>
       </li>
     </ItemTemplate>
     <FooterTemplate>
       </ul>
     </FooterTemplate>
   </asp:Repeater>
```

La balise

```
<asp:Repeater id="rptErreurs" runat="server">
```

sert à répéter un motif HTML sur les différents éléments d'une source de données. Ses différents éléments sont les suivants :

`HeaderTemplate`	le motif HTML à afficher avant que les éléments de la source de données ne soient affichés
`ItemTemplate`	le motif HTML à répéter pour chacun des éléments de la source de données. L'expression [<%# Container.DataItem %>] sert à afficher la valeur de l'élément courant de la source de données
`FooterTemplate`	le motif HTML à afficher après que les éléments de la source de données ont été affichés

La source de données est liée à la balise généralement dans la partie [contrôleur] de la page :

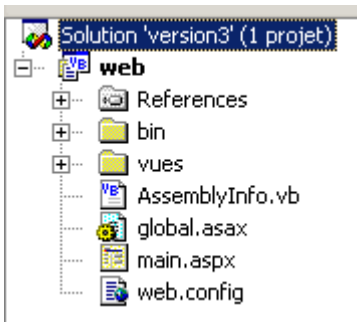
```
Protected WithEvents rptErreurs As System.Web.UI.WebControls.Repeater

Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
..
    ' on lie les erreurs à rptErreurs
    With rptErreurs
      .DataSource = context.Items("erreurs")
      .DataBind()
    End With
End Sub
```

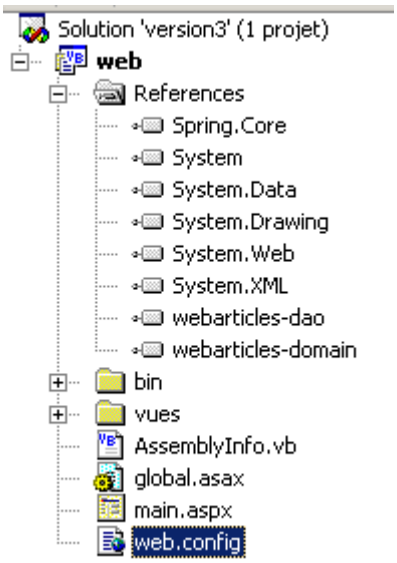
Cette liaison peut se faire également au moment de la conception de la page si la source de données est déjà connue, une base de données existante par exemple.

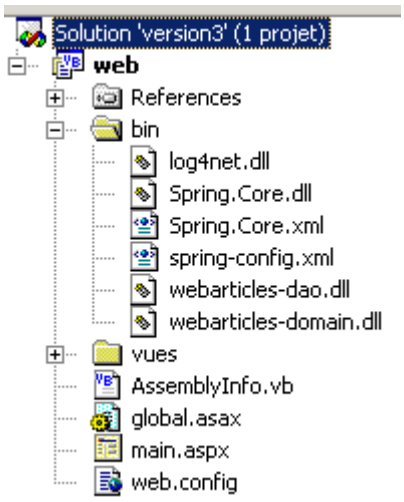
Nous utiliserons dans nos vues une autre balise : <asp:datagrid> qui permet d'afficher une source de données sous forme d'un tableau.

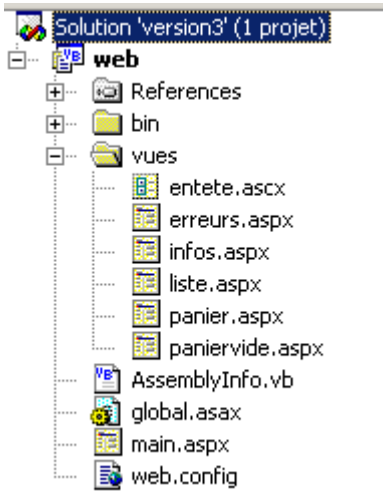
5.6 Structure de la solution Visual Studio de l'application [webarticles]

Une application web est un puzzle avec de nombreux éléments. Lui donner une architecture MVC augmente en général le nombre de ceux-ci. La structure de l'application [webarticles] sous [Visual Studio] est la suivante :









Commentaires :

- le projet [web] est de type [bibliothèque de classes] et non de type [Application web ASP.NET] comme on pourrait logiquement s'y attendre. Le type [Application web ASP.NET] exige la présence du serveur web IIS sur la machine de développement ou sur une machine distante. Le serveur IIS n'existe pas en standard sur les machines Windows XP Familial. Or beaucoup de PC sont vendus avec cette version. Afin de permettre aux lecteurs disposant de Windows XP de mettre en oeuvre l'application étudiée, nous utiliserons le serveur Web Cassini (voir annexe) disponible gratuitement chez Microsoft et nous remplacerons le projet [Application web ASP.NET] par un projet [bibliothèque de classes]. Cela entraîne quelques inconvénients qui sont expliqués en annexes.

- les DLL utilisées par l'application sont les suivantes :

`webarticles-dao.dll`	rassemble les classes de la couche d'accès aux données
`webarticles-domain.dll`	rassemble les classes de la couche métier
`Spring.Core.dll`	contient les classes Spring qui nous permettent d'intégrer les couches web, domain et dao
`log4net.dll`	classes de logs - utilisées par Spring

Ces DLL sont placées dans le dossier [bin] et ajoutées aux références du projet.

5.7 Les vues ASPX

Comme il a été conseillé précédemment, nous utiliserons la bibliothèque de balises <asp:> dans nos vues ASPX.

5.7.1 Le composant utilisateur [entete.ascx]

Afin de donner une certaine homogénéité aux différentes vues, celles-ci partageront un même entête, celui qui affiche le nom de l'application avec le menu :

Le menu est dynamique et fixé par le contrôleur. Celui-ci met dans la requête transmise à la page ASPX, un attribut de clé "actions" ayant pour valeur associée, un tableau d'éléments de type **Hashtable()**. Chaque élément de ce tableau est un dictionnaire destiné à générer une option du menu de l'entête. Chaque dictionnaire a deux clés :

- **href** : l'url associée à l'option de menu
- **lien** : le texte du menu

On fera de l'entête un **contrôle utilisateur**. Un contrôle utilisateur encapsule un morceau de page (présentation et code associé) dans un composant réutilisable ensuite dans d'autres pages. Ici, nous voulons réutiliser le composant [entete] dans les autres vues de

l'application. Le code de présentation sera dans [entete.ascx] et le code de contrôle associé dans [entete.ascx.vb]. Le code de présentation utilisera un composant <asp:repeater> pour afficher le tableau des options du menu :

n°	*type*	*nom*	*rôle*
1	repeater	rptMenu source de données : un tableau de dictionnaires à deux clés : href, lien	afficher les options de menu

Le code de présentation de la page sera le suivant :

```
1. <%@ Control codebehind="entete.ascx.vb" Language="vb" autoeventwireup="false"
   inherits="istia.st.articles.web.EnteteWebArticles" %>
2.    <table>
3.      <tr>
4.        <td>
5.          <h2>Magasin virtuel</h2></td>
6.        <asp:Repeater id="rptMenu" runat="server">
7.          <ItemTemplate>
8.            <td>
9.              |<a href='<%# Container.DataItem("href") %>'>
10.               <%# Container.DataItem("lien") %>
11.             </a>
12.           </td>
13.         </ItemTemplate>
14.       </asp:Repeater>
15.     </tr>
16.   </table>
17.   <hr>
```

Commentaires :

- le composant [repeater] est défini lignes 6-14
- chaque élément de la source de données associée au répéteur est un dictionnaire à deux clés : **href** - ligne 9 et **lien** - ligne 10

Le code de contrôle associé sera le suivant :

```
1. Namespace istia.st.articles.web
2.  Public Class EnteteWebArticles
3.    Inherits System.Web.UI.UserControl
4.
5.    Protected WithEvents rptMenu As System.Web.UI.WebControls.Repeater
6.
7.    Public WriteOnly Property actions() As Hashtable()
8.      Set(ByVal Value As Hashtable())
9.        ' on associe le tableau des actions à son composant
10.       With rptMenu
11.         .DataSource = Value
12.         .DataBind()
13.       End With
14.     End Set
15.   End Property
16. End Class
17.End Namespace
```

Commentaires :

- le composant de type [EnteteWebArticles] a une propriété publique [actions] en écriture seule - ligne 7
- cette propriété permet d'associer au composant <asp:repeater> appelé [rptMenu] - ligne 10 - le tableau des options calculé par le contrôleur de l'application - lignes 11-12.

Les autres vues de l'application utiliseront l'entête défini par [entete.ascx]. La page [erreurs.aspx] par exemple, incluera l'entête à l'aide du code suivant :

```
1. <%@ Register TagPrefix="WA" TagName="entete" Src="entete.ascx" %>
2. <%@ Page inherits="istia.st.articles.web.ErreursWebarticles" autoeventwireup="false" Language="vb" %>
3. <HTML>
4.  <HEAD>
5.    <TITLE>webarticles</TITLE>
6.    <META http-equiv="Content-Type" content="text/html; charset=windows-1252">
7.  </HEAD>
8.  <body>
```

```
9.    <WA:entete id="entete" runat="server"></WA:entete>
10.   <h3>Les erreurs suivantes se sont produites :
11.   </h3>
```

Commentaires :

- La ligne 1 déclare que la balise <WA:entete> devra être associée au composant défini par le fichier [entete.ascx]. Les attributs [TagPrefix] et [TagName] sont libres.
- Ceci fait, l'insertion du composant dans le code de présentation de la page se fait avec la ligne 9. A l'exécution, cette balise aura pour effet d'inclure dans le code de la page ASPX qui la contient, celui de la page [entete.ascx]. Le code de contrôle [erreurs.aspx.vb] prendra soin d'initialiser ce composant. Il peut le faire de la façon suivante :

```
1.  Public Class ErreursWebarticles
2.    Inherits System.Web.UI.Page
3.
4.    ' composants de la page
5. ...
6.    Protected WithEvents entete As New EnteteWebArticles
7.
8.    Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
   MyBase.Load
9. ...
10.     ' on lie les options de menu à rptmenu
11.     entete.actions = CType(context.Items("options"), Hashtable())
12....
13.   End Sub
14. End Class
```

Commentaires :

- la ligne 6 crée un objet de type [EnteteWebArticles] qui est le type du composant créé
- la ligne 11 initialise la propriété [actions] de cet objet

5.7.2 La vue [liste.aspx]

5.7.2.1 Introduction

Cette vue affiche la liste des articles disponibles à la vente :

Elle est affichée à la suite d'une requête **/main?action=liste** ou **/main?action=validationpanier**. Les éléments de la requête du contrôleur sont les suivants :

`actions`	objet Hashtable() - le tableau des options du menu
`listarticles`	ArrayList d'objets de type [Article]
`message`	objet String - message à afficher en bas de page

Chaque lien [Infos] du tableau HTML des articles a une url de la forme [?action=infos&id=ID] où ID est le champ id de l'article affiché.

5.7.2.2 Composants de la page

n°	type	nom	rôle
1	composant utilisateur	entete	afficher l'entête
2	DataGrid	DataGridArticles 3 - colonne connexe : entête : Nom, champ : nom 4 - colonne connexe : entête : Prix, champ : prix 5 - colonne hypertexte : texte : Infos, champ Url : id, format URL : /webarticles/main.aspx?action=infos&id={0}	afficher les articles en vente
6	label	lblMessage	afficher un message

Rappelons comment procéder pour définir ces propriétés :

- dans Visual Studio, on sélectionne le [DataGrid] pour avoir accès à sa feuille de propriétés :

- on utilise le lien [Mise en forme automatique] ci-dessus pour gérer la forme du tableau affiché
- et le lien [Générateur de propriétés] pour gérer son contenu

5.7.2.3 Code de présentation [liste.aspx]

```
1. <%@ Page codebehind="liste.aspx.vb" inherits="istia.st.articles.web.ListeWebarticles"
   autoeventwireup="false" Language="vb" %>
2. <%@ Register TagPrefix="WA" TagName="entete" Src="entete.ascx"%>
3. <HTML>
4.  <HEAD>
5.    <TITLE>webarticles</TITLE>
6.    <META http-equiv="Content-Type" content="text/html; charset=windows-1252">
7.  </HEAD>
8.  <body>
9.    <WA:entete id="entete" runat="server"></WA:entete>
10.   <h2>Liste des articles</h2>
11.   <P>
12.     <asp:DataGrid id="DataGridArticles" runat="server" ForeColor="Black"
   BackColor="LightGoldenrodYellow"
13.       BorderColor="Tan" CellPadding="2" BorderWidth="1px" GridLines="None" AutoGenerateColumns="False">
```

```
14.      <SelectedItemStyle ForeColor="GhostWhite" BackColor="DarkSlateBlue"></SelectedItemStyle>
15.      <AlternatingItemStyle BackColor="PaleGoldenrod"></AlternatingItemStyle>
16.      <HeaderStyle Font-Bold="True" BackColor="Tan"></HeaderStyle>
17.      <FooterStyle BackColor="Tan"></FooterStyle>
18.      <Columns>
19.        <asp:BoundColumn DataField="nom" HeaderText="Nom"></asp:BoundColumn>
20.        <asp:BoundColumn DataField="prix" HeaderText="Prix" DataFormatString="{0:C}"></asp:BoundColumn>
21.        <asp:HyperLinkColumn Text="Infos" DataNavigateUrlField="id"
   DataNavigateUrlFormatString="/webarticles/main.aspx?action=infos&amp;id={0}"></asp:HyperLinkColumn>
22.      </Columns>
23.      <PagerStyle HorizontalAlign="Center" ForeColor="DarkSlateBlue"
   BackColor="PaleGoldenrod"></PagerStyle>
24.    </asp:DataGrid></P>
25.  <P>
26.    <asp:Label id="lblMessage" runat="server" BackColor="#FFC080"></asp:Label></P>
27. </body>
28.</HTML>
```

Commentaires :

- la ligne 9 définit l'entête de la page
- les lignes 12-24 définissent les caractéristiques du [DataGrid]
- la ligne 26 définit le label [lblMessage]

5.7.2.4 Code du contrôleur [liste.aspx.vb]

```
1. Imports System
2. Imports System.Collections
3. Imports System.Data
4. Imports istia.st.articles.dao
5.
6. Namespace istia.st.articles.web
7.
8.  ' gère la page d'affichage de la liste d'articles
9.  Public Class ListeWebarticles
10.  Inherits System.Web.UI.Page
11.
12.  ' composants de la page
13.  Protected WithEvents lblMessage As System.Web.UI.WebControls.Label
14.  Protected WithEvents DataGridArticles As System.Web.UI.WebControls.DataGrid
15.  Protected WithEvents entete As New EnteteWebArticles
16.
17.  Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
   MyBase.Load
18.    ' on prépare la vue [liste] à partir des informations du contexte
19.    ' on lie les options de menu à rptmenu
20.    entete.actions = CType(context.Items("options"), Hashtable())
21.    ' on récupère les articles dans un datatable
22.    Dim articles As ArrayList = CType(context.Items("articles"), ArrayList)
23.    ' on les lie au composant [DataGrid] de la page
24.    With DataGridArticles
25.      .DataSource = articles
26.      .DataBind()
27.    End With
28.    ' on affiche le message
29.    lblMessage.Text = context.Items("message").ToString
30.  End Sub
31. End Class
32.End Namespace
```

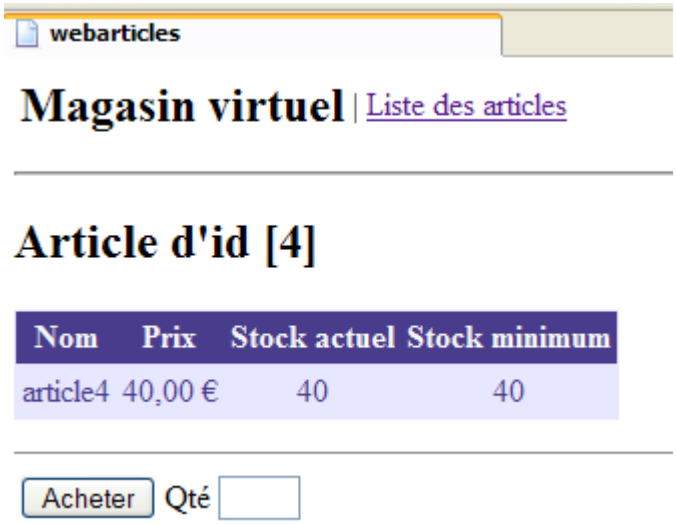
Commentaires :

- les composants de la page apparaissent lignes 13-15. A noter qu'on a eu besoin de créer un objet [EnteteWebArticles] avec un opérateur [new] alors que cela a été inutile avec les autres composants. Sans cette création explicite, on avait une erreur d'exécution indiquant que l'objet [entete] ne référençait rien. Ce point mériterait d'être creusé. Il ne l'a pas été.
- le tableau des options du menu de l'entête est pris dans le contexte pour initialiser le composant [entete] de la page - ligne 20
- la liste des articles est prise dans le contexte - ligne 22
- pour initialiser le composant [DataGridArticles] - lignes 24-27
- le composant [lblMessage] est initialisé avec un message placé dans le contexte - ligne 29

5.7.3 La vue [infos.aspx]

5.7.3.1 Introduction

Cette vue affiche des informations sur un article et permet également son achat :



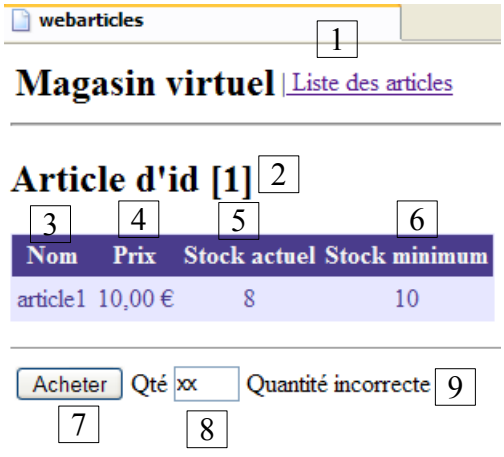
Elle est affichée à la suite d'une requête **/main?action=infos&id=ID** ou d'une requête **/main?action=achat&id=ID** lorsque la quantité achetée est erronée. Les éléments de la requête du contrôleur sont les suivants :

`actions`	objet Hashtable() - le tableau des options du menu
`article`	objet de type [Article] - article à afficher
`msg`	objet String - message à afficher en cas d'erreur sur la quantité
`qte`	objet String - valeur à afficher dans le champ de saisie [Qte]

Les champs [msg] et [qte] sont utilisés en cas d'erreur de saisie sur la quantité :

Cette page contient un formulaire qui est posté par le bouton [Acheter]. L'url cible du POST est [?action=achat&id=ID] où ID est l'id de l'article acheté.

5.7.3.2 Les composants de la page



n°	type	nom	rôle
1	composant utilisateur	**entete**	afficher l'entête
2	literal	**litID**	afficher le n° de l'article
3 à 6	DataGrid	**DataGridArticle** 3 - colonne connexe : entête : Nom, champ : nom 4 - colonne connexe : entête : Prix, champ : prix 5 - colonne connexe : entête : Stock actuel, champ : stockActuel 6 - colonne connexe : entête : Stock minimum, champ : stockMinimum	afficher un article
7	HTML Submit		poster le formulaire
8	HTML Input runat=server	**txtQte**	saisir la quantité achetée
9	label	**lblMsgQte**	message d'erreur éventuel

5.7.3.3 Le code de présentation [infos.aspx]

```
1. <%@ Register TagPrefix="WA" TagName="entete" Src="entete.ascx" %>
2. <%@ Page codebehind="infos.aspx.vb" inherits="istia.st.articles.web.InfosWebarticles"
   autoeventwireup="false" Language="vb" %>
3. <HTML>
4.  <HEAD>
5.    <TITLE>webarticles</TITLE>
6.    <META http-equiv="Content-Type" content="text/html; charset=windows-1252">
7.  </HEAD>
8.  <body>
9.    <WA:entete id="entete" runat="server"></WA:entete>
10.   <h2>Article d'id [<asp:Literal id="litId" runat="server"></asp:Literal>]</h2>
11.   <P>
12.     <asp:DataGrid id="DataGridArticle" runat="server" BackColor="White" BorderColor="#E7E7FF"
   CellPadding="3"
13.       BorderWidth="1px" BorderStyle="None" GridLines="Horizontal" AutoGenerateColumns="False">
14.       <SelectedItemStyle Font-Bold="True" ForeColor="#F7F7F7" BackColor="#738A9C"></SelectedItemStyle>
15.       <AlternatingItemStyle BackColor="#F7F7F7"></AlternatingItemStyle>
16.       <ItemStyle HorizontalAlign="Center" ForeColor="#4A3C8C" BackColor="#E7E7FF"></ItemStyle>
17.       <HeaderStyle Font-Bold="True" HorizontalAlign="Center" ForeColor="#F7F7F7"
   BackColor="#4A3C8C"></HeaderStyle>
18.       <FooterStyle ForeColor="#4A3C8C" BackColor="#B5C7DE"></FooterStyle>
19.       <Columns>
20.         <asp:BoundColumn DataField="nom" HeaderText="Nom">
21.           <HeaderStyle HorizontalAlign="Center"></HeaderStyle>
22.         </asp:BoundColumn>
23.         <asp:BoundColumn DataField="prix" HeaderText="Prix" DataFormatString="{0:C}">
24.           <HeaderStyle HorizontalAlign="Center"></HeaderStyle>
25.         </asp:BoundColumn>
26.         <asp:BoundColumn DataField="stockactuel" HeaderText="Stock actuel">
27.           <HeaderStyle HorizontalAlign="Center"></HeaderStyle>
28.         </asp:BoundColumn>
29.         <asp:BoundColumn DataField="stockminimum" HeaderText="Stock minimum">
30.           <HeaderStyle HorizontalAlign="Center"></HeaderStyle>
31.         </asp:BoundColumn>
32.       </Columns>
33.       <PagerStyle HorizontalAlign="Right" ForeColor="#4A3C8C" BackColor="#E7E7FF"
   Mode="NumericPages"></PagerStyle>
34.     </asp:DataGrid></P>
35.   <HR width="100%" SIZE="1">
36.   <form method="post" action="<%=strAction%>">
37.     <table>
38.       <tr>
39.         <td><input type="submit" value="Acheter"></td>
40.         <td>Qté</td>
41.         <td><INPUT type="text" maxLength="3" size="3" id="txtQte" runat="server"></td>
42.         <td><asp:Label id="lblMsgQte" runat="server" />
43.         </td>
44.       </tr>
45.     </table>
46.   </form>
47. </body>
48.</HTML>
```

Commentaires :

- l'entête est inclus dans la page - ligne 9
- le litéral [litId] est défini ligne 10
- le DataGrid [DataGridArticles] est défini lignes 12-34
- le formulaire est défini lignes 36-46. Il a de type POST.
- la cible du POST est fourni par une variable [strAction] - ligne 36. Cette variable devra être définie par le contrôleur.

- le champ de saisie de la quantité achetée est définie ligne 41. C'est un composant HTML serveur (runat=server). Côté code, on y a accès via un objet.
- la ligne 42 définit le label [lblMsgQte] qui contiendra un éventuel message d'erreur sur la quantité saisie

5.7.3.4 Le code de contrôle [infos.aspx.vb]

```
1. Imports istia.st.articles.dao
2. Imports System
3. Imports System.Collections
4.
5. Namespace istia.st.articles.web
6.
7.  ' gère la page d'informations sur un article
8.  Public Class InfosWebarticles
9.    Inherits System.Web.UI.Page
10.   Protected WithEvents lblMsgQte As System.Web.UI.WebControls.Label
11.   Protected WithEvents litId As System.Web.UI.WebControls.Literal
12.   Protected WithEvents txtQte As System.Web.UI.HtmlControls.HtmlInputText
13.   Protected WithEvents DataGridArticle As System.Web.UI.WebControls.DataGrid
14.   Protected WithEvents entete As New EnteteWebArticles
15.
16.   ' l'url où sera postée le formulaire
17.   Private _strAction As String
18.   Public Property strAction() As String
19.     Get
20.       Return _strAction
21.     End Get
22.     Set(ByVal Value As String)
23.       _strAction = Value
24.     End Set
25.   End Property
26.
27.   ' affichage page
28.   Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
    MyBase.Load
29.     ' on récupère les infos de la requête
30.     Dim unArticle As Article = CType(Session.Item("article"), Article)
31.     ' on lie les options de menu à rptmenu
32.     entete.actions = CType(context.Items("options"), Hashtable())
33.     ' on lie l'article aux [DataGrid]
34.     Dim articles As New ArrayList
35.     articles.Add(unArticle)
36.     With DataGridArticle
37.       .DataSource = articles
38.       .DataBind()
39.     End With
40.     ' le label id
41.     litId.Text = unArticle.id.ToString
42.     ' le message d'erreur
43.     lblMsgQte.Text = context.Items("msg").ToString
44.     ' la qté précédente
45.     txtQte.Value = context.Items("qte").ToString
46.     ' l'url action
47.     strAction = "?action=achat&id=" + unArticle.id.ToString
48.   End Sub
49.End Namespace
```

Commentaires :

- les composants de la page sont définis lignes 10-14
- la classe définit une propriété publique [strAction] qui sert à définir la cible du POST du formulaire - lignes 17-25
- l'article à afficher est récupéré dans le contexte de l'application - ligne 30
- le tableau des options du menu de l'entête est pris dans le contexte pour initialiser le composant [entete] de la page - ligne 32
- lignes 33-39, le composant [DataGridArticle] est liée à une source de données de type [ArrayList] ne contenant que l'article récupéré ligne 30
- les composants [lblMsgQte, txtQte] sont initialisés avec des informations prises dans le contexte - lignes 42-45
- la propriété [straction] est également initialisée avec une information prise dans le contexte - ligne 47. Cette variable sert à générer l'attribut [action] du formulaire HTML présent dans la page :

```
    <form method="post" action="<%=strAction%>">
....
    </form>
```

5.7.4 La vue [panier.aspx]

5.7.4.1 Introduction

Cette vue affiche le contenu du panier :

Elle est affichée à la suite d'une requête **/main?action=panier** ou **/main?action=retirerachat&id=ID**. Les éléments de la requête du contrôleur sont les suivants :

`actions`	objet Hashtable() - le tableau des options du menu
`panier`	objet de type [Panier] - le panier à afficher

Chaque lien [Retirer] du tableau HTML des achats du panier a une url de la forme [?action=retirerachat&id=ID] où ID est le champ [id] de l'article qu'on veut retirer du panier.

5.7.4.2 Les composants de la page

n°	*type*	*nom*	*rôle*
1	composant utilisateur	**entete**	afficher l'entête
2	DataGrid	**DataGridAchats** 3- colonne connexe - entête : Article, champ : nom 4 - colonne connexe - entête : Qté, champ : qte 5 - colonne connexe - entête : Prix, champ : prix 6 - colonne connexe - entête : Total, champ : total, mise en forme {0:C} 7 - colonne hypertexte - Texte : Retirer, p Url : id, format Url : /webarticles/main.aspx?action=retirerachat&id={0}	afficher la liste des articles achetés
8	label	**lblTotal**	afficher le montant à payer

5.7.4.3 Le code de présentation [panier.aspx]

```
1. <%@ Page codebehind="panier.aspx.vb" inherits="istia.st.articles.web.PanierWebarticles"
   autoeventwireup="false" Language="vb" %>
2. <%@ Register TagPrefix="WA" TagName="entete" Src="entete.ascx" %>
3. <HTML>
4.  <HEAD>
```

```
5.    <TITLE>webarticles</TITLE>
6.    <META http-equiv="Content-Type" content="text/html; charset=windows-1252">
7.  </HEAD>
8.  <body>
9.    <WA:entete id="entete" runat="server"></WA:entete>
10.   <h2>Contenu de votre panier</h2>
11.   <P>
12.     <asp:DataGrid id="DataGridAchats" runat="server" BorderWidth="1px" GridLines="Vertical"
    CellPadding="4"
13.       BackColor="White" BorderStyle="None" BorderColor="#DEDFDE" ForeColor="Black"
    AutoGenerateColumns="False">
14.       <SelectedItemStyle Font-Bold="True" ForeColor="White" BackColor="#CE5D5A"></SelectedItemStyle>
15.       <AlternatingItemStyle BackColor="White"></AlternatingItemStyle>
16.       <ItemStyle BackColor="#F7F7DE"></ItemStyle>
17.       <HeaderStyle Font-Bold="True" ForeColor="White" BackColor="#6B696B"></HeaderStyle>
18.       <FooterStyle BackColor="#CCCC99"></FooterStyle>
19.       <Columns>
20.         <asp:BoundColumn DataField="nom" HeaderText="Article"></asp:BoundColumn>
21.         <asp:BoundColumn DataField="qte" HeaderText="Qt&#233;"></asp:BoundColumn>
22.         <asp:BoundColumn DataField="prix" HeaderText="Prix"></asp:BoundColumn>
23.         <asp:BoundColumn DataField="totalAchat" HeaderText="Total"
    DataFormatString="{0:C}"></asp:BoundColumn>
24.         <asp:HyperLinkColumn Text="Retirer" DataNavigateUrlField="id"
    DataNavigateUrlFormatString="/webarticles/main.aspx?action=retirerachat&amp;id={0}"></asp:HyperLinkColu
    mn>
25.       </Columns>
26.       <PagerStyle HorizontalAlign="Right" ForeColor="Black" BackColor="#F7F7DE"
    Mode="NumericPages"></PagerStyle>
27.     </asp:DataGrid></P>
28.   <P>Total de la commande :
29.     <asp:Label id="lblTotal" runat="server"></asp:Label> euros</P>
30. </body>
31.</HTML>
```

Commentaires

- la ligne 9 inclut l'entête
- lignes 12-27, le composant [DataGridAchats] est défini
- ligne 29, le composant [lblTotal] est défini

5.7.4.4 Le code de contrôle [panier.aspx.vb]

```
1. Imports System
2. Imports System.Collections
3. Imports System.Data
4. Imports istia.st.articles.dao
5. Imports istia.st.articles.domain
6.
7. Namespace istia.st.articles.web
8.  ' gère la page d'affichage du panier
9.  Public Class PanierWebarticles
10.   Inherits System.Web.UI.Page
11.   Protected WithEvents DataGridAchats As System.Web.UI.WebControls.DataGrid
12.   Protected WithEvents lblTotal As System.Web.UI.WebControls.Label
13.   Protected WithEvents entete As New EnteteWebArticles
14.
15.   Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
    MyBase.Load
16.     ' on lie les options de menu à rptmenu
17.     entete.actions = CType(context.Items("options"), Hashtable())
18.     ' on récupère le panier
19.     Dim unPanier As Panier = CType(Session.Item("panier"), Panier)
20.     ' on transfère les achats dans un tableau de lignes d'achats
21.     Dim achats(unPanier.achats.Count - 1) As LigneAchat
22.     ' on change le type des éléments du ArrayList
23.     For i As Integer = 0 To achats.Length - 1
24.       achats(i) = New LigneAchat(CType(unPanier.achats(i), Achat))
25.     Next
26.     ' on lie les données aux composants [DataGrid] de la page
27.     With DataGridAchats
28.       .DataSource = achats
29.       .DataBind()
30.     End With
31.     ' on affiche le total à payer
32.     lblTotal.Text = unPanier.totalPanier.ToString
33.   End Sub
34.
35.   ' ligne d'achat construite à partir d'un objet Achat
36.   Private Class LigneAchat
37.     Inherits Achat
38.
39.     ' constructeur reçoit un achat
40.     Public Sub New(ByVal unAchat As Achat)
```

```
41.       Me.article = unAchat.article
42.       Me.qte = unAchat.qte
43.     End Sub
44.
45.     ' id : rend l'id de l'article acheté
46.     Public ReadOnly Property id() As Integer
47.       Get
48.         Return article.id
49.       End Get
50.     End Property
51.
52.     ' nom : nom de l'article acheté
53.     Public ReadOnly Property nom() As String
54.       Get
55.         Return article.nom
56.       End Get
57.     End Property
58.
59.     ' prix de l'article acheté
60.     Public ReadOnly Property prix() As Double
61.       Get
62.         Return article.prix
63.       End Get
64.     End Property
65.
66.   End Class
67. End Class
68.End Namespace
```

Commentaires :

- les composants de la page sont déclarés lignes 11-13
- l'initialisation du composant [entete] est identique à celle trouvée dans les pages déjà étudiées - ligne 17
- le panier à afficher est récupéré dans la session - ligne 19
- l'affichage de ce panier à l'aide du composant [DataGridAchats] pose des problèmes. La difficulté vient de l'initialisation du composant. Rappelons les colonnes de celui-ci :
  - colonne [Article] associée au un champ [nom] de la source de données
  - colonne [Qté] associée au un champ [qte] de la source de données
  - colonne [Prix] associée au un champ [prix] de la source de données
  - colonne [Total] associée au un champ [total] de la source de données

  La source de données dont nous disposons est le panier et sa liste d'achats. Cette dernière sera la source de données du [DataGrid]. Seulement les objets [Achat] qui vont alimenter les lignes du [DataGrid] n'ont pas les propriétés [nom, qte, prix, total] attendues par le [DataGrid]. Aussi crée-t-on ici, spécialement pour le [DataGrid] une source de données dont les éléments ont les caractéristiques attendues par le [DataGrid]. Ces éléments seront de type [LigneAchat] une classe créée pour l'occasion et dérivée de la classe [Achat] - lignes 36-66

- la classe [LigneAchat) définie, la source de données de [DataGridAchats] est construite à partir du panier trouvé dans la session - lignes 20-30
- le montant des achats est affiché grâce à la propriété [totalPanier] de la classe [Panier] - ligne 32

5.7.5 La vue [paniervide.aspx]

5.7.5.1 Introduction

Cette vue affiche l'information indiquant que le panier est vide :

Elle est affichée à la suite d'une requête **/main?action=panier** ou **/main?action=retirerachat&id=ID**. Les éléments de la requête du contrôleur sont les suivants :

actions	objet Hashtable() - le tableau des options du menu

5.7.5.2 Les composants de la page

n°	*type*	*nom*	*rôle*
1	composant utilisateur	**entete**	afficher l'entête

5.7.5.3 Le code de présentation [paniervide.aspx]

```
1. <%@ Register TagPrefix="WA" TagName="entete" Src="entete.ascx"%>
2. <%@ Page codebehind="paniervide.aspx.vb" inherits="istia.st.articles.web.PaniervideWebarticles"
   autoeventwireup="false" Language="vb" %>
3. <HTML>
4.  <HEAD>
5.    <TITLE>webarticles</TITLE>
6.    <META http-equiv="Content-Type" content="text/html; charset=windows-1252">
7.  </HEAD>
8.  <body>
9.    <WA:entete id="entete" runat="server"></WA:entete>
10.   <h2>Contenu de votre panier</h2>
11.   <P>Votre panier est vide</P>
12. </body>
13.</HTML>
```

Commentaires :

- l'entête est inclus ligne 9

5.7.5.4 Le code de contrôle [paniervide.aspx.vb]

```
1. Namespace istia.st.articles.web
2.  ' gère la page d'affichage d'un panier vide
3.  Public Class PaniervideWebarticles
4.    Inherits System.Web.UI.Page
5.    Protected WithEvents entete As New EnteteWebArticles
6.
7.    Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
   MyBase.Load
8.      ' on prépare la vue [paniervide] à partir des informations du contexte
9.      ' on lie les options de menu à rptmenu
10.     entete.actions = CType(context.Items("options"), Hashtable())
11.   End Sub
12. End Class
13.End Namespace
```

Commentaires :

- on se contente d'initialiser le seul composant dynamique de la page - ligne 10

5.7.6 La vue [erreurs.aspx]

5.7.6.1 Introduction

Cette vue est affichée en cas d'erreurs :

Elle est affichée à la suite de toute requête menant à une erreur sauf pour l'action d'achat avec une quantité erronée, qui elle, est traitée par la vue [INFOS]. Les éléments de la requête du contrôleur sont les suivants :

`actions`	objet Hashtable() - le tableau des options du menu
`erreurs`	ArrayList d'objets [String] représentant les messages d'erreurs à afficher

5.7.6.2 Les composants de la page

n°	*type*	*nom*	*rôle*	
1	composant utilisateur	**entete**	afficher l'entête	1
2	repeater	**rptErreurs**	afficher la liste des erreurs	

5.7.6.3 La code de présentation [erreurs.aspx] 2

```
1. <%@ Register TagPrefix="WA" TagName="entete" Src="entete.ascx" %>
2. <%@ Page codebehind="erreurs.aspx.vb" inherits="istia.st.articles.web.ErreursWebarticles"
   autoeventwireup="false" Language="vb" %>
3. <HTML>
4.  <HEAD>
5.    <TITLE>webarticles</TITLE>
6.    <META http-equiv="Content-Type" content="text/html; charset=windows-1252">
7.  </HEAD>
8.  <body>
9.    <WA:entete id="entete" runat="server"></WA:entete>
10.   <h3>Les erreurs suivantes se sont produites :
11.   </h3>
12.   <ul>
13.     <asp:Repeater id="rptErreurs" runat="server">
14.       <ItemTemplate>
15.         <li>
16.           <%# Container.DataItem %>
17.         </li>
18.       </ItemTemplate>
19.     </asp:Repeater></ul>
20. </body>
21.</HTML>
```

Commentaires :

- l'entête est défini ligne 9
- le composant [rptErreurs] est défini lignes 13-19. Son contenu provient d'une source de données qui sera de type [ArrayList] d'objets [String].

5.7.6.4 La code de contrôle [erreurs.aspx.vb]

```
1. Namespace istia.st.articles.web
2.
3.  ' gère la page d'erreurs
4.  Public Class ErreursWebarticles
5.    Inherits System.Web.UI.Page
6.
7.    ' composants de la page
8.    Protected WithEvents rptErreurs As System.Web.UI.WebControls.Repeater
9.    Protected WithEvents entete As New EnteteWebArticles
10.
11.   Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
   MyBase.Load
12.     ' on prépare la vue [erreurs] à partir des informations du contexte
13.     ' on lie les options de menu à rptmenu
14.     entete.actions = CType(context.Items("options"), Hashtable())
15.     ' on lie les erreurs à rptErreurs
16.     With rptErreurs
17.       .DataSource = context.Items("erreurs")
18.       .DataBind()
19.     End With
20.   End Sub
21. End Class
22.
23.End Namespace
```

Commentaires :

- le composant [entete] est initialisé comme à l'habitude, lignes 9 et 14
- le composant [rptErreurs] est initialisé avec la liste d'erreurs de type [ArrayList] trouvée dans le contexte - lignes 16-19

5.8 Les contrôleurs global.asax, main.aspx

Il reste à écrire le coeur de notre application web, le contrôleur. Son rôle consiste à :

- récupérer la requête du client,
- traiter l'action demandée par celui-ci à l'aide des classes métier,
- envoyer en réponse la vue appropriée.

5.8.1 Le contrôleur [global.asax.vb]

Lorsque l'application reçoit sa toute première requête, la procédure [Application_Start] du fichier [global.asax.vb] est exécutée. Ce sera la seule fois. La procédure [Application_Start] a pour but d'initialiser les objets nécessaires à l'application web et qui seront partagés en lecture seule par tous les threads clients. Ces objets partagés peuvent être placés en deux endroits :

- les champs privés du contrôleur
- le contexte d'exécution de l'application (Application)

La méthode [Application_Start] de l'application [global.asax.vb] fera les actions suivantes :

- vérifiera la présence, dans le fichier [web.config], des paramètres nécessaires au bon fonctionnement de l'application. Ceux-ci ont été décrits au paragraphe 5.4.
- mettra dans le contexte de l'application la liste des erreurs éventuelles sous la forme d'un objet [ArrayList erreurs]. Cette liste sera vide s'il n'y a pas d'erreurs mais existera néanmoins.
- s'il y a eu des erreurs, la méthode [Application_Start] s'arrête là. Sinon, elle demande la référence d'un singleton de type [IArticlesDomain] qui sera l'objet métier que le contrôleur utilisera pour ses besoins. Comme il a été expliqué en 5.4.2, le contrôleur demandera au framework Spring ce singleton. Cette opération d'instanciation peut amener différentes erreurs. Si tel est le cas, elles seront là encore, mémorisées dans l'objet [erreurs] du contexte de l'application.

Le contrôleur [global.asax.vb] dispose d'une procédure [Session_Start] exécutée à chaque fois qu'un nouveau client arrive. Dans cette procédure, on créera un panier vide pour le client. Ce panier sera maintenu au fil des requêtes de ce client particulier. Le code pourrait être le suivant :

```
1. Imports System
2. Imports System.Web
3. Imports System.Web.SessionState
4. Imports System.Configuration
5. Imports istia.st.articles.domain
6. Imports System.Collections
7. Imports Spring.Context
```

```
8.
9. Namespace istia.st.articles.web
10.
11. Public Class GlobalWebArticles
12.   Inherits System.Web.HttpApplication
13.
14.   ' init application
15.   Sub Application_Start(ByVal sender As Object, ByVal e As EventArgs)
16.
17.     ' données locales
18.     Dim parameters() As String = {"urlMain", "urlErreurs", "urlInfos", "urlListe", "urlPanier",
   "urlPanierVide"}
19.     Dim erreurs As New ArrayList
20.
21.     ' on récupère les paramètres d'initialisation de l'application
22.     Dim param As String
23.     For i As Integer = 0 To parameters.Length - 1
24.       ' lecture dans fichier de conf
25.       param = ConfigurationSettings.AppSettings(parameters(i))
26.       If param Is Nothing Then
27.         ' on note l'erreur
28.         erreurs.Add("Paramètre [" + parameters(i) + "] absent dans le fichier [web.config]")
29.       Else
30.         ' on mémorise le paramètre dans l'application
31.         Application.Item(parameters(i)) = param
32.       End If
33.     Next
34.     ' des erreurs ?
35.     If erreurs.Count = 0 Then
36.       ' on crée un objet IArticlesDomain d'accès à la couche métier
37.       Dim contexte As IApplicationContext = CType(ConfigurationSettings.GetConfig("spring/context"),
   IApplicationContext)
38.       Dim articlesDomain As IArticlesDomain
39.       Try
40.         articlesDomain = CType(contexte.GetObject("articlesDomain"), IArticlesDomain)
41.         ' on mémorise l'objet dans l'application
42.         Application.Item("articlesDomain") = articlesDomain
43.       Catch ex As Exception
44.         ' on mémorise l'erreur
45.         erreurs.Add("Erreur lors de la construction de l'objet d'accès à la couche métier [" +
   ex.ToString + "]")
46.       End Try
47.     End If
48.     ' les erreurs sont placées dans l'application
49.     Application.Item("erreurs") = erreurs
50.     ' c'est fini s'il y a eu des erreurs
51.     If erreurs.Count <> 0 Then Return
52.     ' on construit un tableau d'options de menu
53.     Dim options As New Hashtable
54.     ' on récupère l'url du contrôleur
55.     Dim urlMain As String = CType(Application.Item("urlMain"), String)
56.     Dim uneOption As Hashtable
57.     ' liste des articles
58.     uneOption = New Hashtable
59.     uneOption.Add("href", urlMain + "?action=liste")
60.     uneOption.Add("lien", "Liste des articles")
61.     options.Add("liste", uneOption)
62.     ' panier
63.     uneOption = New Hashtable
64.     uneOption.Add("href", urlMain + "?action=panier")
65.     uneOption.Add("lien", "Voir le panier")
66.     options.Add("panier", uneOption)
67.     ' validation panier
68.     uneOption = New Hashtable
69.     uneOption.Add("href", urlMain + "?action=validationpanier")
70.     uneOption.Add("lien", "Valider le panier")
71.     options.Add("validationpanier", uneOption)
72.     ' on met les options du menu dans l'application
73.     Application.Item("options") = options
74.     Return
75.   End Sub
76.
77.   ' init session
78.   Sub Session_Start(ByVal sender As Object, ByVal e As EventArgs)
79.     ' on crée un panier pour le client
80.     Session.Item("panier") = New Panier
81.   End Sub
82. End Class
83.End Namespace
```

Commentaires :

- les paramètres attendus dans [web.config] sont définis dans un tableau - ligne 18
- ils sont recherchés dans [web.config]. S'ils sont présents, ils sont mémorisés dans le contexte de l'application, sinon une erreur est enregistrée dans la liste des erreurs [erreurs] - lignes 21-33

- s'il n'y a pas d'erreurs, on demande à Spring une référence du singleton [articlesDomain] qui gère l'accès à la couche [domain] de l'application - lignes 35-47. Les erreurs éventuelles sont enregistrées dans [erreurs].
- les erreurs sont enregistrées dans le contexte de l'application - ligne 49
- on quitte la procédure s'il y a eu des erreurs - ligne 51
- on crée un tableau de trois dictionnaires. Chacun d'eux a deux clés : href et lien. Ce tableau représente les trois options de menu possibles - lignes 52-71
- ce tableau est mémorisé dans le contexte de l'application - ligne 73
- à chaque nouveau client, la procédure [Session_Start] est exécutée. On y crée un panier vide dans la session du client - lignes 78-81

5.8.2 Le contrôleur [main.aspx.vb]

Le contrôleur [main.aspx.vb] voit passer toutes les requêtes des clients. En effet, celles-ci sont toutes de la forme [/webarticles/main.aspx?action=XX]. Une requête sera traitée de la façon suivante :

- l'objet [erreurs] du contexte de l'application sera vérifié. S'il est non vide, cela signifie qu'il y a eu des erreurs lors de l'initialisation de l'application et que celle-ci ne peut pas fonctionner. On enverra alors, en réponse, la vue [ERREURS].
- le paramètre [action] de la requête sera récupéré et vérifié. S'il ne correspond pas à une action connue, la vue [ERREURS] est envoyée avec un message d'erreur approprié.
- si le paramètre [action] est valide, la requête du client est passée à une procédure spécifique à l'action pour traitement :

méthode	demande	traitement	réponses possibles
`doListe`	GET /main?action=liste	- demander la liste des articles à la classe métier - l'afficher	[LISTE] ou [ERREURS]
`doInfos`	GET /main?action=infos&id=ID	- demander l'article d'id=ID à la classe métier - l'afficher	[INFOS] ou [ERREURS]
`doAchat`	POST /main?action=achat&id=ID - la qté achetée fait partie des paramètres postés	- demander l'article d'id=ID à la classe métier - l'inclure dans le panier dans la session client	[LISTE] ou [INFOS] ou [ERREURS]
`doRetirerAchat`	GET /main?action=retirerachat&id=ID	- retirer l'article d'id=ID de la liste des achats du panier de la session client	[PANIER]
`doPanier`	GET /main?action=panier	- faire afficher le panier de la session client	[PANIER] ou [PANIERVIDE]
`doValidationPanier`	GET /main?action=validationpanier	- décrémenter dans la base les stocks de tous les articles présents dans le panier de session du client	[LISTE] ou [ERREURS]

Le squelette du contrôleur [main.aspx.vb] pourrait être le suivant :

```
1. Imports System.Collections
2. Imports System
3. Imports System.Data
4.
5. Imports istia.st.articles.dao
6. Imports istia.st.articles.domain
7.
8. Namespace istia.st.articles.web
9.
10. ' classe contrôleur de l'application web
11. Public Class MainWebArticles
12.   Inherits System.Web.UI.Page
13.
14.   ' champs privés
15.   Private articlesDomain As IArticlesDomain
16.   Private options As Hashtable
17.
18.   ' chargement de la page
19.   Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
   MyBase.Load
20.....
21.   End Sub
22.
23.   ' méthodes de traitement des actions
24.
25.   ' liste des articles
26.   Public Sub doListe()
27....
28.   End Sub
```

```
29.
30.   ' infos sur un article
31.   Public Sub doInfos()
32....
33.   End Sub
34.
35.   ' achat d'un article
36.   Public Sub doAchat()
37....
38.   End Sub
39.
40.   ' suppression d'un achat
41.   Public Sub doRetirerAchat()
42....
43.   End Sub
44.
45.   ' visualisation panier
46.   Public Sub doPanier()
47....
48.   End Sub
49.
50.   ' achat panier
51.   Public Sub doValidationPanier()
52....
53.   End Sub
54.
55. End Class
56.
57.End Namespace
```

Commentaires :

- la classe a deux champs privés qui seront partagés entre les méthodes - lignes 15-16 :
  - **articlesDomain** : le singleton d'accès à la couche [domain]
  - **options** : le tableau des dictionnaires des options de menu
- la procédure [**Page_Load**] :
  - initialisera les deux champs privés de la classe
  - récupèrera le paramètre [action] de la requête et fera exécuter la méthode de traitement de cette action.

5.8.3 La méthode [Page_Load]

Cet événement est le premier à se produire sur la page. Le code est le suivant :

```
1.    ' chargement de la page
2.    Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
   MyBase.Load
3.      ' on vérifie si l'application a démarré correctement
4.      Dim erreurs As ArrayList = CType(Application.Item("erreurs"), ArrayList)
5.      ' s'il y a des erreurs, on envoie la vue [erreurs]
6.      If erreurs.Count <> 0 Then
7.        context.Items("erreurs") = erreurs
8.        context.Items("options") = New Hashtable() {}
9.        Server.Transfer(CType(Application("urlErreurs"), String))
10.     End If
11.     ' on récupère l'objet d'accès à la classe métier
12.     articlesDomain = CType(Application.Item("articlesDomain"), IArticlesDomain)
13.     ' ainsi que les options de menu
14.     options = CType(Application.Item("options"), Hashtable)
15.     ' on récupère l'action à faire
16.     Dim action As String = Request.QueryString("action")
17.     If action Is Nothing Then
18.       action = "liste"
19.     End If
20.     ' on exécute l'action
21.     Select Case action
22.       Case "liste"
23.         doListe()
24.       Case "infos"
25.         doInfos()
26.       Case "achat"
27.         doAchat()
28.       Case "panier"
29.         doPanier()
30.       Case "retirerachat"
31.         doRetirerAchat()
32.       Case "validationpanier"
33.         doValidationPanier()
34.       Case Else
35.         doListe()
36.     End Select
37.   End Sub
```

Commentaires :

- à chaque chargement de page, on s'assure que l'initialisation de l'application faite par [global.asax] s'est bien déroulée.
- pour cela, on récupère dans le contexte de l'application, la liste des erreurs placée là par [global.asax] - ligne 4
- si cette liste est non vide, on fait afficher la vue [ERREURS] - lignes 6-10
- on récupère le singleton [articlesDomain] placé par [global.asax] dans le contexte de l'application et on le mémorise dans le champ privé [articlesDomain] afin qu'il soit disponible aux différentes méthodes de la classe - ligne 12
- on fait un travail analogue avec le tableau des options de menu - ligne 14
- on récupère le paramètre [action] de la requête - ligne 16
- on fait exécuter la méthode correspondant à l'action demandée. Une action non prévue est considérée comme l'action [liste] - lignes 16-36

5.8.4 Traitement de l'action [liste]

Il s'agit d'afficher la liste des articles :

Le code est le suivant :

```
1.    ' liste des articles
2.    Public Sub doListe()
3.      ' gestion des erreurs
4.      Dim erreurs As New ArrayList
5.      ' on demande la liste des articles
6.      Dim articles As IList
7.      Try
8.        articles = articlesDomain.getAllArticles
9.      Catch ex As Exception
10.       ' on note l'erreur
11.       erreurs.Add(ex.ToString)
12.     End Try
13.     ' des erreurs ?
14.     If erreurs.Count <> 0 Then
15.       ' affichage vue [erreurs]
16.       context.Items("erreurs") = erreurs
17.       context.Items("options") = New Hashtable() {CType(options("liste"), Hashtable)}
18.       Server.Transfer(CType(Application.Item("urlErreurs"), String))
19.     End If
20.     ' affichage vue [liste]
21.     context.Items("articles") = articles
22.     context.Items("options") = New Hashtable() {CType(options("panier"), Hashtable)}
23.     If context.Items("message") Is Nothing Then context.Items("message") = ""
24.     Server.Transfer(CType(Application.Item("urlListe"), String))
25.   End Sub
```

Commentaires :

- on met les erreurs éventuelles dans un [ArrayList] - ligne 4
- la liste des articles est demandée au singleton [articlesDomain] - lignes 5-12
- s'il y a eu des erreurs, on envoie la vue [ERREURS] - lignes 13-19
- sinon on envoie la vue [LISTE] - lignes 20-24

5.8.5 Traitement de l'action [infos]

Le client a demandé de l'information sur un article donné :

Le code est le suivant :

```
1.   ' infos sur un article
2.   Public Sub doInfos()
3.     ' gestion des erreurs
4.     Dim erreurs As New ArrayList
5.     ' on récupère l'id de l'article demandé
6.     Dim strId As String = Request.QueryString("id")
7.     ' a-t-on qq chose ?
8.     If strId Is Nothing Then
9.       ' pas normal - on envoie la page d'erreurs
10.      erreurs.Add("action incorrecte (action=infos, id=rien)")
11.      context.Items("erreurs") = erreurs
12.      context.Items("options") = New Hashtable() {CType(options("liste"), Hashtable)}
13.      Server.Transfer(CType(Application.Item("urlErreurs"), String))
14.      Exit Sub
15.    End If
16.    ' a-t-on un entier ?
17.    Dim id As Integer
18.    Try
19.      id = Integer.Parse(strId)
20.    Catch ex As Exception
21.      ' pas normal - on envoie la page d'erreurs
22.      erreurs.Add("action incorrecte (action=infos, id[" + strId + "] invalide)")
23.      context.Items("erreurs") = erreurs
24.      context.Items("options") = New Hashtable() {CType(options("liste"), Hashtable)}
25.      Server.Transfer(CType(Application.Item("urlErreurs"), String))
26.      Exit Sub
27.    End Try
28.    ' on demande l'article de clé id
29.    Dim unArticle As Article
30.    Try
31.      unArticle = articlesDomain.getArticleById(id)
32.    Catch ex As Exception
33.      ' pb d'accès aux données
34.      erreurs.Add("Problème d'accès aux données (" + ex.ToString + ")")
35.      context.Items("erreurs") = erreurs
36.      context.Items("options") = New Hashtable() {CType(options("liste"), Hashtable)}
37.      Server.Transfer(CType(Application.Item("urlErreurs"), String))
38.      Exit Sub
39.    End Try
40.    ' a-t-on récupéré un article ?
41.    If unArticle Is Nothing Then
42.      ' l'article n'existe pas
43.      erreurs.Add("L'article d'id=" + id.ToString + " n'existe pas")
44.      context.Items("erreurs") = erreurs
45.      context.Items("options") = New Hashtable() {CType(options("liste"), Hashtable)}
46.      Server.Transfer(CType(Application.Item("urlErreurs"), String))
47.      Exit Sub
48.    End If
49.    ' on a l'article - on le met dans la session courante
50.    Session.Item("article") = unArticle
51.    ' on prépare son affichage
52.    context.Items("options") = New Hashtable() {CType(options("liste"), Hashtable)}
53.    context.Items("msg") = ""
54.    context.Items("qte") = ""
55.    Server.Transfer(CType(Application.Item("urlInfos"), String))
56.  End Sub
```

Commentaires :

- on met les erreurs éventuelles dans un [ArrayList] - ligne 4
- l'identifiant de l'article demandé est récupéré dans la requête - ligne 6
- cet identifiant est vérifié. Il doit être présent et ce doit être un entier. Si ce n'est pas le cas, la vue [ERREURS] est envoyée avec le message d'erreur approprié - lignes 7-27
- une fois l'identifiant vérifié, l'article est demandé au singleton [articlesDomain]. S'il se produit une exception, la vue [ERREURS] est envoyée - lignes 29-39

- si l'article n'a pas été trouvé, la vue [ERREURS] est envoyée - lignes 41-48
- si l'article a été trouvé, il est placé dans la session de l'utilisateur puis affiché dans la vue [INFOS] - lignes 50-56

5.8.6 Traitement de l'action [achat]

Le client a acheté l'article affiché dans la vue [INFOS].

Le code est le suivant :

```
1.    ' achat d'un article
2.    Public Sub doAchat()
3.      ' achat d'un article
4.      ' on récupère l'article qui était dans la session
5.      Dim unArticle As Article = CType(Session.Item("article"), Article)
6.      ' a-t-on qq chose ?
7.      If unArticle Is Nothing Then
8.        ' pas normal - on affiche la liste des articles
9.        doListe()
10.     End If
11.     ' on récupère la qté postée
12.     Dim strQte As String = Request.Form("txtQte")
13.     ' a-t-on qq chose ?
14.     If strQte Is Nothing Then
15.       ' pas normal - on envoie la liste des articles
16.       doListe()
17.     End If
18.     ' a-t-on un entier ?
19.     Dim qte As Integer
20.     Try
21.       qte = Integer.Parse(strQte)
22.       If (qte <= 0) Then Throw New Exception
23.     Catch ex As Exception
24.       ' pas normal - on envoie la page d'infos avec le msg d'erreur
25.       context.Items("options") = New Hashtable() {CType(options("liste"), Hashtable)}
26.       context.Items("msg") = "Quantité incorrecte"
27.       context.Items("qte") = strQte
28.       Server.Transfer(CType(Application.Item("urlInfos"), String))
29.     End Try
30.     ' tout va bien - on met l'achat dans le panier du client
31.     Dim unPanier As Panier = CType(Session.Item("panier"), Panier)
32.     unPanier.ajouter(New Achat(unArticle, qte))
33.     ' on affiche la liste des articles
34.     doListe()
35.   End Sub
```

Commentaires :

- l'article mis dans la session est récupéré - ligne 5
- s'il n'est pas là (la session a pu expirer), on affiche la vue [LISTE] - lignes 7-10
- la quantité achetée est récupérée dans la requête - ligne 12
- sa validité est vérifiée - lignes 13-29
- en cas d'invalidité, selon les cas, on envoie la vue [LISTE] - ligne 16 ou la vue [INFOS] - lignes 24-28
- si tout est normal, l'achat est enregistré dans le panier - lignes 31-32
- puis la vue [LISTE] est envoyée - ligne 34

5.8.7 Traitement de l'action [panier]

Le client a fait différents achats et il demande à voir le panier :

Le code est le suivant :

```
1.    ' visualisation panier
2.    Public Sub doPanier()
3.      'on récupère le panier dans la session
4.      Dim unPanier As Panier = CType(Session.Item("panier"), Panier)
5.      ' panier vide ?
6.      If unPanier.achats.Count = 0 Then
7.        ' affichage panier vide
8.        context.Items("options") = New Hashtable() {CType(options("liste"), Hashtable)}
9.        Server.Transfer(CType(Application.Item("urlPanierVide"), String))
10.     End If
11.     ' affichage panier non vide
12.     context.Items("options")     =     New     Hashtable()     {CType(options("liste"),     Hashtable),
    CType(options("validationpanier"), Hashtable)}
13.     Server.Transfer(CType(Application.Item("urlPanier"), String))
14.   End Sub
```

Commentaires :

- on récupère le panier dans la session - ligne 4. On ne vérifie pas ici qu'on récupère bien quelque chose. Il faudrait le faire car la session a pu expirer.
- si le panier est vide, on envoie la vue [PANIERVIDE] - lignes 6-10
- sinon on envoie la vue [PANIER] - lignes 11-14

5.8.8 Traitement de l'action [retirerachat]

Le client veut retirer un achat de son panier :

Le code est le suivant :

```
1.    ' suppression d'un achat
2.    Public Sub doRetirerAchat()
3.      'on récupère le panier dans la session
4.      Dim unPanier As Panier = CType(Session.Item("panier"), Panier)
5.      ' on enlève l'achat
6.      Try
7.        ' on récupère l'id de l'article retiré
8.        Dim idArticle As Integer = Integer.Parse(Request.QueryString("id"))
9.        ' on l'enlève du panier
10.       unPanier.enlever(idArticle)
11.     Catch ex As Exception
12.       ' on affiche la liste des articles
13.       doListe()
14.     End Try
15.     ' on affiche le panier
16.     doPanier()
17.   End Sub
```

Commentaires :

- on récupère le panier dans la session - ligne 4. On ne vérifie pas ici qu'on récupère bien quelque chose. Il faudrait le faire car la session a pu expirer.
- on récupère dans la requête l'identifiant [id] de l'article à enlever - ligne 8.
- l'achat correspondant est enlevé du panier - ligne 10

- la validité de l'identifiant de l'article acheté n'a pas été vérifiée ici. Si celui-ci a un type invalide, une exception aura lieu et sera traitée lignes 11-13. S'il est valide mais inexistant, la méthode [panier.enlever] - ligne 10 - ne fait rien.
- on fait afficher le nouveau panier - ligne 16

5.8.9 Traitement de l'action [validerpanier]

Le client veut valider son panier :

Le code est le suivant :

```
1.    ' achat panier
2.    Public Sub doValidationPanier()
3.      ' au départ, pas d'erreurs
4.      Dim erreurs As ArrayList
5.      'on récupère le panier dans la session
6.      Dim unPanier As Panier = CType(Session.Item("panier"), Panier)
7.      ' on tente de valider le panier
8.      Try
9.        articlesDomain.acheter(unPanier)
10.       ' on note les éventuelles erreurs d'achats
11.       erreurs = articlesDomain.erreurs
12.     Catch ex As Exception
13.       ' on note l'erreur
14.       erreurs = New ArrayList
15.       erreurs.Add(String.Format("Erreur lors de la validation du panier [{0}]", ex.Message))
16.     End Try
17.     ' si erreurs alors page d'erreurs
18.     If erreurs.Count <> 0 Then
19.       context.Items("erreurs") = erreurs
20.       context.Items("options")     =     New     Hashtable()     {CType(options("liste"),     Hashtable),
    CType(options("panier"), Hashtable)}
21.       Server.Transfer(CType(Application.Item("urlErreurs"), String))
22.       Exit Sub
23.     End If
24.     ' tout va bien - on affiche la liste des articles avec un message de réussite
25.     context.Items("message") = "Votre panier a été validé"
26.     doListe()
27.   End Sub
```

Commentaires :

- on récupère le panier dans la session - ligne 6. On ne vérifie pas ici qu'on récupère bien quelque chose. Il faudrait le faire car la session a pu expirer.
- dans le cas où la session a expiré, on aura un pointeur [nothing] pour le panier et la méthode [acheter] - ligne 9 - lancera une exception et la vue [ERREURS] sera envoyée. Seulement le message d'erreur sera peu explicite pour l'utilisateur.
- lignes 8-16, on essaie de valider le panier récupéré dans la session. Certains achats peuvent être non validés si la quantité demandée excède le stock de l'article demandé. Ces cas sont mémorisés par la méthode [acheter] dans une liste d'erreurs qui est récupérée ligne 11.
- si on a des erreurs, la vue [ERREURS] est envoyée - lignes 18-23
- sinon c'est la vue [LISTE] qui est envoyée - lignes 25-26

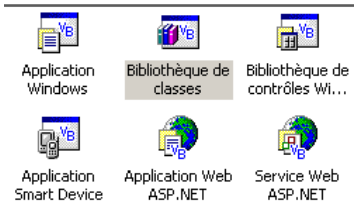
6 Conclusion

Nous avons ici développé une application selon un modèle MVC. Il ne semble pas exister (avril 2005) de "frameworks" professionnels de développement MVC en ASP.NET comme il en existe en Java (Struts, Spring, ...). Le projet [Spring.net] devrait en proposer un rapidement. En attendant cet avènement, la méthode précédente permet un développement MVC viable pour des applications de taille moyenne.

7 Annexes

7.1 Construire un projet web avec Visual Studio.net sur XP familial

Visual Studio.net permet de construire différents types de projets :



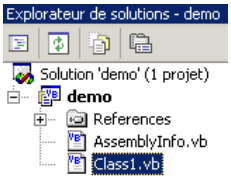
Pour construire un projet d'application web, on choisit normalement le type [Application Web ASP.NET]. Ce type de projet nécessite la présence d'un serveur web IIS local ou distant. Si on travaille sur une machine Windows XP, ce serveur n'existe pas et il n'est pas possible de l'installer. On ne peut donc créer de projet de type [Application Web ASP.NET].

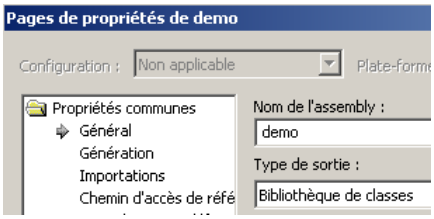
On peut contourner l'obstacle en acceptant quelques inconvénients mineurs. Il suffit :

- d'utiliser le serveur web Cassini en lieu et place du serveur IIS. Il est disponible librement sur le site de Microsoft.
- d'utiliser un projet [Bibliothèque de classes] en lieu et place du projet [Application Web ASP.NET]

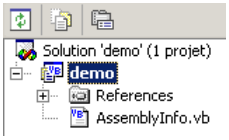
Créons un projet simple qui montre comment procéder.

- créer un projet [Bibliothèque de classes]

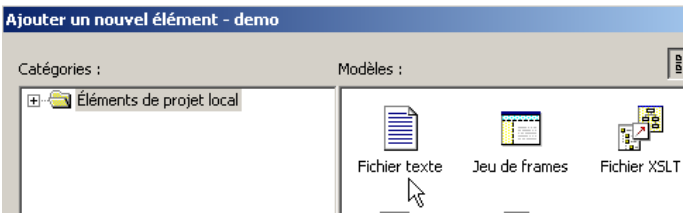


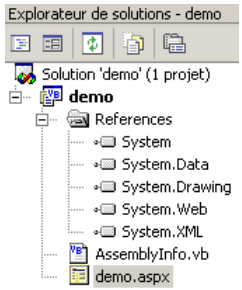


- supprimer [Class1.vb]

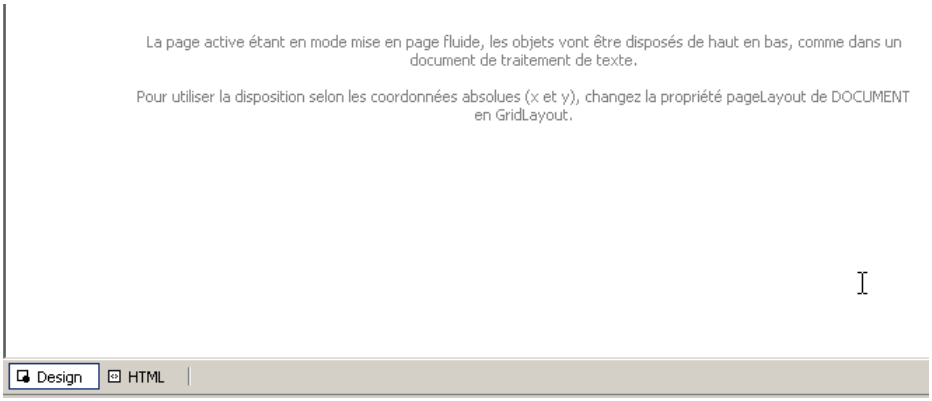


- ajouter un nouvel élément au projet, de type [Fichier texte] et l'appeler [demo.aspx] :

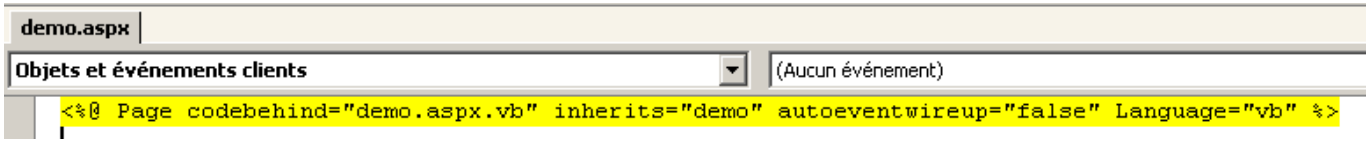




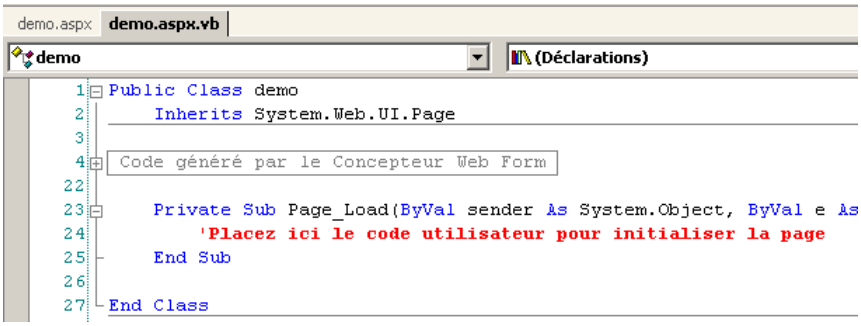
- le fichier [demo.aspx] est reconnu comme une page web et un éditeur de page lui est associé. Celui-ci a deux panneaux :
  - [design] pour construire graphiquement la page
  - [HTML] pour avoir accès au code HTML de la page



- faire [Affichage/Code] pour faire afficher la partie code VB de la page. Rien ne se passe. On n'a pas accès au code VB de la page.
- aller dans le panneau [HTML] et inscrire le code qui va relier la page [demo.aspx] au code [demo.aspx.vb] :



- demander à voir le code de contrôle associé à la page par [Affichage code - F7]. On obtient le fichier [demo.aspx.vb] :



- la page [demo.aspx] est désormais reconnue comme une page web [.aspx] avec un code [.aspx.vb] associé.
- revenons sur le panneau [Design] de [demo.aspx] et dessinons la page suivante :

La page contient du texte et un composant serveur de type [Label] d'identifiant [lblHeure].

- passons dans le panneau [HTML]. On y trouve le code suivant :

```
<%@ Page codebehind="demo.aspx.vb" inherits="demo.demo" autoeventwireup="false" Language="vb" %>
Démo ASPX, il est
<asp:Label id="lblHeure" runat="server"></asp:Label>
```

Ce code est incomplet d'un point de vue syntaxe HTML. Complétons-le :

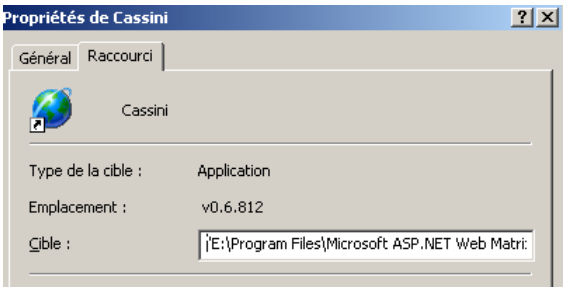
```
<%@ Page codebehind="demo.aspx.vb" inherits="demo.demo" autoeventwireup="false" Language="vb" %>
<html>
  <head>
    <title>démo ASPX</title></head>
  <body>
    Démo ASPX, il est
    <asp:Label id="lblHeure" runat="server"></asp:Label>
  </body>
</html>
```

- passons dans le code [demo.aspx.vb] pour y écrire le code de contrôle qui mettra l'heure dans le composant [lblHeure]. On constate que celui-ci n'est pas reconnu par Intellisense, l'outil d'aide à l'écriture de code.
- fermer [demo.aspx] et [demo.aspx.vb] en les sauvegardant auparavant puis les rouvrir. Passer dans le code [demo.aspx.vb]. Cette fois-ci le composant [lblHeure] de [demo.aspx] est bien reconnu par Intellisense dans le code [demo.aspx.vb]. Compléter le code :

```
1  Public Class demo
2      Inherits System.Web.UI.Page
3
4  Code généré par le Concepteur Web Form
23
24     Private Sub Page_Load(ByVal sender As System.
25     'on met l'heure dans le composant [lblHeure]
26     lblHeure.Text = Date.Now.ToString("t")
27   End Sub
28
29 End Class
30
```

- générer le projet par [Générer/Générer demo]. Si la génération se passe correctement, la DLL est alors générée dans le dossier [bin] du projet :

- Nous sommes prêts pour les tests. Nous configurons le serveur Cassini (cg paragraphe suivant) de la façon suivante :

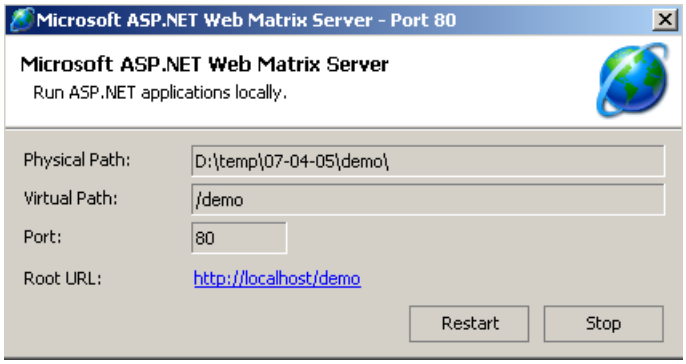


La cible du raccourci vers Cassini est définie comme suit :

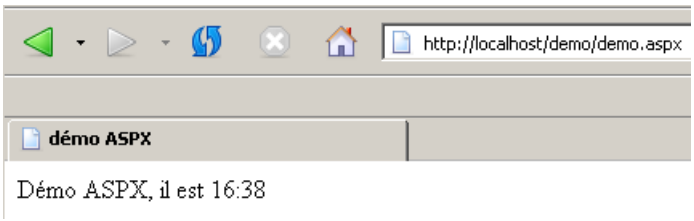
```
"E:\Program Files\Microsoft ASP.NET Web Matrix\v0.6.812\WebServer.exe" /path:"D:\temp\07-04-05\demo" /vpath:"/demo"
```

`E:\..\WebServer.exe`	le chemin de l'exécutable
`D:\temp\07-04-05\demo`	le chemin du dossier du projet web Visual Studio
`/demo`	le chemin virtuel associé

- lancer Cassini. Son icône s'installe dans la barre des tâches. Cliquer droit dessus et prendre l'option [Show details] pour vérifier la correcte configuration du serveur web :



- avec un navigateur, demandons la page que nous avons construite, en demandant l'URL [http://localhost/demo/demo.aspx]. Nous obtenons le résultat suivant :

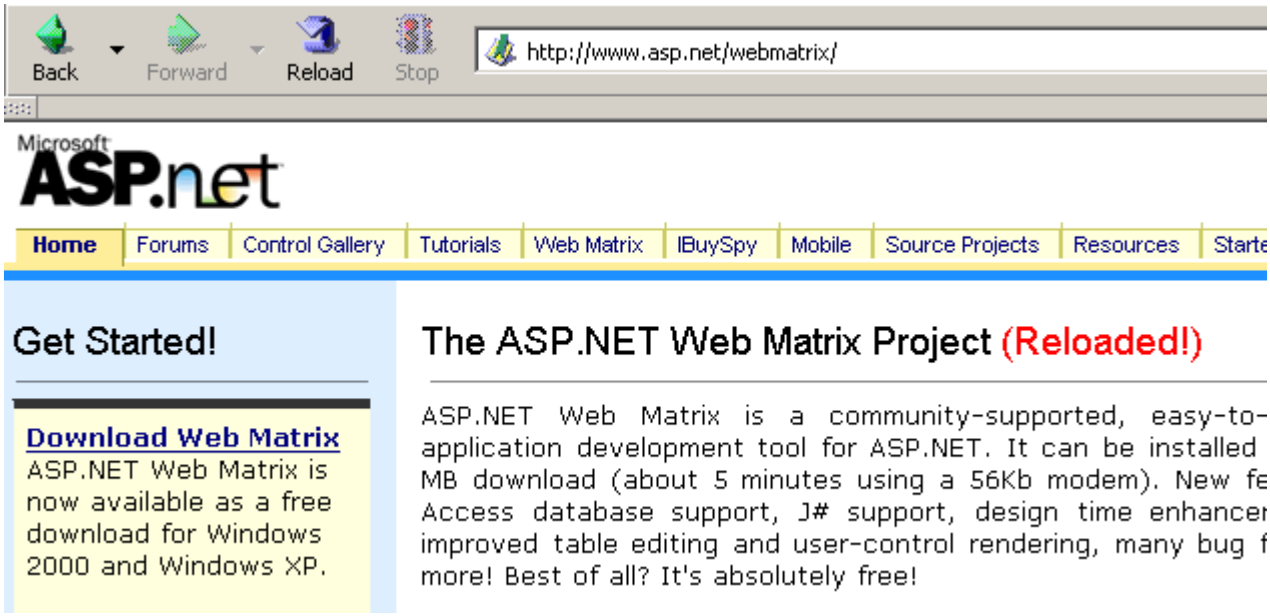


Nous avons réussi à créer une application web sous Visual Studio en utilisant :
- un projet de type [bibliothèque de classes]
- le serveur web Cassini

Nous savons maintenant construire des applications web sur des postes ne disposant pas du serveur IIS, comme c'est le cas des postes windows XP, édition familiale.

7.2 Où trouver le serveur web Cassini ?

Pour travailler avec la plate-forme .NET de Microsoft, on peut utiliser le serveur web Cassini. Celui-ci est disponible via un produit appelé [WebMatrix] qui est un environnement gratuit de développement web sur les plate-formes .NET disponible à l'url :

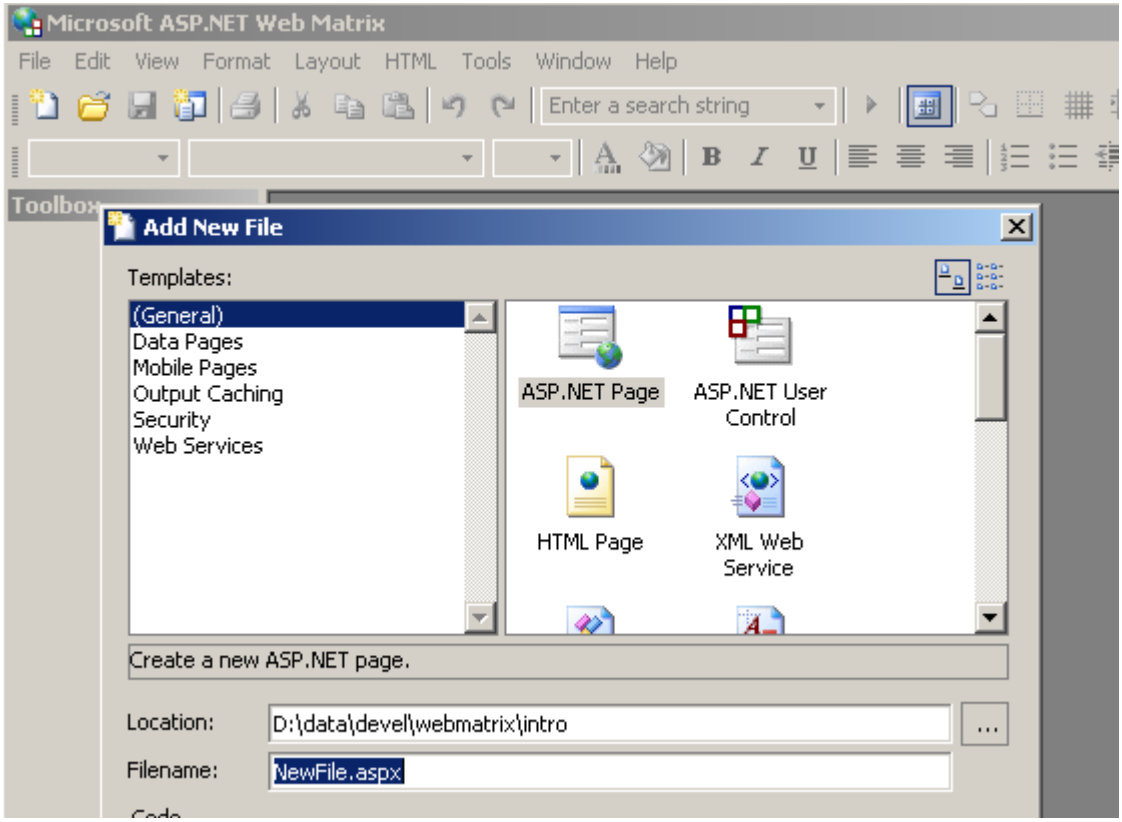


On suivra attentivement la démarche d'installation du produit :

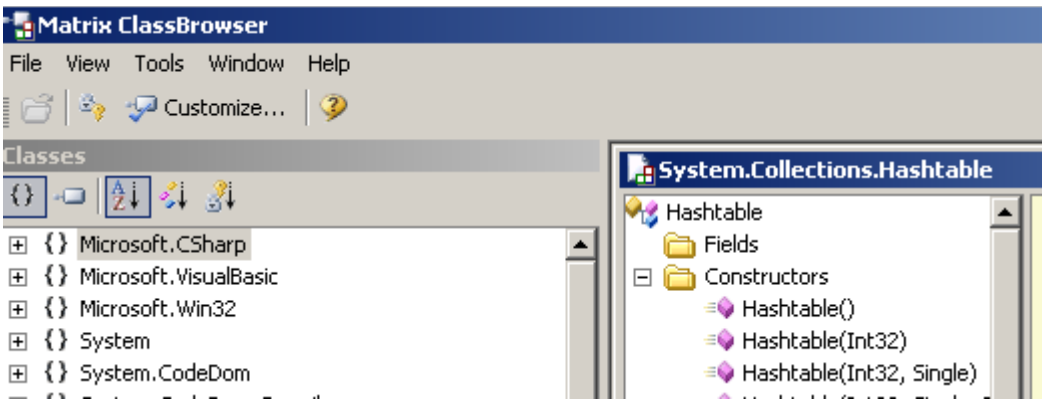
- télécharger et installer la plate-forme .NET (1.1 en mars 2004)
- télécharger et installer WebMatrix
- télécharger et installer MSDE (Microsoft Data Engine) qui est une version limitée de SQL Server.

Une fois l'installation terminée, le produit [WebMatrix] est disponible dans les programmes installés :

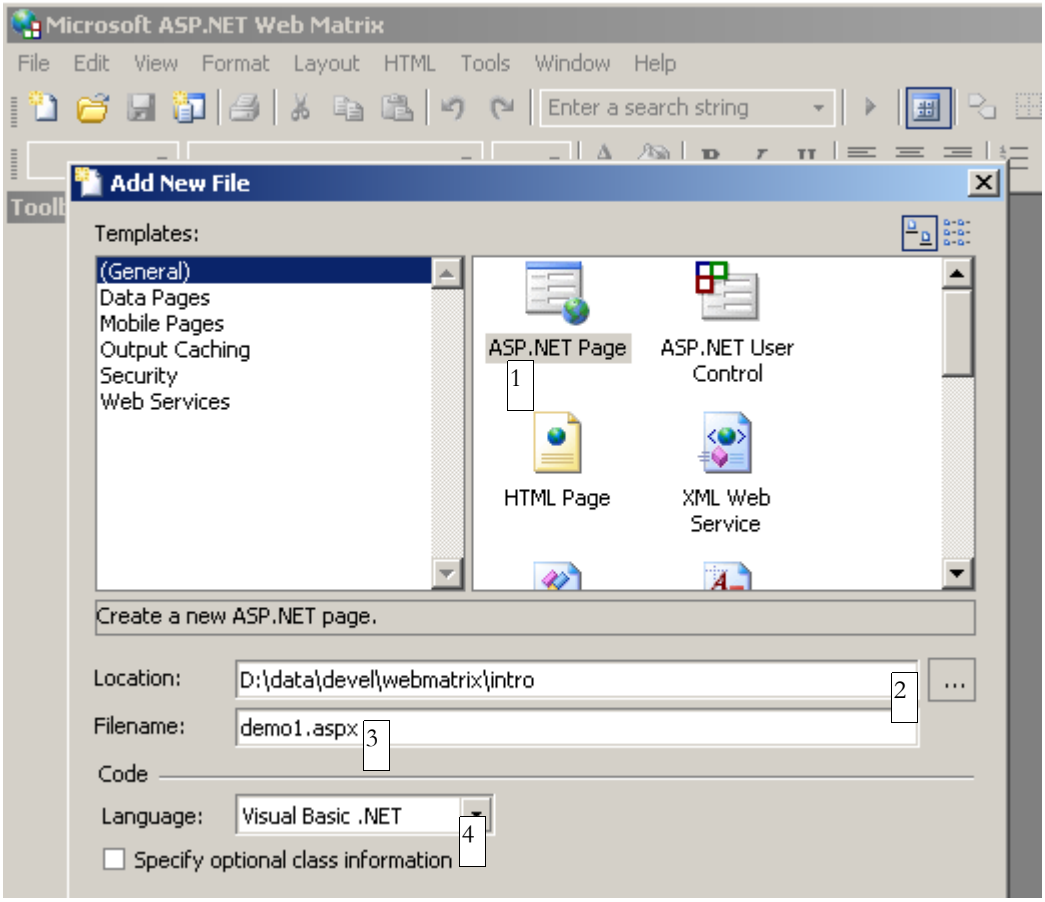
Le lien [ASP.NET] Web Matrix lance l'IDE de développement ASP.NET :



Le lien [Class Browser] lance un outil d'exploration des classes .NET :

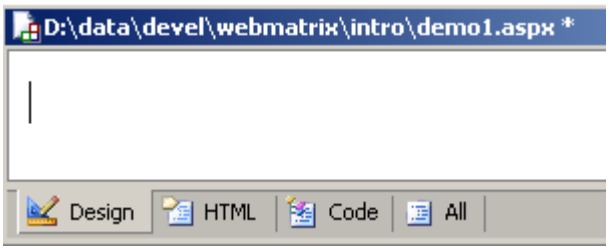


Pour tester l'installation, lançons [WebMatrix] :



Lors du démarrage initial, [WebMatrix] demande les caractéristiques du nouveau projet.C'est sa configuration par défaut. On peut le configurer pour qu'il ne fasse pas apparaître cette boîte de dialogue au démarrage. On l'obtient alors par l'otion [File/New File]. [WebMatrix] permet de construire des squelettes pour différentes applications web. Ci-dessus, nous avons précisé avec (1) que nous voulions construire une application [ASP.ET Page] qui est une page Web. Avec (2), nous précisons le dossier dans lequel sera placée cette page Web. Dans (3) nous donnons le nom de la page. Elle doit avoir le suffixe .aspx. Enfin dans (4), nous précisons que nous voulons travailler avec le langage VB.NET, [WebMatrix] supportant par ailleurs les langages C# et J#.

Ceci fait, [WebMatrix] affiche une page d'édition du fichier [demo1.aspx]. Nous y plaçons le code suivant :



- l'onglet [Design] permet de "dessiner" la page web que l'on veut construire. Cela se passe comme avec un IDE de construction d'applications windows.
- la conception graphique de la page Web dans [Design] va générer du code HTML dans l'onglet [HTML]
- la page Web peut contenir des contrôles générant des événements auxquels il faut réagir, un bouton par exemple. Ces événements seront gérés par du code VB.NET qui sera placé dans l'onglet [Code]
- au final, le fichier demo1.aspx est un fichier texte mélangeant code HTML et code VB.NET, résultat de la conception graphique faite dans [Design], du code HTML qu'on a pu ajouter à la main dans [HTML] et du code VB.NET placé dans [Code]. La totalité du fichier est disponible dans l'onglet [All].
- un développeur ASP.ET expérimenté peut construire le fichier demo1.aspx directement avec un éditeur de texte sans l'aide d'aucun IDE.

Sélectionnons l'option [All] On constate que [WebMatrix] a déjà généré du code :

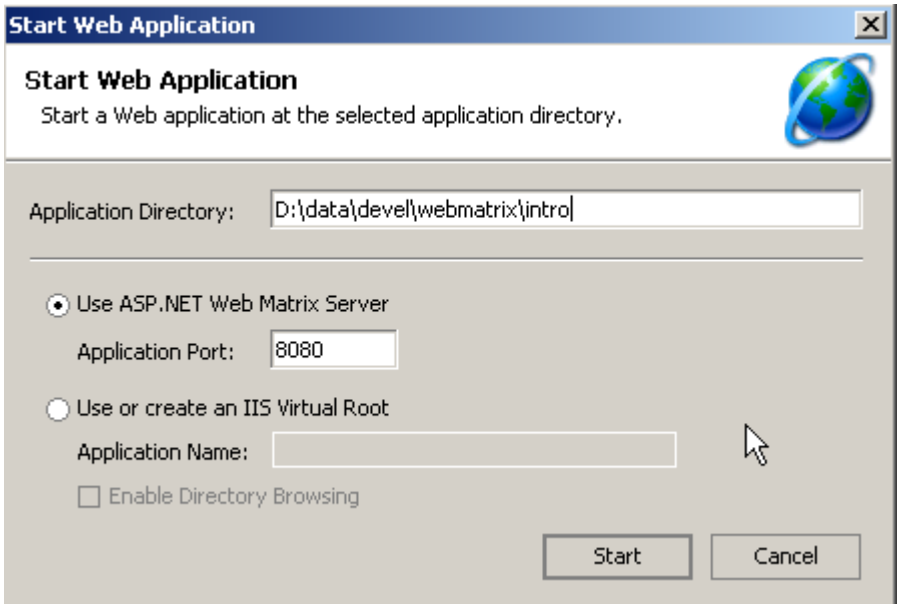
```
<%@ Page Language="VB" %>
<script runat="server">

    ' Insert page code here
```

```
    '

</script>
<html>
<head>
</head>
<body>
    <form runat="server">
        <!-- Insert content here -->
    </form>
</body>
</html>
```

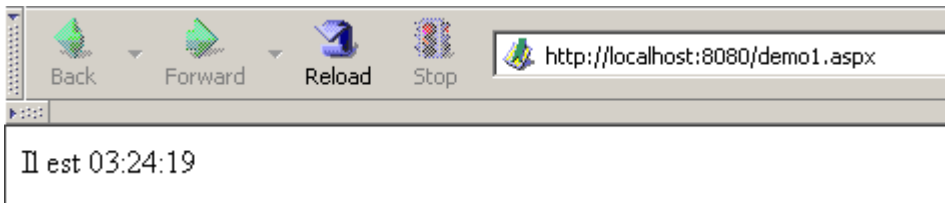
Nous n'allons pas essayer ici d'expliquer ce code. Nous le transformons de la façon suivante :

```
<html>
<head>
 <title>Démo asp.net </title>
</head>
<body>
 Il est <% =Date.Now.ToString("hh:mm:ss") %>
</body>
</html>
```

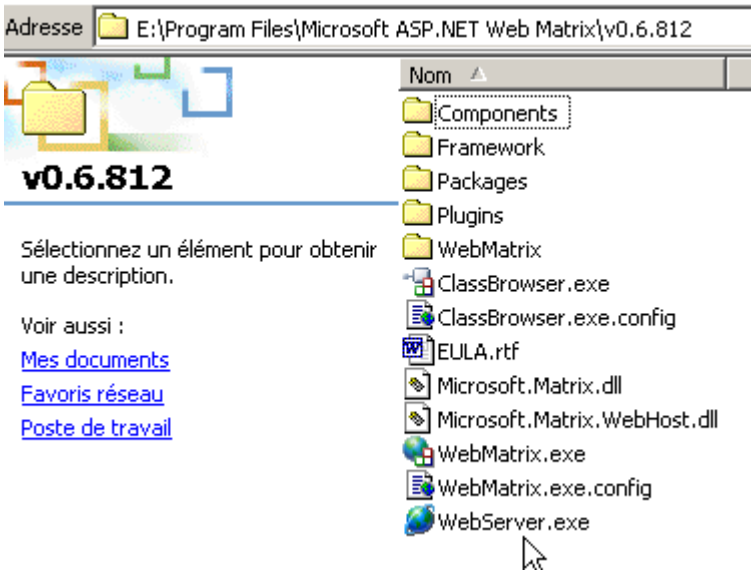
Le code ci-dessus est un mélange de HTML et de code VB.NET. Celui-ci a été placé dans les balises <% ... %>. Pour exécuter ce code, nous utilisons l'option [View/Start]. [WebMatrix] lance alors le serveur Web Cassini s'il n'est pas déjà lancé



On peut accepter les valeurs par défaut proposées dans cette boîte de dialogue et choisir l'option [Start]. Le serveur Web est alors actif. [WebMatrix] va alors lancer le navigateur par défaut de la machine sur laquelle il se trouve et demander l'URL http://localhost:8080/demo1.aspx :



Il est possible d'utiliser le serveur Cassini en-dehors de [WebMatrix]. L'exécutable du serveur se trouve dans <WebMatrix>\<version>\WebServer.exe où <WebMatrix> est le répertoire d'installation de [WebMatrix] et <version> son n° de version :



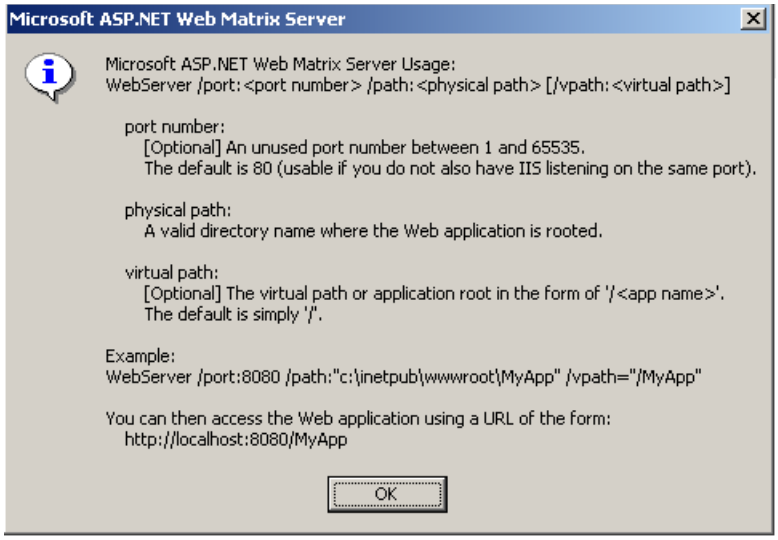
Ouvrons une fenêtre Dos et positionnons-nous dans le dossier du serveur Cassini :

```
E:\Program Files\Microsoft ASP.NET Web Matrix\v0.6.812>dir
...
29/05/2003  11:00               53 248 WebServer.exe
...
```

Lançons [WebServer.exe] sans paramètres :

```
E:\Program Files\Microsoft ASP.NET Web Matrix\v0.6.812>webserver
```

Nous obtenons une fenêtre d'aide :



L'application [WebServer] appelée également serveur web Cassini admet trois paramètres :

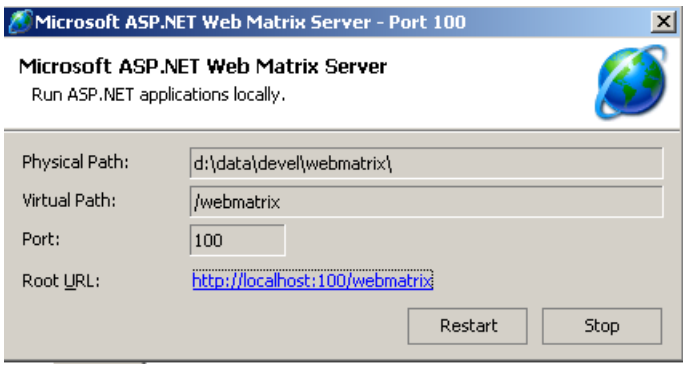
- **/port** : n° de port du service web. Peut-être quelconque. A par défaut la valeur 80
- /**path** : chemin physique d'un dossier du disque
- /**vpath** : dossier virtuel associé au dossier physique précédent. On prêtera attention au fait que la syntaxe n'est pas /path=chemin mais /vpath:chemin, contrairement à ce que dit l'exemple [Example] du panneau d'aide ci-dessus.

Plaçons le fichier [demo1.aspx] dans le dossier suivant :

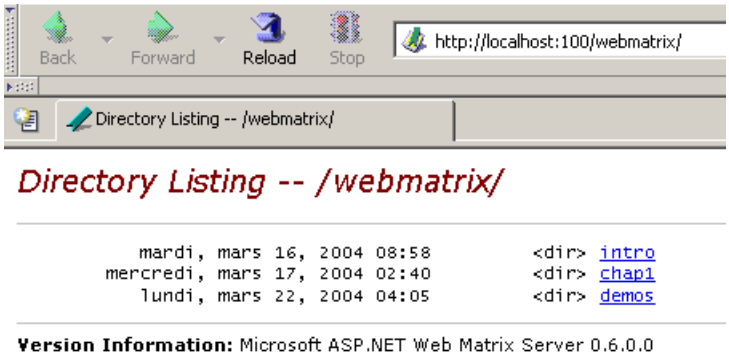
Associons au dossier physique [d:\data\devel\webmatrix] le dossier virtuel [/webmatrix]. Le serveur web pourrait être lancé de la façons suivante :

```
E:\Program Files\Microsoft ASP.NET Web Matrix\v0.6.812>webserver /port:100 /path:"d:\data\devel\webmatrix"
/vpath:"/webmatrix"
```

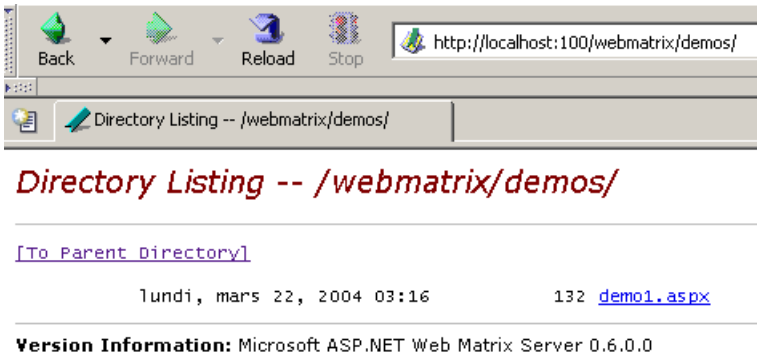
Le serveur Cassini est alors actif et son icône apparaît dans la barre des tâches. Si on double-clique dessus :



On retrouve les paramètres de lancement du serveur. On dispose également de la possibilité d'arrêter [Stop] ou de relancer [Restart] le serveur web. Si on clique sur le lien [Root URL], on obtient la racine de l'arborescence web du serveur dans un navigateur :



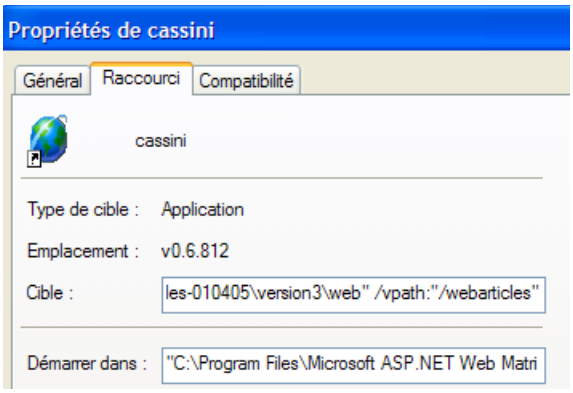
Suivons le lien [demos] :



puis le lien [demo1.aspx] :

On voit donc que si le dossier physique P=[d:\data\devel\webmatrix] a été associé au dossier virtuel V=[/webmatrix] et que le serveur travaille sur le port 100, la page web [demo1.aspx] se trouvant physiquement dans [P\demos] sera accessible localement via l'URL [http://localhost:100/V/demos/demo1.aspx].

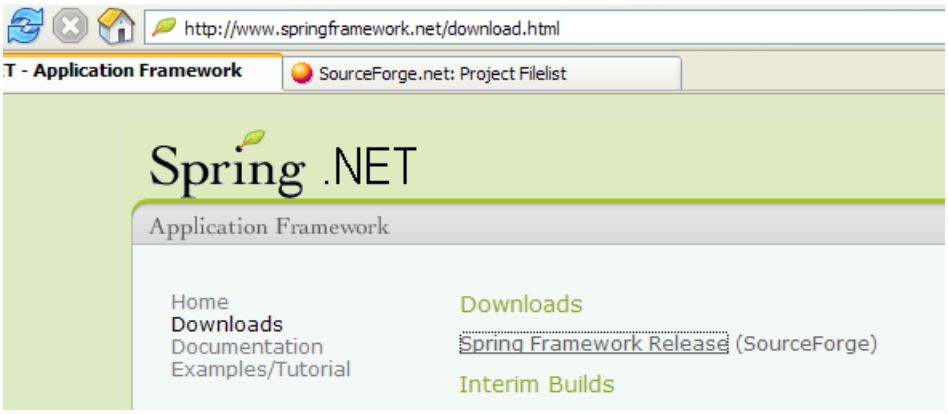
Afin de ne pas être obligé de passer par une fenêtre DOS pour lancer le serveur Cassini, on pourra créer un raccourci vers l'exécutable du serveur avec des propriétés analogues aux suivantes :



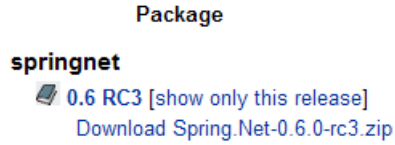
`Cible` "C:\Program Files\Microsoft ASP.NET Web Matrix\v0.6.812\WebServer.exe" /port:80 /path:"D:\data\serge\travail\2004-2005\aspnet\webarticles-010405\version3\web" /vpath:"/webarticles"
`Démarrer dans` "C:\Program Files\Microsoft ASP.NET Web Matrix\v0.6.812"

7.3 Où trouver Spring ?

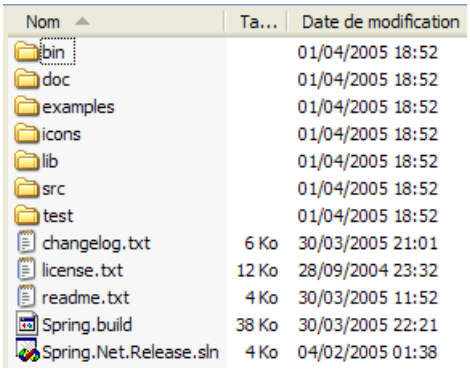
Le site principal de Spring est [http://www.springframework.org/]. C'est le site de la version Java. La version .Net en cours de développement (avril 2005) est à l'url [http://www.springframework.net/].



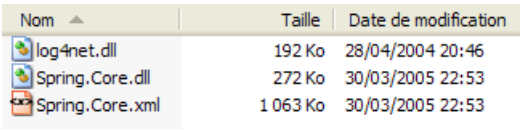
Le site de téléchargement est chez [SourceForge] :



Une fois le zip ci-dessus récupéré, le décompresser :



Dans ce document, nous n'avons utilisé que le contenu du dossier [bin] de la distribution :



Dans un projet Visual Studio utilisant Spring, il faut faire systématiquement deux choses :

- mettre les fichiers ci-dessus dans le dossier [bin] du projet
- ajouter au projet une référence à l'assembly [Spring.Core.dll]

7.4 Où trouver Nunit ?

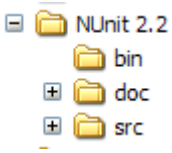
Le site principal de Nunit est [http://www.nunit.org/]. La version disponible en avril 2005 est la 2.2.0 :

Current Production Release

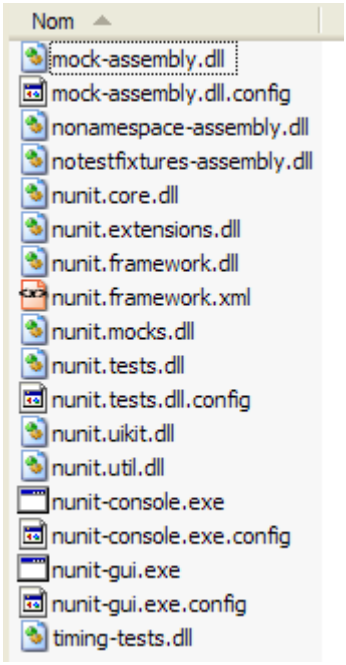
Production releases are stable releases that have gone through a beta review and had most of the problems fixed. Most people will want to use a production release.

NUnit 2.2 Production Release	
win	NUnit-2.2.0.msi
mono	NUnit-2.2.0-mono.zip
src	NUnit-2.2.0-src.zip

Téléchargez cette version et installez la. L'installation crée un dossier où on trouvera la version graphique de test :

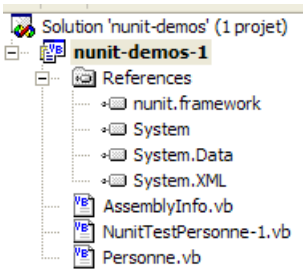


Ce qui est intéressant se trouve dans le dossier [bin] :



La flèche ci-dessus désigne l'utilitaire graphique de test. L'installation a également ajouté de nouveaux éléments au référentiel d'assemblages de Visual Studio que nous allons découvrir maintenant.

Créons le projet Visual Studio suivant :



La classe testée est dans [Personne.vb] :

```
Public Class Personne

  ' champs privés
  Private _nom As String
  Private _age As Integer

  ' constructeur par défaut
  Public Sub New()
  End Sub

  ' propriétés associées aux champs privés
  Public Property nom() As String
    Get
      Return _nom
    End Get
    Set(ByVal Value As String)
      _nom = Value
    End Set
  End Property

  Public Property age() As Integer
    Get
      Return _age
    End Get
    Set(ByVal Value As Integer)
      _age = Value
    End Set
  End Property

  ' chaîne d'identité
  Public Overrides Function tostring() As String
    Return String.Format("[{0},{1}]", nom, age)
  End Function

  ' méthode init
```

```
  Public Sub init()
    Console.WriteLine("init personne {0}", Me.ToString)
  End Sub

  ' méthode close
  Public Sub close()
    Console.WriteLine("destroy personne {0}", Me.ToString)
  End Sub

End Class
```

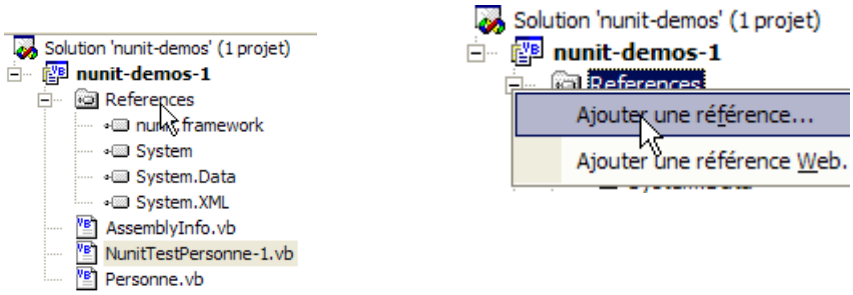
La classe de test est dans [NunitTestPersonne-1.vb] :

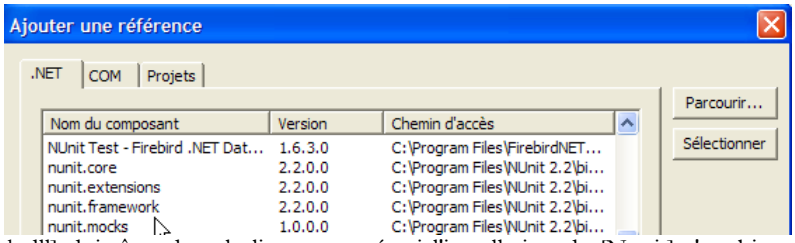
```
Imports System
Imports NUnit.Framework

<TestFixture()> _
 Public Class NunitTestPersonne

  ' objet testé
  Private personne1 As Personne

  <SetUp()> _
  Public Sub init()
    ' on crée une instance de Personne
    personne1 = New Personne
    ' log
    Console.WriteLine("setup test")
  End Sub

  <Test()> _
  Public Sub demo()
    ' log écran
    Console.WriteLine("début test")
    ' init personne1
    With personne1
      .nom = "paul"
      .age = 10
    End With
    ' tests
    Assert.AreEqual("paul", personne1.nom)
    Assert.AreEqual(10, personne1.age)
    ' log écran
    Console.WriteLine("fin test")
  End Sub

  <TearDown()> _
  Public Sub destroy()
    ' suivi
    Console.WriteLine("teardown test")
  End Sub

End Class
```

Plusieurs choses sont à noter :

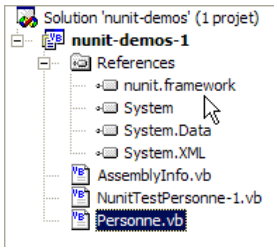
- les méthodes sont dotées d'attributs tels <Setup()>, <TearDown()>, ...
- pour que ces attributs soient reconnus, il faut que :
  - le projet référence l'assembly [nunit.framework.dll]
  - la classe de test importe l'espace de noms [NUnit.Framework]

La référence est obtenue en cliquant droit sur [References] dans l'explorateur de solutions :





L'assembly [nunit.framework.dll] doit être dans la liste proposée si l'installation de [Nunit] s'est bien passée. Il suffit de double-cliquer sur l'assembly pour l'ajouter au projet :



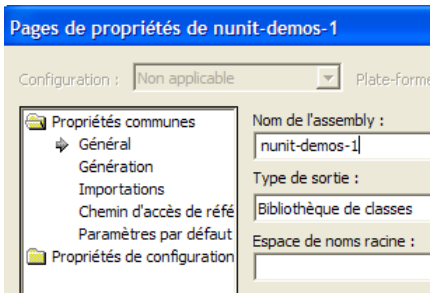
Ceci fait, la classe de test [NunitTestPersonne] doit importer l'espace de noms [NUnit.Framework] :

```
Imports NUnit.Framework
```

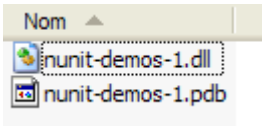
Les attributs de la classe de test [NunitTestPersonne] doivent alors être reconnus.

- l'attribut <Test()> désigne une méthode à tester
- l'attribut <Setup()> désigne la méthode à exécuter avant chaque méthode testée
- l'attribut <TearDown()> désigne la méthode à exécuter après chaque méthode testée
- la méthode **Assert.AreEqual** permet de tester l'égalité de deux entités.Il existe de nombreuses autres méthodes de type **Assert.xx**.
- l'utilitaire NUnit arrête l'exécution d'une méthode testée dès qu'une méthode [Assert] échoue et affiche un message d'erreur. Sinon il affiche un message de réussite.

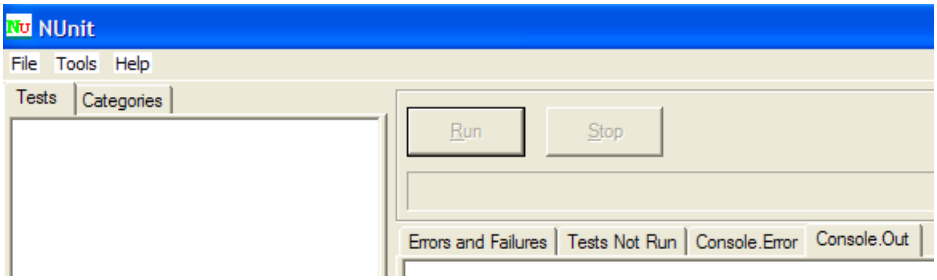
Configurons notre projet pour qu'il génère une DLL :



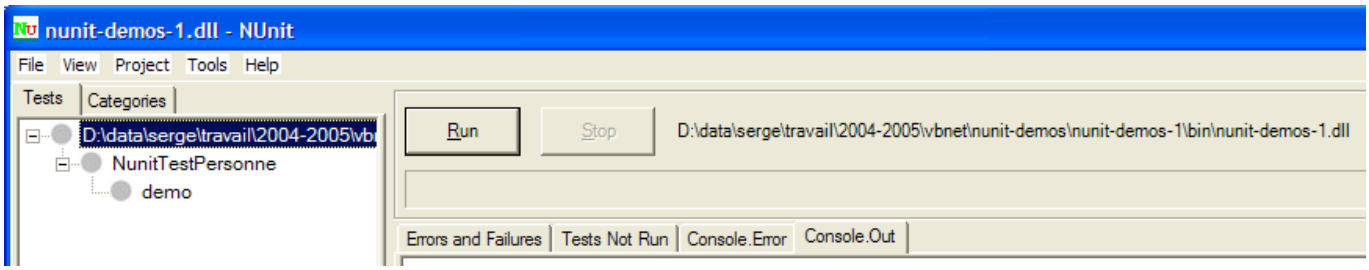
La DLL générée s'appellera [nunit-demos-1.dll] et sera placée par défaut dans le dossier [bin] du projet. Générons notre projet. Nous obtenons dans le dossier [bin] :



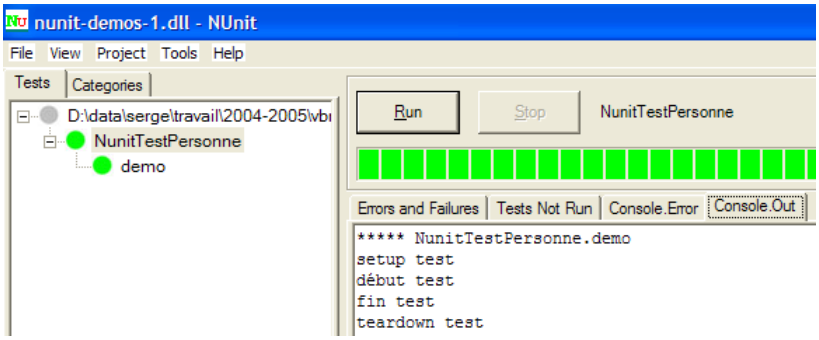
Lançons maintenant l'utilitaire de test graphique Nunit. Rappelons qu'il se trouve dans <Nunit>\bin et qu'il s'appelle [nunit-gui.exe]. <Nunit> désigne le dossier d'installation de [Nunit]. On obtient l'interface suivante :



Utilisons l'option de menu [File/Open] pour charger la DLL [nunit-demos-1.dll] de notre projet :



[Nunit] est capable de détecter automatiquement les classes de test qui se trouvent dans la DLL chargée. Ici, il trouve la classe [NunitTestPersonne]. Il affiche alors toutes les méthodes de la classe ayant l'attribut <Test()>. Le bouton [Run] permet de lancer les tests sur l'objet sélectionné. Si celui-ci est la classe [NunitTestPersonne], toutes les méthodes affichées sont testées. On peut demander le test d'une méthode particulière en la sélectionnant et en demandant son exécution par [Run]. Demandons l'exécution de la classe :



Un test réussi sur une méthode est symbolisé par un point vert à côté de la méthode dans la fenêtre de gauche. Un test raté est symbolisé par un point rouge.

La fenêtre [Console.Out] à droite montre les affichages écran produits par les méthodes testées. Ici, nous avons voulu suivre le déroulement d'un test :

```
1. setup test
2. début test
3. fin test
4. teardown test
```

- la ligne 1 montre que la méthode d'attribut <Setup()> est exécutée avant le test
- les lignes 2-3 sont produites par la méthode [demo] testée (voir le code plus haut)
- la ligne 4 montre que la méthode d'attribut <TearDown()> est exécutée après le test