Construction d'applications à trois couches avec

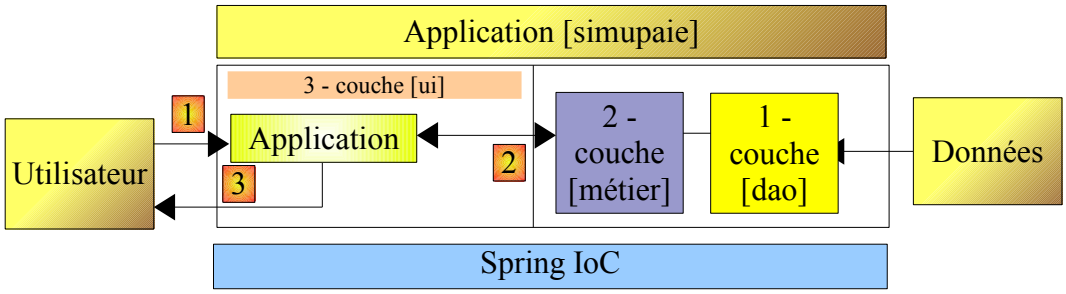
VB.Net 2005
ASP.Net 2005
Spring.Net

serge.tahe@istia.univ-angers.fr, février 2007

1 Introduction

Nous souhaitons écrire une application .NET permettant à un utilisateur de faire des simulations de calcul de la paie des assistantes maternelles de l'association " Maison de la petite enfance " d'une commune. Nous nous intéresserons autant à l'organisation du code DotNet de l'application qu'au code lui-même.

L'application finale, que nous appellerons [SimuPaie] aura la structure à trois couches suivante :



- la couche [1-dao] (dao=Data Access Object) s'occupera de l'accès aux données. Celles-ci seront placées dans une base de données.
- la couche [2-métier] s'occupera de l'aspect métier de l'application, le calcul de la paie.
- la couche [3-ui] (ui=User Interface) s'occupera de la présentation des données à l'utilisateur et de l'exécution de ses requêtes. Nous appelons [Application] l'ensemble des modules assurant cette fonction. Elle est l'interlocuteur de l'utilisateur.
- les trois couches seront rendues indépendantes grâce à l'utilisation d'interfaces .NET
- l'intégration des différentes couches sera réalisée par **Spring IoC**

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

1. le client fait une demande à l'application.
2. l'application traite cette demande. Pour ce faire, elle peut avoir besoin de l'aide de la couche [métier] qui elle-même peut avoir besoin de la couche [dao] si des données doivent être échangées avec la base de données. L'application reçoit une réponse de la couche [métier].
3. selon celle-ci, elle envoie la vue (= la réponse) appropriée au client.

L'interface présentée à l'utilisateur peut avoir diverses formes :

1. une application console : dans ce cas, la vue est une suite de lignes de texte.
2. une application graphique windows : dans ce cas, la vue est une fenêtre windows
3. une application web : dans ce cas, la vue est une page HTML
4. ...

Nous écrirons plusieurs versions de cette application :

1. une versions windows VB.NET avec un seul formulaire où tout est fait dans une unique couche, celle de l'interface utilisateur. C'est la méthode la plus rapide, dite RAD (Rapid Application Development). Cette méthode est justifiée lorsque le code n'est pas destiné à être réutilisé. Il est simplement souvent difficile d'affirmer qu'un code écrit là ne sera pas utile ailleurs.
2. une versions windows VB.NET proche de la précédente mais utilisant de nouvelles fonctionnalités apportées par VB.NET 2005.
3. une version web ASPNET qui sera quasiment un copier / coller de la version windows précédente. On veut montrer là qu'une application web ASP.NET peut être développée avec la même philosophie qu'une application windows .NET classique. C'est là l'intérêt majeur de cette technologie. Elle permet de migrer rapidement vers le web certaines applications windows, celles à un ou deux formulaires comme celle que nous allons écrire. Pour des applications plus complexes, le portage vers le web est plus lourd et nécessite de la part des développeurs, une compréhension plus fine des mécanismes sous-tendant les échanges sur le web.
4. une version ASP.NET proche de la précédente et qui commence à introduire la notion de séparation des tâches d'une application web : présentation des données au client, calcul métier, accès aux données persistantes.
5. la même version que précédemment avec des extensions AJAX.
6. une versions windows VB.NET avec une interface utilisateur identique à celle de la version 1 mais s'appuyant sur une architecture à trois couches.

7. une versions windows ASP.NET s'appuyant sur l'architecture à trois couches développée dans la version 6.
8. la même version que précédemment avec des extensions AJAX.
9. une version web ASP.NET multi-vues et mono-page.
10. une version web ASP.NET multi-vues et multi-pages.

Pré-requis

Dans une échelle [débutant-intermédiaire-avancé], ce document est dans la partie [intermédiaire]. Sa compréhension nécessite divers pré-requis qu'on pourra trouver dans certains des documents que j'ai écrits :

1. **programmation ASP.NET** [http://tahe.developpez.com/dotnet/aspnet/vol1] et [http://tahe.developpez.com/dotnet/aspnet/vol2]
2. **langage VB.net** : [http://tahe.developpez.com/dotnet/vbnet/] : classes, interfaces, héritage, polymorphisme
3. [**Spring IoC**], disponible à l'url [http://tahe.developpez.com/dotnet/springioc]. Présente les bases de l'inversion de contrôle (Inversion of Control) ou injection de dépendances (Dependency Injection) du framework **Spring.Net** [http://www.springframework.net].
4. [**Construction d'une application web à trois couches avec Spring et VB.NET - Parties 1 et 2**], disponibles aux url [http://tahe.developpez.com/dotnet/web3tier-part1] et [http://tahe.developpez.com/dotnet/web3tier-part2]. Ces deux articles présentent une application simplifiée d'achats de produits sur le web. Son architecture 3tier implémente le modèle MVC.

Des conseils de lecture sont parfois donnés au début des paragraphes de ce document. Ils référencent les documents précédents.

Outils

Les outils utilisés dans cet étude de cas sont librement disponibles sur le web. Ce sont les suivants :

- **Visual Basic Express 2005, Visual Web Developer Express 2005**, **SQL Server Express 2005** disponibles à l'url [http://www.microsoft.com/france/msdn/vstudio/express/default.mspx].
- Visual Web Developer Express 2005 est livré avec un serveur web. On peut également installer le serveur web **Cassini** disponible à l'url [http://www.asp.net/Projects/Cassini/Download/] - voir également annexes de [4, Partie 1].
- **Spring IoC** pour l'instanciation des services nécessaires à l'application, disponible à l'url [http://www.springframework.net/] - voir également [3]
- **Ibatis SqlMap** pour la couche d'accès aux données du SGBD [http://ibatis.apache.org] - voir annexes de [4, Partie 2]
- **SQL Server Management Studio Express** [http://msdn.microsoft.com/vstudio/express/sql/] qui permet d'administrer graphiquement SQL Server Express 2005.
- **NUnit** : [http://www.nunit.org] pour les tests unitaires.

Pour approfondir cette étude de cas et en particulier le concept MVC (Modèle - Vue - Contrôleur), on pourra lire les articles suivants :

A) [M2VC-win, un moteur MVC pour des applications windows .NET], disponible à l'url [http://tahe.developpez.com/dotnet/m2vc-win]. [M2VC] est un framework MVC pour des applications Windows, inspiré de [Struts]. M2VC signifie **M**oteur **MVC**. On peut utiliser M2VC lorsqu'on veut donner une architecture MVC à une application windows.

B) [Construction d'une application windows à trois couches avec Spring, M2VC-win et VB.NET], disponible à l'url [http://tahe.developpez.com/dotnet/win3tier]. Cet article reprend l'application web de [article 4] pour en faire une application windows "standalone". L'architecture MVC initiale de l'application web est reproduite grâce au moteur M2VC décrit dans [article A].

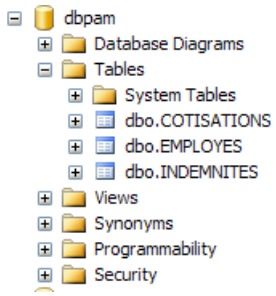
C) [Construction en VB.NET d'une application web MVC multi-couches formée d'un client riche et d'un service web], disponible à l'url [http://tahe.developpez.com/dotnet/web3tier-part3]. Cet article décrit la transformation de l'application web de [article 4] en une application client-serveur formée côté client, du client riche décrit dans [article A] et côté serveur, de services web.

D) [M2VC-aspnet, un moteur MVC pour ASP.NET], disponible à l'url [http://tahe.developpez.com/dotnet/m2vc-aspnet]. Cet article décrit un moteur MVC pour les applications ASP.NET. Appelé [M2VC-aspnet], il s'inspire du moteur [M2VC-win]. Son utilisation est illustrée par la réécriture de l'application web étudiée dans [article 4].

2 L'étude de cas

2.1 La base de données

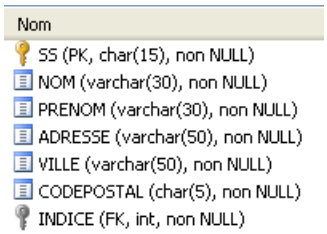
Les données statiques utiles pour construire la fiche de paie sont placées dans une base de données SQL Server Express nommée **dbpam** (**pam**=Paie Assistante Maternelle). Cette base a un administrateur appelé **sa** ayant le mot de passe **msde**.



La base a trois tables, **EMPLOYES, COTISATIONS** et **INDEMNITES,** dont la structure est la suivante :

Table **EMPLOYES** : rassemble des informations sur les différentes assistantes maternelles

Structure :



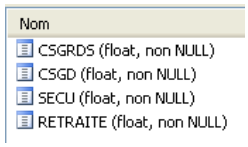
SS	numéro de sécurité sociale de l'employé - clé primaire
NOM	nom de l'employé
PRENOM	son prénom
ADRESSE	son adresse
VILLE	sa ville
CODEPOSTAL	son code postal
INDICE	son indice de traitement - clé étrangère sur le champ [INDICE] de la table [INDEMNITES]

Son contenu pourrait être le suivant :

SS	NOM	PRENOM	ADRESSE	VILLE	CODEPOSTAL	INDICE
254104940426058	Jouveinal	Marie	5 rue des Oiseaux	St Corentin	49203	2
260124402111742	Laverti	Justine	la Brûlerie	St Marcel	49014	1

Table **COTISATIONS** : rassemble les taux des cotisations sociales prélevées sur le salaire

Structure :



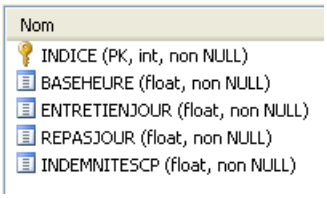
CSGRDS	pourcentage : contribution sociale généralisée + contribution au remboursement de la dette sociale
CSGD	pourcentage : contribution sociale généralisée déductible
SECU	pourcentage : sécurité sociale
RETRAITE	pourcentage : retraite complémentaire + assurance chômage

Son contenu pourrait être le suivant :

CSGRDS	CSGD	SECU	RETRAITE
3,49	6,15	9,39	7,88

Les taux des cotisations sociales sont indépendants du salarié. La table précédente n'a qu'une ligne.

Table **INDEMNITES** : rassemble les différentes indemnités dépendant de l'indice de l'employé



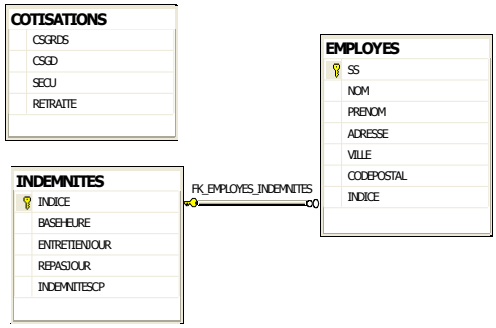
INDICE	indice de traitement - clé primaire
BASEHEURE	prix net en euro d'une heure de garde
ENTRETIENJOUR	indemnité d'entretien en euro par jour de garde
REPASJOUR	indemnité de repas en euro par jour de garde
INDEMNITESCP	indemnité de congés payés. C'est un pourcentage à appliquer au salaire de base.

Son contenu pourrait être le suivant :

INDICE	BASEHEURE	ENTRETIENJOUR	REPASJOUR	INDEMNITESCP
1	1,93	2	3	12
2	2,1	2,1	3,1	15

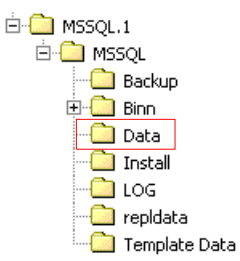
On notera que les indemnités peuvent varier d'une assistante maternelle à une autre. Elles sont en effet associées à une assistante maternelle précise via l'indice de traitement de celle-ci. Ainsi Mme Marie Jouveinal qui a un indice de traitement de 2 (table EMPLOYES) a un salaire horaire de 2,1 euros (table INDEMNITES).

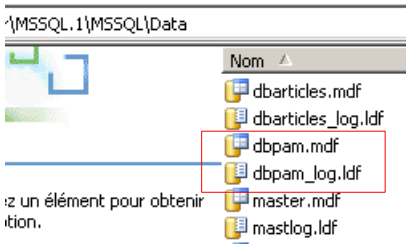
Les relations entre les trois tables sont les suivantes :



Il y a une relation de clé étrangère entre la colonne EMPLOYES(INDICE) et la colonne INDEMNITES(INDICE).

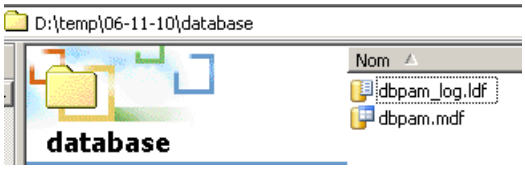
La base de données [dbpam] ainsi créée donne naissance à deux fichiers dans le dossier de SQL Server Express :



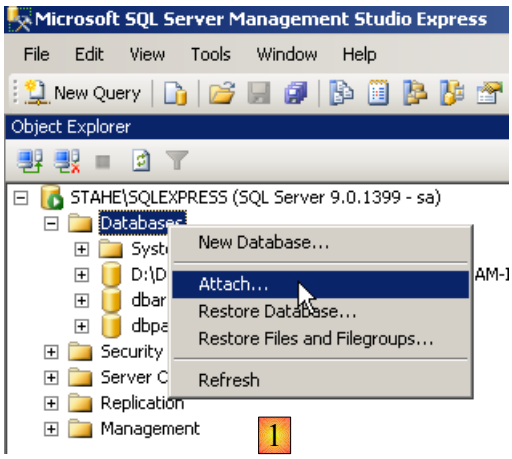


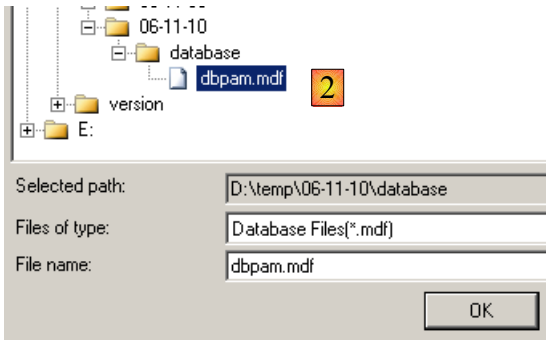
Les fichiers [dbpam.mdf, dbpam_log.ldf] peuvent être transportés sur une autre machine et être réattachés au SGBD SQL Server Express de cette machine. Voici comment procéder :

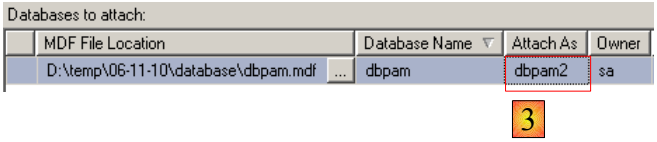
- les fichiers de la BD [dbpam] sont dupliqués dans un dossier

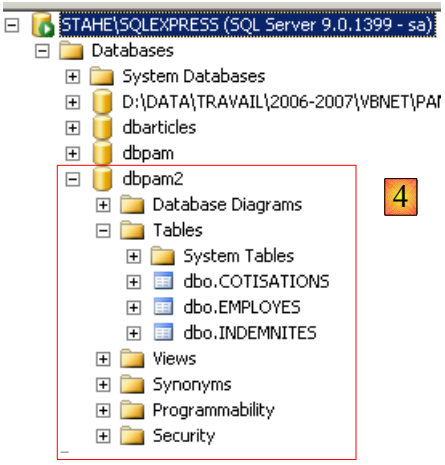


- on lance SQL Server Express
- avec le client SQL Server Management Studio Express, on attache le fichier [dbpam.mdf] au SGBD :









1. clic droit sur [Databases] / Attach
2. choix du fichier [dbpam.mdf] via un bouton [Add] non représenté
3. le fichier attaché va donner naissance à une BD qui ne doit pas déjà exister. Ici, dans le champ [Attach As] on a donné le nom [dbpam2] à la nouvelle BD.
4. on voit la nouvelle BD et ses tables

Cette technique de l'attachement d'une BD est pratique pour transporter une BD d'un poste à un autre et nous l'utiliserons ici.

2.2 Mode de calcul du salaire d'une assistante maternelle

Nous présentons maintenant le mode de calcul du salaire mensuel d'une assistante maternelle. Nous prenons pour exemple, le salaire de Mme Marie Jouveinal qui a travaillé 150 h sur 20 jours pendant le mois à payer.

Les éléments suivants sont pris en compte :	`[TOTALHEURES]: total des heures travaillées dans le mois`
	
	`[TOTALJOURS]: total des jours travaillés dans le mois`
		`[TOTALHEURES]=150`
		`[TOTALJOURS]= 20`

Le salaire de base de l'assistante maternelle est donné par la formule suivante :	`[SALAIREBASE]=([TOTALHEURES]*[BASEHEURE])*(1+[INDEMNITESCP]/100)`	`[SALAIREBASE]=(150*[2.1])*(1+0.15) = 362,25`

Un certain nombre de cotisations sociales doivent être prélevées sur ce salaire de base :	`Contribution sociale généralisée et contribution au remboursement de la dette sociale : [SALAIREBASE]*[CSGRDS/100]`
	
	`Contribution sociale généralisée déductible : [SALAIREBASE]*[CSGD/100]`
	
	`Sécurité sociale, veuvage,`
		`CSGRDS : 12,64`
		
		`CSGD : 22,28`
		
		`Sécurité sociale : 34,02`
		
		`Retraite : 28,55`

Les éléments suivants sont pris en compte : [TOTALHEURES]: total des heures travaillées dans le mois [TOTALJOURS]: total des jours travaillés dans le mois vieillesse : [SALAIREBASE]*[SECU/100] Retraite Complémentaire + AGPF + Assurance Chômage : [SALAIREBASE]*[RETRAITE/100]		[TOTALHEURES]=150 [TOTALJOURS]= 20
Total des cotisations sociales :	[COTISATIONSSOCIALES]=[SALAIREBASE]*(CSGRDS+CSGD+SECU+RETRAITE)/100	[COTISATIONSSOCIALES]=97,48
Par ailleurs, l'assistante maternelle a droit, chaque jour travaillé, à une indemnité d'entretien ainsi qu'à une indemnité de repas. A ce titre elle reçoit les indemnités suivantes :	[INDEMNITÉS]=[TOTALJOURS]*(ENTRETIENJOUR+REPASJOUR)	[INDEMNITES]=104
Au final, le salaire net à payer à l'assistante maternelle est le suivant :	[SALAIREBASE]-[COTISATIONSSOCIALES]+[INDEMNITÉS]	[salaire NET]=368,77

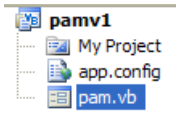
3 L'application [SimuPaie] – version 1 – VB.NET

Lectures conseillées : référence [2], chapitre 6 : " Accès aux bases de données "

Nous présentons ici les éléments de l'application simplifiée de calcul de paie que nous souhaitons écrire.

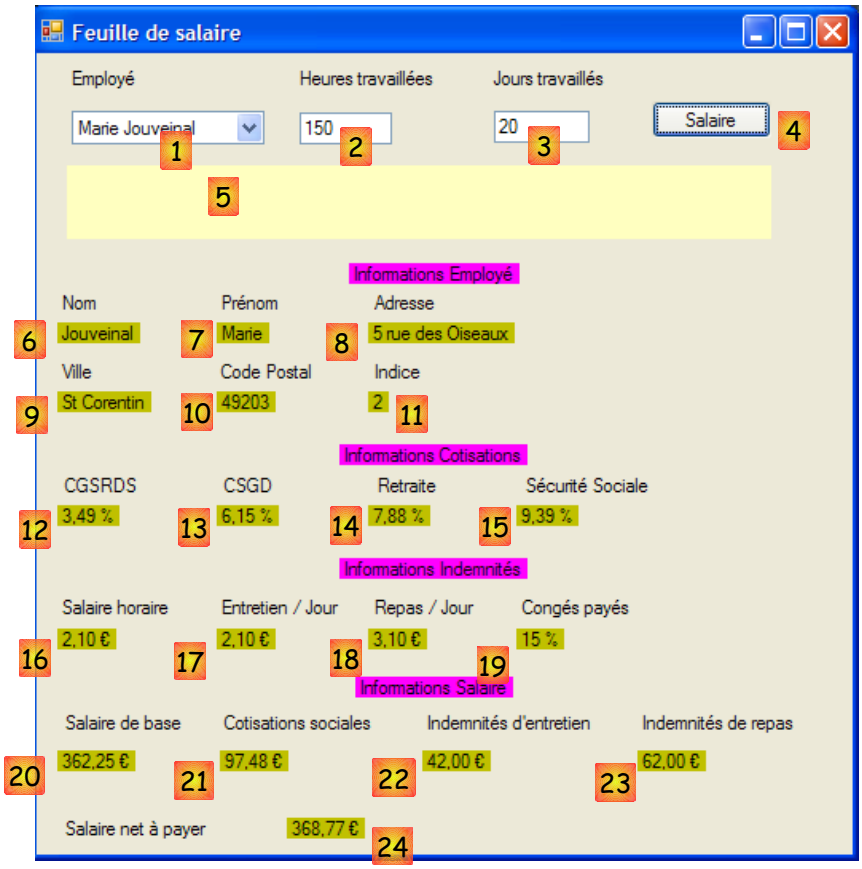
3.1 Le formulaire

Le projet VB Express 2005 pourrait être le suivant :



- **pam.vb** : le formulaire décrit ci-dessous
- **app.config** : le fichier de configuration de l'application

L'interface graphique sera la suivante :



N°	Type	Nom	Rôle
1	ComboBox	ComboBoxEmployes	Contient la liste des noms des employés
2	TextBox	TextBoxHeures	Nombre d'heures travaillées – nombre réel
3	TextBox	TextBoxJours	Nombre de jours travaillés – nombre entier
4	Button	ButtonSalaire	Demande le calcul du salaire
5	Label	LabelErreurs	Messages d'informations à destination de l'utilisateur
6	Label	LabelNom	Nom de l'employé sélectionné en (1)

N°	Type	Nom	Rôle
7	Label	LabelPrénom	Prénom de l'employé sélectionné en (1)
8	Label	LabelAdresse	Adresse
9	Label	LabelVille	Ville
10	Label	LabelCP	Code postal
11	Label	LabelIndice	Indice
12	Label	LabelCSGRDS	Taux de cotisation CSGRDS
13	Label	LabelCSGD	Taux de cotisation CSGD
14	Label	LabelRetraite	Taux de cotisation Retraite
15	Label	LabelSS	Taux de cotisation Sécurité Sociale
16	Label	LabelSH	Salaire horaire de base pour l'indice indiqué en (11)
17	Label	LabelEJ	Indemnité journalière d'entretien pour l'indice indiqué en (11)
18	Label	LabelRJ	Indemnité journalière de repas pour l'indice indiqué en (11)
19	Label	LabelCongés	Taux d'indemnités de congés payés à appliquer au salaire de base
20	Label	LabelSB	Montant du salaire de base
21	Label	LabelCS	Montant des cotisations sociales à verser
22	Label	LabelIE	Montant des indemnités d'entretien de l'enfant gardé
23	Label	LabelIR	Montant des indemnités de repas de l'enfant gardé
24	Label	LabelSN	Salaire net à payer à l'employé(e)

3.2 Configuration de l'application

Le fichier [app.config] qui configure l'application sera le suivant :

```
1.  <?xml version="1.0" encoding="utf-8" ?>
2.  <configuration>
3.    <connectionStrings>
4.      <add name="dbpam" connectionString="Data Source=.\SQLEXPRESS;AttachDbFilename=C:\data\2006-
    2007\vbnet\pam\database\dbpam.mdf;User Id=sa;Password=msde;Connect Timeout=30;"
5.             providerName="System.Data.SqlClient" />
6.    </connectionStrings>
7.    <appSettings>
8.      <add key="selectEMPLOYES" value="select NOM,PRENOM,SS from EMPLOYES"/>
9.      <add key="selectEMPLOYE" value="select
    NOM,PRENOM,ADRESSE,VILLE,CODEPOSTAL,EMPLOYES.INDICE,BASEHEURE,ENTRETIENJOUR,REPASJOUR,INDEMNITESCP from
    EMPLOYES,INDEMNITES where SS=@SS and EMPLOYES.INDICE=INDEMNITES.INDICE"/>
10.     <add key="selectCOTISATIONS" value="select CSGRDS,CSGD,SECU,RETRAITE from COTISATIONS"/>
11.   </appSettings>
12. </configuration>
```

- ligne 4 : la chaîne de connexion à la base de données. Celle-ci a diverses composantes :

`Data Source`	l'identifiant du SGBD qui gère la source de données sous la forme [\\machine\instance SQL Server]. Ici [.\SQLEXPRESS] désigne l'instance nommée [SQLEXPRESS] sur la machine locale.
`AttachDbFileName`	la base de données gérée sera obtenue par attachement d'un fichier .mdf (voir paragraphe 2.1, page 5).
`User Id`	le propriétaire de la connexion
`Password`	son mot de passe
`Connect Timeout`	durée en secondes au terme de laquelle la tentative de connexion est abandonnée si elle n'a pas abouti.

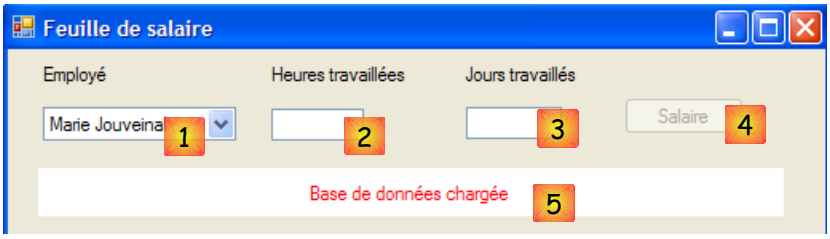
- ligne 8 : l'ordre SELECT pour obtenir l'identité de tous les employés
- ligne 9 : l'ordre SELECT pour obtenir les informations concernant un employé particulier
- ligne 10 : l'ordre SELECT pour obtenir les taux de cotisations

Le constructeur du formulaire [pam.vb] pourra récupérer ces éléments de la façon suivante :

```
1.    ' champs privés de l'instance du formulaire
2.     ...
3.     ' requêtes SQL
4.     Private SelectEmployes As String
5.     Private SelectEmploye As String
6.     Private SelectCotisations As String
7.     ' chaîne de connexion à la BD
8.     Private DatabaseConnectionString As String
9.
10.      ' constructeur
11.   ...
12.      ' chaîne de connexion
13.      DatabaseConnectionString = ConfigurationManager.ConnectionStrings("dbpam").ConnectionString
14.      ' requêtes SQL
15.      SelectEmployes = ConfigurationManager.AppSettings("selectEMPLOYES")
16.      SelectEmploye = ConfigurationManager.AppSettings("selectEMPLOYE")
17.      SelectCotisations = ConfigurationManager.AppSettings("selectCOTISATIONS")
18.  ...
```

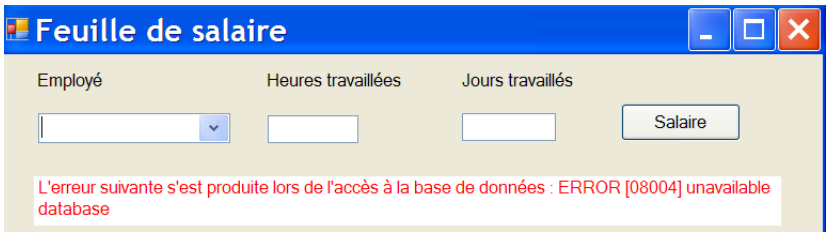
3.3 Démarrage de l'application

Lorsque l'application VB démarre, elle va chercher dans la base de données décrite au paragraphe 2.1, page 3, les noms des employés pour les mettre dans le ComboBox [1] :



- le ComboBox [1] a été rempli
- le bouton [4] est désactivé. Il sera activé lorsque les champs [2] et [3] seront non vides
- le label [5] indique que la base des données a été lue correctement

S'il se produit des erreurs d'accès à la base de données, le label [5] l'indique :



Question 1 : écrire le constructeur du formulaire qui, exécuté au démarrage de l'application, garantit le fonctionnement précédent.

3.4 Calcul du salaire

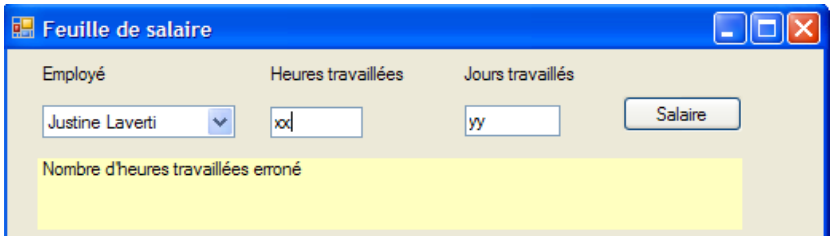
Pour calculer un salaire, l'utilisateur :

- sélectionne un employé en [1]
- saisit le nombre d'heures travaillées en [2]. Ce nombre peut être décimal, comme 2,5 pour 2 h 30 mn.
- saisit le nombre de jours travaillés en [3]. Ce nombre est entier.
- demande le salaire avec le bouton [4]

Le clic sur le bouton [4] provoque l'exécution du gestionnaire :

```
Private Sub ButtonSalaire_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles ButtonSalaire.Click
```

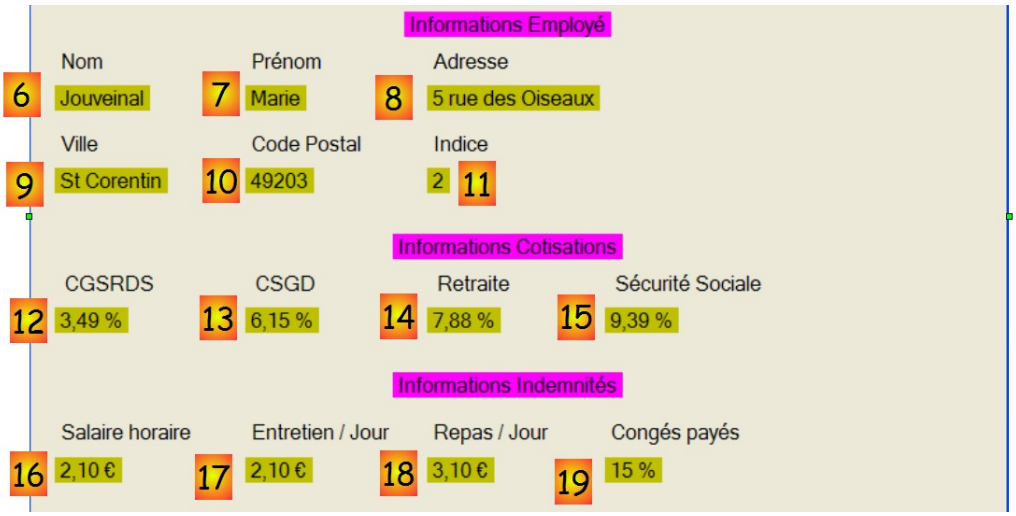
Le gestionnaire [ButtonSalaire_Click] commence par vérifier la validité des saisies faites en [2] et [3]. Si l'une des deux est incorrecte, l'erreur est signalée comme ci-dessous :



Question 2 : écrire le code de la procédure [ButtonSalaire_Click] qui vérifie la validité des saisies faites en [2] et [3].

Une fois les saisies [2] et [3] vérifiées et trouvées valides, l'application doit afficher des données complémentaires sur l'utilisateur sélectionné en [1]. Ces données sont trouvées dans les différentes tables de la base de données.

Question 3 : écrire le code de la procédure [ButtonSalaire_Click] qui va permettre d'obtenir les informations [6] à [19] ci-dessous.



Une fois les données ci-dessus obtenues, le salaire peut être calculé selon l'algorithme métier écrit au paragraphe 2.2, page 6.

Question 4 : écrire le code de la procédure [ButtonSalaire_Click] qui va permettre d'obtenir les informations [20] à [24] ci-dessous.

Question 5 : écrire les gestionnaires d'événements permettant de gérer l'état du bouton [Salaire] qui doit être inactif lorsque l'un des champs (1) ou (2) est vide ou blanc.

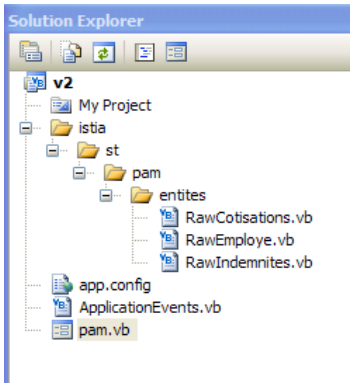
4 L'application [SimuPaie] – version 2 – VB.NET

Cette version modifie la version précédente de la façon suivante :

- elle tire avantage de nouvelles fonctionnalités apportées par VB.NET 2005
- elle met en cache les données les plus demandées de la base de données afin d'améliorer les performances

4.1 Le projet VB.NET

Le nouveau projet VB Express 2005 est le suivant :



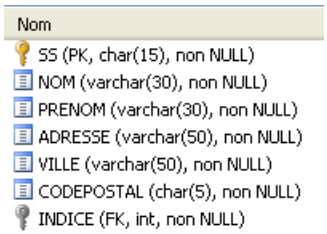
- **pam.vb** : le formulaire de l'application précédente
- **app.config** : le fichier de configuration de l'application
- **ApplicationEvents.vb** : le code de gestion des événements principaux de l'application (démarrage / arrêt)
- [**RawCotisations.vb**, **RawEmploye.vb**, **RawIndemnites.vb**] : des classes destinées à encapsuler les données des tables [COTISATIONS, EMPLOYES, INDEMNITES]

4.2 Les objets de l'application

Les entités (objets) nécessaires à la couche [dao] ont été rassemblées dans le dossier [entites] du projet. Nous les décrivons maintenant.

4.2.1 La classe [RawEmploye]

Un objet de type [RawEmploye] sert à encapsuler une ligne de la table [EMPLOYES] :



SS	numéro de sécurité sociale de l'employé - clé primaire
NOM	nom de l'employé
PRENOM	son prénom
ADRESSE	son adresse
VILLE	sa ville
CODEPOSTAL	son code postal
INDICE	son indice de traitement - clé étrangère sur le champ [INDICE] de la table [INDEMNITES]

Le type [RawEmploye] est défini comme suit :

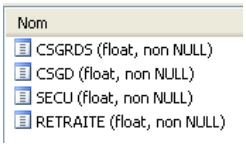
```
1.  Namespace istia.st.pam.entites
2.
3.    Public Class RawEmploye
4.      ' champs privés
5.      Private _SS As String
6.      Private _Nom As String
7.      Private _Prenom As String
8.      Private _Adresse As String
9.      Private _Ville As String
10.     Private _CodePostal As String
```

```
11.     Private _Indice As Integer
12.
13.     ' propriétés associées
14.     Public Property SS() As String
15.       Get
16.         Return _SS
17.       End Get
18.       Set(ByVal value As String)
19.         _SS = value
20.       End Set
21.     End Property
22.
23. ' les autres propriétés
24. ....
25.
26.     ' constructeurs
27.     Public Sub New()
28.
29.     End Sub
30.
31.     Public Sub New(ByVal ss As String, ByVal nom As String, ByVal prenom As String, ByVal adresse
    As String, ByVal codePostal As String, ByVal ville As String,ByVal indice As Integer)
32.       Me.SS = ss
33.       Me.Nom = nom
34.       Me.Prenom = prenom
35.       Me.Adresse = adresse
36.       Me.CodePostal = codePostal
37.       Me.Ville = ville
38.       Me.Indice = indice
39.     End Sub
40.
41.     ' ToString
42.     Public Overrides Function ToString() As String
43.       Return String.Format("[{0},{1},{2},{3},{4},{5},{6}]", SS, Nom, Prenom, Adresse, Ville,
    CodePostal, Indice)
44.     End Function
45.   End Class
46. End Namespace
```

- ligne 1 : les entités de l'application appartiendront toutes à l'espace de noms [istia.st.pam.entites]
- lignes 5-11 : les champs privés qui reçoivent les valeurs des colonnes de même nom d'une ligne de la table [EMPLOYES]
- lignes 14-25 : les propriétés publiques associées aux champs privés. Les méthodes [Set] ne vérifient pas la validité des données injectées dans l'objet. C'est pourquoi la classe a été appelée [RawEmploye] plutôt que [Employe]. Il en sera de même pour toutes les classes dont le nom est préfixé par [Raw].
- lignes 27-29 : les deux constructeurs
- lignes 42-44 : la méthode [ToString] de la classe

4.2.2 La classe [RawCotisations]

Un objet de type [RawCotisations] sert à encapsuler l'unique ligne de la table [COTISATIONS] :



CSGRDS	pourcentage : contribution sociale généralisée + contribution au remboursement de la dette sociale
CSGD	pourcentage : contribution sociale généralisée déductible
SECU	pourcentage : sécurité sociale
RETRAITE	pourcentage : retraite complémentaire + assurance chômage

Le type [RawCotisations] est défini comme suit :

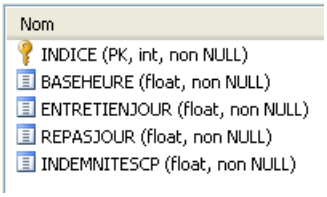
```
1.  Namespace istia.st.pam.entites
2.    Public Class RawCotisations
3.      ' champs privés
4.      Private _CsgRds As Double
5.      Private _Csgd As Double
6.      Private _Secu As Double
7.      Private _Retraite As Double
8.
9.      ' propriétés associées
10.     Public Property CsgRds() As Double
11.       Get
12.         Return _CsgRds
13.       End Get
14.       Set(ByVal value As Double)
15.         _CsgRds = value
16.       End Set
17.     End Property
18. ...
19.
```

```
20.     ' constructeurs
21.     Public Sub New()
22.
23.     End Sub
24.
25.     Public Sub New(ByVal _csgRds As Double, ByVal _csgd As Double, ByVal _secu As Double, ByVal
    _retraite As Double)
26.       Me.CsgRds = _csgRds
27.       Me.Csgd = _csgd
28.       Me.Secu = _secu
29.       Me.Retraite = _retraite
30.     End Sub
31.
32.     ' ToString
33.     Public Overrides Function ToString() As String
34.       Return String.Format("[{0},{1},{2},{3}]", CsgRds, Csgd, Secu, Retraite)
35.     End Function
36.   End Class
37.
38. End Namespace
```

- ligne 1 : la classe appartient à l'espace de noms [istia.st.pam.entites]
- lignes 4-7 : les champs privés qui reçoivent les valeurs des colonnes de l'unique ligne de la table [COTISATIONS]
- lignes 10-19 : les propriétés publiques associées aux champs privés
- lignes 21-30 : les deux constructeurs
- lignes 33-35 : la méthode [ToString] de la classe

4.2.3 La classe [RawIndemnites]

Un objet de type [RawIndemnites] sert à encapsuler une ligne de la table [INDEMNITES] :



INDICE	indice de traitement - clé primaire
BASEHEURE	prix net en euro d'une heure de garde
ENTRETIENJOUR	indemnité d'entretien en euro par jour de garde
REPASJOUR	indemnité de repas en euro par jour de garde
INDEMNITESCP	indemnité de congés payés. C'est un pourcentage à appliquer au salaire de base.

Le type [RawIndemnites] est défini comme suit :

```
1.  Namespace istia.st.pam.entites
2.    Public Class RawIndemnites
3.
4.      ' champs privés
5.      Private _Indice As Integer
6.      Private _BaseHeure As Double
7.      Private _EntretienJour As Double
8.      Private _RepasJour As Double
9.      Private _IndemnitesCP As Double
10.
11.     ' propriétés publiques associées
12.     Public Property Indice() As Integer
13.       Get
14.         Return _Indice
15.       End Get
16.       Set(ByVal Value As Integer)
17.         _Indice = Value
18.       End Set
19.     End Property
20. ...
21.
22.     ' constructeurs
23.     Public Sub New()
24.
25.     End Sub
26.
27.     Public Sub New(ByVal _indice As Integer, ByVal _baseHeure As Double, ByVal _entretienJour As
    Double, ByVal _repasJour As Double, ByVal _indemnitesCP As Double)
28.       Me.Indice = _indice
29.       Me.BaseHeure = _baseHeure
30.       Me.EntretienJour = _entretienJour
31.       Me.RepasJour = _repasJour
32.       Me.IndemnitesCP = _indemnitesCP
33.     End Sub
34.
35.     ' identité
36.     Public Overrides Function toString() As String
```

```
37.       Return String.Format("[{0}, {1}, {2}, {3}, {4}]", Indice, BaseHeure, EntretienJour,
    RepasJour, IndemnitesCP)
38.     End Function
39.
40.   End Class
41. End Namespace
```

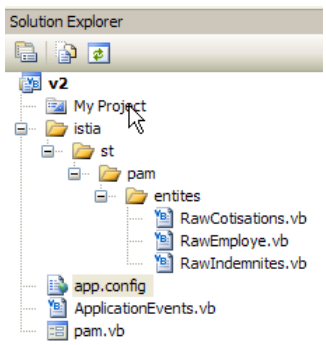
- ligne 1 : la classe appartient à l'espace de noms [istia.st.pam.entites]
- lignes 5-9 : les champs privés qui reçoivent les valeurs des colonnes d'une ligne de la table [INDEMNITES]
- lignes 12-20 : les propriétés publiques associées aux champs privés
- lignes 23-33 : les deux constructeurs
- lignes 36-38 : la méthode [ToString] de la classe

4.3 Configuration et initialisation de l'application

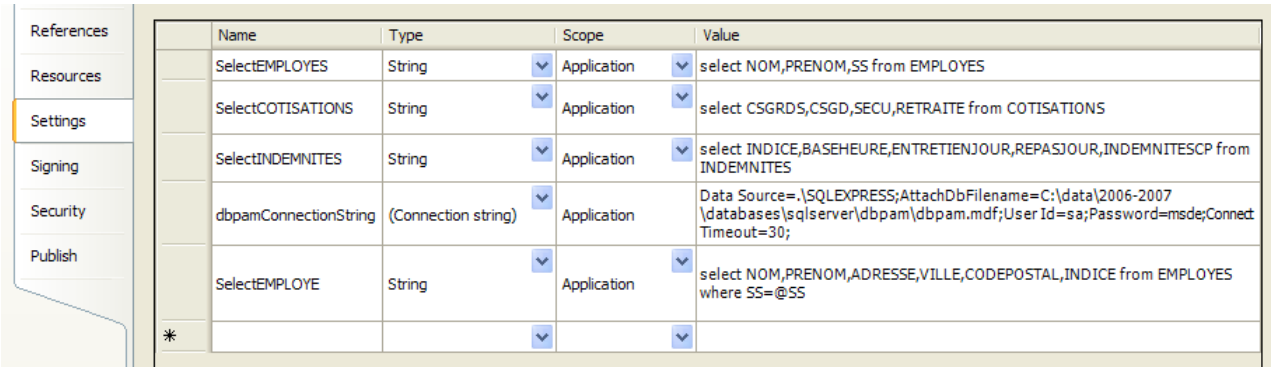
Le fichier [app.config] qui configure l'application sera le suivant :

```
1.  <?xml version="1.0" encoding="utf-8" ?>
2.  <configuration>
3.    <configSections>
4.      <sectionGroup name="applicationSettings" type="System.Configuration.ApplicationSettingsGroup,
    System, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" >
5.        <section name="My.MySettings" type="System.Configuration.ClientSettingsSection, System,
    Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"/>
6.      </sectionGroup>
7.    </configSections>
8.    <connectionStrings>
9.     <add name="My.MySettings.dbpamConnectionString" connectionString="Data
    Source=.\SQLEXPRESS;AttachDbFilename=C:\data\2006-2007\databases\sqlserver\dbpam\dbpam.mdf;User
    Id=sa;Password=msde;Connect Timeout=30;"
10.     providerName="System.Data.SqlClient" />
11.   </connectionStrings>
12.   <applicationSettings>
13.     <My.MySettings>
14.     <setting name="SelectEMPLOYES" serializeAs="String">
15.       <value>select NOM,PRENOM,SS from EMPLOYES</value>
16.     </setting>
17.     <setting name="SelectCOTISATIONS" serializeAs="String">
18.       <value>select CSGRDS,CSGD,SECU,RETRAITE from COTISATIONS</value>
19.     </setting>
20.     <setting name="SelectINDEMNITES" serializeAs="String">
21.       <value>select INDICE,BASEHEURE,ENTRETIENJOUR,REPASJOUR,INDEMNITESCP from INDEMNITES</value>
22.     </setting>
23.     <setting name="SelectEMPLOYE" serializeAs="String">
24.       <value>select NOM,PRENOM,ADRESSE,VILLE,CODEPOSTAL,INDICE from EMPLOYES where SS=@SS</value>
25.     </setting>
26.    </My.MySettings>
27.   </applicationSettings>
28. </configuration>
```

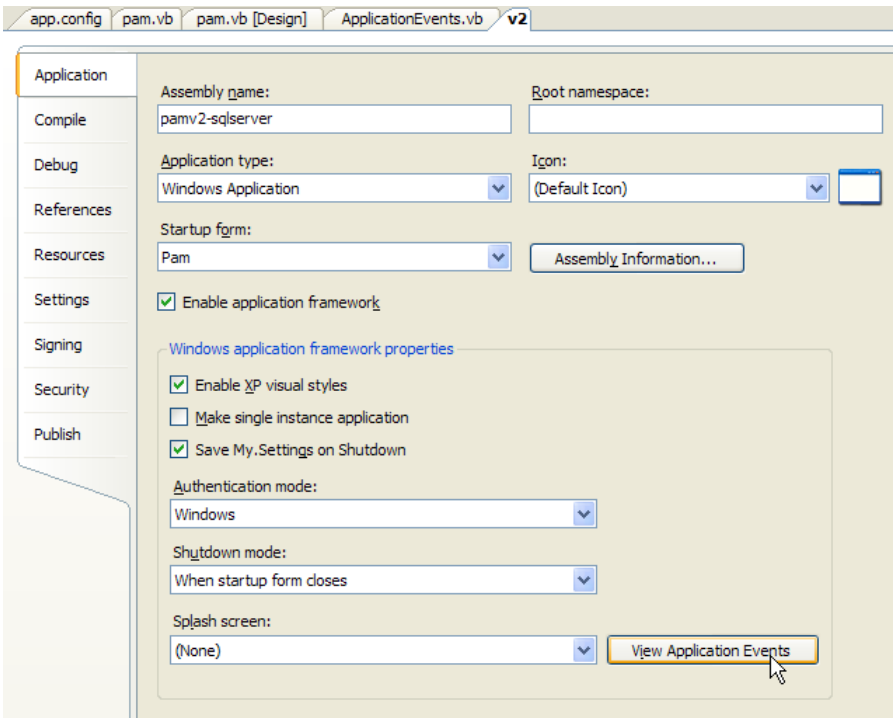
Ce fichier de configuration a été généré par Visual Sudio via l'option [Settings] des propriétés du projet :

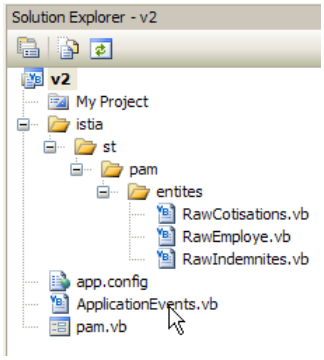


double-clic sur [My Project]



Le squelette de la classe gérant les événements de l'application peut être généré par Visual Studio à partir de la page de propriétés de l'application :





Le code généré pour [ApplicationEvents.vb] est le suivant :

```
1.  Namespace My
2.
3.      ' The following events are availble for MyApplication:
4.      '
5.      ' Startup: Raised when the application starts, before the startup form is created.
6.      ' Shutdown: Raised after all application forms are closed.  This event is not raised if the
    application terminates abnormally.
7.      ' UnhandledException: Raised if the application encounters an unhandled exception.
8.      ' StartupNextInstance: Raised when launching a single-instance application and the application
    is already active.
9.      ' NetworkAvailabilityChanged: Raised when the network connection is connected or disconnected.
10.     Partial Friend Class MyApplication
11.
12.     End Class
13.
14. End Namespace
```

- lignes 10-12 : le squelette vide de la classe [MyApplication] qui gère certains événements de la vie de l'application VB. Le nom de cette classe ne peut être changé. Une instance unique de cette classe est créée pour gérer les événements décrits lignes 5-9. Dans le code des formulaires de l'application VB, cette instance est accessible via la référence [My.Application].
- lignes 5 à 9 : la liste de événements que la classe [MyApplication] peut gérer.

Ici, nous gérerons l'événement [Startup], qui signale le démarrage de l'application, de la façon suivante :

```
1.  Imports System.Data.SqlClient
2.  Imports istia.st.pam.entites
3.
4.  Namespace My
5.
6.    Partial Friend Class MyApplication
7.
8.      ' variables d'instance
9.      Public Employes() As RawEmploye
10.     Public Cotisations As RawCotisations
11.     Public Indemnites As New Dictionary(Of Integer, RawIndemnites)
12.     Public MsgErreur As String = String.Empty
13.     Public Erreur As Boolean = False
14.     Public Connexion As SqlConnection = Nothing
15.
16.     ' démarrage de l'application
17.             Private   Sub   MyApplication_Startup(ByVal   sender   As   Object,   ByVal   e   As
    Microsoft.VisualBasic.ApplicationServices.StartupEventArgs) Handles Me.Startup
18.       ' exploitation du fichier de configuration
19.       ' chaîne de connexion
20.       Dim DatabaseConnectionString As String = My.Settings.dbpamConnectionString
21.       ' requêtes SQL
22.       Dim SelectEmployes As String = My.Settings.SelectEMPLOYES
23.       Dim SelectIndemnites As String = My.Settings.SelectINDEMNITES
24.       Dim SelectCotisations As String = My.Settings.SelectCOTISATIONS
25.       ' chargement des identités des employés dans un tableau
26. ...
27.         ' informations sur les taux de cotisation
28. ...
29.         ' indemnites
30. ...
31.         ' on a réussi
32.         MsgErreur = "Base chargée..."
33.       Catch ex As Exception
34.         ' on note l'erreur
35.          MsgErreur = String.Format("L'erreur suivante s'est produite lors de l'accès à la base de
    données : {0}", ex.Message)
36.         Erreur = True
37.       Finally
38.         ' libération des ressources
39. ...
40.       End Try
41.     End Sub
42.   End Class
43.
44. End Namespace
```

Le rôle de la procédure [MyApplication_Startup] est ici de :

- mettre en cache la liste simplifiée (SS, NOM, PRENOM) de tous les employés, les taux de cotisations et les indemnités liées aux différents indices des employés
- signaler une erreur éventuelle de cette initialisation

Les champs ou méthodes publics de l'objet [My.Application] ci-dessus seront accessibles aux différents formulaires du projet.

- ligne 9 : le tableau d'objets de type [RawEmploye] qui mémorisera la liste simplifiée (SS, NOM, PRENOM) de tous les employés
- ligne 10 : l'objet de type [RawCotisations] qui mémorisera les taux de cotisations
- ligne 11 : le dictionnaire qui mémorisera les indemnités liés aux différents indices des employés. Il sera indexé par l'indice de l'employé et ses valeurs seront de type [RawIndemnites]
- ligne 13 : le booléen signalant ou non une erreur d'initialisation de l'application
- ligne 12 : un message indiquant comment s'est terminée l'initialisation de l'application
- lignes 20-24 : on exploite le fichier de configuration [app.config]. On obtient la valeur d'un élément nommé *Param* dans les *Settings* de l'application par la notation [My.Settings.*Param*].

Question : compléter le code de la procédure [MyApplication_Startup].

4.4 Le code du formulaire [pam.vb]

Le code du formulaire [pam.vb] va différer de celui de la version précédente sur les points suivants :

- l'instance de type [My.MyApplication] qui a été créée pour gérer le démarrage de l'application est accessible du code des formulaires de l'application via la référence [My.Application]. On notera la différence : [My.MyApplication] est un type de données alors que [My.Application] est une référence à une instance de ce type.

- dans la version précédente, le constructeur faisait un certain nombre d'initialisations désormais faites dans la procédure [My.Application.MyApplication_Startup]. Nous n'ajouterons donc rien au code du constructeur généré par défaut par Visual Studio.
- dans la version précédente, l'événement *Load* n'était pas géré. Ici, nous le gèrerons. Dans cette procédure, il faut vérifier comment s'est passée la phase d'initialisation de l'application et pour cela tester la valeur du champ [My.Application.Erreur]. S'il y a eu erreur, le message d'erreur [My.Application.MsgErreur] est affiché et la gestion de l'événement *Load* s'arrête là. S'il y a pas eu erreur, le combo doit être rempli avec le contenu du tableau [My.Application.Employes].
- dans la version précédente, le calcul du salaire nécessitait des accès à la base de données. Certains d'entre-eux sont désormais devenus inutiles, ceux qui demandaient des données que la nouvelle application a mises en cache.

Question : écrire le code des procédures [Pam_Load] et [ButtonSalaire_Click]

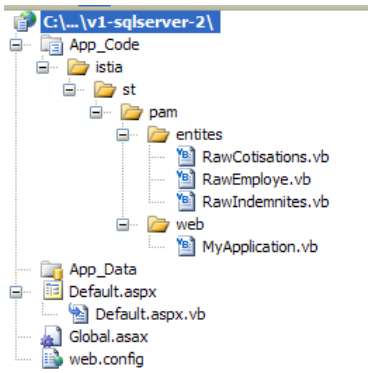
Travail pratique : mettre en oeuvre sur machine cette application VB.NET

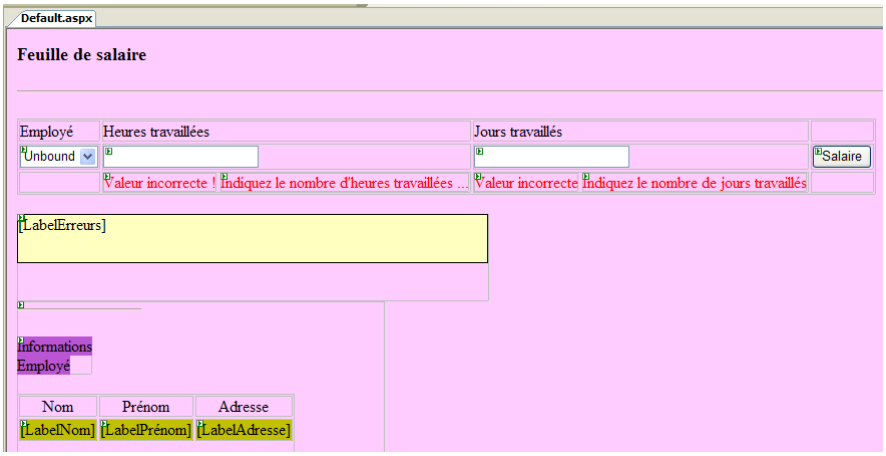
5 L'application [SimuPaie] – version 3 – ASP.NET

Lectures conseillées : référence [1], paragraphe 1 " Composants serveur ASP ", pages 1-40 de ASPNET, vol2.

Nous reprenons l'application précédente en lui donnant maintenant une interface web.

5.1 Le projet Visual Web 2005





Le formulaire [Default.aspx] est le suivant :

Les composants ont gardé le même nom que dans l'application VB.NET afin de faciliter le portage de cette dernière vers l'application ASP.NET :

N°	Type	Nom	Rôle
1	DropDownList	ComboBoxEmployes	Contient la liste des noms des employés
2	TextBox	TextBoxHeures	Nombre d'heures travaillées – nombre réel
3	TextBox	TextBoxJours	Nombre de jours travaillés – nombre entier
4	Button	ButtonSalaire	Demande le calcul du salaire
5	Label	LabelErreurs	Messages d'informations à destination de l'utilisateur
6	Label	LabelNom	Nom de l'employé sélectionné en (1)
7	Label	LabelPrénom	Prénom de l'employé sélectionné en (1)
8	Label	LabelAdresse	Adresse
9	Label	LabelVille	Ville
10	Label	LabelCP	Code postal
11	Label	LabelIndice	Indice
12	Label	LabelCSGRDS	Taux de cotisation CSGRDS
13	Label	LabelCSGD	Taux de cotisation CSGD
14	Label	LabelRetraite	Taux de cotisation Retraite
15	Label	LabelSS	Taux de cotisation Sécurité Sociale
16	Label	LabelSH	Salaire horaire de base pour l'indice indiqué en (11)
17	Label	LabelEJ	Indemnité journalière d'entretien pour l'indice indiqué en (11)
18	Label	LabelRJ	Indemnité journalière de repas pour l'indice indiqué en (11)
19	Label	LabelCongés	Taux d'indemnités de congés payés à appliquer au salaire de base
20	Label	LabelSB	Montant du salaire de base
21	Label	LabelCS	Montant des cotisations sociales à verser
22	Label	LabelIE	Montant des indemnités d'entretien de l'enfant gardé
23	Label	LabelIR	Montant des indemnités de repas de l'enfant gardé
24	Label	LabelSN	Salaire net à payer à l'employé(e)

Le formulaire contient en outre deux conteneurs de type [Panel] :

`PanelErreurs`	contient le composant (5) *LabelErreurs*
`PanelSalaire`	contient les composants (6) à (24)

On rappelle qu'un composant de type [Panel] peut être rendu visible ou non par programmation grâce à sa propriété booléenne [Panel].Visible.

5.2 Configuration de l'application

Lectures conseillées : référence [1], paragraphe 3.1 " La notion d'application web ASP.NET " dans ASP.NET, vol1 [1], pages 49-70.

Le fichier [web.config] qui configure l'application est analogue au fichier [app.config] configurant l'application VB.NET de la version 2 étudiée au paragraphe 4.3, page 15, si ce n'est qu'on utilise une syntaxe différente, celle utilisée par le fichier [app.config] de la version 1 décrit au paragraphe 3.2, page 9. En effet, la nouvelle syntaxe apportée par VB Express / .NET 2.0 n'a pas été portée sur Web Developer. Le fichier [web.config] sera donc le suivant :

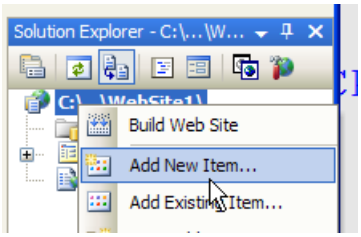
```
1.  <?xml version="1.0" encoding="utf-8" ?>
2.  <configuration>
3.    <connectionStrings>
4.      <add name="dbpam" connectionString="Data Source=.\SQLEXPRESS;AttachDbFilename=C:\data\2006-
    2007\databases\sqlserver\dbpam\dbpam.mdf;User Id=sa;Password=msde;Connect Timeout=30;"
5.          providerName="System.Data.SqlClient" />
6.    </connectionStrings>
7.    <appSettings>
8.      <add key="selectEMPLOYES" value="select NOM,PRENOM,SS from EMPLOYES"/>
```

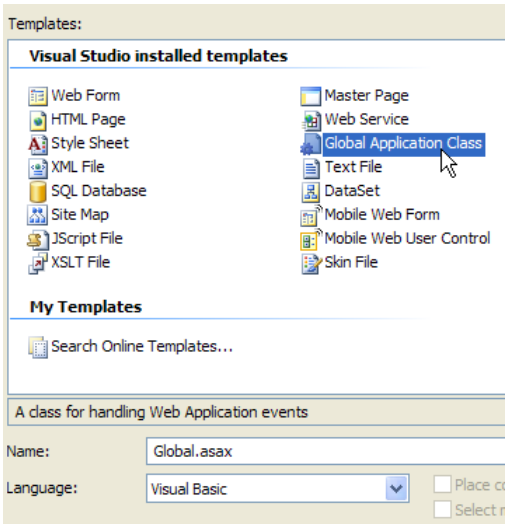
```
9.      <add key="selectEMPLOYE" value="select NOM,PRENOM,ADRESSE,VILLE,CODEPOSTAL,INDICE from EMPLOYES
    where SS=@SS"/>
10.     <add key="selectCOTISATIONS" value="select CSGRDS,CSGD,SECU,RETRAITE from COTISATIONS"/>
11.     <add key="SelectINDEMNITES" value="select INDICE,BASEHEURE,ENTRETIENJOUR,REPASJOUR,INDEMNITESCP
    from INDEMNITES"/>
12.   </appSettings>
13. </configuration>
```

Ce fichier est exploité au démarrage de l'application par le code du couple [Global.asax, Global.asax.vb] :

Global.asax

Le fichier [Global.asax] sert à définir les procédures de gestion des événements de l'application web (démarrage / arrêt de l'application, démarrage / arrêt d'une session utilisateur, ...). Pour créer le fichier [Global.asax], on peut procéder comme suit :





Le contenu du fichier [Global.asax] généré est le suivant :

```
1.  <%@ Application Language="VB" %>
2.
3.  <script runat="server">
4.
5.      Sub Application_Start(ByVal sender As Object, ByVal e As EventArgs)
6.          ' Code that runs on application startup
7.      End Sub
8.
9.      Sub Application_End(ByVal sender As Object, ByVal e As EventArgs)
10.         ' Code that runs on application shutdown
11.     End Sub
12.
13.     Sub Application_Error(ByVal sender As Object, ByVal e As EventArgs)
14.         ' Code that runs when an unhandled error occurs
15.     End Sub
16.
17.     Sub Session_Start(ByVal sender As Object, ByVal e As EventArgs)
18.         ' Code that runs when a new session is started
19.     End Sub
20.
21.     Sub Session_End(ByVal sender As Object, ByVal e As EventArgs)
22.         ' Code that runs when a session ends.
23.         ' Note: The Session_End event is raised only when the sessionstate mode
24.         ' is set to InProc in the Web.config file. If session mode is set to StateServer
25.         ' or SQLServer, the event is not raised.
26.     End Sub
27.
28. </script>
```

- lignes 5-7 : la procédure exécutée au démarrage de l'application
- lignes 9-11 : la procédure exécutée à la fin de l'application, lorsque le serveur web la décharge de la mémoire
- lignes 13-15 : la procédure exécutée lorsqu'une exception remonte jusqu'au serveur web
- lignes 17-19 : la procédure exécutée lors de l'arrivée d'un nouvel utilisateur (absence du cookie de session)
- lignes 21-26 : la procédure exécutée à la fin d'une session de l'utilisateur (expiration de la durée de vie de la session, abandon de la session par programmation, ...)

Le fichier [Global.asax] mélange directives (ligne 1) et code VB. Ecrire du code dans ce fichier est malaisé. Ainsi on écrira :

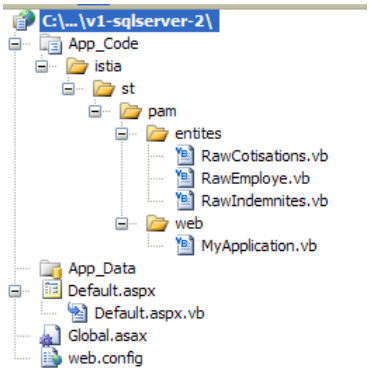
```
1.<%@ Application Language="VB" %>
2.<%@ Import Namespace="System.Collections.Generic" %>
3.
4.<script RunAt="server">
5.
6.  Dim Liste As New List(Of String)
7.
8.  Sub Application_Start(ByVal sender As Object, ByVal e As EventArgs)
9.    ' Code that runs on application startup
10.End Sub
```

- ligne 6 : déclaration d'une variable globale
- ligne 2 : la balise <%@ Import ...%> qui importe l'espace de noms définissant la classe *List*.

On perd nos repères de la programmation objet. Ainsi, ci-dessus, il n'y a pas de classe. De façon implicite, on a une classe qui dérive de la classe [System.Web.HttpApplication]. On préfèrerait un code plus explicite. C'est possible. Nous limitons le fichier [Global.asax] à la seule ligne suivante :

```
<%@ Application Language="VB" inherits="istia.st.pam.web.MyApplication" %>
```

Cette ligne indique que la classe chargée de gérer les événements de l'application web est la classe [istia.st.pam.web.MyApplication]. Cette classe est à placer dans le dossier [App_Code] du projet comme toute classe de l'application web :



La classe [istia.st.pam.web.MyApplication] a été placée dans le fichier [MyApplication.vb] ci-dessus, lequel est dans un dossier dont le chemin est identique l'espace de noms de la classe. Il n'y a aucune obligation à cela. La classe [istia.st.pam.web.MyApplication] est la suivante :

```
1.  Imports System.Data.SqlClient
2.  Imports System.Configuration
3.  Imports System.Collections.Generic
4.  Imports istia.st.pam.entites
5.
6.  Namespace istia.st.pam.web
7.
8.    Public Class MyApplication
9.      Inherits System.Web.HttpApplication
10.
11.     ' --- données statiques de l'application ---
12.     Public Shared Employes() As RawEmploye
13.     Public Shared Cotisations As RawCotisations
14.     Public Shared Indemnites As New Dictionary(Of Integer, RawIndemnites)
15.     Public Shared MsgErreur As String = String.Empty
16.     Public Shared Erreur As Boolean = False
17.     Public Shared Connexion As SqlConnection = Nothing
18.
19.     ' démarrage de l'application
20.     Sub Application_Start(ByVal sender As Object, ByVal e As EventArgs)
21. ...
22.     End Sub
23.
24.   End Class
25.
26.
27. End Namespace
```

- ligne 9 : la classe [MyApplication] (on aurait pu l'appeler autrement) dérive de la classe [HttpApplication]. Les pages web d'une application ASP.NET auront accès à l'instance de cette classe, via la référence [Application]. Cette référence représente une instance de la classe dérivée de [System.Web.HttpApplication] et définie dans le fichier [Global.asax] par l'attribut **inherits** :

```
<%@ Application Language="VB" inherits="istia.st.pam.web.MyApplication" %>
```

- lignes 20-22 : la méthode [Application_Start] est exécutée au démarrage de l'application web. Elle n'est exécutée qu'une fois.
- lignes 12-17 : champs publics et **statiques** de la classe. Un champ statique est partagé par toutes les instances de la classe. Ainsi, si plusieurs instances de la classe [MyApplication] sont créées, elles partagent toutes le même champ statique [Employes], accessible via la référence [MyApplication.Employes], c.a.d. *[NomDeClasse].ChampStatique*. On notera la différence entre les deux notations suivantes :
  - [MyApplication] est le nom d'une classe. C'est donc un type de donnée. Ce nom est arbitraire. La classe dérive toujours de [System.Web.HttpApplication].
  - [Application] est ici une référence sur une instance de la classe [MyApplication] ci-dessus. Dans toute application ASP.NET, [Application] désigne une instance de la classe définie dans le fichier [Global.asax] par l'attribut **inherits** ou bien à l'intérieur du fichier [Global.asax] lui-même.

  Revenons à notre classe [MyApplication]. On peut se demander s'il est nécessaire de déclarer **statiques** ses champs. En fait, il semble que dans certains cas, on puisse avoir plusieurs exemplaires de la classe [MyApplication], ce qui justifie le fait de rendre statiques les champs qui ont à être partagés par toutes ces instances. Il existe une autre façon de partager des données de portée " application " entre les différentes pages d'une application web. Ainsi le tableau *Employes* des employés pourrait être mémorisé dans la procédure *Application_Start* de la façon suivante :

```
Application.Item("employes") = Employes
```

  Le conteneur *Application* peut mémoriser des objets de tout type. Le tableau *Employes* pourrait ensuite être récupéré dans le code d'une page de l'application web de la façon suivante :

```
Dim Employes() As RawEmploye=CType(Application.Item("employes"),RawEmploye())
```

  Parce que le conteneur *Application* mémorise des objets de tout type, on récupère un type *Object* qu'il faut ensuite transtyper. Cette méthode est toujours utilisable. Partager des données via des champs statiques typés de l'objet [MyApplication] évite les transtypages et permet au compilateur de faire des vérifications de type qui aident le développeur. C'est la méthode qui sera utilisée ici.

Le code de la classe [MyApplication] est proche de celui de la classe [My.MyApplication] décrit au paragraphe 4.3, page 17 aux détails près suivants :

- là où la classe VB [My.MyApplication] récupérait le paramètre *Param* du fichier [app.config] avec la syntaxe :

```
Dim Param As String = My.Settings.Param
```

  la classe [istia.st.pam.web.MyApplication] de l'application web le récupère dans le fichier [web.config] avec la syntaxe :

```
Dim Param As String = ConfigurationManager.AppSettings("Param")
```

- là où la classe VB [My.MyApplication] récupérait la chaîne de connexion à la base de données du fichier [app.config] avec la syntaxe :

```
Dim DatabaseConnectionString As String = My.Settings.dbpamConnectionString
```

  la classe [istia.st.pam.web.MyApplication] de l'application web la récupère dans le fichier [web.config] avec la syntaxe :

```
Dim DatabaseConnectionString As String = ConfigurationManager.ConnectionStrings("dbpam").ConnectionString
```

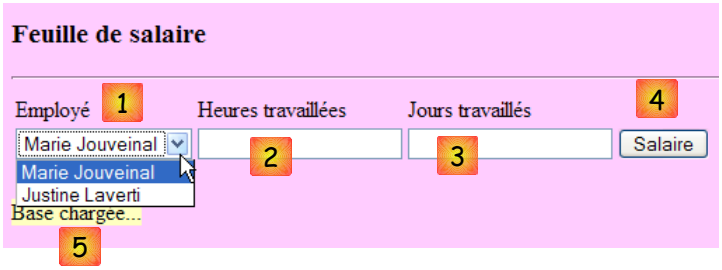
La classe [ConfigurationManager] nécessite l'import de l'espace de noms de la ligne 2 du code ci-dessus.

Question : compléter le code de la classe [MyApplication].

5.3 Le formulaire [Default.aspx]

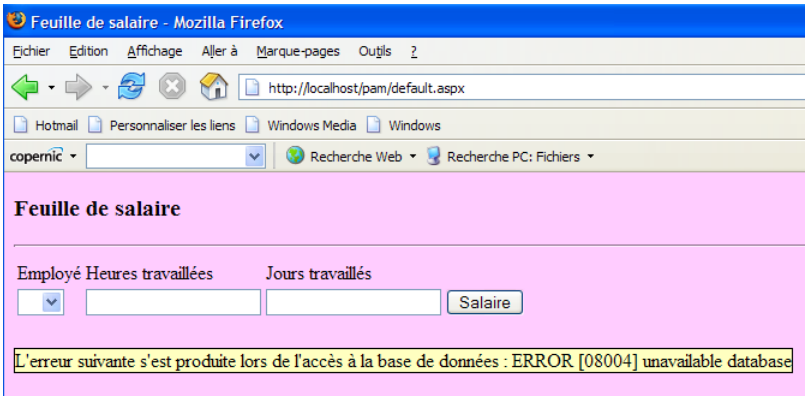
5.3.1 La procédure [Page_Load]

Lorsque le formulaire [Default.aspx] est chargé, il va chercher dans le tableau [MyApplication.Employes] (ligne 12) les noms des employés pour les mettre dans la liste déroulante [1] :



- la liste déroulante [1] a été remplie
- le label [5] indique que la base des données a été lue correctement

S'il s'est produit des erreurs d'initialisation lors du démarrage de l'application, le label [5] l'indique :



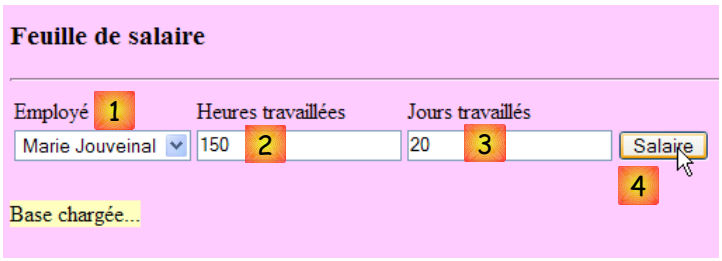
Question 1 : écrire la procédure [Page_Load] de la page web [default.aspx] qui, exécutée au démarrage de l'application, garantit le fonctionnement précédent. On s'inspirera de la procédure analogue de l'application VB.NET de la version 2.

5.3.2 La vérification des saisies

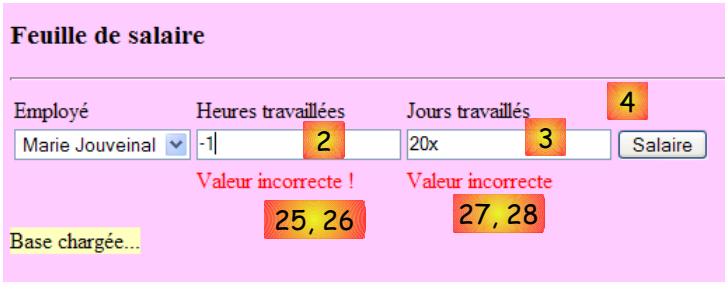
Lectures conseillées : référence [1], paragraphe 2.2 de ASP.NET, vol2 : " Les composants de validation de données ".

Pour calculer un salaire, l'utilisateur :

- sélectionne un employé en [1]
- saisit le nombre d'heures travaillées en [2]. Ce nombre peut être décimal, comme 2,5 pour 2 h 30 mn.
- saisit le nombre de jours travaillés en [3]. Ce nombre est entier.
- demande le salaire avec le bouton [4]



Lorsque l'utilisateur entre des données erronées dans [2] et [3], celles-ci sont vérifiées dès que l'utilisateur change de champ de saisie. Ainsi, la copie d'écran ci-dessous a été obtenue avant même que l'utilisateur ne clique sur le bouton [4] :



Le comportement précédent n'est obtenu qu'avec un navigateur capable d'exécuter le code Javascript embarqué dans une page HTML. Sinon, les données ne sont vérifiées que lorsque le navigateur poste les saisies du formulaire au serveur. C'est alors le code situé sur ce dernier et qui traite la demande du navigateur qui doit vérifier la validité des données. On rappelle que cette vérification doit être toujours faite même si la page affichée dans le navigateur client embarque du code javascript faisant cette même vérification. En effet, le serveur ne peut être assuré que la demande POST qui lui est faite vient réellement de cette page et que donc la vérification des données a été faite.

N°	Type	Nom	Rôle
25	RequiredFieldValidator	RequiredFieldValidatorHeures	vérifie que le champ [2] [TextBoxHeures] n'est pas vide
26	RegularExpressionValidator	RegularExpressionValidatorHeures	vérifie que le champ [2] [TextBoxHeures] est un nombre réel >=0
27	RequiredFieldValidator	RequiredFieldValidatorJours	vérifie que le champ [3] [TextBoxJours] n'est pas vide
28	RegularExpressionValidator	RegularExpressionValidatorJours	vérifie que le champ [3] [TextBoxJours] est un nombre entier >=0

Question 2 : donner les expressions régulières associées aux contrôles de validation [26] et [28].

5.3.3 Le calcul du salaire

Le clic sur le bouton [4] provoque l'exécution du gestionnaire :

```
Protected Sub ButtonSalaire_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles ButtonSalaire.Click
```

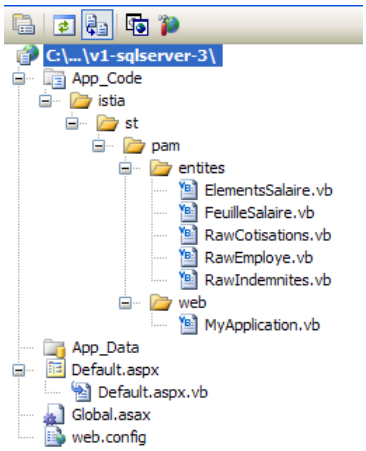
Ce gestionnaire commence par vérifier la validité des saisies faites en [2] et [3]. Si l'une des deux est incorrecte, l'erreur est signalée comme il a été montré précédemment. Une fois les saisies [2] et [3] vérifiées et trouvées valides, l'application doit afficher des données complémentaires sur l'utilisateur sélectionné en [1] ainsi que son salaire.

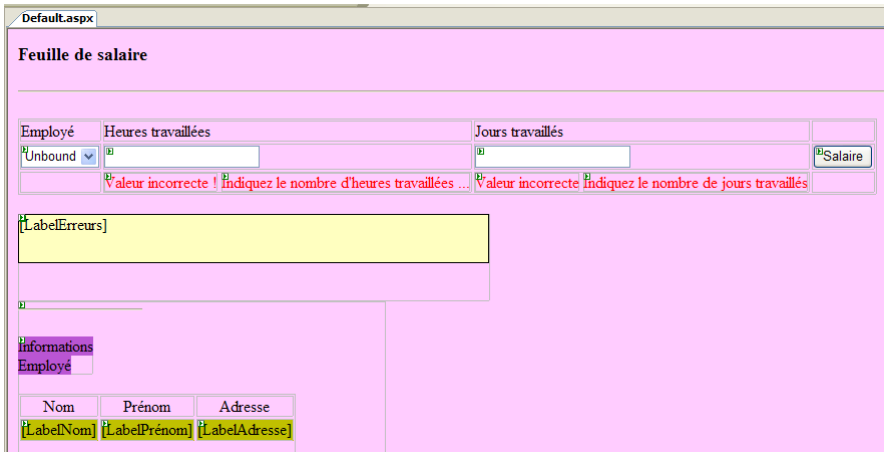
Question 3 : écrire le code de la procédure [ButtonSalaire_Click]. On s'inspirera de la procédure analogue de l'application VB.NET de la version 2.

6 L'application [SimuPaie] – version 4 – ASP.NET

Nous reprenons l'application précédente pour réorganiser son code de façon différente.

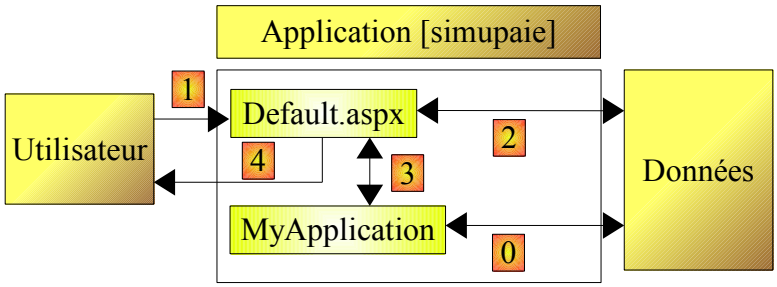
6.1 Le projet Visual Web 2005





6.2 La nouvelle architecture de l'application

L'application ASP.NET précédente avait l'architecture suivante :



- lors de la première requête faite à l'application, un objet de type [MyApplication] dérivé du type [system.web.HttpApplication] est instancié. C'est lui qui va exploiter le fichier de configuration [web.config] de l'application web et mettre en cache certaines données de la base de données (opération 0 ci-dessus).
- lors de la première requête (opération 1) faite à la page [Default.aspx] qui est l'unique page de l'application, l'événement [Load] est traité. On y traite le remplissage du combo des employés. Les données nécessaires au combo sont demandées à l'objet [MyApplication] qui les a mises en cache. C'est l'opération 3 ci-dessus. L'opération 4 envoie la page ainsi initialisée à l'utilisateur.
- lors de la demande du calcul du salaire (opération 1) faite à la page [Default.aspx], l'événement [Load] est de nouveau traité. Rien n'est fait car il s'agit d'une demande de type POST, et le gestionnaire [Pam_Load] a été écrit pour ne rien faire dans ce cas (utilisation du booléen IsPostBack). L'événement [Load] traité, c'est l'événement [Click] sur le composant [ButtonSalaire] qui l'est ensuite. Celui-ci a besoin de données qui n'ont pas été mises en cache dans [MyApplication]. Aussi le gestionnaire [ButtonSalaire_Click] les demande-t-il à la base de données. C'est l'opération 2 ci-dessus. Le calcul du salaire est ensuite fait et l'opération 4 envoie les résultats à l'utilisateur.

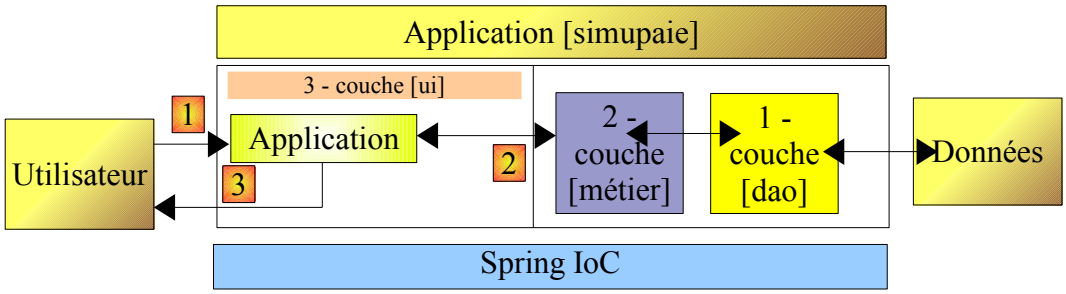
On constate que le code de la page [Default.aspx] fait diverses choses :

1. de la présentation de données : on y trouve des instructions qui donnent des valeurs aux composants de la page [Default.aspx].
2. de l'accès aux données : on y trouve des instructions qui accèdent aux données d'un SGBD
3. du calcul métier : le calcul du salaire

La procédure [ButtonSalaire_Click] par exemple contient ces trois types d'instructions. On pourrait vouloir séparer ces trois types d'activités dans des procédures différentes :

- si des procédures bien identifiées s'occupaient de l'accès aux données, on connaîtrait tout de suite les procédures à modifier, dans le cas par exemple où les données sont trouvées dans un fichier Excel au lieu d'un SGBD. Lorsque les gestionnaires d'événements des pages mélangent les trois types d'activité (présentation, métier, accès aux données), il faut alors parcourir tout le code pour trouver les instructions d'accès aux données et les modifier. Cela demande plus de temps et présente davantage de risques d'erreurs.
- un autre avantage de la séparation des tâches est qu'elle facilite les tests. Si les instructions d'accès aux données ont été isolées dans des procédures qui ne font que ça, on peut tester ces dernières de façon isolée, en-dehors de l'application web dans notre cas. Si elles sont mélangées avec le reste, ces tests deviennent difficiles.
- enfin, la séparation des tâches facilite la réutisabilité. Si on passe d'une interface web à une interface windows, les procédures d'accès aux données ou de calcul métier pourront être réutilisées. Cette réutilisabilité peut prendre diverses formes : du simple copier / coller à l'utilisation de composants encapsulés dans une DLL.

Nous l'avons dit au début de cette étude de cas, nous souhaitons arriver à l'architecture finale suivante pour notre application :

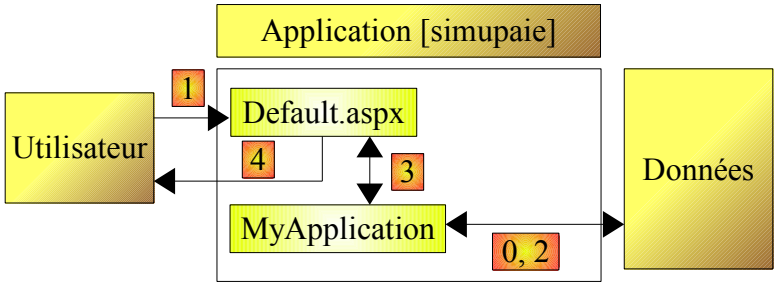


- la couche [1-dao] (dao=Data Access Object) s'occupera de l'accès aux données.
- la couche [2-métier] s'occupera de l'aspect métier de l'application, le calcul de la paie.
- la couche [3-ui] (ui=User Interface) s'occupera de la présentation des données à l'utilisateur et de l'exécution de ses requêtes. Nous appelons [Application] l'ensemble des modules assurant cette fonction. Elle est l'interlocuteur de l'utilisateur.
- les trois couches seront rendues indépendantes grâce à l'utilisation d'interfaces .NET
- l'intégration des différentes couches sera réalisée par **Spring IoC**

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

1. le client fait une demande à l'application.
2. l'application traite cette demande. Pour ce faire, elle peut avoir besoin de l'aide de la couche [métier] qui elle-même peut avoir besoin de la couche [dao] si des données doivent être échangées avec la base de données. L'application reçoit une réponse de la couche [métier].
3. selon celle-ci, elle envoie la vue (= la réponse) appropriée au client.

L'architecture précédente est assez complexe et n'est pas préconisée pour une application simple qui doit être développée rapidement par un unique développeur. Nous allons ici adopter une architecture moins complexe :



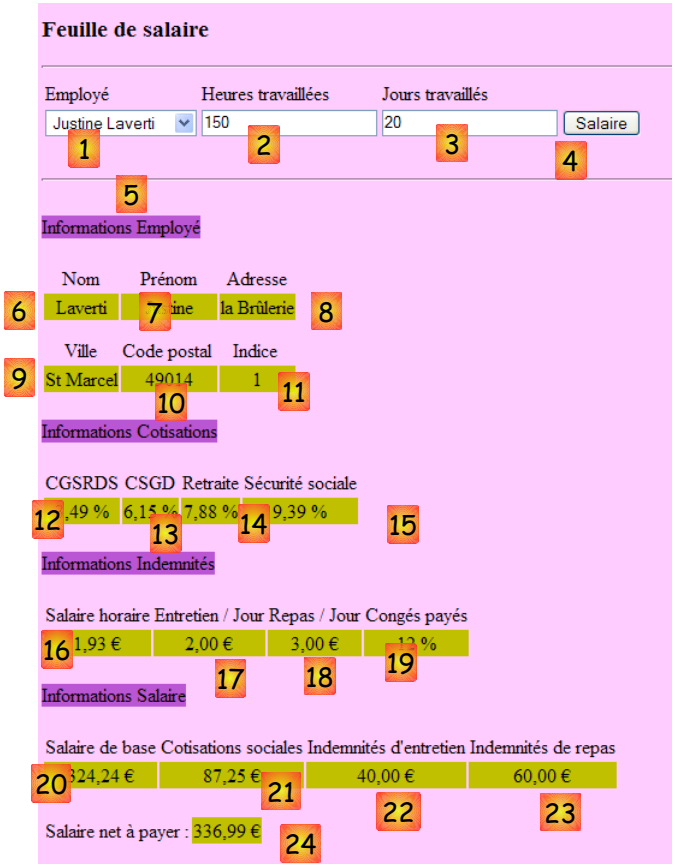
Dans [Default.aspx], la procédure [ButtonSalaire_Click] ne s'occupera plus que de la présentation des données. Les parties " métier " et " accès aux données " qu'elle contenait sont transférées dans des procédures partagées (Shared) de l'objet [MyApplication]. Le déroulement du calcul du salaire se fera ainsi :

- lors de la demande du calcul du salaire (opération 1) faite à la page [Default.aspx], l'événement [Load] est d'abord traité. L'événement [Click] sur le composant [ButtonSalaire] l'est ensuite. Celui-ci va demander à l'objet [MyApplication] les données qu'il a besoin d'afficher. C'est ce dernier qui demandera au SGBD les données dont il a besoin (opération 2 ci-dessus) et fera le calcul du salaire. Il renverra les résultats au gestionnaire [ButtonSalaire_Click] qui les affectera aux composants de la page [Default.aspx]. L'opération 4 enverra ces résultats à l'utilisateur.

6.3 Les entités de l'application web

Le dossier [entites] du projet Visual Studio (cf paragraphe 6.1, page 26) contient les objets manipulés par :
- les procédures d'accès aux données : [RawEmploye, RawCotisations, RawIndemnites]
- les procédures " métier " : [FeuilleSalaire] et [ElementsSalaire].

Rappelons l'interface web présentée à l'utilisateur :



La classe [FeuilleSalaire] encapsule les informations 6 à 24 du formulaire précédent :

```
1.
2.
3.  Namespace istia.st.pam.entites
4.
5.    Public Class FeuilleSalaire
6.
7.      ' champs privés
8.      Private _Employe As RawEmploye
9.      Private _Cotisations As RawCotisations
10.     Private _Indemnites As RawIndemnites
11.     Private _ElementsSalaire As ElementsSalaire
12.
13.     ' propriétés associées
14.     Public Property Employe() As RawEmploye
15.       Get
16.         Return _Employe
17.       End Get
18.       Set(ByVal value As RawEmploye)
19.         _Employe = value
20.       End Set
21.     End Property
22.
```

```
23.  ...
24.
25.     ' constructeurs
26.     Public Sub New()
27.
28.     End Sub
29.
30.     Public Sub New(ByVal employe As RawEmploye, ByVal cotisations As RawCotisations, ByVal
    indemnites As RawIndemnites, ByVal elementsSalaire As ElementsSalaire)
31.       With Me
32.         .Employe = employe
33.         .Cotisations = cotisations
34.         .Indemnites = indemnites
35.         .ElementsSalaire = elementsSalaire
36.       End With
37.     End Sub
38.
39.     ' ToString
40.     Public Overrides Function ToString() As String
41.       Return String.Format("[{0},{1},{2},{3}", Employe, Cotisations, Indemnites, ElementsSalaire)
42.     End Function
43.   End Class
44.
45. End Namespace
```

- ligne 8 : les informations 6 à 11 sur l'employé dont on calcule le salaire
- ligne 9 : les informations 12 à 15
- ligne 10 : les informations 16 à 19
- ligne 11 : les informations 20 à 24
- lignes 14-23 : les propriétés publiques associées aux 4 champs privés précédents
- lignes 26-37 : les constructeurs
- lignes 40-42 : la méthode [ToString]

La classe [ElementsSalaire] encapsule les informations 20 à 24 du formulaire :

```
1.  Namespace istia.st.pam.entites
2.    Public Class ElementsSalaire
3.      ' champs privés
4.      Private _SalaireBase As Double
5.      Private _CotisationsSociales As Double
6.      Private _IndemnitesEntretien As Double
7.      Private _IndemnitesRepas As Double
8.      Private _SalaireNet As Double
9.
10.     ' propriétés publiques associées
11.     Public Property SalaireBase() As Double
12.       Get
13.         Return _SalaireBase
14.       End Get
15.       Set(ByVal value As Double)
16.         _SalaireBase = value
17.       End Set
18.     End Property
19.
20. ...
21.
22.     ' constructeurs
23.     Public Sub New()
24.
25.     End Sub
26.
27.     Public Sub New(ByVal salaireBase As Double, ByVal cotisationsSociales As Double, ByVal
    indemnitesEntretien As Double, ByVal indemnitesRepas As Double, ByVal salaireNet As Double)
28.       With Me
29.         .SalaireBase = salaireBase
30.         .CotisationsSociales = cotisationsSociales
31.         .IndemnitesEntretien = indemnitesEntretien
32.         .IndemnitesRepas = indemnitesRepas
33.         .SalaireNet = salaireNet
34.       End With
35.     End Sub
36.
37.     ' ToString
38.     Public Overrides Function ToString() As String
39.       Return String.Format("[{0} : {1} : {2} : {3} : {4} ]", SalaireBase, CotisationsSociales,
    IndemnitesEntretien, IndemnitesRepas, SalaireNet)
40.     End Function
41.   End Class
42. End Namespace
```

- lignes 4-8 : les éléments du salaire tels qu'expliqués dans les règles métier décrites page 6.
- ligne 4 : le salaire de base de l'employé, fonction du nombre d'heures travaillées

- ligne 5 : les cotisations prélevées sur ce salaire de base
- lignes 6 et 7 : les indemnités à ajouter au salaire de base, fonction de l'indice de l'employé et du nombre de jours travaillés
- ligne 8 : le salaire net à payer
- lignes 11-20 : les propriétés publiques associées aux 5 champs privés précédents
- lignes 23-35 : les constructeurs
- lignes 38-40 : la méthode [ToString] de la classe.

6.4 Configuration de l'application

Le fichier de configuration [web.config] est identique à celui de la version précédente.

6.5 La classe [MyApplication]

La classe [MyApplication] est la classe dérivée de [HttpApplication] qui se trouve dans le fichier [MyApplication.vb]. Le squelette de son code est le suivant :

```
1.  Imports System.Data.SqlClient
2.  Imports System.Configuration
3.  Imports System.Collections.Generic
4.  Imports istia.st.pam.entites
5.
6.  Namespace istia.st.pam.web
7.
8.   Public Class MyApplication
9.     Inherits System.Web.HttpApplication
10.
11.    ' --- données statiques de l'application ---
12.    Public Shared Employes() As RawEmploye
13.    Public Shared Cotisations As RawCotisations
14.    Public Shared Indemnites As New Dictionary(Of Integer, RawIndemnites)
15.    Public Shared MsgErreur As String = String.Empty
16.    Public Shared Erreur As Boolean = False
17.    Public Shared Connexion As SqlConnection = Nothing
18.
19.    ' démarrage de l'application
20.    Sub Application_Start(ByVal sender As Object, ByVal e As EventArgs)
21. ...
22.    End Sub
23.
24.    ' calcul du salaire
25.    Public Shared Function GetSalaire(ByVal ss As String, ByVal heuresTravaillées As Double, ByVal
    joursTravaillés As Integer) As FeuilleSalaire
26.      ' SS : n° SS de l'employé
27.      ' heuresTravaillées : le nombre d'heures travaillés
28.      ' jours Travaillés : nbre de jours travaillés
29. ...
30.    End Function
31.
32.    ' obtenir un employé
33.    Private Shared Function GetEmploye(ByVal ss As String) As RawEmploye
34. ...
35.    End Function
36.
37.  End Class
38. End Namespace
```

- lignes 12-17 : les données publiques partagées n'ont pas changé. Il est à noter qu'il serait peut-être préférable de déclarer privées ces données et de les rendre disponibles via des méthodes publiques.
- lignes 20-22 : la procédure [Application_Start] qui gère l'initialisation de l'application n'a pas changé
- lignes 25-30 : le calcul du salaire n'est plus fait par la page [Default.aspx.vb] mais par la fonction partagée (Shared) [MyApplication.GetSalaire] qui rend un objet de type [FeuilleSalaire].
- lignes 33-35 : le calcul du salaire nécessite d'avoir des informations complémentaires sur l'employé, notamment son indice de traitement. La méthode [GetEmploye] permet d'obtenir auprès du SGBD toutes les informations associées à un employé identifié par son n° SS. Elle est déclarée privée car elle n'est utile qu'au sein de la classe [MyApplication].

Question : compléter le code ci-dessus. On reprendra le code de la procédure [ButtonSalaire_Click] de la page [Default.aspx.vb] de la version précédente.

6.6 Le formulaire [Default.aspx.vb]

Dans le code de la page [Default.aspx], la procédure [ButtonSalaire_Click] est désormais allégée. Elle fait appel au code de l'objet [MyApplication] pour calculer le salaire.

Question : donner le nouveau code de la procédure [ButtonSalaire_Click].

Travail pratique : mettre en oeuvre sur machine cette application.

7 L'application [SimuPaie] – version 5 – AJAX / ASP.NET

7.1 Introduction

La technologie AJAX (**A**synchronous **J**avascript **A**nd **X**ml) réunit en fait un ensemble de technologies :

- **Javascript** : une page HTML affichée dans un navigateur peut embarquer du code Javascript. Ce code est exécuté par le navigateur si l'utilisateur n'a pas inhibé l'exécution du code Javascript sur son navigateur. Cette technologie est apparue dès les débuts du web et suscite depuis 2005 un regain d'intérêt grâce à AJAX.
- **DOM** : **D**ocument **O**bject **M**odel. Le code Javascript embarqué dans une page HTML a accès au document sous la forme d'un arbre d'objets, le DOM. Chaque élément du document (un champ de saisie, une balise nommée, une balise <select>,... est représenté par un noeud de l'arbre. Le code Javascript peut changer le document affiché en modifiant le DOM. Par exemple, il peut changer le texte affiché dans un champ de saisie via le noeud du DOM représentant ce champ.

AJAX utilise Javascript pour faire des échanges navigateur / serveur en tâche de fond. Pour réagir à un événement tel que le clic sur un bouton, du code Javascript va être exécuté par le navigateur pour envoyer une requête HTTP au serveur. Cette requête va contenir des paramètres indiquant au serveur ce qu'il doit faire. Celui-ci va exécuter l'action demandée. Il va envoyer en réponse au navigateur, un flux HTTP contenant un document XML permettant une mise à jour partielle de la page actuellement affichée. Celle-ci sera réalisée via le DOM du document et l'exécution de code Javascript embarqué dans le document.

L'intérêt de la technologie réside dans cette mise à jour partielle du document affiché. Cela signifie :
- moins de données échangées entre le client et le serveur et donc une réponse plus rapide
- une interface graphique plus fluide à utiliser car les actions de l'utilisateur ne provoquent pas le rechargement complet de la page.

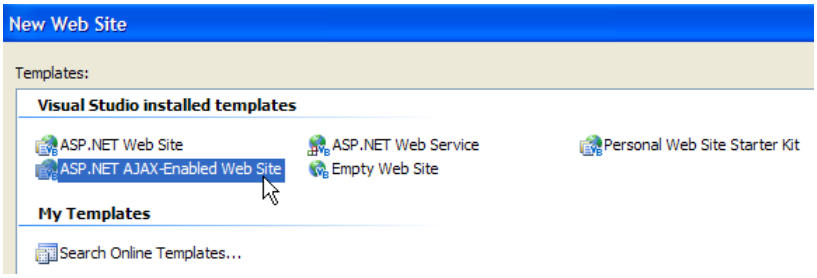
Dans un intranet où les échanges réseau sont rapides, AJAX permet de créer des applications web ayant un comportement qui se rapproche de celui des interfaces windows classiques. On peut en effet gérer des événements tels que le changement de sélection dans une liste et rafraîchir immédiatement et partiellement la page pour refléter ce changement de sélection. Le fait que la page ne soit pas totalement rechargée donne à l'utilisateur l'impression de fluidité qu'il a avec les applications windows. Pour des applications Internet, où les temps de réponse peuvent être longs, la technologie AJAX reste utilisable mais l'impression de fluidité est dépendante de celle du réseau.

La principale difficulté dans AJAX est le langage Javascript. Créé dès les débuts du web,

- il s'avère peu pratique pour faire de la programmation orientée objet. Ce n'est pas, par exemple, un langage typé. On n'y déclare pas le type des données manipulées. On se rapproche là de langages tels que Perl ou PHP.
- il n'a pas un standard accepté par tous les navigateurs. Chacun d'eux a ses extensions "Javascript" propriétaires qui font qu'un code Javascript écrit pour IE pourra ne pas fonctionner dans Mozilla Firefox et vice-versa.
- il est difficile à déboguer, les navigateurs n'offrant pas d'outils performants de débogage du code Javascript qu'ils exécutent.

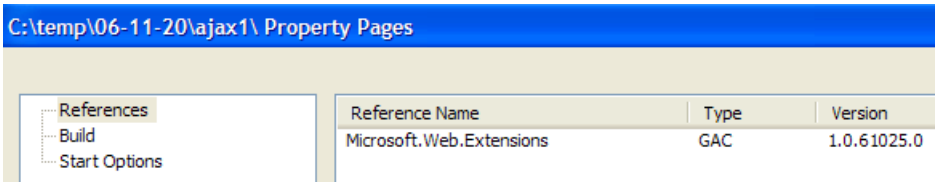
Tous ces maux du Javascript ont fait qu'il était peu utilisé avant l'arrivée d'AJAX. Une fois compris l'intérêt de faire exécuter du code Javascript en tâche de fond, pour faire des requêtes HTTP au serveur web et utiliser la réponse XML de celui-ci pour faire, grâce au DOM, une mise à jour partielle du document affiché, les équipes de développement se sont mises au travail et ont proposé des frameworks permettant de faire de l'AJAX sans que le développeur ait à écrire du code Javascript. Pour cela, des bibliothèques Javascript qui savent s'adapter au navigateur client ont été écrites. L'insertion de code Javascript dans la page HTML envoyée au navigateur est faite côté serveur, selon des techniques qui diffèrent selon le framework AJAX utilisé. Il existe des frameworks Ajax aussi bien pour Java ou .NET. Nous utiliserons ici l'extension AJAX pour ASP.NET de Microsoft [http://ajax.asp.net] :

Le téléchargement et l'installation d'ASP/AJAX permet de créer un nouveau type de projets web avec Visual Web Developer [File / new WebSite] :

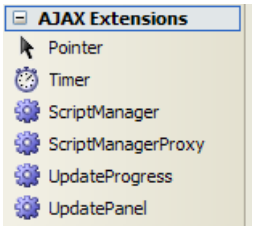


Le type [Ajax-Enabled] permet de créer un site web incluant la technologie Ajax. Le projet créé présente quelques différences avec un projet classique :

– une référence est automatiquement ajoutée au projet :



– une nouvelle boîte de composants apparaît

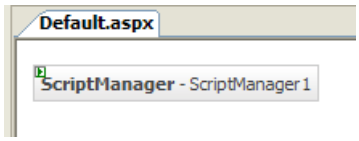


– une balise <asp:ScriptManager> est automatiquement insérée dans le code source du formulaire [Default.aspx] (ligne 10 ci-dessous) :

```
1.  <%@ Page Language="VB" AutoEventWireup="true" CodeFile="Default.aspx.vb" Inherits="_Default" %>
2.
3.  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
4.  <html xmlns="http://www.w3.org/1999/xhtml">
5.  <head runat="server">
6.      <title>Untitled Page</title>
7.  </head>
8.  <body>
9.      <form id="form1" runat="server">
10.         <asp:ScriptManager ID="ScriptManager1" runat="server" />
11.         <div>
12.         </div>
```

```
13.     </form>
14. </body>
15. </html>
```

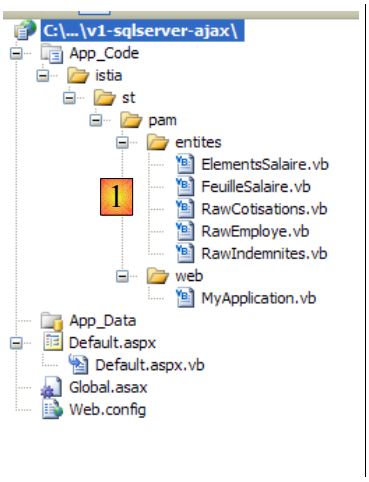
- la présence de la balise <asp:ScriptManager> se traduit par la présence d'un composant dans l'onglet [Design] du formulaire :

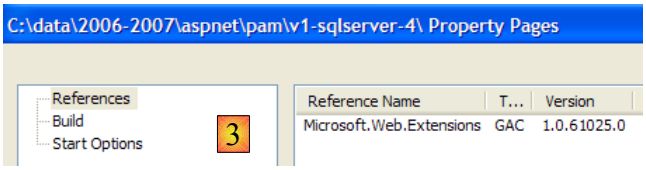


- le fichier de configuration [web.config] subit d'importantes modifications, que nous ne chercherons pas à expliquer, afin que les extensions ASP / AJAX soient disponibles à l'application :

```
1.  <?xml version="1.0"?>
2.  <configuration>
3.    <configSections>
4.      <sectionGroup name="microsoft.web" type="Microsoft.Web.Configuration.MicrosoftWebSectionGroup,
    Microsoft.Web.Extensions, Version=1.0.61025.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35">
5.  ...
6.      </sectionGroup>
7.    </configSections>
8.
9.    <system.web>
10.     <pages>
11.       <controls>
12.         <add tagPrefix="asp" namespace="Microsoft.Web.UI" assembly="Microsoft.Web.Extensions,
    Version=1.0.61025.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35"/>
13. ...
14.       </controls>
15.       <tagMapping>
16.         <add tagType="System.Web.UI.WebControls.CompareValidator"
    mappedTagType="Microsoft.Web.UI.Compatibility.CompareValidator, Microsoft.Web.Extensions,
    Version=1.0.61025.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35"/>
17. ...
18.       </tagMapping>
19.     </pages>
20.
21.     <compilation debug="false">
22.       <assemblies>
23.         <add assembly="Microsoft.Web.Extensions, Version=1.0.61025.0, Culture=neutral,
    PublicKeyToken=31bf3856ad364e35"/>
24.       </assemblies>
25.     </compilation>
26.
27.     <httpHandlers>
28.       <remove verb="*" path="*.asmx"/>
29.       <add verb="*" path="*.asmx" validate="false"
    type="Microsoft.Web.Script.Services.ScriptHandlerFactory, Microsoft.Web.Extensions,
    Version=1.0.61025.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35"/>
30. ...
31.     </httpHandlers>
32.
33.     <httpModules>
34.       <add name="WebResourceCompression"
    type="Microsoft.Web.Handlers.WebResourceCompressionModule, Microsoft.Web.Extensions,
    Version=1.0.61025.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35"/>
35. ...
36.     </httpModules>
37.   </system.web>
38.
39.   <microsoft.web>
40.     <scripting>
41.       <webServices>
42. ....
43.       </webServices>
44.     </scripting>
45.   </microsoft.web>
46.
47.   <system.webServer>
48. ...
49.   </system.webServer>
50. </configuration>
```

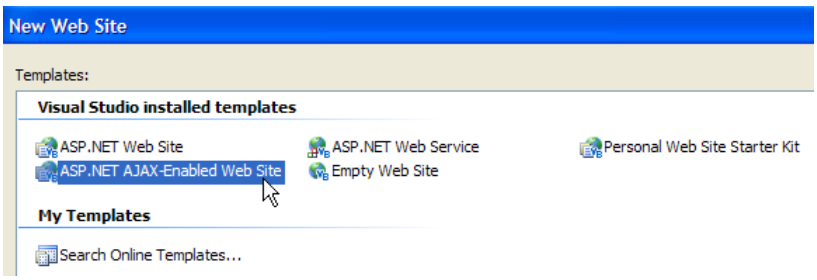
7.2 Le projet Visual Web Developer de la couche [web-ui-ajax]



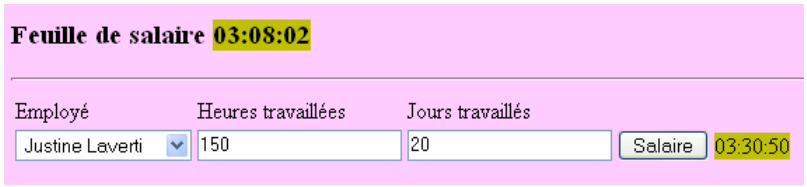


Le projet ASP/Ajax a été construit à partir du projet ASP.NET de la version précédente de la façon suivante :

- création d'un projet Ajax :



- via l'explorateur windows, copie dans le dossier du nouveau projet de tous les fichiers et dossiers du projet précédent (cf [1] ci-dessus) à **l'exception** notable du fichier [web.config], puis rafraîchissement du projet (bouton *Refresh* dans la fenêtre *Solution Explorer*).
- ajout de composants Ajax dans le formulaire [Default.aspx] (cf [2] ci-dessus).
  - 2A : le composant <asp:ScriptManager> est nécessaire pour tout projet AJAX
  - 2B : le composant <asp:UpdatePanel> sert à délimiter la zone à rafraîchir lors d'un POST de l'utilisateur. Ce composant évite le rechargement total de la page.
  - 2C : le composant <asp:UpdateProgress> sert à afficher un texte ou une image pendant la durée de la mise à jour afin d'en avertir l'utilisateur.
  - 2D : un type Label nommé *LabelHeureLoad* qi va afficher l'heure à laquelle s'exécute le gestionnaire de l'événement *Load* de la page.
  - 2E : un type Label nommé *LabelHeureLoad* qui va afficher l'heure à laquelle s'exécute le gestionnaire de l'événement *Click* sur le bouton [Salaire]. On veut montrer qu'Ajax ne recharge pas toute la page lors du clic sur le bouton [Salaire]. Aussi devrait-on voir deux heures différentes dans les deux *Labels* :



- en [3], on voit la référence du projet sur la DLL permettant les extensions AJAX

Le fichier [web.config] généré par le projet de type [AJAX-enabled] est complété avec le contenu du fichier [web.config] de la version 3, de la façon suivante :

```
1.  ...
2.     </sectionGroup>
3.   </configSections>
4.
5.   <connectionStrings>
6.     <add name="dbpam" connectionString="Data Source=.\SQLEXPRESS;AttachDbFilename=C:\data\2006-
    2007\aspnet\pam\database\sqlserver\dbpam.mdf;User Id=sa;Password=msde;Connect Timeout=30;"
7.         providerName="System.Data.SqlClient" />
8.   </connectionStrings>
9.   <appSettings>
10.    <add key="selectEMPLOYES" value="select NOM,PRENOM,SS from EMPLOYES"/>
11.    <add key="selectEMPLOYE" value="select
    NOM,PRENOM,ADRESSE,VILLE,CODEPOSTAL,EMPLOYES.INDICE,BASEHEURE,ENTRETIENJOUR,REPASJOUR,INDEMNITESCP
    from EMPLOYES,INDEMNITES where SS=@SS and EMPLOYES.INDICE=INDEMNITES.INDICE"/>
12.    <add key="selectCOTISATIONS" value="select CSGRDS,CSGD,SECU,RETRAITE from COTISATIONS"/>
13.    <add key="SelectINDEMNITES" value="select INDICE,BASEHEURE,ENTRETIENJOUR,REPASJOUR,INDEMNITESCP
    from INDEMNITES"/>
14.  </appSettings>
15.
16.  <system.web>
17.    <pages>
18. ....
```

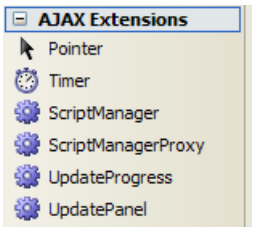
L'ajout des constantes (chaîne de connexion et textes des requêtes SQL) se fait entre la section <configSections> (ligne 3) et <system.web> (ligne 16) du fichier [web.config] généré.

L'Ajaxification du formulaire peut se faire directement dans le code source de [Default.aspx] par l'ajout de balises d'extensions Ajax :

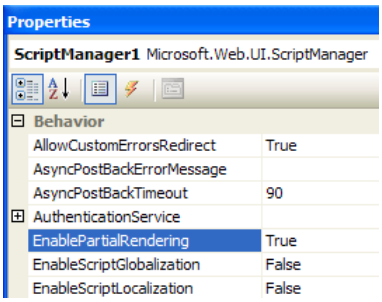
```
1.  ...
2.  <body style="text-align: left" bgcolor="#ffccff">
3.    <h3>
4.      Feuille de salaire
5.      <asp:Label ID="LabelHeureLoad" runat="server" BackColor="#C0C000"></asp:Label></h3>
6.    <hr />
7.    <form id="form1" runat="server">
8.      <asp:ScriptManager ID="ScriptManager1" runat="server" EnablePartialRendering="true" />
9.      <asp:UpdatePanel runat="server" ID="UpdatePanelPam" UpdateMode="Conditional">
10.       <ContentTemplate>
11.         <div>
12.           <table>
13.             <tr>
14. ...
15.             <tr>
16.               <td>
17.                 <asp:DropDownList ID="ComboBoxEmployes" runat="server">
18.                 </asp:DropDownList></td>
19.               <td>
20.                 <asp:TextBox ID="TextBoxHeures" runat="server" EnableViewState="False">
21.                 </asp:TextBox></td>
22.               <td>
23.                 <asp:TextBox ID="TextBoxJours" runat="server" EnableViewState="False">
24.                 </asp:TextBox></td>
25.               <td>
26.                 <asp:Button ID="ButtonSalaire" runat="server" Text="Salaire"
    CausesValidation="False" /></td>
27.                 <td>   <asp:UpdateProgress ID="UpdateProgress1" runat="server">
28.                   <ProgressTemplate>
29.                     <img src="images/indicator.gif" />
30.                     <asp:Label ID="Label5" runat="server" BackColor="#FF8000"
    EnableViewState="False"
31.                       Text="Calcul en cours. Patientez ...."></asp:Label>
32.                   </ProgressTemplate>
33.                 </asp:UpdateProgress>
34.           <asp:Label ID="LabelHeureSalaire" runat="server" BackColor="#C0C000"></asp:Label></td>
```

```
35.             </tr>
36.             <tr>
37. ...
38.             </tr>
39.           </table>
40.         </div>
41.         <br />
42. ....
43.           <table>
44.             <tr>
45.               <td>
46.                 Salaire net à payer :
47.               </td>
48.               <td align="center" bgcolor="#C0C000" height="20px">
49.                 <asp:Label ID="LabelSN" runat="server" EnableViewState="False"></asp:Label></td>
50.             </tr>
51.           </table>
52.            </asp:Panel>
53.       </ContentTemplate>
54.     </asp:UpdatePanel>
55.   </form>
56. </body>
57. </html>
```

- ligne 8 : tout formulaire Ajaxifié doit inclure la balise <asp:ScriptManager>. C'est cette balise qui permet l'utilisation des nouveaux composants Ajax :



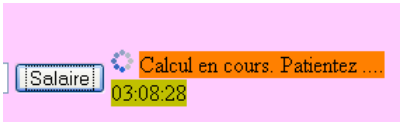
- la balise <asp:ScriptManager> peut être obtenue par double-clic sur le composant [ScriptManager] ci-dessus. Il faut prendre soin de vérifier que cette balise est contenue dans la balise <form> du code source. L'attribut [EnablePartialRendering= "true "] de la ligne 8 est absent pas défaut. La valeur " true " étant sa valeur par défaut, il n'est pas indispensable.



- ligne 9 : la balise <asp:UpdatePanel> permet de délimiter les zones de la page qui doivent être mises à jour, lors d'une mise à jour partielle de la page. L'attribut [UpdateMode="Conditional"] indique que la zone ne doit être mise à jour que sur un événement AJAX d'un composant de la zone. L'autre valeur est [UpdateMode="Always"] et c'est la valeur par défaut. Avec cet attribut, la zone *UpdatePanel* est mise à jour systématiquement même si l'événement Ajax qui s'est produit provient d'un composant d'une autre zone *UpdatePanel*. En général, ce comportement n'est pas désirable.

  La balise <asp:UpdatePanel> admet deux balises enfant : <ContentTemplate> et <Triggers>.

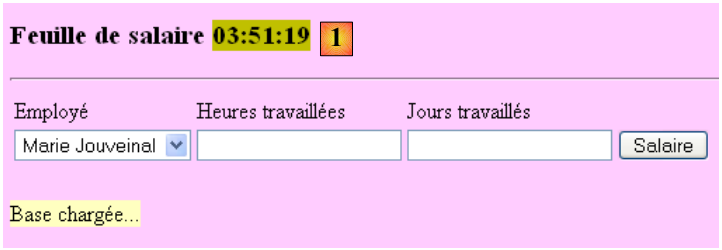
- les balises <asp:ContentTemplate>, lignes 10 et 53, délimitent la zone de la page à mettre à jour partiellement.
- lignes 27-33 : un composant Ajax <asp:UpdateProgress> permettant d'afficher un texte pendant toute la durée de la mise à jour de la page. Par exemple, le clic sur le bouton [Salaire] va provoquer un POST en arrière-plan. Le navigateur ne met alors pas le sablier et l'utilisateur peut être ainsi tenté de continuer à utiliser le formulaire. La balise <asp:UpdateProgress> permet d'afficher un texte avertissant qu'une mise à jour de la page est en cours. On peut également afficher une image. Ici, on affiche une image animée (ligne 29) ainsi qu'un texte (lignes 30-31) :



- lignes 28-32 : la balise <ProgressTemplate> délimite le contenu qui sera affiché pendant toute la durée de la mise à jour de la zone *UpdatePanel* dans laquelle se trouve la balise *UpdateProgress*.
- lignes 29-31 : l'image animée et le texte qui seront affichés pendant la mise à jour de la zone *UpdatePanel* .

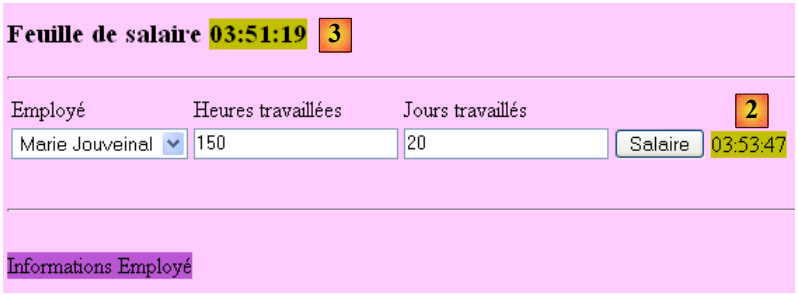
7.3 Tests de la solution Ajax

Lorsqu'on demande l'exécution du projet (CTRL-F5), on obtient la page suivante :

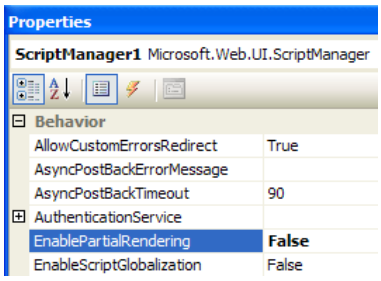


- en [1], l'heure de chargement de la page

Faites des essais et vérifiez que le bouton [Salaire] ne provoque pas le rechargement complet de la page. Vous pouvez le voir avec l'heure affichée pour le traitement du clic sur le bouton [Salaire] [2] qui n'est pas le même que l'heure du chargement initial de la page [3] :



Refaire les tests en mettant la propriété *EnablePartialRendering* du composant *ScriptManager1* à *False* :



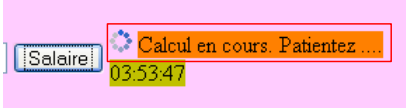
Constater qu'alors on retrouve le comportement d'une page non ajaxifiée. Il y a rechargement total de la page lors du clic sur le bouton [Salaire].

Refaire les tests avec un autre navigateur. Le test multi-navigateurs a pour but de montrer que le javascript généré par les composants serveur ASP / AJAX est correctement interprété par les différents navigateurs.

Pour finir, mettons en lumière le rôle de la balise <asp:UpdateProgress>. Dans le code de la procédure [ButtonSalaire_Click] de [Default.aspx.vb], ajoutons une instruction qui arrête la procédure pendant 5 secondes :

```
1.   Imports System.Threading
2.
3.  ....
4.  Private Sub ButtonSalaire_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
    Handles ButtonSalaire.Click
5.
6.  ...
7.     ' attente
8.     Thread.Sleep(5000)
9.  End Sub
```

La ligne 8 fait attendre 5 secondes le thread qui exécute la procédure [ButtonSalaire_Click]. Ceci fait, refaisons des tests. Cette fois, on voit le texte de [UpdateProgress] ainsi que l'image animée pendant le calcul du salaire :



7.4 Conclusion

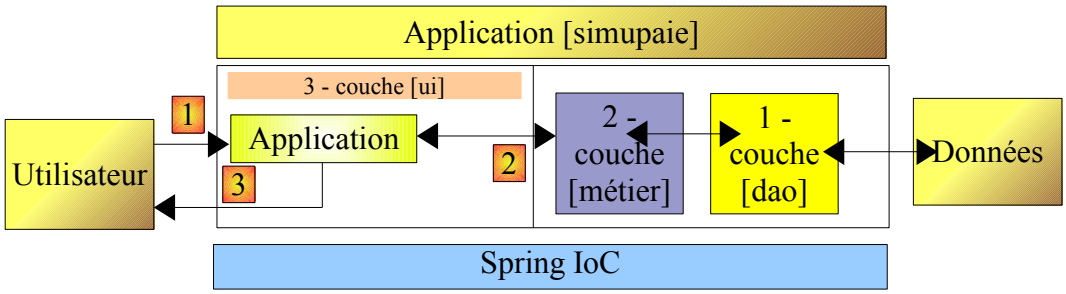
L'étude précédente nous a montré qu'il était possible d'ajaxifier des applications ASP.NET existantes. Les extensions AJAX d'ASP.NET vont plus loin que ce qui vient d'être vu. Le lecteur est invité à consulter le site [http://ajax.asp.net].

8 L'application [SimuPaie] – version 6 – VB.NET / 3 couches

Lectures conseillées : référence [4] " **Construction d'une application web à trois couches avec Spring et VB.NET - Partie 1** ".

8.1 Architecture générale de l'application

L'application [SimuPaie] aura maintenant la structure à trois couches suivante :



- la couche [1-dao] (dao=Data Access Object) s'occupera de l'accès aux données.
- la couche [2-métier] s'occupera de l'aspect métier de l'application, le calcul de la paie.
- la couche [3-ui] (ui=User Interface) s'occupera de la présentation des données à l'utilisateur et de l'exécution de ses requêtes. Nous appelons [Application] l'ensemble des modules assurant cette fonction. Elle est l'interlocuteur de l'utilisateur.
- les trois couches seront rendues indépendantes grâce à l'utilisation d'interfaces .NET
- l'intégration des différentes couches sera réalisée par **Spring IoC**

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

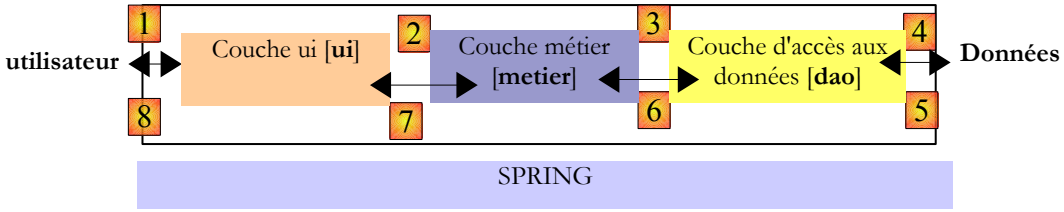
1. le client fait une demande à l'application.
2. l'application traite cette demande. Pour ce faire, elle peut avoir besoin de l'aide de la couche [métier] qui elle-même peut avoir besoin de la couche [dao] si des données doivent être échangées avec la base de données.
3. l'application reçoit une réponse de la couche [métier]. Selon celle-ci, elle envoie la vue (= la réponse) appropriée au client.

Nous commençons par une application VB.NET où la couche [ui] est implémentée par le formulaire de la version 1 (cf page 8).

8.2 Les interfaces des différentes couches de l'application

Rappelons qu'une interface est un contrat définissant les signatures d'un groupe de méthodes auxquelles les classes implémentant l'interface doivent donner un contenu. C'est le compilateur qui vérifie le respect du contrat.

Revenons à l'architecture 3 couches de notre application :



Dans ce type d'architecture, c'est souvent l'utilisateur qui prend les initiatives. Il fait une demande en [1] et reçoit une réponse en [8]. On appelle cela le cycle demande - réponse. Prenons l'exemple du calcul de la paie d'une assistante maternelle. Celui-ci va nécessiter plusieurs étapes :

(a) la couche [ui] va devoir demander à l'utilisateur
- l'identité de la personne dont on veut faire la paie
- le nombre de jours travaillés de celle-ci

- le nombre d'heures travaillées

(b) Pour cela elle va devoir présenter à celui-ci la liste des personnes (nom, prénom, SS) présentes dans la table [EMPLOYES] afin que l'utilisateur choisisse l'une d'elles. La couche [ui] va utiliser le chemin [2, 3, 4, 5, 6, 7] pour les obtenir. L'opération [2] est la demande de la liste des employés, l'opération [7] la réponse à cette demande. Ceci fait, la couche [ui] peut présenter la liste des employés à l'utilisateur par [8].

(c) l'utilisateur va transmettre à la couche [ui] le nombre de jours travaillés ainsi que le nombre d'heures travaillées. C'est l'opération [1] ci-dessus. Au cours de cette étape, l'utilisateur n'interagit qu'avec la couche [ui]. C'est celle-ci qui va notamment vérifier la validité des données saisies. Ceci fait, l'utilisateur va demander le calcul de la paie.

(d) la couche [ui] va demander à la couche métier de faire ce calcul. Pour cela elle va lui transmettre les données qu'elle a reçues de l'utilisateur. C'est l'opération [2].

(e) la couche [metier] a besoin de certaines informations pour mener à bien son travail :
- des informations plus complètes sur la personne (adresse, indice, ...)
- les indemnités liées à son indice
- les taux des différentes cotisations sociales à prélever sur le salaire brut

Elle va demander ces informations à la couche [dao] avec le chemin [3, 4, 5, 6]. [3] est la demande initiale et [6] la réponse à cette demande.

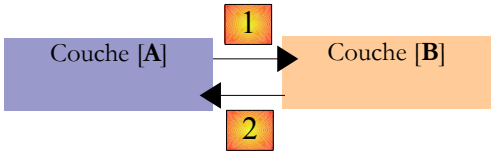
(f) ayant toutes les données dont elle avait besoin, la couche [metier] calcule la paie de la personne choisie par l'utilisateur.

(g) la couche [metier] peut maintenant répondre à la demande de la couche [ui] faite en (d). C'est le chemin [7].

(h) la couche [ui] va mettre en forme ces résultats pour les présenter à l'utilisateur sous une forme appropriée puis les présenter. C'est le chemin [8].

(i) on peut imaginer que ces résultats doivent être mémorisés dans un fichier ou une base de données. Cela peut être fait de façon automatique. Dans ce cas, après l'opération (f), la couche [metier] va demander à la couche [dao] d'enregistrer les résultats. Ce sera le chemin [3, 4, 5, 6]. Cela peut être fait également sur demande de l'utilisateur. Ce sera le chemin [1-8] qui sera utilisé par le cycle demande - réponse.

On voit dans cette description qu'une couche est amenée à utiliser les ressources de la couche qui est à sa droite, jamais de celle qui est à sa gauche. Considérons deux couches contigües :



La couche [A] fait des demandes à la couche [B]. Dans les cas les plus simples, une couche est implémentée par une unique classe. Une application évolue au cours du temps. Ainsi la couche [B] peut avoir des classes d'implémentation différentes [B1, B2, ...]. Si la couche [B] est la couche [dao], celle-ci peut avoir une première implémentation [B1] qui va chercher des données dans un fichier. Quelques années plus tard, on peut vouloir mettre les données dans une base de données. On va alors construire une seconde classe d'implémentation [B2]. Si dans l'application initiale, la couche [A] travaillait directement avec la classe [B1] on est obligés de réécrire partiellement le code de la couche [A]. Supposons par exemple qu'on ait écrit dans la couche [A] quelque chose comme suit :

```
1. Dim _B1 as B1=new B1(...)
2. ..
3. _B1.getData(...)
```

- ligne 1 : une instance de la classe [B1] est créée
- ligne 3 : des données sont demandées à cette instance

Si on suppose, que la nouvelle classe d'implémentation [B2] utilise des méthodes de même signature que celle de la classe [B1], il suffit de changer la ligne :

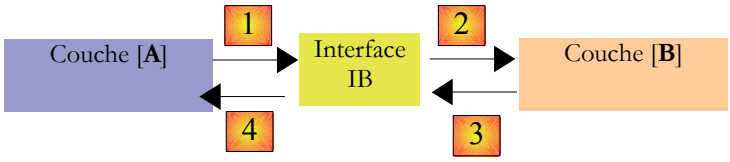
```
Dim _B1 as B1=new B1(...)
```

en

```
Dim _B1 as B2=new B2(...)
```

Ca, c'est le cas très favorable, et assez improbable si on n'a pas prêté attention à ces signatures de méthodes. Dans la pratique, il est fréquent que les classes [B1] et [B2] n'aient pas les mêmes signatures de méthodes et que donc une bonne partie de la couche [A] doive être totalement réécrite.

On peut améliorer les choses si on met une interface entre les couches [A] et [B]. Cela signifie qu'on fige dans une interface les signatures des méthodes présentées par la couche [B] à la couche [A]. Le schéma précédent devient alors le suivant :



La couche [A] ne s'adresse désormais plus directement à la couche [B] mais à son interface [IB]. Ainsi dans le code de la couche [A], la classe d'implémentation [Bi] de la couche [B] n'apparaît qu'une fois, au moment de l'implémentation de l'interface [IB]. Ailleurs, c'est le nom de l'interface [IB] qui apparaît. Le code précédent devient celui-ci :

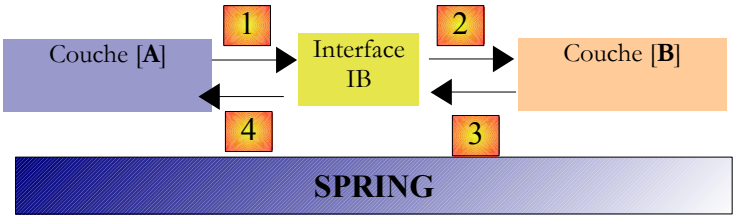
```
1. dim _IB as IB=new B1(...)
2. ..
3. _IB.getData(...);
```

- ligne 1 : une instance [_IB] implémentant l'interface [IB] est créée par instanciation de la classe [B1]
- ligne 3 : des données sont demandées à l'instance [_IB]

Si on remplace l'implémentation [B1] de la couche [B] par une implémentation [B2], et que ces deux implémentations respectent la même interface [IB], seule la ligne 1 de la couche [A] doit être modifiée et aucune autre. C'est un grand avantage qui à lui seul justifie l'usage systématique des interfaces entre deux couches.

On peut aller encore plus loin et rendre la couche [A] totalement indépendante de la couche [B]. Dans le code ci-dessus, la ligne 1 pose problème parce qu'elle référence en dur la classe [B1]. L'idéal serait que la couche [A] puisse disposer d'une implémentation de l'interface [IB] sans avoir à nommer de classe. Ce serait cohérent avec notre schéma ci-dessus. On y voit que la couche [A] s'adresse à l'interface [IB] et on ne voit pas pourquoi elle aurait besoin de connaître le nom de la classe qui implémente cette interface. Ce détail n'est pas utile à la couche [A].

Le framework **Spring** (http://www.springframework.net) permet d'obtenir ce résultat. L'architecture précédente évolue de la façon suivante :



La couche transversale [Spring] à une couche d'obtenir par configuration une référence sur la couche située à sa droite sans avoir à connaître le nom de la classe d'implémentation de celle-ci. Ce nom sera dans les fichiers de configuration et non pas dans le code .NET. Le code de la couche [A] pourrait être le suivant :

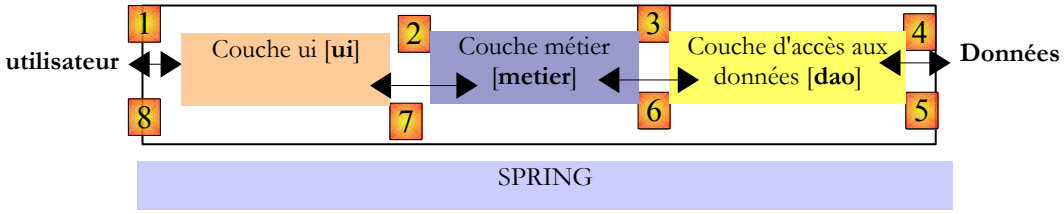
```
1. dim _IB as IB ' initialisé par Spring
2. ..
3. _IB.getData(...);
```

- ligne 1 : une instance [_IB] implémentant l'interface [IB] de la couche [B]. Cette instance est créée par Spring sur la base d'informations trouvées dans un fichier de configuration. Spring va s'occuper de créer :
    - l'instance [b] implémentant la couche [B]
    - l'instance [a] implémentant la couche [A]. Cette instance sera initialisée. Le champ [_IB] ci-dessus recevra pour valeur la référence [b] de l'objet implémentant la couche [B]
- ligne 3 : des données sont demandées à l'instance [_IB]

On voit que maintenant, la classe d'implémentation [B1] de la couche B n'apparaît nulle part dans le code de la couche [A]. Lorsque l'implémentation [B1] sera remplacée par une nouvelle implémentation [B2], rien ne changera dans le code de la classe [A]. On changera simplement les fichiers de configuration de Spring pour instancier un objet de type [B2] au lieu d'un objet de type [B1].

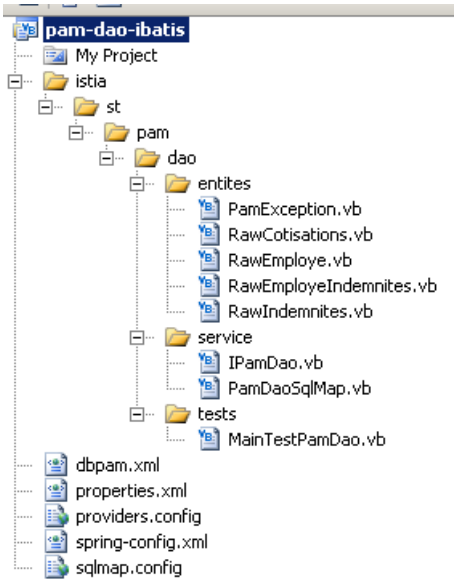
Le couple **Spring** et **interfaces** apporte une amélioration décisive à la maintenance d'applications en rendant les couches de celles-ci étanches entre elles. C'est cette solution que nous utiliserons pour l'application [SimuPaie].

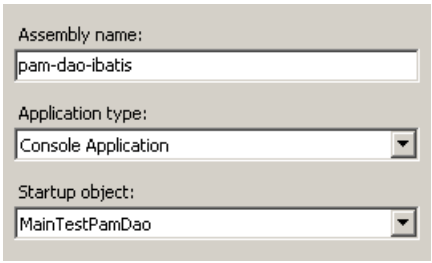
8.3 La couche [dao] d'accès aux données



8.3.1 Le projet Visual Studio de la couche [dao]

Le projet Visual Studio de la couche [dao] est le suivant :



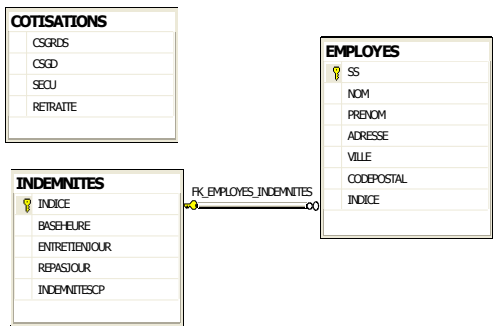


Le projet est configuré pour générer l'exécutable [pam-dao-ibatis.exe]

Les entités (objets) nécessaires à la couche [dao] ont été rassemblées dans le dossier [entites] du projet. Certaines nous sont déjà connues : [RawCotisations] décrite page 13, [RawEmploye] décrite page 12, [RawIndemnites] décrite page 14. On les a simplement placées dans un autre espace de noms : *istia.st.pam.dao*. Nous décrivons maintenant les autres entités.

8.3.2 La classe [RawEmployeIndemnites]

Un objet de type [RawEmployeIndemnites] sert à encapsuler une ligne de la table [EMPLOYES] associée à la ligne de la table [INDEMNITES] à laquelle elle est liée via sa clé étrangère [indice] :



Le type [RawEmployeIndemnites] est défini comme suit :

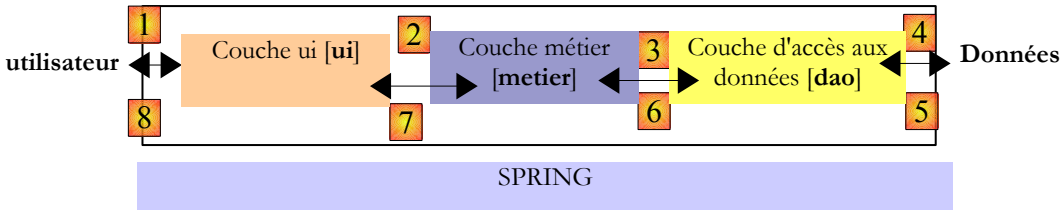
```
1.  Namespace istia.st.pam.dao.entites
2.
3.    Public Class RawEmployeIndemnites
```

```
4.      Inherits RawEmploye
5.
6.      ' champs privés
7.      Private _Indemnites As RawIndemnites
8.
9.      ' propriétés associées
10.     Public Property Indemnites() As RawIndemnites
11.       Get
12.         Return _Indemnites
13.       End Get
14.       Set(ByVal value As RawIndemnites)
15.         _Indemnites = value
16.       End Set
17.     End Property
18.
19.     ' constructeurs
20.     Public Sub New()
21.
22.     End Sub
23.
24.     Public Sub New(ByVal ss As String, ByVal nom As String, ByVal prenom As String, ByVal adresse
    As String, ByVal codePostal As String, ByVal ville As String, ByVal indice As Integer, ByVal
    indemnites As RawIndemnites)
25.       ' init parent
26.       MyBase.New(ss, nom, prenom, adresse, codePostal, ville, indice)
27.       ' init Me
28.       Me.Indemnites = indemnites
29.     End Sub
30.
31.     Public Sub New(ByVal employe As RawEmploye, ByVal indemnites As RawIndemnites)
32.       ' init parent
33.       MyBase.New(employe.SS, employe.Nom, employe.Prenom, employe.Adresse, employe.CodePostal,
    employe.Ville, employe.Indice)
34.       ' init Me
35.       Me.Indemnites = indemnites
36.     End Sub
37.
38.     ' ToString
39.     Public Overrides Function ToString() As String
40.       Return String.Format("[{0},{1}]", MyBase.ToString, Indemnites)
41.     End Function
42.   End Class
43. End Namespace
```

- ligne 1 : la classe appartient à l'espace de noms [istia.st.pam.dao.entites]
- lignes 3-4 : la classe dérive de la classe [RAWEMPLOYE]
- ligne 7 : le champ privé qui reçoit la ligne de la table [INDEMNITES] référencée par le champ [_Indice] de l'employé.
- lignes 10-17 : la propriété publique associée au champ privé précédent
- lignes 20-22 : le constructeur par défaut
- lignes 24-29 : le constructeur qui a pour paramètres les 8 informations qui vont initialiser les 8 champs privés
- lignes 31-36 : le constructeur qui a pour paramètres deux objets de types respectifs [RawEmploye] pour initialiser l'objet parent et [RawIndemnites] pour initialiser le champ privé [_Indemnites] de la ligne 7.
- lignes 24-26 : la méthode [ToString] de la classe

8.3.3 L'interface [IPamDao] de la couche [dao]

Revenons à l'architecture à trois couches de notre application :



Dans les cas simples, on peut partir de la couche [metier] pour découvrir les interfaces de l'application. Pour travailler, elle a besoin de données :

- déjà disponibles dans des fichiers, bases de données ou via le réseau. Elles sont fournies par la couche [dao].
- pas encore disponibles. Elles sont alors fournies par la couche [ui] qui les obtient auprès de l'utilisateur de l'application.

Quelle interface doit offrir la couche [dao] à la couche [metier] ? Quelles sont les interactions possibles entre ces deux couches ? La couche [dao] doit fournir les données suivantes à la couche [metier] :

- la liste des assistantes maternelles afin de permettre à l'utilisateur d'en choisir une en particulier
- des informations complètes sur la personne choisie (adresse, indice, ...)
- les indemnités liées à l'indice de la personne
- les taux des différentes cotisations sociales

Ces informations sont en effet connues avant le calcul de la paie et peuvent donc être mémorisées. Dans le sens [metier] -> [dao], la couche [metier] peut demander à la couche [dao] d'enregistrer le résultat du calcul de la paie. Nous ne le ferons pas ici.

Avec ces informations, on pourrait tenter une première définition de l'interface de la couche [dao] :

```
1.  Imports istia.st.pam.dao.entites
2.
3.  Namespace istia.st.pam.dao.service
4.    Public Interface IPamDao
5.      ' tableau de toutes les identités des employés
6.      Function GetAllIdentitesEmployes() As RawEmploye()
7.      ' un employé particulier
8.      Function GetEmployeIndemnites(ByVal SS As String) As RawEmployeIndemnites
9.      ' liste de toutes les cotisations
10.     Function GetCotisations() As RawCotisations
11.   End Interface
12. End Namespace
```

- ligne 1 : on importe l'espace de noms des entités de la couche [dao].
- ligne 3 : la couche [dao] est dans l'espace de noms [istia.st.pam.dao.service]. Les éléments de l'espace de noms [entites] peuvent être créés en plusieurs exemplaires. Les éléments de l'espace de noms [service] sont créés en un unique exemplaire (singleton). C'est ce qui a justifié le choix des noms des espaces de noms.
- ligne 4 : l'interface s'appelle [IPamDao]. Elle définit trois méthodes :
    - [GetAllIdentitesEmployes] rend un tableau d'objets de type [RawEmploye] qui représente la liste des assistantes maternelles sous une forme simplifiée (nom, prénom, SS).
    - [GetEmployeIndemnites] rend un objet [RawEmployeIndemnites]. La partie [RawEmploye] de l'objet est l'employé qui a le n° de sécurité sociale passé en paramètre à la méthode et la partie [RawIndemnites] encapsule les indemnités liées à l'indice de l'employé.
    - [GetCotisations] rend l'objet [RawCotisations] qui encapsule les taux des différentes cotisations sociales à prélever sur le salaire brut.

8.3.4 La classe [PamException]

La couche [dao] est chargée d 'échanger des données avec une source extérieure. Cet échange peut échouer. Par exemple, si les informations sont demandées à un service distant sur Internet, leur obtention échouera sur une panne quelconque du réseau. Sur ce type d'erreurs, il est classique en Java de lancer une exception. Si l'exception n'est pas de type [RunTimeException] ou dérivé, il faut indiquer dans la signature de la méthode que celle-ci lance (throws) une exception. En .NET, toutes les exceptions sont non contrôlées, c.a.d. équivalentes au type [RunTimeException] de Java. Il n'y a alors pas lieu de déclarer que les méthodes [GetAllIdentitesEmployes, GetEmployeIndemnites, GetCotisations] sont susceptibles de lancer une exception.

Il est cependant intéressant de pouvoir différencier les exceptions les unes des autres car leur traitement peut différer. Ainsi le code gérant divers types d'exceptions peut être écrit de la façon suivante :

```
try
  ... code pouvant générer divers types d'exceptions
catch ex1 as Exception1
...on gère un type d'exceptions
catch ex2 as Exception2
...on gère un autre type d'exceptions
end try
```

Nous créons donc un type d'exceptions pour la couche [dao] de notre application. C'est le type [PamException] suivant :

```
1.  Namespace istia.st.pam.dao.entites
2.
3.    Public Class PamException
4.      Inherits Exception
5.
6.      ' le code de l'erreur
7.      Private _Code As Integer
8.
9.      ' propriété associée
10.     Public Property Code() As Integer
```

```
11.       Get
12.         Return _Code
13.       End Get
14.       Set(ByVal value As Integer)
15.         _Code = value
16.       End Set
17.     End Property
18.
19.     ' constructeurs
20.     Public Sub New()
21.
22.     End Sub
23.
24.     Public Sub New(ByVal Code As Integer)
25.       MyBase.New()
26.       Me.Code = Code
27.     End Sub
28.
29.     Public Sub New(ByVal message As String, ByVal Code As Integer)
30.       MyBase.new(message)
31.       Me.Code = Code
32.     End Sub
33.
34.     Public Sub New(ByVal message As String, ByVal ex As Exception, ByVal Code As Integer)
35.       MyBase.New(message, ex)
36.       Me.Code = Code
37.     End Sub
38.   End Class
39. End Namespace
```

- ligne 1 : la classe appartient à l'espace de noms [istia.st.pam.dao.entites]
- lignes 3-4 : la classe dérive de la classe [Exception]
- ligne 7 : elle a un champ privé [_Code] qui est un code d'erreur
- lignes 10-17 : la propriété publique associée au champ privé précédent
- nous utiliserons dans notre couche [dao] deux sortes de constructeur :
  - celui des lignes 29-32 qu'on peut utiliser comme montré ci-dessous :

```
throw new PamException("Problème d'accès aux données",5)
```

  - ou celui des lignes 34-37 destiné à faire remonter une exception déjà survenue en l'encapsulant dans une exception de type [PamException] :

```
try
....
catch ex as IOException
  ' on encapsule l'exception
  throw new PamException("Problème d'accès aux données",ex,10);
end try
```

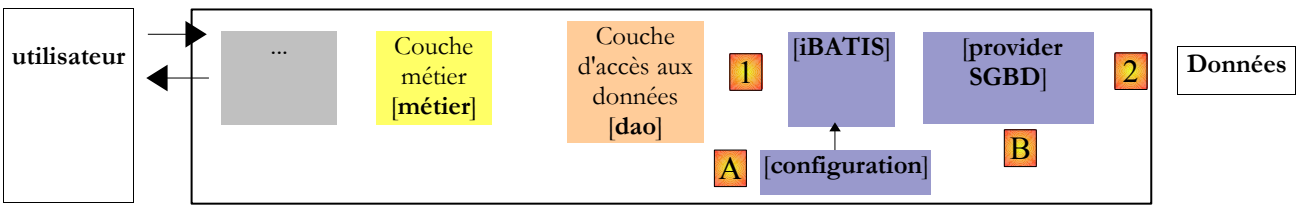
Cette seconde méthode a l'avantage de ne pas perdre l'information que peut contenir la première exception.

8.4 Implémentation de la couche [dao] avec [Ibatis SqlMap]

Lectures conseillées : référence [4] " **Construction d'une application web à trois couches avec Spring et VB.NET - Partie 2** ".

8.4.1 Le produit iBatis SqlMap

Nous nous proposons d'implémenter maintenant l'interface **IPamDao** en utilisant un produit Open Source appelé **iBatis SqlMap** (http://ibatis.apache.org). Cet outil rend possible le développement de couches d'accès aux données, indépendantes de la nature réelle de la source de données.



L'accès aux données est assuré à l'aide :

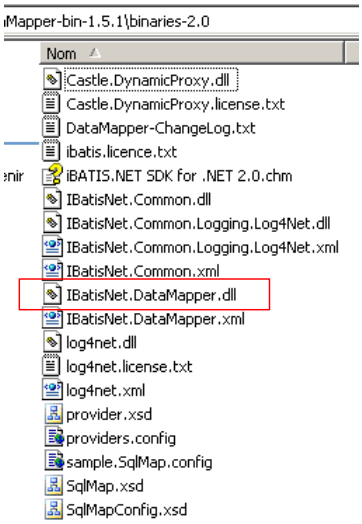
- [A] : de fichiers de configuration dans lesquels sont placées les informations définissant la source de données et les opérations que l'on veut faire dessus
- [B] : d'une bibliothèque de classes propre au SGBD utilisé pour accéder aux données

L'outil Ibatis SqlMap a été développé initialement pour la plate-forme Java puis a été porté sur la plate-forme .NET.

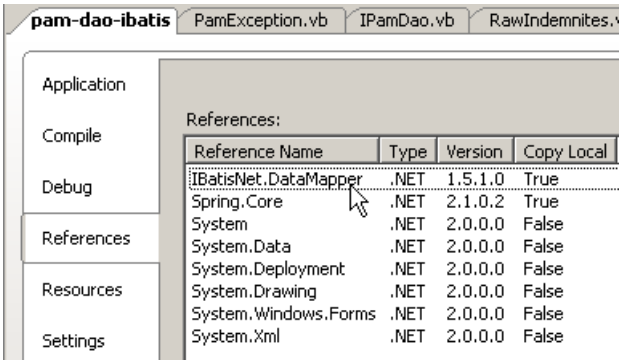
8.4.2 Où trouver IBATIS SqlMap ?

Le site principal d'Ibatis est [http://ibatis.apache.org/]. La page de téléchargements offre les liens suivants :

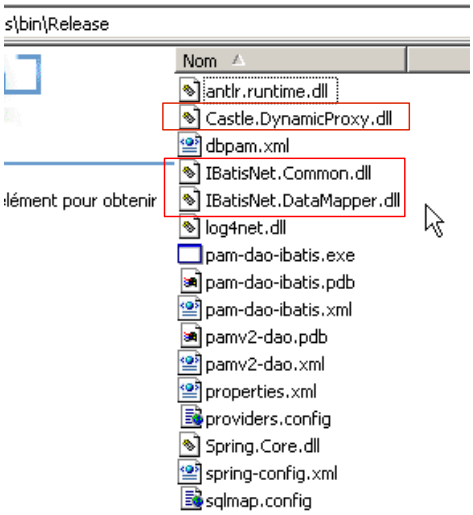
On choisira le lien [DataMapper] qui nous emmène chez [SourceForge.net]. Suivre le processus de téléchargement jusqu'au bout. On obtient un zip contenant les fichiers suivants :



Dans un projet Visual Studio utilisant Ibatis SqlMap, il faut ajouter au projet une référence à la DLL [IbatisNet.DataMapper]. Voici par exemple, les références du projet [pam-dao-ibatis] :



La DLL [IbatisNet.DataMapper] est copiée (Copy Local=true ci-dessus) dans le dossier [bin/Release] du projet :



La DLL [IbatisNet.DataMapper] utilise elle-même d'autres DLL [IbatisNet.Common, Castle.DynamicProxy] qui elles aussi seront recopiées dans le dossier [/bin/release].

8.4.3 Les fichiers de configuration d'iBatis SqlMap

Une source de données [SqlMap] va être définie au moyen des fichiers de configuration suivants :

1. **providers.config** : définit les bibliothèques de classes à utiliser pour accéder aux données
2. **sqlmap.config** : définit les caractéristiques de la connexion à établir
3. **fichiers de mapping** : définissent les opérations à faire sur les données

La logique de ces fichiers est la suivante :

- pour accéder aux données, il va nous falloir une connexion. En .NET, il existe différentes bibliothèques d'accès aux bases de données selon le type d'accès :
    - accès à une source ODBC
    - accès à une source OleDb
    - accès à une source SqlServer
    - ...

  De la même façon qu'en Java, les SGBD fournissent des pilotes JDBC permettant à du code Java d'accéder directement aux données gérées par le SGBD, certains d'entre-eux fournissent également des bibliothèques de classes qui permettent à du code .NET d'accéder aux données. L'inconvénient est que chaque SGBD vient avec sa propre bibliothèque de classes et que le code .NET est alors dépendant du SGBD utilisé. Changer de SGBD n'est pas transparent : il faut changer le code d'accès aux données. C'est un inconvénient important de la plate-forme .NET 1.1 corrigée depuis dans la version 2.0. Ibatis SqlMap apporte également une solution à ce problème et va un peu plus loin que la solution apportée par .NET 2.0.
- le fichier [**providers.config**] définit des fournisseurs d'accès aux données. A chacun d'eux est associée une bibliothèque de classes. Utiliser un SGBD particulier avec Ibatis SqlMap revient à préciser quel fournisseur d'accès (provider) utiliser. IBatis sait alors quelle bibliothèque de classes utiliser pour les opérations courantes d'accès aux données :
    - ouvrir une connexion
    - émettre un ordre SQL de type *Select* et exploiter le résultat

        - émettre un ordre SQL de type *Update*, *Insert*, *Delete* et exploiter le résultat
        - fermer la connexion
- le fichier [**sqlmap.config**] définit :
  - le *provider* de providers.config à utiliser
  - la chaîne de connexion à la base qui contient les données.

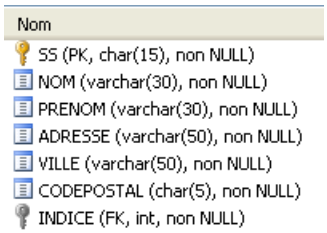
  La connexion à la base sera ouverte par instanciation de la classe [Connection] définie par le *provider* choisi, au constructeur duquel sera passée la chaîne de connexion définie dans [sqlmap.config].
- les **fichiers de mapping** définissent :
  - des associations entre lignes de tables de données et classe .NET dont les instances contiendront ces lignes
  - les opérations SQL à exécuter. Celles-ci sont identifiées par un nom. Le code .NET exécute ces opérations via leur nom, ce qui a pour conséquence d'éliminer tout code SQL du code .NET.

8.4.4 Les fichiers de configuration du projet [pam-dao-ibatis]

Examinons sur notre exemple, la nature exacte des fichiers de configuration de SqlMap. Revenons sur la base SQL Server [dbpam] que nous allons utiliser. Elle a trois tables, **EMPLOYES, COTISATIONS** et **INDEMNITES,** dont la structure est la suivante :

Table **EMPLOYES** : rassemble des informations sur les différentes assistantes maternelles

Structure :



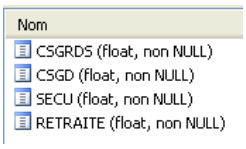
SS	numéro de sécurité sociale de l'employé - clé primaire
NOM	nom de l'employé
PRENOM	son prénom
ADRESSE	son adresse
VILLE	sa ville
CODEPOSTAL	son code postal
INDICE	son indice de traitement - clé étrangère sur le champ [INDICE] de la table [INDEMNITES]

Son contenu pourrait être le suivant :

SS	NOM	PRENOM	ADRESSE	VILLE	CODEPOSTAL	INDICE
254104940426058	Jouveinal	Marie	5 rue des Oiseaux	St Corentin	49203	2
260124402111742	Laverti	Justine	la Brûlerie	St Marcel	49014	1

Table **COTISATIONS** : rassemble les taux des cotisations sociales prélevées sur le salaire

Structure :

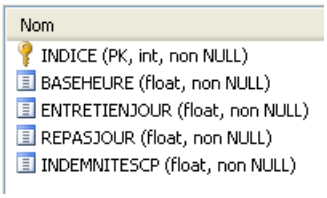


CSGRDS	pourcentage : contribution sociale généralisée + contribution au remboursement de la dette sociale
CSGD	pourcentage : contribution sociale généralisée déductible
SECU	pourcentage : sécurité sociale
RETRAITE	pourcentage : retraite complémentaire + assurance chômage

Son contenu pourrait être le suivant :

CSGRDS	CSGD	SECU	RETRAITE
3,49	6,15	9,39	7,88

Table **INDEMNITES** : rassemble les différentes indemnités dépendant de l'indice de l'employé



INDICE	indice de traitement - clé primaire
BASEHEURE	prix net en euro d'une heure de garde
ENTRETIENJOUR	indemnité d'entretien en euro par jour de garde
REPASJOUR	indemnité de repas en euro par jour de garde
INDEMNITESCP	indemnité de congés payés. C'est un pourcentage à appliquer au salaire de base.

Son contenu pourrait être le suivant :

INDICE	BASEHEURE	ENTRETIENJOUR	REPASJOUR	INDEMNITESCP
1	1,93	2	3	12
2	2,1	2,1	3,1	15

8.4.4.1 providers.config

Le fichier [providers.config] pour une source SQL Server est le suivant :

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <providers
3.  xmlns="http://ibatis.apache.org/providers"
4.  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
5.
6.  <clear/>
7.  <provider
8.      name="sqlServer2.0"
9.      enabled="true"
10.     description="Microsoft SQL Server, provider V2.0.0.0 in framework .NET V2.0"
11.     assemblyName="System.Data, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
12.     connectionClass="System.Data.SqlClient.SqlConnection"
13.     commandClass="System.Data.SqlClient.SqlCommand"
14.     parameterClass="System.Data.SqlClient.SqlParameter"
15.     parameterDbTypeClass="System.Data.SqlDbType"
16.     parameterDbTypeProperty="SqlDbType"
17.     dataAdapterClass="System.Data.SqlClient.SqlDataAdapter"
18.     commandBuilderClass=" System.Data.SqlClient.SqlCommandBuilder"
19.     usePositionalParameters = "false"
20.     useParameterPrefixInSql = "true"
21.     useParameterPrefixInParameter = "true"
22.     parameterPrefix="@"
23.     allowMARS="false"
24.     />
25. </providers>
```

Commentaires :

- un fichier [providers.config] est distribué avec le framework [iBatis / SqlMap]. Il propose plusieurs fournisseurs d'accès (provider) standard. Le code ci-dessus provient de ce fichier. Nous n'avons gardé que le provider [sqlServer2.0] (ligne 8) qui est le fournisseur d'accès pour le SGBD SQL Serveur Express 2005.
- un <provider> est défini par les lignes 7-24 ci-dessus. Le fichier [providers.config] distribué avec le framework iBatis offre des providers pour divers SGBD. En voici une liste à la date de novembre 2006 : SQL Server 1.0, SQL Server 1.1, SQL Server 2.0, OleDb1.1, OleDb2.0, Odbc1.1, Odbc2.0, oracle9.2, oracle10.1, oracleClient1.0, ByteFx (MySQL), MySql, SQLite3, Firebird1.7, Firebird2.0, PostgreSql0.99.1.0, iDb2.10, Informix. Le fichier [providers.config] n'est pas figé. On peut lui ajouter soi-même des sections <provider>, pour peu qu'on ait les renseignements suivants :
    - nom, version et jeton de la DLL contenant les bibliothèques de classe d'accès aux données
    - nom de chaque classe spécialisée. Par exemple l'attribut [connectionClass] (ligne 12) de la balise <provider> doit recevoir pour valeur le nom de la classe permettant la connexion au SGBD.
- un <provider> a un nom - ligne 8 - peut être quelconque
- ligne 9 : un <provider> peut être activé [enabled=true] ou non [enabled=false]. S'il est activé, la DLL référencée ligne 11 doit être accessible car elle est chargée à la lecture du fichier [providers.config]. En général, on ne conservera dans [providers.config] que la seule balise <provider> dont on a besoin.
- ligne 10 - description du <provider> - peut être quelconque
- ligne 11 - nom de l'assembly qui contient les classes définies lignes 12-18
- ligne 12 - classe à utiliser pour créer une connexion
- ligne 13 - classe à utiliser pour créer un objet [Command] d'émission de commandes SQL
- ligne 14 - classe à utiliser pour gérer les paramètres d'une commande SQL paramétrée
- ligne 15 - classe d'énumération des types de données possibles pour les champs d'une table
- ligne 16 - nom de la propriété d'un objet [Parameter] qui contient le type de la valeur de ce paramètre
- ligne 17 - nom de la classe [Adapter] permettant de créer des objets [DataSet] à partir de la source de données

- ligne 18 - nom de la classe [CommandBuilder] qui, associée à un objet [Adapter], permet de générer automatiquement les propriétés [InsertCommand, DeleteCommand, UpdateCommand] de celui-ci à partir de sa propriété [SelectCommand]
- lignes 19 - 21 - définissent comment sont gérées les commandes SQL paramétrées. Selon les cas, il faut écrire par exemple :

```
insert into ARTICLES(id,nom,prix,stockactuel,stockminimum) values (?,?,?,?,?)
```

ou bien

```
insert into ARTICLES(id,nom,prix,stockactuel,stockminimum) values
(@id,@nom,@prix,@sa,@sm)
```

Dans le premier cas, on parle de paramètres positionnels formels. Les valeurs effectives de ceux-ci doivent être fournies dans l'ordre des paramètres formels. Dans le second cas, on a affaire à des paramètres nommés. On fournit une valeur à un tel paramètre en précisant son nom. L'ordre n'a plus d'importance.

- ligne 19 - indique que le provider SQLServer2.0 utilise des paramètres nommés
- ligne 20 - indique que le paramètre nommé utilisé dans un ordre SQL est précédé d'un préfixe
- ligne 21 - indique que le paramètre nommé utilisé dans une instruction .NET est précédé d'un préfixe
- ligne 22 - indique que le préfixe est @ pour SQL Server Express 2005

Ces informations permettent à Ibatis-SqlMap de savoir par exemple, quelle classe doit être instanciée pour créer une connexion. Ici ce sera la classe [System.Data.SqlClient.SqlConnection] (ligne 12).

8.4.4.2 sqlmap.config

Le fichier [providers.config] définit les classes à utiliser pour accéder à une source de données gérée SQL Server 2.0 mais n'indique aucune source. C'est le fichier [**sqlmap.config**] qui le fait :

```
1.  <?xml version="1.0" encoding="utf-8" ?>
2.  <sqlMapConfig
3.    xmlns="http://ibatis.apache.org/dataMapper"
4.    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
5.   <properties resource="properties.xml" />
6.   <providers resource="providers.config"/>
7.   <database>
8.       <provider name="${provider}"/>
9.     <dataSource name="dbpam"  connectionString="${connectionString}"/>
10.  </database>
11.  <sqlMaps>
12.    <sqlMap resource="dbpam.xml" />
13.  </sqlMaps>
14. </sqlMapConfig>
```

Commentaires :

- ligne 5 - on définit un fichier de propriétés [**properties.xml**]. Celui-ci définit des couples (clé, valeur). Les clés peuvent être quelconques. La valeur associée à une clé C est obtenue par la notation ${C} dans [**sqlmap.config**]. Voici le fichier [properties.xml] qui sera associé au fichier [sqlmap.config] précédent :

```
1.  <?xml version="1.0" encoding="utf-8" ?>
2.  <settings>
3.    <add key="provider" value="sqlServer2.0" />
4.    <add key="connectionString" value="Data
    Source=.\SQLEXPRESS;AttachDbFilename=D:\data\travail\2006-2007\vbnet\pam\pam-ibatis-3\dao-
    ibatis\database\dbpam.mdf;User Id=sa;Password=msde;Connect Timeout=30;" />
5.  </settings>
```

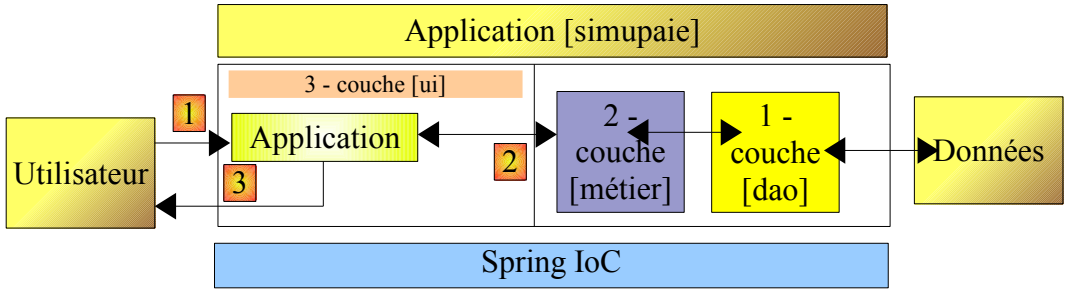
ligne 3 - la clé [provider] est définie. Sa valeur est le nom de la balise <provider> à utiliser dans [providers.config]. Ici, on utilise le provider [sqlServer2.0] qui permet de gérer des sources de données SQL Server Express 2005.
ligne 4 - la clé [connectionString] est définie. Sa valeur est la chaîne de connexion à utiliser pour ouvrir la source de données. Cette chaîne de connexion a été expliquée au paragraphe 3.2, page 9.

Revenons au fichier [sqlmap.config] :

- lignes 7-10 - on définit les caractéristiques de la source de données :
  - ligne 8 - nom du <provider> à utiliser dans [providers.config]
  - ligne 9 - [connectionString] : chaîne de connexion à la source de données, [name] : nom de la source de données. Ce nom est libre.
- lignes 11-13 - liste des fichiers définissant les opérations SQL à effectuer sur la source de données. Ici il n'y en a qu'un : [dbpam.xml]

8.4.4.3 dbpam.xml

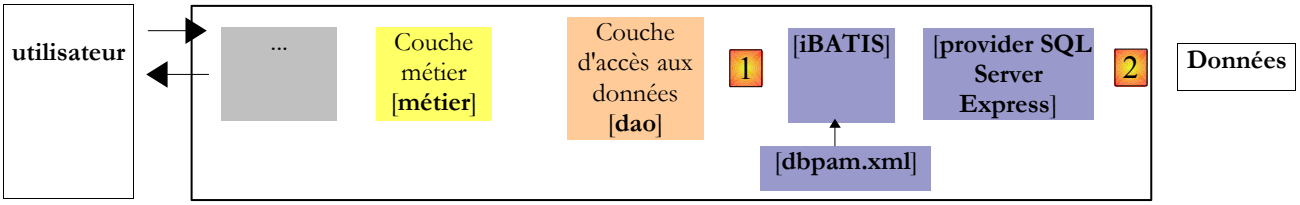
Rappelons l'interface [IPamDao] de la couche [dao] de notre architecture à trois couches :




```
1.  Imports istia.st.pam.dao.entites
2.
3.  Namespace istia.st.pam.dao.service
4.    Public Interface IPamDao
5.      ' liste de toutes les identités des employés
6.      Function GetAllIdentitesEmployes() As RawEmploye()
7.      ' un employé particulier
8.      Function GetEmployeIndemnites(ByVal SS As String) As RawEmployeIndemnites
9.      ' liste de toutes les cotisations
10.     Function GetCotisations() As RawCotisations
11.   End Interface
12. End Namespace
```

La couche [métier] interagit avec la couche [dao] via les méthodes de l'interface [IPamDao]. Ces interactions amènent des échanges d'objets de type [RawCotisations, RawEmploye, RawEmployeIndemnites] entre les deux couches. La couche [métier] ignore le mode de persistance de ces objets (BD, fichiers plats, fichiers XML, services web, ...) et n'a pas à le connaître.

Dans l'implémentation présente de l'interface [IPamDao], le fichier [dbpam.xml] fait le lien entre le monde des bases de données relationnelles (2) et celui des objets demandés par la couche [dao] (1).

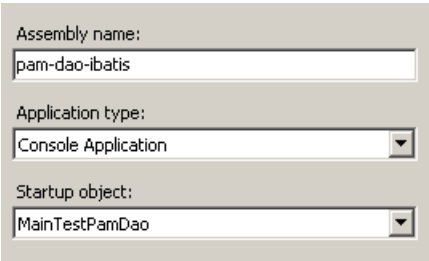


Le fichier [dbpam.xml] est référencé par le fichier principal [**sqlmap.config**] comme définissant des relations entre tables de données et objets .NET ainsi que les ordres SQL à exécuter sur la base. Son contenu sera le suivant :

```
1.  <?xml version="1.0" encoding="utf-8" ?>
2.  <sqlMap
3.    namespace="Pam"
4.    xmlns="http://ibatis.apache.org/mapping"
5.    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
6.    <!-- les resultMaps -->
7.    <resultMaps>
8.      <resultMap id="employe_identite" class="istia.st.pam.dao.entites.RawEmploye, pam-dao-ibatis">
9.        <result property="Nom" column="NOM" />
10.       <result property="Prenom" column="PRENOM" />
11.       <result property="SS" column="SS" />
12.     </resultMap>
13.     <resultMap id="employe" class="istia.st.pam.dao.entites.RawEmploye, pam-dao-ibatis">
14.       <result property="Nom" column="NOM" />
15.       <result property="Prenom" column="PRENOM" />
16.       <result property="Adresse" column="ADRESSE" />
17.       <result property="CodePostal" column="CODEPOSTAL" />
18.       <result property="Ville" column="VILLE" />
19.       <result property="SS" column="SS" />
20.       <result property="Indice" column="Indice" />
21.     </resultMap>
22.     <resultMap id="cotisations" class="istia.st.pam.dao.entites.RawCotisations, pam-dao-ibatis">
23.       <result property="CsgRds" column="CSGRDS" />
24.       <result property="Csgd" column="CSGD" />
25.       <result property="Secu" column="SECU" />
26.       <result property="Retraite" column="RETRAITE" />
27.     </resultMap>
```

```
28.     <resultMap id="indemnites" class="istia.st.pam.dao.entites.RawIndemnites, pam-dao-ibatis">
29.       <result property="Indice" column="INDICE" />
30.       <result property="BaseHeure" column="BASEHEURE" />
31.       <result property="EntretienJour" column="ENTRETIENJOUR" />
32.       <result property="RepasJour" column="REPASJOUR" />
33.       <result property="IndemnitesCP" column="INDEMNITESCP" />
34.     </resultMap>
35.   </resultMaps>
36.   <!-- les requêtes SQL -->
37.   <statements>
38.     <!-- obtention des identités (SS, NOM, PRENOM) de tous les employes -->
39.     <select id="getAllIdentitesEmployes" resultMap="employe_identite">
40.       select SS, NOM, PRENOM FROM EMPLOYES
41.     </select>
42.     <!-- obtention d'un employé-->
43.     <select id="getEmploye" resultMap="employe" parameterClass="string">
44.       select NOM, PRENOM, ADRESSE, VILLE, CODEPOSTAL, SS, INDICE FROM EMPLOYES WHERE SS=#value#
45.     </select>
46.     <!-- obtention des coefficients de cotisations -->
47.     <select id="getCotisations" resultMap="cotisations">
48.       select CSGRDS, CSGD, SECU, RETRAITE FROM COTISATIONS
49.     </select>
50.     <!-- obtention des indemnités avec indice -->
51.     <select id="getAllIndemnites" resultMap="indemnites">
52.       select INDICE, BASEHEURE, ENTRETIENJOUR, REPASJOUR, INDEMNITESCP FROM INDEMNITES
53.     </select>
54.   </statements>
55. </sqlMap>
```

- lignes 2-5 : la balise racine est <sqlMap>. Son attribut [namespace] permet de mettre les identifiants du fichier dans un espace de noms. Ainsi, ligne 8, la balise <resultMap> a en réalité un identifiant (id) égal à [Pam.employe_identite]. Cela est utile lorsque l'application utilise plusieurs fichiers de ressource, ce que *SqlMap* permet. On peut alors imaginer que des développeurs différents utilisent les mêmes identifiants dans des fichiers de ressources différents. Lorsque ceux-ci vont être réunis par SqlMap, il peut y avoir un conflit d'identifiants. Pour éviter cela, chaque fichier de ressources utilisera un attribut [namespace] différent.
- lignes 37-54 (<statements>) : les ordres SQL à exécuter sur la base [dbpam]. Ici nous ne trouvons que des ordres <select>. D'autres applications pourraient utiliser des ordres <insert>, <delete> ou <update>.
- lignes 7-35 (<resultMaps>) : les liaisons entre colonnes des tables résultats des ordres SQL *Select* et les propriétés publiques des objets utilisés par la couche [dao]. Le type de ces objets est défini par l'attribut *class* de la balise <resultMap> sous la forme " nom de la classe, nom de l'assembly ". Ainsi si notre projet actuel est défini comme suit (cf paragraphe 8.3.1, page 43) :



L'*assembly* du projet est [pam-dao-ibatis.exe]. C'est dans cet *assembly* que seront trouvées les classes [Raw*].

Chacune des méthodes de l'interface [IPamDao] est liée à un ou des ordres SQL du fichier [dbpam.xml]. Examinons-les une à une.

```
1.      ' liste de toutes les identités des employés
2.      Function GetAllIdentitesEmployes() As RawEmploye()
```

Cette méthode rend un tableau d'objets de type [RawEmploye] avec pour chaque objet, seulement certains champs renseignés : [Nom, Prenom, SS]. C'est l'ordre SQL *Select* des lignes 39-41 de [dbpam.xml] qui va rendre ces informations.

```
<select id="getAllIdentitesEmployes" resultMap="employe_identite">
  select SS, NOM, PRENOM FROM EMPLOYES
</select>
```

Le <resultMap> d'id="*employe_identite*" des lignes 8-12 de [dbpam.xml] indique comment faire la liaison entre les colonnes [SS, NOM, PRENOM] de la table résultat du *Select* et les propriétés [SS, Nom, Prenom] d'un objet de type [RawEmploye].

```
' liste de toutes les cotisations
Function GetCotisations() As RawCotisations
```

Cette méthode rend une liste d'objets de type [RawCotisations]. C'est l'ordre SQL *Select* des lignes 47-49 de [dbpam.xml] qui va rendre ces informations :

```
<select id="getCotisations" resultMap="cotisations">
  select CSGRDS, CSGD, SECU, RETRAITE FROM COTISATIONS
</select>
```

Le <resultMap> d'id="*cotisations*" des lignes 22-27 de [dbpam.xml] indique comment faire la liaison entre les colonnes [CSGRDS, CSGD, SECU,RETRAITE] de la table résultat du *Select* et les propriétés [CsgRds, Csgd, Secu, Retraite] d'un objet de type [RawCotisations].

```
' un employé particulier avec ses indemnités
Function GetEmployeIndemnites(ByVal SS As String) As RawEmployeIndemnites
```

Cette méthode rend un objet de type [RawEmployeIndemnites]. L'ordre SQL *Select* des lignes 43-45 de [dbpam.xml] va obtenir une partie de ces informations. On notera que l'ordre SQL est paramétré :

```
<select id="getEmploye" resultMap="employe" parameterClass="string">
  select NOM, PRENOM, ADRESSE, VILLE, CODEPOSTAL, SS, INDICE FROM EMPLOYES WHERE SS=#value#
</select>
```

value est un mot clé qui est remplacé à l'exécution de l'ordre SQL par une valeur de type simple (nombre, chaîne, booléen ...). Le type du paramètre est indiqué par l'attribut **parameterClass** de la balise <**select**>, ici le type **string**.

Le <resultMap> d'id="*employe*" des lignes 13-21 de [dbpam.xml] indique comment faire la liaison entre les colonnes [NOM, PRENOM, ADRESSE, CODEPOSTAL, VILLE, SS, INDICE] de la table résultat du *Select* et les propriétés [Nom, Prenom, Adresse, CodePostal, Ville, SS, Indice] d'un objet de type [RawEmploye].

La classe [RawEmployeIndemnites] est dérivée de la classe [RawEmploye] et contient des informations supplémentaires qui n'ont pas été obtenues par la requête SQL précédente : les indemnités liées à l'indice de l'employé. La table des indemnités est obtenue avec l'ordre SQL des lignes 51-53 de [dbpam.xml] :

```
<!-- obtention des indemnités avec indice -->
<select id="getAllIndemnites" resultMap="indemnites">
  select INDICE, BASEHEURE, ENTRETIENJOUR, REPASJOUR, INDEMNITESCP FROM INDEMNITES
</select>
```

Le <resultMap> d'id="*indemnites*" des lignes 28-34 de [dbpam.xml] indique comment faire la liaison entre les colonnes [INDICE, BASEHEURE, ENTRETIENJOUR, REPASJOUR, INDEMNITESCP] de la table résultat du *Select* et les propriétés [Indice, BaseHeure, EntretienJour, RepasJour, IndemnitesCP] d'un objet de type [RawIndemnites].

8.4.5 L'API de SqlMap

Les applications .NET utilisant les classes iBatis / SqlMap doivent importer l'espace de noms suivant :

```
Imports IBatisNet.DataMapper
```

Toutes les opérations SQL se font au travers d'un singleton de type [IBatisNet.DataMapper.SqlMapper]. Ce singleton peut être obtenu de la façon suivante :

```
Private Mappeur As ISqlMapper = Mapper.Instance
```

Cette opération peut lancer une exception. On voit qu'aucun paramètre n'est utilisé dans cette opération. C'est le fichier [**sqlmap.config**] qui est automatiquement utilisé pour construire le singleton [SqlMapper]. Ce fichier doit être dans le dossier d'exécution du projet Visual Studio, par défaut [bin/Debug] en mode débogage et [bin/Release] en mode normal.

A chaque type de balise du fichier de configuration de *SqlMap* décrivant une instruction SQL, correspond une méthode particulière de l'API pour faire exécuter cette instruction SQL et en récupérer les résultats. Nous décrivons certaines de ces intructions SQL et les méthodes .NET qui leur sont associées. Revenons sur le fichier de mapping [dbpam.xml] que nous avons défini :

```
1.  <?xml version="1.0" encoding="utf-8" ?>
2.  <sqlMap
3.    namespace="Pam"
4.    xmlns="http://ibatis.apache.org/mapping"
5.    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
6.    <!-- les resultMaps -->
7.    <resultMaps>
8.      <resultMap id="employe_identite" class="istia.st.pam.dao.entites.RawEmploye, pam-dao-ibatis">
9.        <result property="Nom" column="NOM" />
10.       <result property="Prenom" column="PRENOM" />
11.       <result property="SS" column="SS" />
12.     </resultMap>
13.     <resultMap id="employe" class="istia.st.pam.dao.entites.RawEmploye, pam-dao-ibatis">
14.       <result property="Nom" column="NOM" />
```

```
15.      <result property="Prenom" column="PRENOM" />
16.      <result property="Adresse" column="ADRESSE" />
17.      <result property="CodePostal" column="CODEPOSTAL" />
18.      <result property="Ville" column="VILLE" />
19.      <result property="SS" column="SS" />
20.      <result property="Indice" column="Indice" />
21.    </resultMap>
22. ....
23.  </resultMaps>
24.  <!-- les requêtes SQL -->
25.  <statements>
26.    <!-- obtention des identités (SS, NOM, PRENOM) de tous les employes -->
27.    <select id="getAllIdentitesEmployes" resultMap="employe_identite">
28.      select SS, NOM, PRENOM FROM EMPLOYES
29.    </select>
30.    <!-- obtention d'un employé-->
31.    <select id="getEmploye" resultMap="employe" parameterClass="string">
32.      select NOM, PRENOM, ADRESSE, VILLE, CODEPOSTAL, SS, INDICE FROM EMPLOYES WHERE SS=#value#
33.    </select>
34.
35. ...
36.  </statements>
37. </sqlMap>
```

Une requête de type [SELECT] est définie lignes 27-29. Cette requête SELECT rend des lignes de données qui vont être encapsulées dans des objets. L'attribut [resultMap] indique que le type de ces objets est la classe "*istia.st.pam.dao.entites.RawEmploye, pam-dao-ibatis*" définie ligne 8.

Pour exécuter cette requête *select* nommée "*getAllIdentitesEmployes*", on pourra écrire dans le code VB.NET :

```
Dim Employes as IList=Mappeur.QueryForList("getAllIdentitesEmployes", Nothing);
```

- [Mappeur] est le singleton de type [SqlMapper] obtenu par la ligne :

```
Private Mappeur As ISqlMapper = Mapper.Instance
```

- la méthode [QueryForList] permet d'obtenir le résultat d'une commande SELECT dans une objet de type [IList]. Elle peut lancer une exception.
- le premier paramètre de [QueryForList] est le nom de la commande SQL à exécuter (attribut id, ligne 27)
- le second paramètre est le paramètre à transmettre à la requête SQL. Doit correspondre à l'attribut [parameterClass] de la commande SqlMap exécutée. La balise <select> définie ligne 27 n'a pas cet attribut. Aussi passe-t-on ici le pointeur *Nothing*.
- le résultat est de type **IList**, une interface .NET. Un exemple d'implémentation de l'interface *IList* est la classe [ArrayList]. Les objets de cette liste sont du type indiqué par l'attibut [resultMap], ligne 8, donc de type [RawEmploye]. Nous obtenons dans la variable [Employes] la totalité de la table [EMPLOYES] en une seule instruction. Si la table [EMPLOYES] est vide, on obtient un objet [**IList**] sans éléments.

Considérons une variante de la balise <select>, celle des lignes :

```
1.    <!-- obtention d'un employé-->
2.    <select id="getEmploye" resultMap="employe" parameterClass="string">
3.      select NOM, PRENOM, ADRESSE, VILLE, CODEPOSTAL, SS, INDICE FROM EMPLOYES WHERE SS=#value#
4.  </select>
```

L'attribut [parameterClass] est ici défini. Cela signifie que la requête [SELECT] est paramétrée. Ces paramètres apparaissent sous la forme #param# dans la requête, par exemple #*value*# ligne 3.

Il y a plusieurs façons de passer les paramètres à une requête SqlMap selon la valeur du paramètre [parameterClass].

- dans le cas où l'instruction SQL n'admet qu'un paramètre, on indique [parameterClass="T"] où T est le type (Integer, Double, Float, String, ...) du paramètre. Dans la requête, le paramètre est représenté par le mot clé **#value#**.
- dans le cas où l'instruction SQL admet plusieurs paramètres, il y a deux façons courantes de les passer :
  - dans un **dictionnaire**. Dans ce cas, on indique [parameterClass="Hashtable"] et on trouve dans le texte de l'instruction SQL des mots clés du type #clé1#, #clé2#, ... où [cléi] est une **clé du dictionnaire**. C'est la valeur associée à cette clé du dictionnaire qui est insérée dans l'instruction SQL.
  - dans un **objet**. Dans ce cas, on indique [parameterClass="C"] où C est une classe. On trouve dans le texte de l'instruction SQL des mots clés du type #clé1#, #clé2#, ... où [cléi] est une **propriété publique de l'objet**.

Nous allons présenter ces diverses méthodes de passage de paramètres. La balise <select> des lignes 1-4 ci-dessus définit un paramètre unique de type *string*. Pour l'exécuter, on pourra écrire par exemple :

```
Dim Employe as RawEmploye=Ctype(Mappeur.QueryForObject("getEmploye", "1451053072022"), RawEmploye)
```

- [Mappeur] est le singleton de type [SqlMapper] qui contrôle l'accès aux données.

- la méthode [QueryForObject] permet d'obtenir le résultat unique de la commande SELECT dans une objet de type [RawEmploye]. Si la requête SELECT ne rend aucune ligne, la méthode [QueryForObject] rend le pointeur *Nothing*. La méthode [QueryForObject] peut lancer une exception si l'accès au SGBD rencontre un problème.

Considérons maintenant une instruction SQL [INSERT]. Elle pourrait être définie comme suit :

```
1.    <!-- insertion d'un nouvel employé -->
2.    <insert id="insertEmploye" parameterClass="istia.st.pam.dao.entites.RawEmploye">
3.        insert into EMPLOYES (SS, NOM, PRENOM, ADRESSE, VILLE, CODEPOSTAL, INDICE) values
4.        ( #SS#, #Nom#, #Prenom#, #Adresse#, #Ville#, #CodePostal#, #Indice#)
5.      </insert>
```

- ligne 2 : définit une instruction SQL nommée "insertEmploye" qui admet pour paramètre un objet de type [istia.st.pam.dao.entites.RawEmploye]
- ligne 4 : utilise les champs de l'objet [RawEmploye] passé en paramètre. L'exploitation de cet ordre *SqlMap* dans le code VB.NET pourrait se faire de la façon suivante :

```
Dim Employe as new RawEmploye("1451053072022", "TOUCHON", "Paul", "2 rue des acacias", "Segré", "49400",2)
Dim nbLignesInsérées as Integer=Mappeur.Insert("insertEmploye", Employe)
```

- [Mappeur] est le singleton de type [SqlMapper] qui contrôle l'accès aux données.
- le premier paramètre de [Insert] est le nom de la commande SQL à exécuter (attribut id, ligne 2)
- le second paramètre est le paramètre à transmettre à la requête SQL de type [parameterClass] (ligne 2).
- le résultat est le nombre de lignes insérées, ici 0 ou 1. La méthode [Insert] peut également lancer une exception. Ce sera par exemple le cas si l'insertion enfreint l'une des contraintes de la BD, par exemple une duplication de clé primaire.

La balise <insert> aurait pu être écrite différemment :

```
1.    <!-- insertion d'un nouvel employé -->
2.    <insert id="insertEmploye" parameterClass="Hashtable">
3.        insert into EMPLOYES (SS, NOM, PRENOM, ADRESSE, VILLE, CODEPOSTAL, INDICE) values
4.        ( #SS#, #Nom#, #Prenom#, #Adresse#, #Ville#, #CodePostal#, #Indice#)
5.      </insert>
```

- ligne 2 : déclare que les paramètres sont dans un dictionnaire.

L'exploitation de cet ordre *SqlMap* dans le code VB.NET pourrait se faire alors de la façon suivante :

```
Dim EmployeInfos as new Hashtable()
    EmployeInfos("SS")= "1451053072022"
    EmployeInfos("Nom")= "TOUCHON"
    EmployeInfos("Prenom")= "Paul"
    EmployeInfos("Adresse")="2 rue des acacias"
    EmployeInfos("Ville")="Segré"
    EmployeInfos("CodePostal")="49400"
    EmployeInfos("Indice")=2
    Dim nbLignesInsérées as Integer=Mappeur.Insert("insertEmploye", EmployeInfos)
```

Introduisons maintenant une instruction SQL [DELETE]. Elle pourrait être définie comme suit :

```
1.    <!-- suppression d'un employé -->
2.    <delete id="deleteEmploye" parameterClass="string">
3.        delete FROM EMPLOYES where SS= #value#
4.      </delete>
```

- ligne 2 : définit une instruction SQL nommée "deleteEmploye" qui admet un unique paramètre de type [string]
- ligne 3 : définit l'instruction SQL DELETE associée.

L'exploitation de cet ordre *SqlMap* dans le code VB.NET pourrait se faire de la façon suivante :

```
        Dim NbLignesDétruites as Integer=Mappeur.Delete("deleteEmploye", "1451053072022")
```

- [Mappeur] est le singleton de type [SqlMapper] qui contrôle l'accès aux données.
- le premier paramètre de [Delete] est le nom de la commande SQL à exécuter (attribut id, ligne 2)
- le second paramètre est le paramètre à transmettre à la requête SQL de type [string] (ligne 2).
- le résultat est le nombre de lignes détruites, ici 0 ou 1. La méthode [Delete] peut lancer une exception si l'accès au SGBD se passe mal.

Terminons par l'instruction SQL [UPDATE]. Elle pourrait être définie comme suit :

```
1.    <!-- modification d'un employé -->
2.    <update id="modifyEmploye" parameterClass="istia.st.pam.dao.entites.RawEmploye">
```

```
3.       update EMPLOYES set ADRESSE= #Adresse# where SS= #SS#
4.    </update>
```

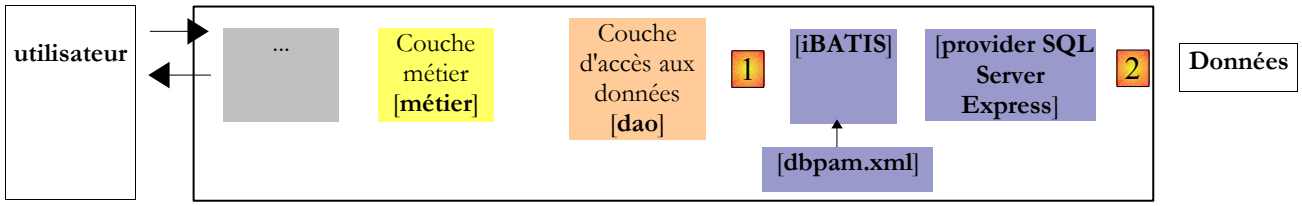
- ligne 2 : définit une instruction SQL nommée "modifyEmploye" qui admet pour paramètre un objet de type [istia.st.pam.dao.entites.RawEmploye]
- ligne 3 : définit une instruction SQL UPDATE paramétrée par les champs de l'objet [RawEmploye] passé en paramètre.

L'exploitation de cet ordre *SqlMap* dans le code VB.NET pourrait se faire de la façon suivante :

```
1.  Dim Employe as RawEmploye=Mappeur.QueryForObject("getEmploye", "1451053072022")
2.  Employe.Adresse="2 rue des oiseaux"
3.  Dim NbLignesModifiées as Integer=Mappeur.Update("modifyEmploye", Employe)
```

- ligne 1 : [Mappeur] est le singleton de type [SqlMapper] qui contrôle l'accès aux données. On demande un employé identifié par son n° de sécurité sociale.
- ligne 2 : on modifie la propriété [Adresse] de cet employé.
- ligne 3 : on réinjecte la ligne modifiée dans la base.
- le premier paramètre de [Update] est le nom de la commande SQL à exécuter (attribut id, ligne 2)
- le second paramètre est le paramètre à transmettre à la requête SQL, du type indiqué par l'attribut [parameterClass] de la balise <update> (ligne 2).
- le résultat est le nombre de lignes modifiées, ici 0 ou 1. La méthode [Update] peut lancer une exception pour les mêmes raisons que la méthode [Insert].

8.5 Implémentation et tests de la couche [dao]



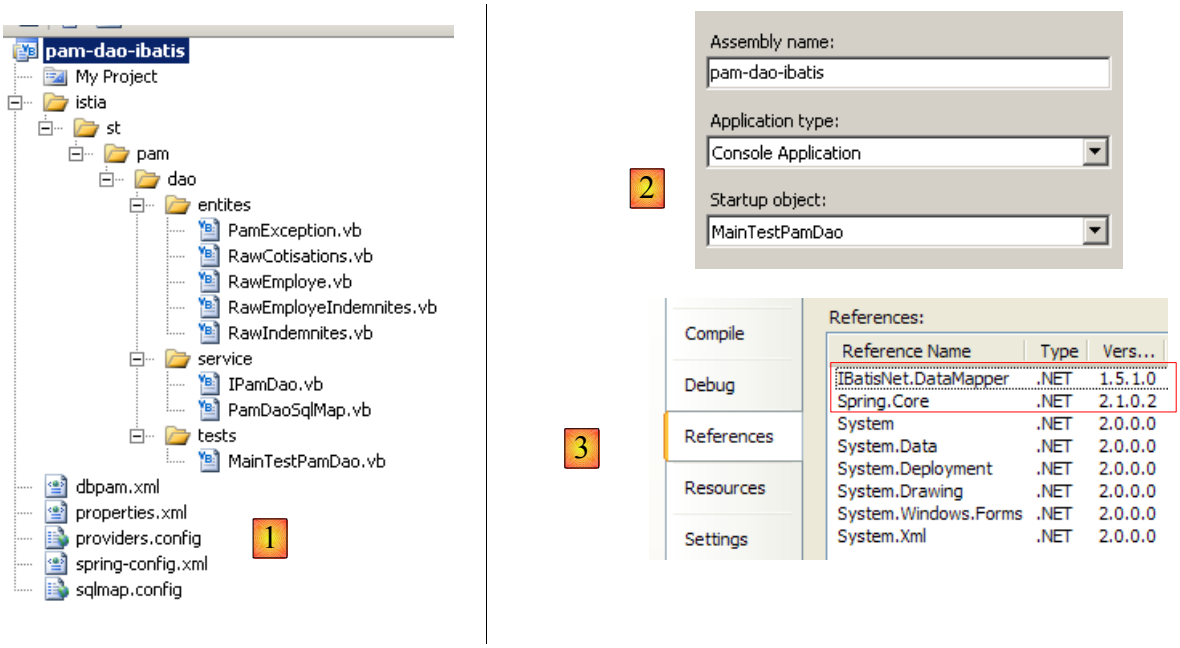
L'interface IPamDao implémentée par la couche [dao] est la suivante :

```
Imports istia.st.pam.dao.entites

Namespace istia.st.pam.dao.service
  Public Interface IPamDao
    ' liste de toutes les identités des employés
    Function GetAllIdentitesEmployes() As RawEmploye()
    ' un employé particulier avec ses indemnités
    Function GetEmployeIndemnites(ByVal SS As String) As RawEmployeIndemnites
    ' liste de toutes les cotisations
    Function GetCotisations() As RawCotisations
  End Interface
End Namespace
```

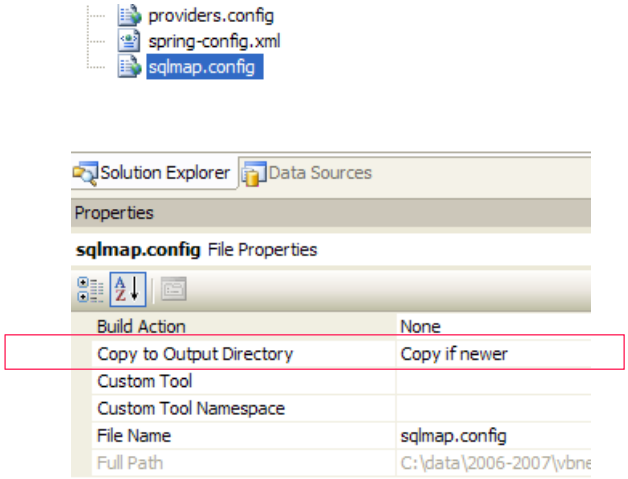
8.5.1 Le projet Visual Studio

Le projet Visual Studio a déjà été présenté. Rappelons-le :



- dans [1] on voit que :

- le dossier [entites] contient les objets manipulés par la couche [dao]. Ils ont été présentés au paragraphe 8.3.1, page 43.
- le dossier [service] implémente l'interface [IPamDao] définie au paragraphe 8.2, page 40.
- le dossier [tests] contient un programme de tests de la couche [dao]
- les fichiers [sqlmap.config, providers.config, properties.xml, dbpam.xml] sont les fichiers de configuration de la couche [iBatis / SqlMap]. [spring-config.xml] est un fichier de configuration Spring utilisé par le programme de test [MainTestPamDao]. Ces fichiers sont à la racine du projet VS. On a configuré leur propriété [Copy to Output Directory] à vrai :



  En effet, tous ces fichiers doivent être présents dans le dossier de l'exécutable (.exe ou .dll) au moment de l'exécution car c'est là qu'ils sont cherchés, que ce soit par iBatis ou Spring.
- dans [2] on voit que le projet génère un assembly nommé "pam-dao-ibatis"
- dans [3] on voit que le projet le projet détient des références sur les DLL [IbatisNet.DataMapper] et [Spring.Core] qui sont les DLL respectives des outils tierces *iBatis* et *Spring*.

8.5.2 Le programme de test

Le programme de test [MainTestPamDao] est chargé de tester les méthodes de l'interface [IPamDao]. Un exemple basique pourrait être le suivant :

```
1.  Imports istia.st.pam.dao.service
2.  Imports istia.st.pam.dao.entites
3.  Imports Spring.Objects.Factory.Xml
```

```
4.  Imports Spring.Core.IO
5.
6.  Module MainTestPamDao
7.    Public Sub main()
8.      Try
9.        ' on récupère une image mémoire du fichier de configuration Spring
10.       Dim Factory As XmlObjectFactory = New XmlObjectFactory(New FileSystemResource("spring-
    config.xml"))
11.       ' on demande une référence sur la couche [dao]
12.       Dim PamDao As IPamDao = CType(Factory.GetObject("pamdao"), IPamDao)
13.       ' liste des identités des employés
14.       For Each Employe As RawEmploye In pamDao.GetAllIdentitesEmployes
15.         Console.WriteLine(Employe.ToString)
16.       Next
17.       ' un employé avec ses indemnités
18.       Console.WriteLine("-----------------------------------")
19.       Console.WriteLine(pamDao.GetEmployeIndemnites("254104940426058"))
20.       Console.WriteLine("-----------------------------------")
21.       ' liste des cotisations
22.       Dim Cotisations As RawCotisations = PamDao.GetCotisations
23.       Console.WriteLine(Cotisations.ToString)
24.     Catch Ex As Exception
25.       ' affichage exception
26.       Console.WriteLine(Ex.ToString)
27.     End Try
28.   End Sub
29. End Module
```

- ligne 10 : on exploite le fichier de configuration [spring-config.xml]. Celui-ci est décrit ci-dessous.
- ligne 12 : on demande à Spring une référence sur la couche [dao]
- lignes 14-16 : test de la méthode [GetAllIdentitesEmployes] de l'interface [IPamDao]
- lignes 18-20 : test de la méthode [GetEmployeIndemnites] de l'interface [IPamDao]
- lignes 22-23 : test de la méthode [GetCotisations] de l'interface [IPamDao]

Le programme de test utilise le fichier de configuration Spring [spring-config.xml] suivant :

```
1.  <?xml version="1.0" encoding="iso-8859-1" ?>
2.  <objects xmlns="http://www.springframework.net" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
    instance"
3.    xsi:schemaLocation="http://www.springframework.net
4.  http://www.springframework.net/xsd/spring-objects.xsd">
5.    <object id="pamdao" type="istia.st.pam.dao.service.PamDaoSqlMap, pam-dao-ibatis"/>
6.  </objects>
```

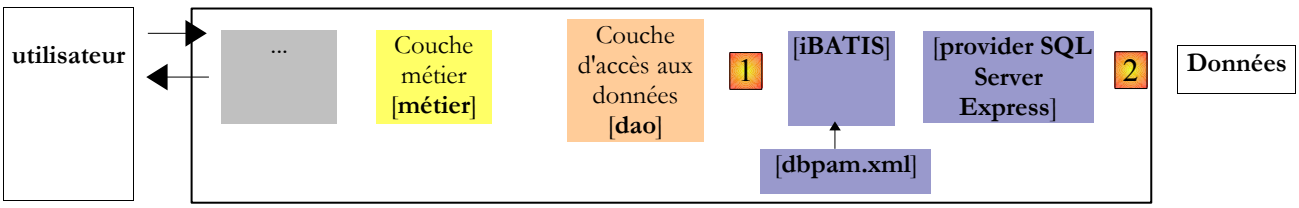
- ligne 5 : l'objet d'id "pamdao" a le type [istia.st.pam.dao.service.PamDaoSqlMap] et est trouvé dans l'assembly [pam-dao-ibatis].

L'exécution faite avec la base de données décrite au paragraphe 2.1, page 3, donne le résultat console suivant :

```
1.  [254104940426058,Jouveinal,Marie,,,,0]
2.  [260124402111742,Laverti,Justine,,,,0]
3.  -----------------------------------
4.  [[254104940426058,Jouveinal,Marie,5 rue des Oiseaux,St Corentin,49203,0],[2, 2,1, 2,1, 3,1, 15]]
5.  -------------------------------------
6.  [3,49,6,15,9,39,7,88]
```

- lignes 1-2 : les 2 employés de type [RawEmploye] avec les seules informations [SS, Nom, Prenom]
- ligne 4 : l'employé de type [RawEmployeIndemnites] ayant le n° de sécurité sociale [254104940426058]
- ligne 6 : les taux de cotisations

8.5.3 Écriture de la classe [PamDaoSqlMap]



L'interface IPamDao implémentée par la couche [dao] est la suivante :

```
Imports istia.st.pam.dao.entites
```

```
Namespace istia.st.pam.dao.service
  Public Interface IPamDao
    ' liste de toutes les identités des employés
    Function GetAllIdentitesEmployes() As RawEmploye()
    ' un employé particulier avec ses indemnités
    Function GetEmployeIndemnites(ByVal SS As String) As RawEmployeIndemnites
    ' liste de toutes les cotisations
    Function GetCotisations() As RawCotisations
  End Interface
End Namespace
```

Question : écrire le code de la classe [PamDaoSqlMap] implémentant l'interface [IPamDao] ci-dessus à l'aide du produit [Ibatis SqlMap] configuré tel qu'il a été présenté précédemment (paragraphe 8.4.3, page 48).

Spécifications :

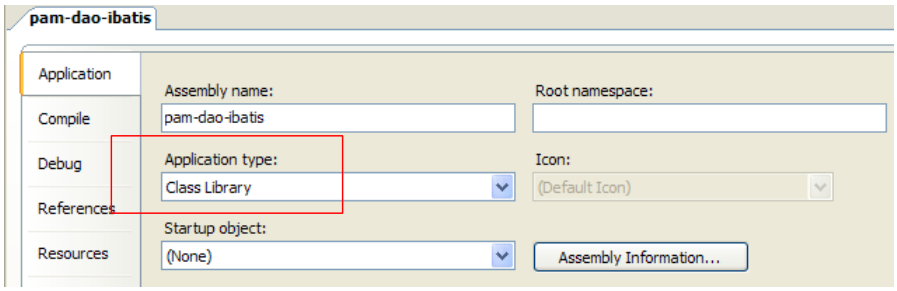
On supposera que les données demandées à la couche [dao] peuvent entièrement tenir en mémoire. Ainsi, pour améliorer les performances, la classe [PamDaoSqlMap] mémorisera :

- la table [EMPLOYES] sous la forme (SS, NOM, PRENOM) nécessitée par la méthode [GetAllIdentitesEmployes] sous la forme d'un tableau d'objets de type [RawEmploye]
- la table [COTISATIONS] sous la forme d'un unique objet de type [RawCotisations]
- la table [INDEMNITES] sous la forme d'un dictionnaire indexé par le champ [INDICE] et dont les valeurs seront des objets de type [RawIndemnites]

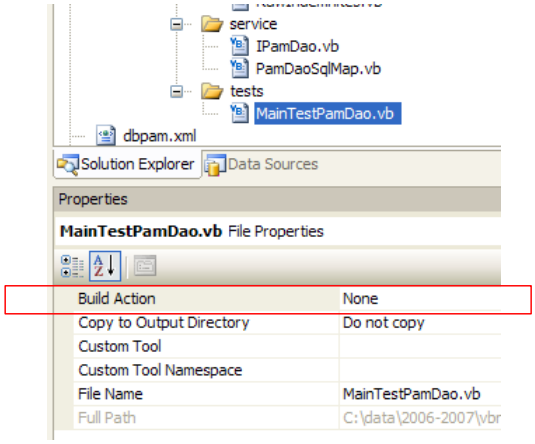
8.5.4 Génération de la DLL de la couche [dao]

Une fois écrite et testée la classe [PamDaoSqlMap], on génèrera la DLL de la couche [dao] de la façon suivante :

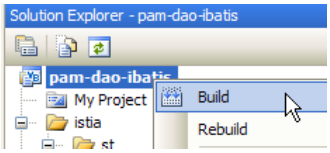
- changer la nature du projet en " Class Library ":



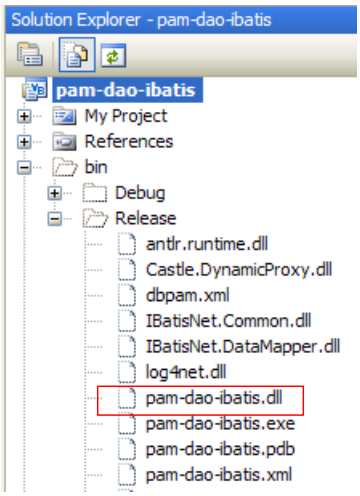
- exclure le programme de test du projet généré :



- générer le projet



- la DLL est générée dans le dossier [bin/Release]

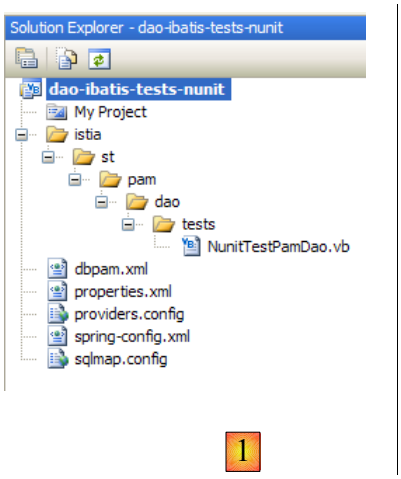


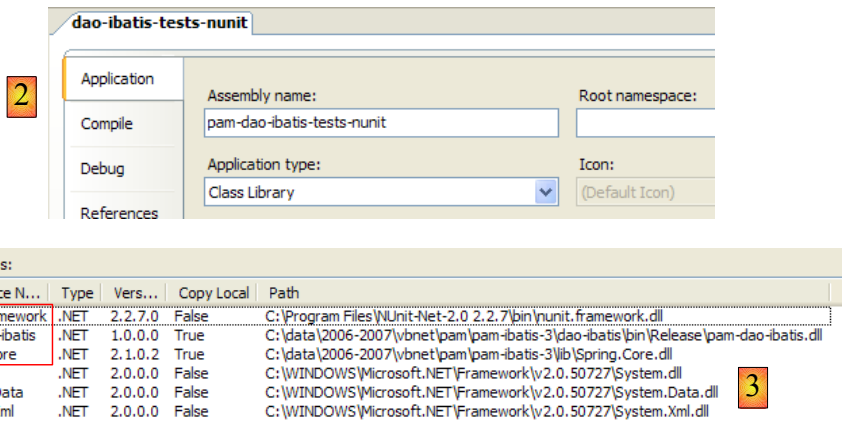
8.6 Tests NUnit de la classe [PamDaoSqlMap]

Les tests précédents étaient visuels : on vérifiait à l'écran qu'on obtenait bien les résultats attendus. C'est une méthode insuffisante en milieu professionnel. Les tests doivent toujours être automatisés au maximum et viser à ne nécessiter aucune intervention humaine. L'être humain est en effet sujet à la fatigue et sa capacité à vérifier des tests s'émousse au fil de la journée. L'outil [NUnit] aide à réaliser cette automatisation. Il est disponible à l'Url [http://www.nunit.org/].

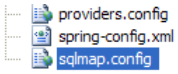
8.6.1 Le projet Visual studio des tests Nunit

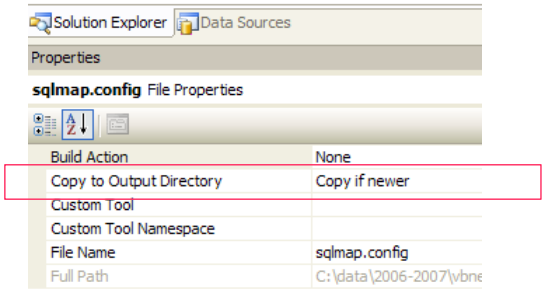
Le projet Visual Studio du projet de tests NUnit de la couche [dao] pourrait être le suivant :





- dans [1], on trouve :
  - le programme de test [NUnitTestPamDao.vb]
  - les fichiers de configuration *iBatis* et *Spring*. Ils sont identiques à ceux de même nom étudiés précédemment. Ils ont eux aussi la propriété [Copy to Output Directory] à vrai.





- dans [2], on voit que le projet va générer une DLL nommé [pam-dao-ibatis-tests-nunit.dll]

- dans [3], on voit :
  - dans [References], les références aux DLL :
    - [pam-dao-ibatis.dll] qui contient les classes de la couche [dao] construite précédemment
    - [nunit.framework.dll] qui contient les classes de l'outil [NUnit]
    - [Spring.Core.dll] la DLL de Spring

8.6.2 La classe de test [NunitTestPamDao.vb]

La classe de test NUnit sera la suivante :

```
1.  Imports System.Collections
2.  Imports NUnit.Framework
3.  Imports istia.st.pam.dao.service
4.  Imports istia.st.pam.dao.entites
5.  Imports Spring.Objects.Factory.Xml
6.  Imports Spring.Core.IO
7.
8.  Namespace istia.st.pam.dao.tests
9.
10.   <TestFixture()> _
11.   Public Class NunitTestPamDao
12.     ' la couche [dao] à tester
13.     Private PamDao As IPamDao = Nothing
14.
15.     ' constructeur
16.     Public Sub New()
17.       ' on crée l'image mémoire du fichier de configuration Spring
18.       Dim Factory As XmlObjectFactory = New XmlObjectFactory(New FileSystemResource("spring-
    config.xml"))
19.       ' on demande une référence sur la couche [dao]
20.       PamDao = CType(Factory.GetObject("pamdao"), IPamDao)
21.     End Sub
22.
23.     ' init
24.     <SetUp()> _
25.     Public Sub Init()
26.
27.     End Sub
28.
29.     <Test()> _
30.     Public Sub GetAllIdentitesEmployes()
31.       ' vérification nbre d'employes
32.       Assert.AreEqual(2, PamDao.GetAllIdentitesEmployes.Length)
33.     End Sub
34.
35.     <Test()> _
36.     Public Sub GetCotisations()
37.       ' vérification taux de cotisations
38.       Dim Cotisations As RawCotisations = PamDao.GetCotisations
39.       Assert.AreEqual(3.49, Cotisations.CsgRds, 0.000001)
40.       Assert.AreEqual(6.15, Cotisations.Csgd, 0.000001)
41.       Assert.AreEqual(9.39, Cotisations.Secu, 0.000001)
42.       Assert.AreEqual(7.88, Cotisations.Retraite, 0.000001)
43.     End Sub
44.
45.     <Test()> _
46.     Public Sub GetEmployeIdemnites()
```

```
47.         ' vérification individus
48.         Dim Employe1 As RawEmployeIndemnites = PamDao.GetEmployeIndemnites("254104940426058")
49.         Dim Employe2 As RawEmployeIndemnites = PamDao.GetEmployeIndemnites("260124402111742")
50.         Assert.AreEqual("Jouveinal", Employe1.Nom)
51.         Assert.AreEqual(2.1, Employe1.Indemnites.BaseHeure, 0.000001)
52.         Assert.AreEqual("Laverti", Employe2.Nom)
53.         Assert.AreEqual(1.93, Employe2.Indemnites.BaseHeure, 0.000001)
54.       End Sub
55.
56.       <Test()> _
57.   Public Sub GetEmployeIdemnites2()
58.         ' vérification individu inexistant
59.         Dim Erreur As Boolean = False
60.         Try
61.           Dim Employe1 As RawEmployeIndemnites = PamDao.GetEmployeIndemnites("xx")
62.         Catch ex As PamException
63.           Erreur = True
64.         End Try
65.         Assert.IsTrue(Erreur)
66.       End Sub
67.     End Class
68.   End Namespace
```

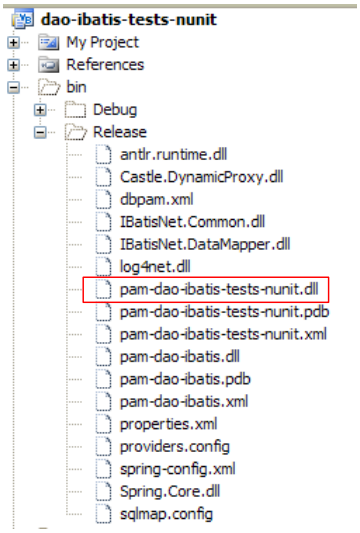
- ligne 10 : la classe a l'attribut <TestFixture()> qui en fait une classe de test [NUnit].
- ligne 13 : le champ privé [PamDao] est une instance de l'interface d'accès à la couche [dao]. On notera que le type de ce champ est une **interface** et non une **classe**. Cela signifie que l'instance [PamDao] ne rend accessibles que des méthodes, celles de l'interface [IPamDao].
- les méthodes testées dans la classe sont celles ayant l'attribut <Test()>. Pour toutes ces méthodes, le processus de test est le suivant :
  - la méthode ayant l'attribut <SetUp()> est tout d'abord exécutée. Elle sert à préparer les ressources (connexions réseau, connexions aux bases de données, ...) nécessaires au test.
  - puis la méthode à tester est exécutée
  - et enfin la méthode ayant l'attribut <TearDown()> est exécutée. Elle sert généralement à libérer les ressources mobilisées par la méthode d'attribut <SetUp()>.
- dans notre test, il n'y a pas de ressources à allouer avant chaque test et à désallouer ensuite. Aussi n'avons-nous pas besoin de méthode avec les attributs <SetUp()> et <TearDown()>. Pour l'exemple, nous avons présenté, lignes 25-27, une méthode avec l'attribut <SetUp()>.
- lignes 16-21 : le constructeur de la classe initialise le champ privé [PamDao] à partir du contenu d'un fichier [Spring] appelé ici "spring-config.xml". Dans ce fichier de configuration, est précisé le type exact de la classe du champ privé [PamDao]. C'est là une propriété fondamentale de Spring, appelée "Injection de dépendance". Cette capacité de Spring, nous évite de donner au champ [PamDao] un type de classe particulier. On se contente de dire, comme nous l'avons fait, qu'il implémente l'interface [IPamDao].
- lignes 30-33 : testent la méthode [GetAllIdentitesEmployes]
- lignes 36-43 : testent la méthode [GetCotisations]
- lignes 46-54 : testent la méthode [GetEmployesIndemnites]
- lignes 57-66 : testent la méthode [GetEmployesIndemnites] lors d'une exception. La ligne 62 dit que cette exception doit être de type [PamException] une exception définie dans les entités de la couche [dao].

Le fichier de configuration Spring "spring-config.xml" est le suivant :

```
1.  <?xml version="1.0" encoding="iso-8859-1" ?>
2.  <objects xmlns="http://www.springframework.net" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
    instance"
3.    xsi:schemaLocation="http://www.springframework.net
4.  http://www.springframework.net/xsd/spring-objects.xsd">
5.    <object id="pamdao" type="istia.st.pam.dao.service.PamDaoSqlMap, pam-dao-ibatis"/>
6.  </objects>
```

8.6.3 Mise en oeuvre des tests

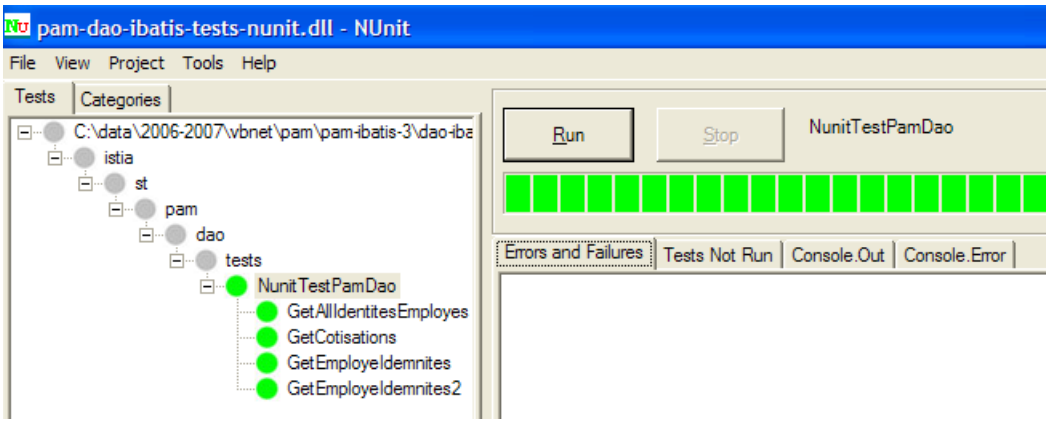
Le projet ci-dessus génère la DLL [pam-ibatis-dao-tests-nunit.dll] dans le dossier [bin/Release].



Le dossier [bin/Release] contient en outre :

- les DLL qui font partie des références du projet et qui ont l'attribut [Copie locale] à vrai : [pam-dao-ibatis.dll, Spring.Core.dll] (cf paragraphe 8.6.1, page 61). Ces DLL sont accompagnées des copies des DLL qu'elles utilisent elles-mêmes :
  - [Castle.DynamicProxy.dll, IbatisNet.Common.dll, IbatisNet.DataMapper.dll] pour l'outil *iBatis*
  - [antlr.runtime.dll, log4net.dll] pour l'outil *Spring*
- les fichiers de configuration du projet qui ont leur attribut [Copy to Output Directory] à vrai : [dbpam.xml, properties.xml, providers.config, spring-config.xml, sqlmap.config]

On charge la DLL [pam-ibatis-dao-tests-nunit.dll] avec l'outil [NUnit-Gui] et on exécute les tests :



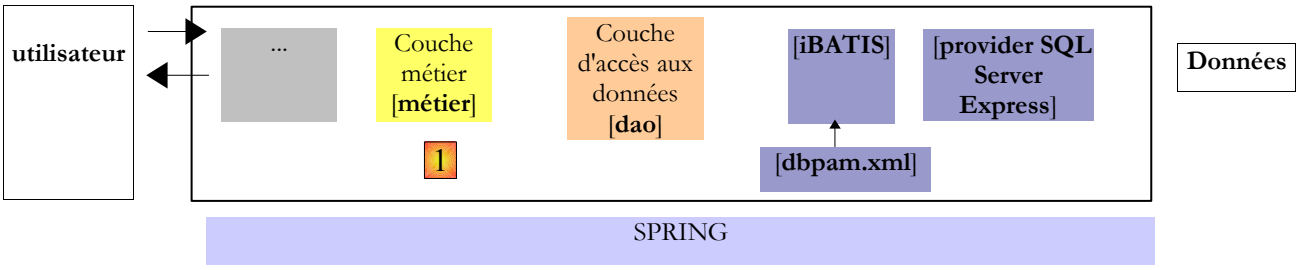
Ci-dessus, les tests ont été réussis.

Travail pratique :

- mettre en oeuvre sur machine les tests de la classe [PamDaoSqlMap].
- utiliser différents fichiers de configuration SqlMap afin d'utiliser des SGBD différents (Firebird, MySQL, Postgres, SQL Server)

8.7 La couche métier

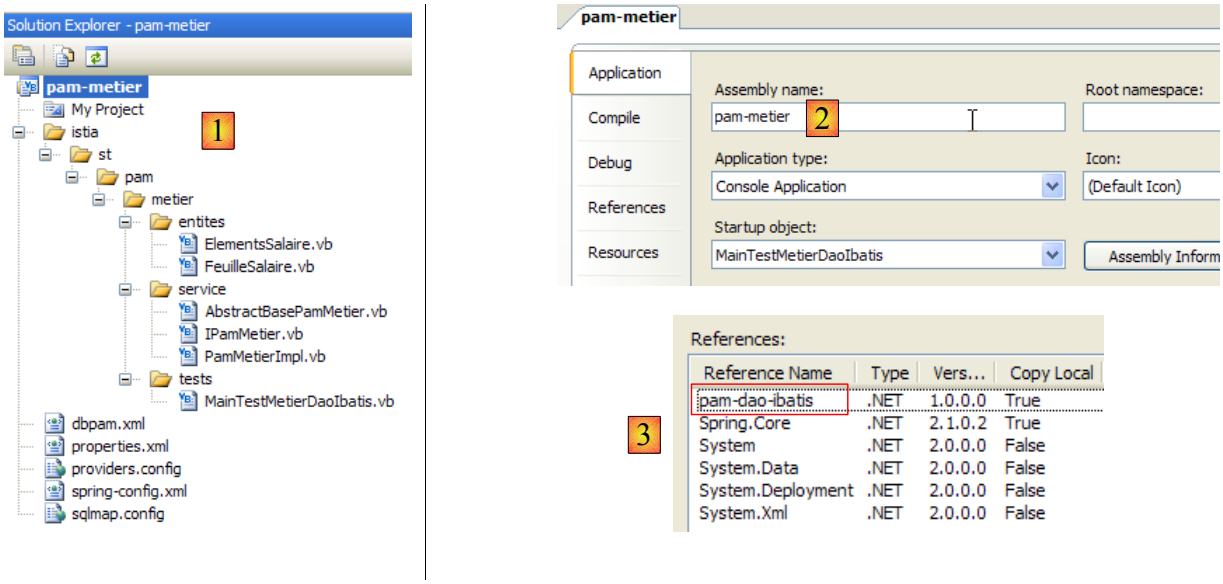
Revenons sur l'architecture générale de l'application [SimuPaie] :



Nous considérons désormais que la couche [dao] est acquise et qu'elle a été encapsulée dans la DLL [pam-dao-ibatis.dll]. Nous nous intéressons maintenant à la couche [metier]. C'est elle qui implémente les règles métier, ici les règles de calcul d'un salaire.

8.7.1 Le projet Visual Studio de la couche [metier]

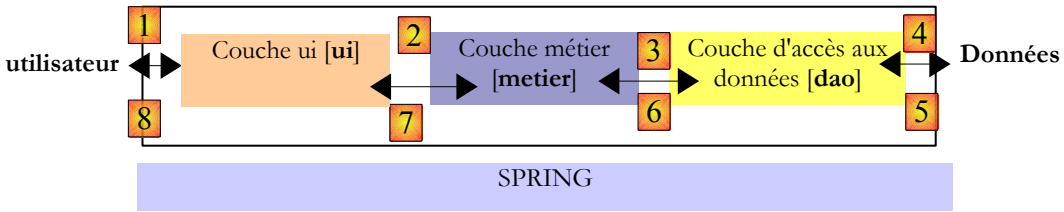
Le projet Visual Studio de la couche métier pourrait ressemble à ce qui suit :



- en [1] l'ensemble du projet. La couche [metier] est formée des deux dossiers [entites, service]. Le reste du projet est formé d'un programme de test [MainTestMetierDaoIbatis] et des fichiers de configuration de ce test [dbpam.xml, properties.xml, providers.config, spring-config.xml, sqlmap.config]. Ces éléments n'ont pas vocation à faire partie de la DLL qui ultérieurement encapsulera la couche [metier].
- en [2] le type du projet (Console Application) et le nom de l'assembly généré (pam-metier). Le projet est de type Console uniquement dans la phase de test. Il deviendra un projet de type [Class Library] pour générer la DLL de la couche [metier].
- en [3] les références utilisées par le projet. On notera la DLL [pam-dao-ibatis] de la couche [dao] étudiée précédemment.

8.7.2 L'interface [IPamMetier] de la couche [metier]

Revenons à l'architecture générale de l'application :



Quelle interface doit offrir la couche [metier] à la couche [ui] ? Quelles sont les interactions possibles entre ces deux couches ? Rappelons-nous l'interface graphique qui sera présentée à l'utilisateur :

1. à l'affichage initial du formulaire, on doit trouver en [1] la liste des employés. Une liste simplifiée suffit (Nom, Prénom, SS). Le n° SS est nécessaire pour avoir accès aux informations complémentaires sur l'employé sélectionné (informations 6 à 11).
2. les informations 12 à 15 sont les différents taux de cotisations.
3. les informations 16 à 19 sont les indemnités liées à l'indice de l'employé
4. les informations 20 à 24 sont les éléments du salaire calculés à partir des saisies 1 à 3 faites par l'utilisateur.

L'interface [IPamMetier] offerte à la couche [ui] par la couche [metier] doit répondre aux exigences ci-dessus. Il existe de nombreuses interfaces possibles. Nous proposons la suivante :

```
1.  Imports istia.st.pam.dao.entites
2.  Imports istia.st.pam.dao.service
3.  Imports istia.st.pam.metier.entites
4.
5.  Namespace istia.st.pam.metier.service
6.    Public Interface IPamMetier
7.      ' liste de toutes les identités des employés
8.      Function GetAllIdentitesEmployes() As RawEmploye()
9.
10.     ' ------- le calcul du salaire
11.     Function GetSalaire(ByVal SS As String, ByVal HeuresTravaillées As Double, ByVal
    JoursTravaillés As Integer) As FeuilleSalaire
12.   End Interface
13. End Namespace
```

- ligne 8 : la méthode qui permettra le remplissage du combo [1]
- ligne 11 : la méthode qui permettra d'obtenir les renseignements 6 à 24. Ceux-ci ont été rassemblés dans un objet de type [FeuilleSalaire].

8.7.3 Les entités de la couche [metier]

Le dossier [entites] du projet Visual Studio (cf paragraphe 8.7.1, page 65) contient les objets manipulés par la classe métier : [FeuilleSalaire] et [ElementsSalaire]. Ces entités ont déjà été rencontrées et décrites : [FeuilleSalaire] page 28 et [ElementsSalaire] page 29.

8.7.4 Implémentation de la couche [metier]

L'interface de la couche [metier] a été décrite précédemment :

```
1.  Imports istia.st.pam.dao.entites
2.  Imports istia.st.pam.dao.service
3.  Imports istia.st.pam.metier.entites
4.
5.
6.  Namespace istia.st.pam.metier.service
7.    Public Interface IPamMetier
8.      ' liste de toutes les identités des employés
9.      Function GetAllIdentitesEmployes() As RawEmploye()
10.
11.     ' ------- le calcul du salaire
12.     Function GetSalaire(ByVal SS As String, ByVal HeuresTravaillées As Double, ByVal
    JoursTravaillés As Integer) As FeuilleSalaire
13.   End Interface
14. End Namespace
```

Nous allons l'implémenter avec deux classes :

- [AbstractBasePamMetier] qui est une classe abstraite dans laquelle on implémentera l'accès aux données de l'interface [IPamMetier]. Cette classe aura une référence sur la couche [dao].
- [PamMetierImpl] une classe dérivée de [AbstractBasePamMetier] qui elle, implémentera les règles métier de l'interface [IPamMetier]. Elle sera ignorante de la couche [dao].

La classe [AbstractBasePamMetier] sera la suivante :

```
1.  Imports istia.st.pam.dao.entites
2.  Imports istia.st.pam.dao.service
3.  Imports istia.st.pam.metier.entites
4.
5.
6.  Namespace istia.st.pam.metier.service
7.    Public MustInherit Class AbstractBasePamMetier
8.      Implements IPamMetier
9.
10.     ' l'objet d'accès aux données
11.     Private _PamDao As IPamDao
12.
13.     ' propriété publique associée
14.     Public Property PamDao() As IPamDao
15.       Get
16.         Return _PamDao
17.       End Get
18.       Set(ByVal Value As IPamDao)
19.         _PamDao = Value
20.       End Set
21.     End Property
22.
23.     ' liste de toutes les identités des employés
24.     Function GetAllIdentitesEmployes() As RawEmploye() Implements IPamMetier.GetAllIdentitesEmployes
25.       Return PamDao.GetAllIdentitesEmployes
26.     End Function
27.
28.     ' un employé particulier avec ses indemnités
29.     Protected Function GetEmployeIndemnites(ByVal SS As String) As RawEmployeIndemnites
30.       Return PamDao.GetEmployeIndemnites(SS)
31.     End Function
32.
33.     ' les cotisations
34.     Protected Function GetCotisations() As RawCotisations
35.       Return PamDao.GetCotisations
36.     End Function
37.
38.     ' le calcul du salaire
39.     Public MustOverride Function GetSalaire(ByVal SS As String, ByVal HeuresTravaillées As Double, ByVal
    JoursTravaillés As Integer) As FeuilleSalaire Implements IPamMetier.GetSalaire
40.   End Class
41. End Namespace
```

- ligne 7 : la classe est abstraite (attribut *MustInherit*)
- ligne 8 : la classe implémente l'interface [IPamMetier]
- ligne 11 : la classe détient une référence sur la couche [dao]
- lignes 14-21 : propriété publique liée au champ privé précédent
- lignes 24-26 : implémentation de la méthode [GetAllIdentitesEmployes] de l'interface [IPamMetier] – utilise la méthode de même nom de la couche [dao]
- lignes 29-31 : méthode interne (protected) [GetEmployeIdemnites] qui fait appel à la méthode de même nom de la couche [dao] – déclarée *protected* pour que les classes dérivées puissent y avoir accès sans qu'elle soit publique.
- lignes 34-36 : méthode interne (protected) [GetCotisations] qui fait appel à la méthode de même nom de la couche [dao]
- ligne 39 : implémentation abstraite (attribut *MustOverride*) de la méthode [GetSalaire] de l'interface [IPamMetier].

Le calcul du salaire est implémenté par la classe [PamMetierImpl] suivante :

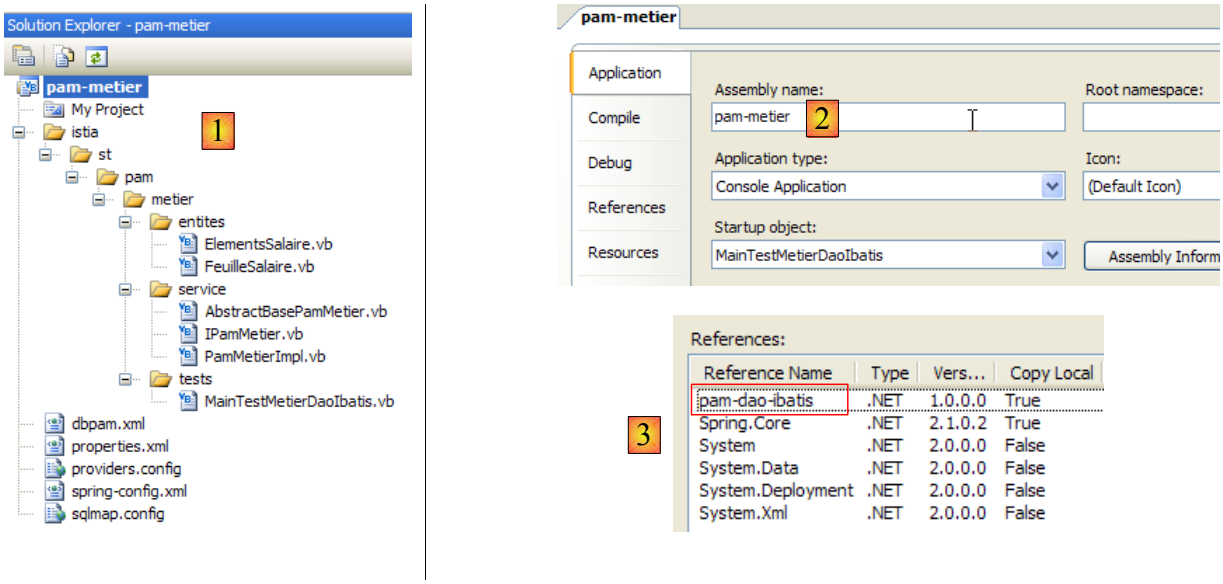
```
1.  Imports istia.st.pam.dao.entites
2.  Imports istia.st.pam.metier.entites
3.
4.  Namespace istia.st.pam.metier.service
5.
6.    Public Class PamMetierImpl
7.      Inherits AbstractBasePamMetier
8.
9.      ' calcul du salaire
10.     Public Overrides Function getSalaire(ByVal ss As String, ByVal heuresTravaillées As Double,
    ByVal joursTravaillés As Integer) As FeuilleSalaire
11.       ' SS : n° SS de l'employé
12.       ' HeuresTravaillées : le nombre d'heures travaillés
13.       ' Jours Travaillés : nbre de jours travaillés
14.       ' on récupère l'employé avec ses indemnités
15. ...
16.       ' on récupère les divers taux de cotisation
17. ...
18.       ' on calcule les éléments du salaire
19.       Dim ElementsSalaire As New ElementsSalaire
20. ...
21.       ' on rend la feuille de salaire
22. ...
23.     End Function
24.   End Class
25. End Namespace
```

- ligne 7 : la classe dérive de [AbstractBasePamMetier] et donc implémente de ce fait l'interface [IPamMetier]
- ligne 10 : la méthode [GetSalaire] à implémenter

Question : écrire le code de la méthode [GetSalaire].

8.7.5 Un premier programme de test de la couche [metier]

Rappelons le projet Visual Studio de la couche [metier] :



Le programme de test [MainTestMetierDaoIbatis] ci-dessus teste les méthodes de l'interface [IPamMetier]. Un exemple basique pourrait être le suivant :

```
1.  Imports istia.st.pam.metier.entites
2.  Imports istia.st.pam.metier.service
3.  Imports Spring.Objects.Factory.Xml
4.  Imports Spring.Core.IO
5.  Imports istia.st.pam.dao.entites
6.
7.  Module MainTestMetierDaoIbatis
8.    Public Sub main()
9.      Try
10.       ' on crée une image mémoire du fichier de configuration spring
11.       Dim Factory As XmlObjectFactory = New XmlObjectFactory(New FileSystemResource("spring-
    config.xml"))
```

```
12.       ' on demande l'instanciation de la couche [metier]
13.       Dim PamMetier As IPamMetier = CType(Factory.GetObject("pammetier"), IPamMetier)
14.       ' calculs de feuilles de salaire
15.       Console.WriteLine(pamMetier.GetSalaire("260124402111742", 30, 5))
16.       Console.WriteLine(pamMetier.GetSalaire("254104940426058", 150, 20))
17.       Try
18.         Console.WriteLine(pamMetier.GetSalaire("xx", 150, 20))
19.       Catch ex As PamException
20.         Console.WriteLine(String.Format("PamException : {0}", ex.Message))
21.       End Try
22.     Catch ex As Exception
23.       Console.WriteLine(String.Format("Exception : {0}", ex.ToString))
24.     End Try
25.   End Sub
26. End Module
```

- ligne 11 : on exploite le fichier de configuration [spring-config.xml]. Celui-ci est décrit ci-dessous.
- ligne 13 : on demande à Spring une référence sur la couche [metier]
- lignes 15-16 : tests de la méthode [GetSalaire] de l'interface [IPamMetier]
- lignes 17-24 : test de la méthode [GetSalaire] lorsqu'il se produit une exception

Le programme de test utilise le fichier de configuration Spring [spring-config.xml] suivant :

```
1.  <?xml version="1.0" encoding="iso-8859-1" ?>
2.  <objects xmlns="http://www.springframework.net" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
    instance"
3.    xsi:schemaLocation="http://www.springframework.net
4.  http://www.springframework.net/xsd/spring-objects.xsd">
5.    <object id="pamdao" type="istia.st.pam.dao.service.PamDaoSqlMap, pam-dao-ibatis"/>
6.    <object id="pammetier" type="istia.st.pam.metier.service.PamMetierImpl, pam-metier">
7.      <property name="PamDao" ref="pamdao"/>
8.    </object>
9.  </objects>
```

- ligne 5 : l'objet d'id "pamdao" a le type [istia.st.pam.dao.service.PamDaoSqlMap] et est trouvé dans l'assembly [pam-dao-ibatis]. On voit qu'ici la couche [dao] est celle étudiée précédemment, c.a.d. l'implémentation iBatis.
- lignes 6-8 : l'objet d'id "pammetier" a le type [istia.st.pam.dao.service.PamMetierImpl] et est trouvé dans l'assembly [pam-metier].
- ligne 7 : l'objet [PamMetierImpl] instancié par Spring a une propriété publique [PamDao] qui est une référence sur la couche [dao]. Cette propriété est initialisée avec la référence de la couche [dao] créée ligne 5.

Parce que la couche [dao] est celle implémentée avec l'outil iBatis, un certain nombre de fichiers de configuration sont nécessaires [dbpam.xml, properties.xml, providers.config, sqlmap.config]. Ceux-ci sont identiques à ceux utilisés pour les tests de la couche [dao] (cf paragraphes 8.4.4.1, 8.4.4.2, 8.4.4.3).

L'exécution faite avec la base de données décrite au paragraphe 2.1, page 3, donne le résultat console suivant :

```
1.  [[[260124402111742,Laverti,Justine,la Brûlerie,St Marcel,49014,0],[1, 1,93, 2, 3,
    12]],[3,49,6,15,9,39,7,88],[1, 1,93, 2, 3, 12],[64,85 : 17,45 : 10 : 15 : 72,4 ]
2.  [[[254104940426058,Jouveinal,Marie,5 rue des Oiseaux,St Corentin,49203,0],[2, 2,1, 2,1, 3,1,
    15]],[3,49,6,15,9,39,7,88],[2, 2,1, 2,1, 3,1, 15],[362,25 : 97,48 : 42 : 62 : 368,77 ]
3.  PamException : L'employé de n° SS [xx] n'existe pas
```

- lignes 1-2 : les 2 feuilles de salaire demandées
- ligne 3 : l'exception de type [PamException] provoquée par un employé inexistant.

8.7.6 Génération de la DLL de la couche [metier]

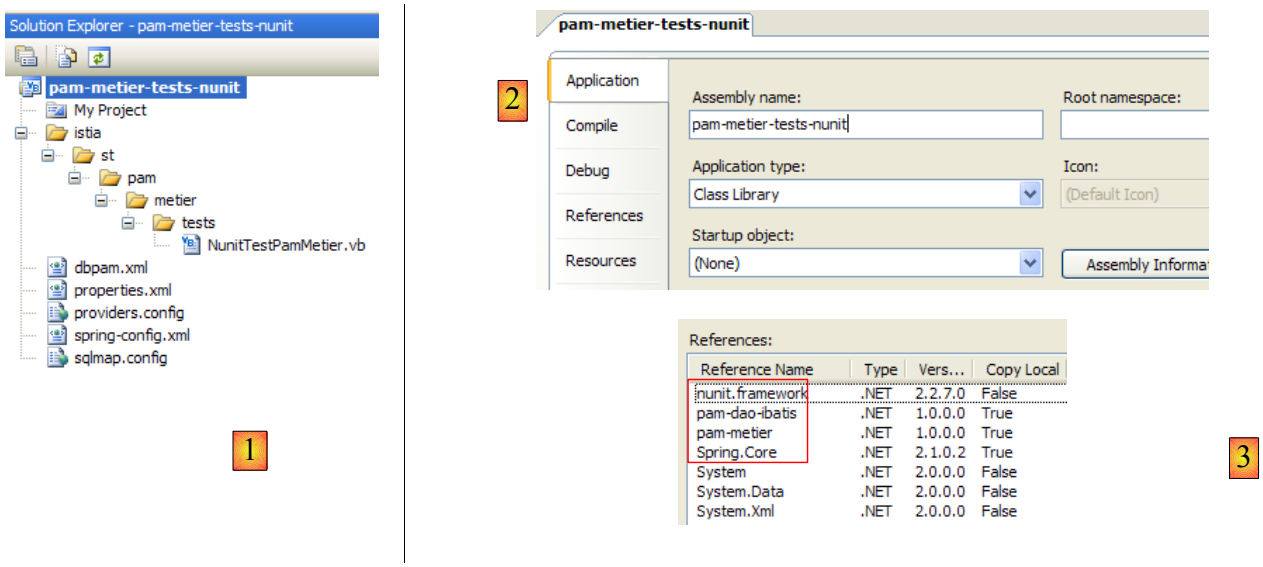
Une fois écrite et testée la classe [PamMetierImpl], on génèrera la DLL [pam-metier.dll) de la couche [metier] en suivant la méthode décrite au paragraphe 8.5.4, page 60.

8.8 Tests unitaires de la couche métier

Les tests précédents étaient visuels : on vérifiait à l'écran qu'on obtenait bien les résultats attendus. Nous passons maintenant aux tests non visuels NUnit.

8.8.1 Le projet Visual studio des tests Nunit

Le projet Visual Studio du projet de tests NUnit de la couche [metier] pourrait être le suivant :



- dans [1], on trouve :
  - le programme de test [NUnitTestPamMetier.vb]
  - les fichiers de configuration *iBatis* et *Spring*. Ils sont identiques à ceux de même nom étudiés précédemment. Ils ont leur propriété [Copy to Output Directory] à vrai.

- dans [2], on voit que le projet va générer une DLL nommé [pam-metier-tests-nunit.dll]

- dans [3], on voit :
  - dans [References], les références aux DLL
    - [pam-dao-ibatis.dll] qui contient les classes de la couche [dao] construite précédemment
    - [pam-metier.dll] qui contient les classes de la couche [metier] construite précédemment
    - [nunit.framework.dll] qui contient les classes de l'outil [NUnit]
    - [Spring.Core.dll] la DLL de Spring

8.8.2 La classe de test [NunitTestPamMetier.vb]

La classe de test NUnit sera la suivante :

```
1.  Imports System
2.  Imports System.Collections
3.  Imports NUnit.Framework
4.  Imports istia.st.pam.dao.service
5.  Imports istia.st.pam.dao.entites
6.  Imports istia.st.pam.metier.service
7.  Imports istia.st.pam.metier.entites
8.  Imports Spring.Objects.Factory.Xml
9.  Imports Spring.Core.IO
10.
11. Namespace istia.st.pam.metier.tests
12.
13.   <TestFixture()> _
14.   Public Class NunitTestPamMetier
15.
16.     ' la couche [metier] à tester
17.     Private PamMetier As IPamMetier
18.
19.     Public Sub New()
20.       ' on récupère une image mémoire du fichier de configuration Spring
21.       Dim Factory As XmlObjectFactory = New XmlObjectFactory(New FileSystemResource("spring-
    config.xml"))
22.       ' on demande une référence sur la couche [metier]
23.       PamMetier = CType(Factory.GetObject("pammetier"), IPamMetier)
24.     End Sub
25.
26.     <Test()> _
27.     Public Sub GetAllIdentitesEmployes()
28.       ' vérification nbre d'employes
29.       Assert.AreEqual(2, PamMetier.GetAllIdentitesEmployes.Length)
30.     End Sub
31.
32.     <Test()> _
33. Public Sub GetSalaire1()
34.       ' calcul d'une feuille de salaire
```

```
35.       Dim FeuilleSalaire As FeuilleSalaire = PamMetier.GetSalaire("254104940426058", 150, 20)
36.       ' vérifications
37.       Assert.AreEqual(368.77, FeuilleSalaire.ElementsSalaire.SalaireNet, 0.000001)
38.       ' feuille de salaire d'un employé inexistant
39.       Dim Erreur As Boolean
40.       Try
41.         FeuilleSalaire = PamMetier.GetSalaire("xx", 150, 20)
42.       Catch ex As PamException
43.         Erreur = True
44.       End Try
45.       Assert.IsTrue(Erreur)
46.     End Sub
47.
48.   End Class
49. End Namespace
```

- ligne 17 : le champ privé [PamMetier] est une instance de l'interface d'accès à la couche [metier]. On notera que le type de ce champ est une **interface** et non une **classe**. Cela signifie que l'instance [PamMetier] ne rend accessibles que des méthodes, celles de l'interface [IPamMetier].
- lignes 19-24 : le constructeur de la classe initialise le champ privé [PamDao] à partir du contenu d'un fichier [Spring] appelé ici "spring-config.xml". Ce fichier est identique à celui utilisé pour le test de type [Main].
- lignes 27-30 : testent la méthode [GetAllIdentitesEmployes]
- lignes 33-46 : testent la méthode [GetSalaire]

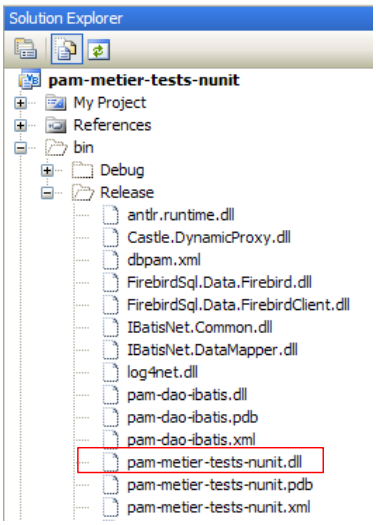
Le fichier de configuration Spring "spring-config.xml" est le suivant :

```
1.  <?xml version="1.0" encoding="iso-8859-1" ?>
2.  <objects xmlns="http://www.springframework.net" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
    instance"
3.    xsi:schemaLocation="http://www.springframework.net
4.  http://www.springframework.net/xsd/spring-objects.xsd">
5.    <object id="pamdao" type="istia.st.pam.dao.service.PamDaoSqlMap, pam-dao-ibatis"/>
6.    <object id="pammetier" type="istia.st.pam.metier.service.PamMetierImpl, pam-metier">
7.      <property name="PamDao" ref="pamdao"/>
8.    </object>
9.  </objects>
```

Il est identique à celui utilisé lors du test [MainTestMetierDaoIbatis] décrit au paragraphe 8.7.5, page 68. Il en est de même pour les fichiers de configuration de la couche iBatis.

8.8.3 Mise en oeuvre des tests

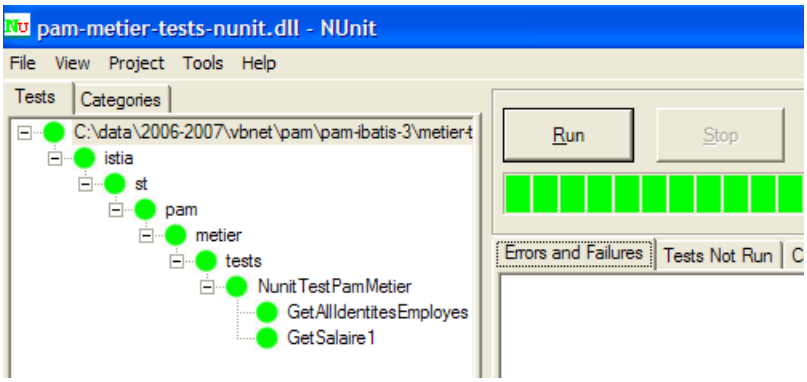
Le projet ci-dessus génère la DLL [pam-metier-tests-nunit.dll] dans le dossier [bin/Release].



Le dossier [bin/Release] contient en outre :

- les DLL qui font partie des références du projet et qui ont l'attribut [Copie locale] à vrai : [pam-dao-ibatis.dll, pam-metier.dll, Spring.Core.dll]. Ces DLL sont accompagnées des copies des DLL qu'elles utilisent elles-mêmes :
  - [Castle.DynamicProxy.dll, IbatisNet.Common.dll, IbatisNet.DataMapper.dll] pour l'outil iBatis
  - [antlr.runtime.dll, log4net.dll] pour l'outil Spring
- les fichiers de configuration du projet qui ont leur attribut [Copy to Output Directory] à vrai : [dbpam.xml, properties.xml, providers.config, spring-config.xml, sqlmap.config]

On charge la DLL [pam-metier-tests-nunit.dll] avec l'outil [NUnit-Gui] et on exécute les tests :



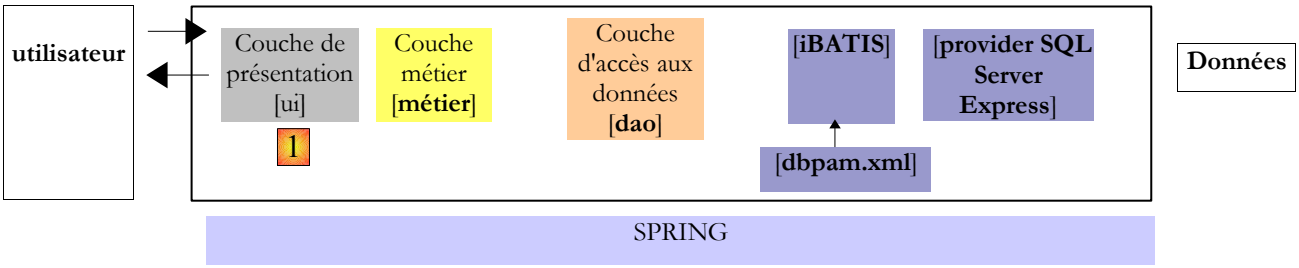
Ci-dessus, les tests ont été réussis.

Travail pratique :

- mettre en oeuvre sur machine les tests de la classe [PamMetierImpl].
- utiliser différents fichiers de configuration SqlMap afin d'utiliser des SGBD différents (Firebird, MySQL, Postgres, SQL Server)

8.9 La couche [ui]

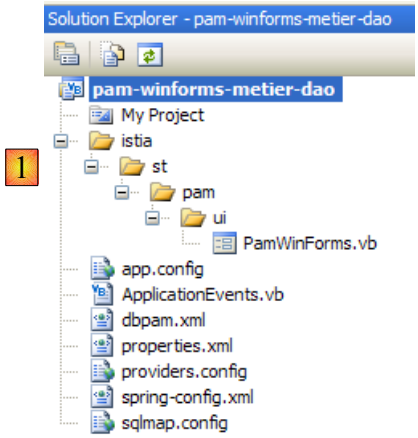
Revenons sur l'architecture générale de l'application [SimuPaie] :

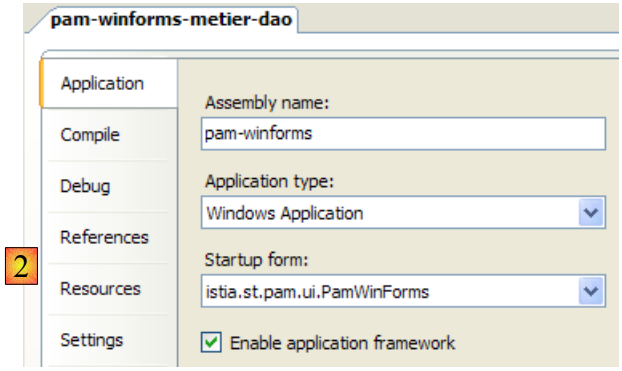


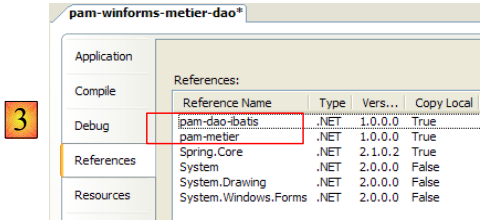
Nous considérons désormais que la couche [métier] est acquise et qu'elle a été encapsulée dans la DLL [pam-metier.dll]. Nous nous intéressons maintenant à la couche [ui] implémentée ici sous la forme d'une interface graphique VB.NET.

8.9.1 Le projet Visual Studio de la couche [ui]

Le projet Visual Studio de la couche [ui] pourrait ressembler à ce qui suit :







- en [1] l'ensemble du projet. La couche [ui] est formée de deux classes :
  - [PamWinForms.vb] : l'interface graphique du projet
  - [ApplicationEvents.vb] : le code de gestion des événements de l'application

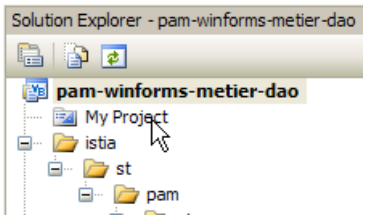
  Le reste du projet est formé de fichiers de configuration [app.config, dbpam.xml, properties.xml, providers.config, spring-config.xml, sqlmap.config].
- en [2] le type du projet (Windows Application) et le nom de l'assembly généré (pam-winforms).
- en [3] les références utilisées par le projet. On notera les DLL [pam-dao-ibatis, pam-metier] des couches [dao] et [métier] étudiées précédemment.

8.9.2 Configuration de l'application

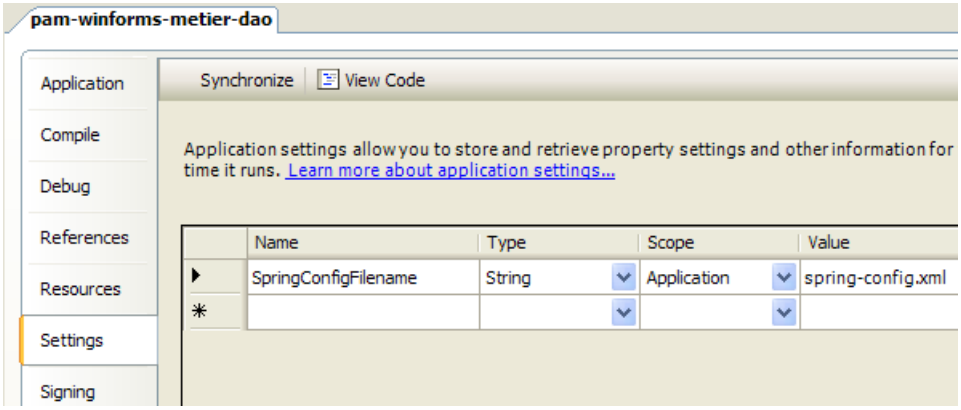
Le fichier [app.config] qui configure l'application sera le suivant :

```
1.  <?xml version="1.0" encoding="utf-8" ?>
2.  <configuration>
3.    <configSections>
4.     <sectionGroup name="applicationSettings" type="System.Configuration.ApplicationSettingsGroup,
    System, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" >
5.       <section name="My.MySettings" type="System.Configuration.ClientSettingsSection, System,
    Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
6.     </sectionGroup>
7.   </configSections>
8.   <applicationSettings>
9.     <My.MySettings>
10.     <setting name="SpringConfigFilename" serializeAs="String">
11.       <value>spring-config.xml</value>
12.     </setting>
13.   </My.MySettings>
14.  </applicationSettings>
15. </configuration>
```

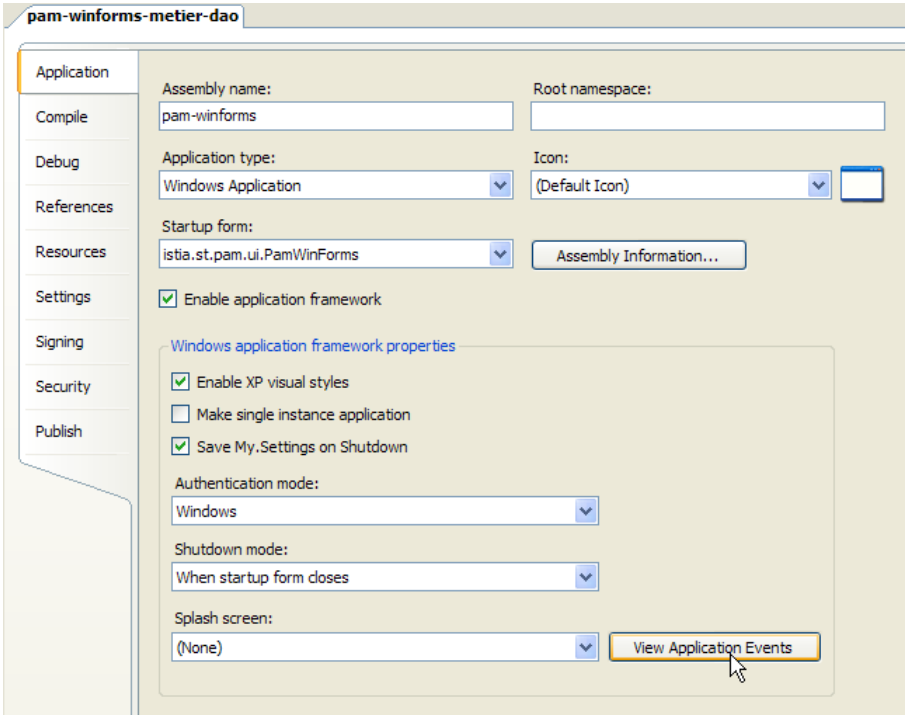
Ce fichier de configuration a été généré par Visual Sudio via l'option [Settings] des propriétés du projet :

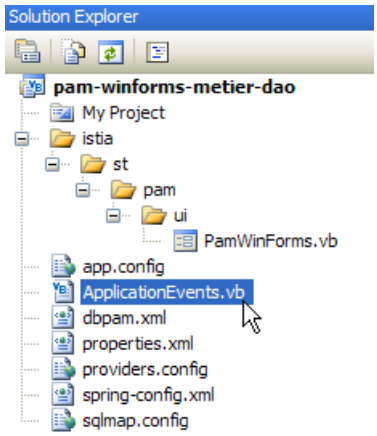


double-clic sur [My Project]



Le squelette de la classe gérant les événements de l'application est généré par Visual Studio à partir de la page de propriétés de l'application :





Le code généré est le suivant :

```
1.  Namespace My
2.
3.      ' The following events are availble for MyApplication:
4.      '
5.      ' Startup: Raised when the application starts, before the startup form is created.
6.      ' Shutdown: Raised after all application forms are closed.  This event is not raised if the
    application terminates abnormally.
7.      ' UnhandledException: Raised if the application encounters an unhandled exception.
8.      ' StartupNextInstance: Raised when launching a single-instance application and the application
    is already active.
9.      ' NetworkAvailabilityChanged: Raised when the network connection is connected or disconnected.
10.    Partial Friend Class MyApplication
11.
12.    End Class
13.
14. End Namespace
```

Les lignes 5 à 9 ci-dessus donnent la liste de événements que la classe [MyApplication] peut gérer. Ici, nous gèrerons l'événement [Startup], qui signale le démarrage de l'application, de la façon suivante :

```
1.  Imports istia.st.pam.metier.entites
2.  Imports istia.st.pam.metier.service
3.  Imports Spring.Objects.Factory.Xml
4.  Imports Spring.Core.IO
```

```
5.  Imports istia.st.pam.dao.entites
6.
7.  Namespace My
8.
9.    Partial Friend Class MyApplication
10.
11.     ' variables d'instance
12.     Public Msg As String = String.Empty
13.     Public Erreur As Boolean = False
14.     Public PamMetier As IPamMetier
15.     Public Employes As RawEmploye()
16.
17.     ' démarrage de l'application
18.     Private Sub MyApplication_Startup(ByVal sender As Object, ByVal e As
    Microsoft.VisualBasic.ApplicationServices.StartupEventArgs) Handles Me.Startup
19.       ' on gère les erreurs de configuration
20.       Try
21.         ' on récupère une instance de la réserve d'objets Spring
22.         Dim Factory As XmlObjectFactory = New XmlObjectFactory(New
    FileSystemResource(My.Settings.SpringConfigFilename))
23.         ' on demande l'instanciation de l'objet [pammetier]
24.         PamMetier = CType(Factory.GetObject("pammetier"), IPamMetier)
25.         ' on récupère la liste des employés
26.         Employes = PamMetier.GetAllIdentitesEmployes
27.         ' on note le succès
28.         Msg = "Base de données chargée ..."
29.       Catch ex As Exception
30.         ' on note l'erreur
31.         Msg = String.Format("L'erreur suivante s'est produite lors de l'initialisation de
    l'application : {0}", ex.Message)
32.         Erreur = True
33.       End Try
34.     End Sub
35.   End Class
36.
37. End Namespace
```

Le rôle de la procédure [MyApplication_Startup] est ici de :

- mettre en place l'architecture à trois couches de l'application
- mémoriser la liste simplifiée (SS, NOM, PRENOM) de tous les employés
- signaler une erreur éventuelle de cette initialisation

Il faut se rappeler que les champs ou méthodes publics de l'objet [My.Application] sont accessibles aux différents formulaires du projet.

- ligne 14 : une référence sur la couche [métier] avec laquelle conversera le formulaire [PamWinForms.vb]
- ligne 15 : le tableau qui mémorise la liste simplifiée (SS, NOM, PRENOM) de tous les employés
- ligne 13 : le booléen signalant ou non une erreur d'initialisation de l'application
- ligne 12 : un message indiquant comment s'est terminée l'initialisation de l'application
- lignes 22-24 : on exploite le fichier de configuration Spring afin d'instancier la couche [métier]. Le nom de ce fichier a été mémorisé dans les [Settings] de l'application (voir plus haut). Le contenu du fichier de configuration Spring est le suivant :

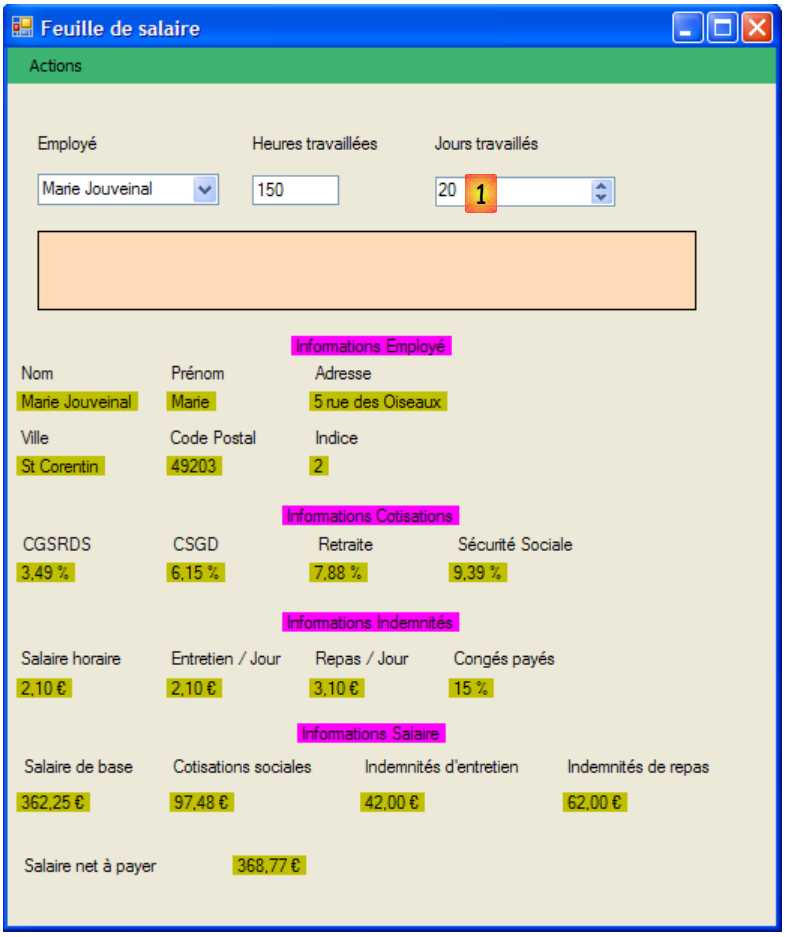
```
1.  <?xml version="1.0" encoding="iso-8859-1" ?>
2.  <objects xmlns="http://www.springframework.net"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3.    xsi:schemaLocation="http://www.springframework.net
4.  http://www.springframework.net/xsd/spring-objects.xsd">
5.    <object id="pamdao" type="istia.st.pam.dao.service.PamDaoSqlMap, pam-dao-ibatis"/>
6.    <object id="pammetier" type="istia.st.pam.metier.service.PamMetierImpl, pam-metier">
7.      <property name="PamDao" ref="pamdao"/>
8.    </object>
9.  </objects>
```

Ce fichier a déjà été utilisé lors des tests de la couche métier (cf page 69). On sait que l'instanciation du bean [pammetier] provoque l'instanciation des couches [métier] et [dao] de l'application. L'instanciation de la couche [dao] va utiliser les fichiers de configuration [sqlmap.config, properties.xml, providers.config, dbpam.xml] nécessaires à la couches [iBatis / SqlMap]. Ici, ils sont identiques à ceux utilisé lors des tests de la couche [dao] (cf paragraphes 8.4.4.1, 8.4.4.2, 8.4.4.3).

- ligne 26 : on récupère une référence sur le tableau simplifié (SS, NOM, PRENOM) des employés.
- ligne 28 : le message en cas de succès
- lignes 29 – 33 : gèrent une éventuelle exception
- ligne 31 : le message en cas d'erreur
- ligne 32 : le booléen *Erreur* est mis à vrai pour signaler l'erreur

8.9.3 L'interface graphique

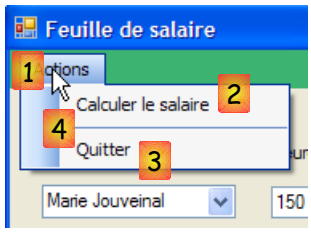
L'interface graphique sera la suivante :



Les composants du formulaire sont identiques à ceux du formulaire décrit au paragraphe 3.1, page 8 au détail près suivant :

N°	Type	Nom	Rôle
1	NumericUpDown	NumericUpDownJoursTravaillés	Nombre de jours travaillés – nombre entier Minimum : 0, Maximum : 31, Increment : 1

Le menu a deux options :



N°	Type	Nom	Rôle
1	ToolStripMenuItem	ActionsToolStripMenuItem	liste d'actions possibles
2	ToolStripMenuItem	ToolStripMenuItemCalculerSalaire	pour lancer le calcul du salaire. N'est actif que si le champ [TextBoxHeuresTravaillées] est non vide.
3	ToolStripMenuItem	QuitterToolStripMenuItem	pour quitter l'application
4	ToolStripSeparator		séparateur d'options de menu

Question : écrire le code du formulaire [PamWinForms.vb] en vous inspirant des versions VB.NET précédentes.

Travail pratique :

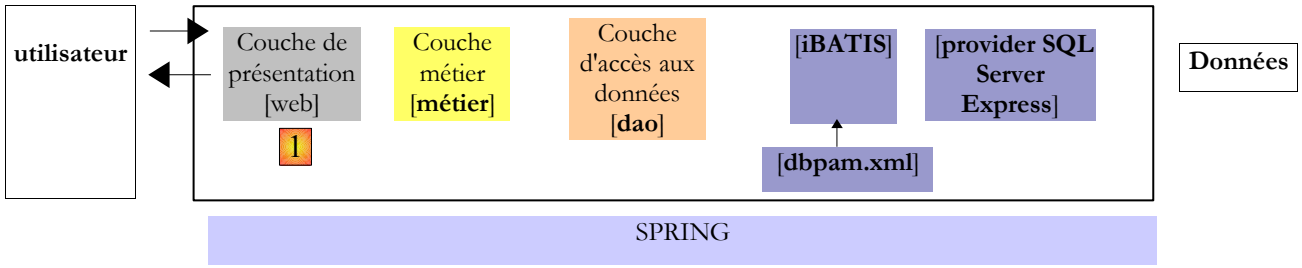
- mettre en oeuvre sur machine les tests de l'application VB.NET précédente
- utiliser différents fichiers de configuration *SqlMap* afin d'utiliser des SGBD différents (Firebird, MySQL, Postgres, SQL Server)

9 L'application [SimuPaie] – version 7 – ASP.NET / 3 couches

Nous reprenons l'application précédente en lui donnant maintenant une interface web.

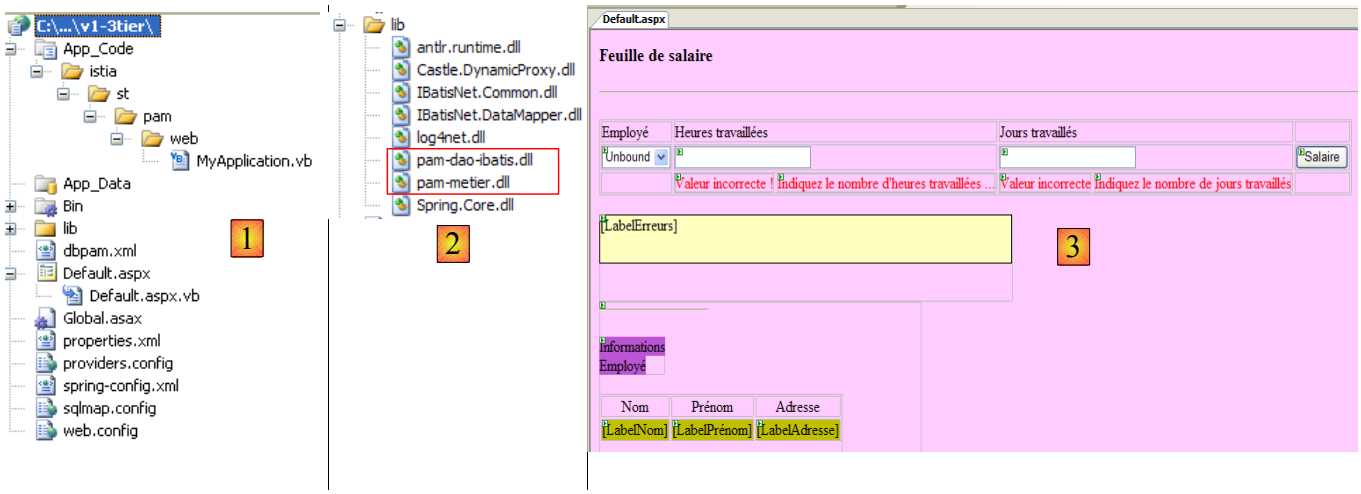
9.1 La couche [ui]

Revenons sur l'architecture générale de l'application [SimuPaie] :



La couche [ui-web] est une couche web. Comme auparavant, nous considérons que les couche [dao] et [métier] sont acquises et encapsulées dans les DLL [pam-dao-ibatis, pam-metier.dll].

9.2 Le projet Visual Web Developer de la couche [web-ui]



- en [1] on trouve :
  - le fichier de configuration [web.config] de l'application – joue un rôle analogue au fichier de configuration [app.config] de l'application VB.NET précédente
  - le fichier [MyApplication.vb] qui gère les événements de l'application web, ici son démarrage - joue un rôle analogue au fichier [ApplicationEvents.vb] de l'application VB.NET précédente
  - le formulaire [Default.aspx.vb] de l'application - joue un rôle analogue au formulaire [PamWinForms.vb] de l'application VB.NET précédente
  - les fichiers de configuration des différentes couches de l'application web [spring-config.xml, sqlmap.config, providers.config, properties.xml, dbpam.xml] – sont identiques aux fichiers de même nom de l'application VB.NET précédente

- en [2] on trouve le dossier [lib] dans lequel on a mis toutes les DLL nécessaires au projet. Les DLL [pam-dao-ibatis, pam-metier, Spring.Core] ont été ajoutées aux références du projet ce qui a provoqué leur recopie et celles des DLL associées dans le dossier [/Bin].

- en [3] on trouve le formulaire [Default.aspx]. Celui-ci est identique à celui de la version 3 de notre application (paragraphe 5.1, page 19) qu'on pourra recopier. Son code [Default.aspx.vb] devra lui évoluer.

9.3 Configuration de l'application

Le fichier [web.config] qui configure l'application définit les mêmes données que le fichier [app.config] configurant l'application VB.NET étudiée précédemment, avec cependant une syntaxe différente :

```
1.  <?xml version="1.0" encoding="utf-8" ?>
2.  <configuration>
3.    <appSettings>
4.      <add key="SpringConfigFileName" value="~/spring-config.xml"/>
5.    </appSettings>
6.  </configuration>
```

Il se contente, ligne 4, d'indiquer le nom du fichier de configuration Spring. On notera l'écriture [~/spring-config.xml] où ~désigne le dossier de l'application web. Ce fichier est exploité au démarrage de l'application par le code du couple [Global.asax, MyApplication.vb] :

Global.asax

```
<%@ Application Language="VB" inherits="istia.st.pam.web.MyApplication"  %>
```

MyApplication.vb

```
1.  Imports System.Configuration
2.  Imports istia.st.pam.metier.entites
3.  Imports istia.st.pam.metier.service
4.  Imports Spring.Objects.Factory.Xml
5.  Imports Spring.Core.IO
6.  Imports istia.st.pam.dao.entites
7.
8.  Namespace istia.st.pam.web
9.
10.   Public Class MyApplication
11.     Inherits System.Web.HttpApplication
12.
13.     ' variables d'instance
14.     Public Shared Msg As String = String.Empty
15.     Public Shared PamMetier As IPamMetier
16.     Public Shared Employes As RawEmploye()
17.     Public Shared Erreur As Boolean = False
18.
19.     ' démarrage de l'application
20.     Sub Application_Start(ByVal sender As Object, ByVal e As EventArgs)
21.       ' on gère les erreurs de configuration
22.       Try
23.         ' on crée une image mémoire de la configuration Spring
24.         Dim Factory As XmlObjectFactory = New XmlObjectFactory(New
    FileSystemResource(ConfigurationManager.AppSettings("SpringConfigFilename")))
25.         ' on demande une référence sur l'objet [pammetier]
26.         PamMetier = CType(Factory.GetObject("pammetier"), IPamMetier)
27.         ' on récupère le tableau simplifié des employés
28.         Employes = PamMetier.GetAllIdentitesEmployes
29.         ' on note le succès
30.         Msg = "Base de données chargée ..."
31.       Catch ex As Exception
32.         ' on note l'erreur
33.         Erreur = True
34.         Msg = String.Format("L'erreur suivante s'est produite lors de l'initialisation de
    l'application : {0}", ex.ToString)
35.       End Try
36.     End Sub
37.
38.   End Class
39.
40. End Namespace
```

Ce code est très proche de celui de [ApplicationEvents.vb] expliqué au paragraphe 8.9.2, page 75. On a simplement adapté certains détails :

- ligne 8 : l'espace de noms
- lignes 10-11 : la classe dérive de [HttpApplication]
- ligne 24 : le paramètre [SpringConfigFilename] est obtenu différemment

9.4 Le formulaire [Default.aspx]

Le formulaire est celui décrit au paragraphe 5.1, page 19 :

Question : Le fonctionnement du formulaire a été décrit au paragraphe 5.3.1, page 23. En vous inspirant du code VB.NET écrit précédemment, écrire le code [Default.aspx.vb] assurant ce fonctionnement.

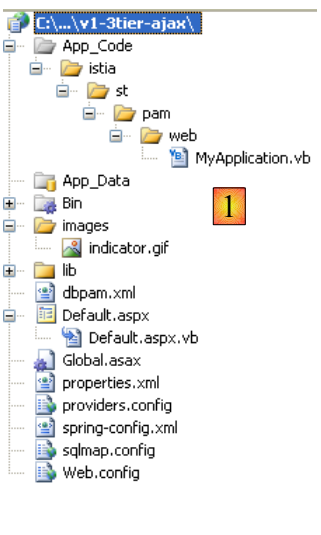
Travail pratique :

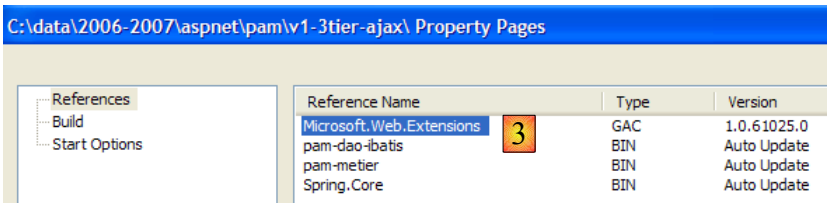
- mettre en oeuvre sur machine l'application web précédente
- utiliser différents fichiers de configuration SqlMap afin d'utiliser des SGBD différents (Firebird, MySQL, Postgres, SQL Server)

10 L'application [SimuPaie] – version 8 – AJAX / ASP.NET

Nous suivons la démarche décrite au paragraphe 7.2, page 34, pour ajaxifier l'application web précédente.

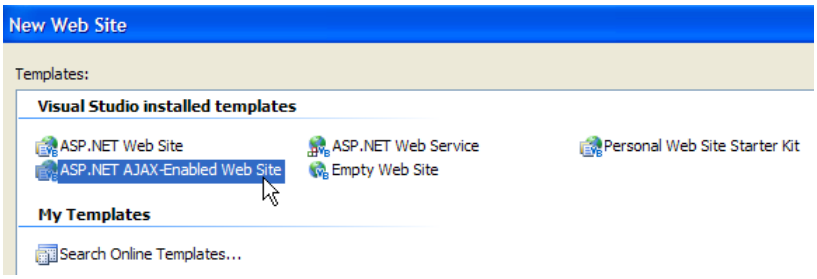
10.1 Le projet Visual Web Developer de la couche [web-ui-ajax]





Le projet ASP/Ajax a été construit à partir du projet ASP.NET de la version précédente de la façon suivante :

- création d'un projet Ajax :



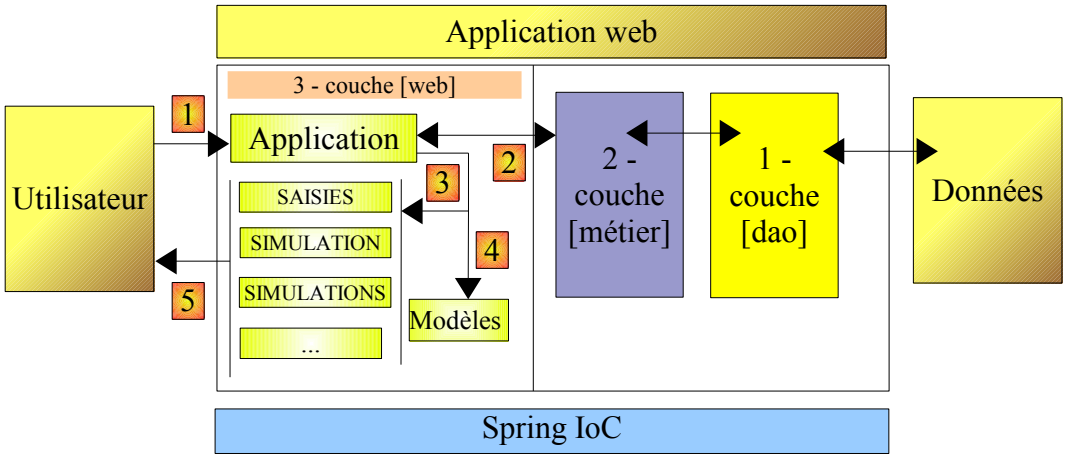
- copie dans le dossier du projet de tous les fichiers et dossiers du projet précédent (cf [1] ci-dessus)
- mise à jour des références du nouveau projet (cf [3] ci-dessus)
- ajout de composants Ajax dans le formulaire [Default.aspx] (cf [2] ci-dessus) et de labels pour vérifier l'ajaxification de l'application

Cette démarche a été décrite page 34. On la suivra pour construire le projet web de cette version et tester celle-ci.

11 L'application [SimuPaie] – version 9 – ASP.NET / multi-vues / mono-page

Lectures conseillées : référence [1], **programmation ASP.NET vol2**, paragraphes :
- 1.1.3 : Composants serveur et contrôleur d'application
- 1.1.4 : Exemples d'applications MVC avec composants serveurs ASP

Nous étudions maintenant une version dérivée de l'application ASP.NET à trois couches étudiée précédemment et qui lui ajoute de nouvelles fonctionnalités. L'architecture de notre application évolue de la façon suivante :



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

1. le client fait une demande à l'application.
2. l'application traite cette demande. Pour ce faire, elle peut avoir besoin de l'aide de la couche [métier] qui elle-même peut avoir besoin de la couche [dao] si des données doivent être échangées avec la base de données. L'application reçoit une réponse de la couche [métier].
3. selon celle-ci, elle choisit (3) la vue (= la réponse) à envoyer au client en lui fournissant (4) les informations (le modèle) dont elle a besoin.
4. la réponse est envoyée au client (5)

On a ici une architecture web dite MVC (Modèle – Vue – Contrôleur) :

- [Application] est le **contrôleur**. Il voit passer toutes les requêtes du client.
- [Saisies, Simulation, Simulations, ...] sont les **vues**. Une vue en .NET est du code ASP / HTML standard qui contient des composants qu'il convient d'initialiser. Les valeurs qu'il faut fournir à ces composants forment le **modèle** de la vue.

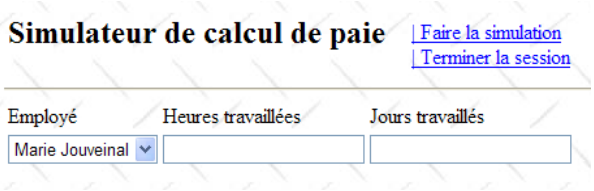
Nous allons ici implémenter le modèle (design pattern) MVC de la façon suivante :
- les vues seront des composants [View] au sein d'une page unique [Default.aspx]
- le contrôleur est alors le code [Default.aspx.vb] de cette page unique.

Seules des applications basiques peuvent supporter cette implémentation MVC. En effet, à chaque requête, tous les composants de la page [Default.aspx] sont instanciés, donc toutes les vues. Au moment d'envoyer la réponse, l'une d'elles est choisie par le code de contrôle de l'application simplement en rendant visible le composant [View] correspondant et en cachant les autres. Si l'application a de nombreuses vues, la page [Default.aspx] aura de nombreux composants et son coût d'instanciation peut devenir prohibitif. Par ailleurs, le mode [Design] de la page risque de devenir ingérable parce qu'ayant trop de vues. Ce type d'architecture convient pour des applications avec peu de vues et développées par une unique personne. Lorsqu'elle peut être adoptée, elle permet de développer une architecture MVC très simplement. C'est ce que nous allons voir dans cette nouvelle version.

11.1 Les vues de l'application

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

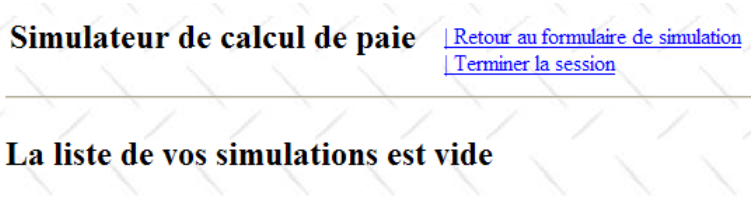
- la vue [VueSaisies] qui présente le formulaire de simulation



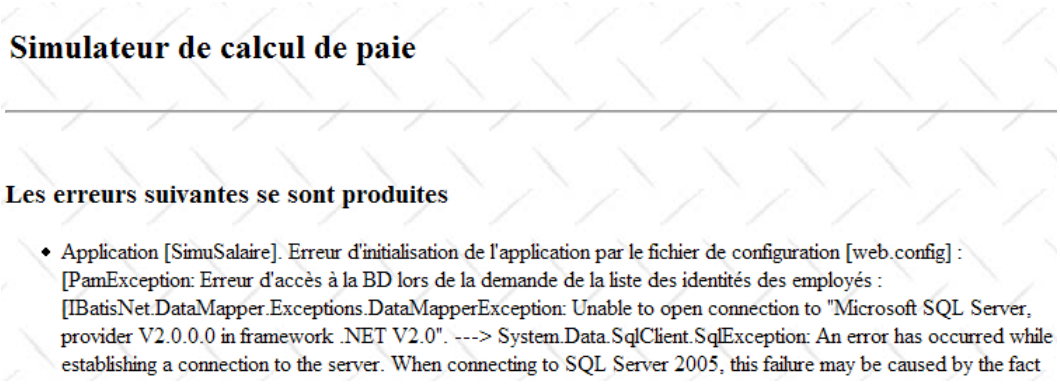
- la vue [VueSimulation] utilisée pour afficher le résultat détaillé de la simulation :

- la vue [VueSimulations] qui donne la liste des simulations faites par le client

- la vue [VueSimulationsVides] qui indique que le client n'a pas ou plus de simulations :

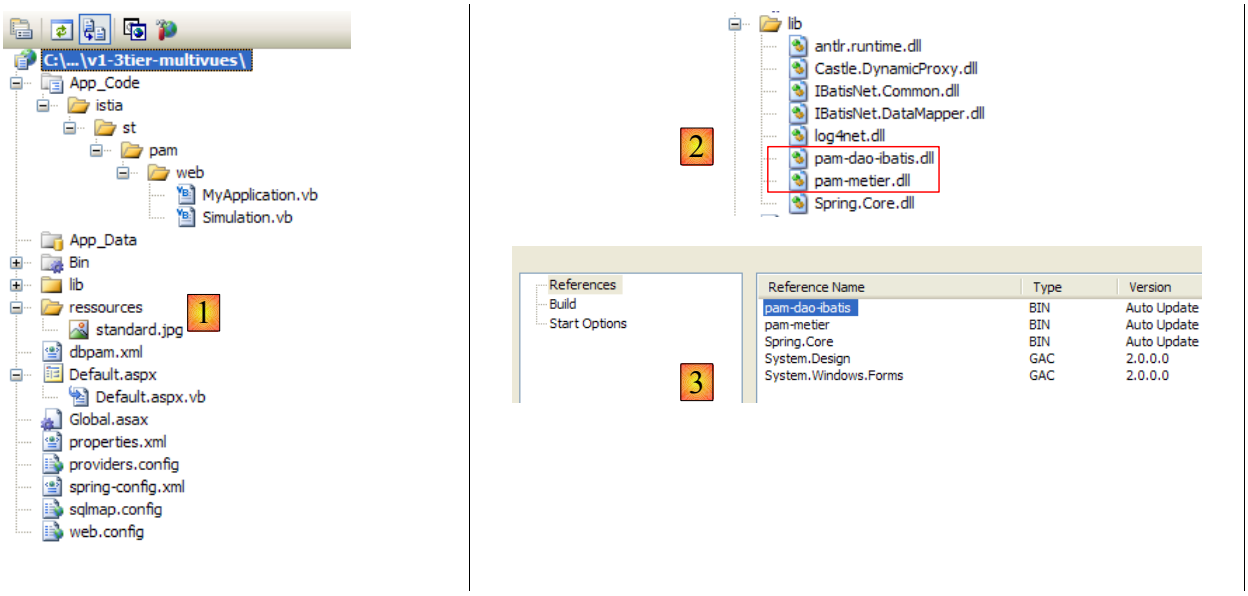


– la vue [VueErreurs] qui indique une ou plusieurs erreurs :



11.2 Le projet Visual Web Developer de la couche [web]

Le projet Visual Web Developer de la couche [web] est le suivant :



– en [1] on trouve :
  – le fichier de configuration [web.config] de l'application – est identique à celui de l'application précédente.
  – le fichier [MyApplication.vb] qui gère les événements de l'application web, ici son démarrage - est identique à celui de l'application précédente si ce n'est qu'il gère également le démarrage de la session utilisateur.
  – le formulaire [Default.aspx] de l'application – contient les différentes vues de l'application.
  – les fichiers de configuration des différentes couches de l'application web : [spring-config.xml, sqlmap.config, providers.config, properties.xml, dbpam.xml] – sont identiques aux fichiers de mêmes noms de l'application précédente

– en [2] on trouve le dossier [lib] dans lequel on a mis toutes les DLL nécessaires au projet – son contenu est identique au dossier [lib] de l'application précédente.

– en [3] on voit les références du projet.

11.3 Le fichier [MyApplication.vb]

Le fichier [MyApplication.vb] qui gère les événements de l'application web, est identique à celui de l'application précédente si ce n'est qu'il gère en plus le démarrage de la session utilisateur :

Global.asax

```
<%@ Application Language="VB" inherits="istia.st.pam.web.MyApplication"  %>
```

MyApplication.vb

```
1.  Imports istia.st.pam.metier.entites
2.  Imports istia.st.pam.metier.service
3.  Imports System.Collections.Generic
4.  Imports istia.st.pam.dao.entites
5.  Imports Spring.Objects.Factory.Xml
6.  Imports Spring.Core.IO
7.
8.  Namespace istia.st.pam.web
9.
10.  Public Class MyApplication
11.    Inherits System.Web.HttpApplication
12.
13.    ' données statiques de l'application
14.    Public Shared Msg As String = String.Empty
15.    Public Shared PamMetier As IPamMetier
16.    Public Shared Employes As RawEmploye()
17.    Public Shared Erreur As Boolean = False
18.
19.    ' démarrage de l'application
20.    Sub Application_Start(ByVal sender As Object, ByVal e As EventArgs)
21. ...
22.    End Sub
23.
24.
25.    ' démarrage de l'application
26.    Sub Session_Start(ByVal sender As Object, ByVal e As EventArgs)
27.      ' on met une liste de simulations vide dans la session
28.      Dim Simulations As New List(Of Simulation)
29.      Session("simulations") = Simulations
30.      ' id de la 1ère simulation
31.      Session("id") = 1
32.    End Sub
33.  End Class
34.
35. End Namespace
```

- lignes 26-32 : on gère le démarrage de la session. Nous mettrons dans celle-ci la liste des simulations faites par l'utilisateur.
- ligne 28 : une liste de simulations vide est créée. Une simulation est un objet de type [Simulation] que nous allons détailler prochainement.
- ligne 29 : la liste de simulation est placée dans la session associée à la clé " simulations "
- ligne 31 : chaque simulation est identifiée par un n° qui doit être incrémenté à chaque nouvelle simulation. Le n° de la prochaine simulation est stocké dans la session, associé à la clé " id ".

11.4 La classe [Simulation]

Un objet de type [Simulation] sert à encapsuler une ligne du tableau des simulations :

Son code est le suivant :

```
1.  Namespace istia.st.pam.web
2.
3.   Public Class Simulation
4.     ' données d'une simulation
5.     Private _Id As Integer
6.     Private _Nom As String
7.     Private _Prenom As String
8.     Private _HeuresTravaillees As Double
9.     Private _JoursTravailles As Integer
10.    Private _SalaireBase As Double
11.    Private _Indemnites As Double
12.    Private _CotisationsSociales As Double
13.    Private _SalaireNet As Double
14.
15.    ' propriétés publiques associées (Id, Nom, Prenom, HeuresTravaillees, JoursTravailles,
16.    ' SalaireBase, Indemnites, CotisationsSociales, SalaireNet
17. ...
18.
19.    ' constructeurs
20.    Public Sub New()
21.
22.    End Sub
23.
24.    Public Sub New(ByVal id As Integer, ByVal nom As String, ByVal prenom As String, ByVal
    heuresTravaillees As Double, ByVal joursTravailles As Integer, ByVal salaireBase As Double, ByVal
    indemnites As Double, ByVal cotisationsSociales As Double, ByVal salaireNet As Double)
25.      With Me
26.        .Id = id
27.        .Nom = nom
28.        .Prenom = prenom
29.        .HeuresTravaillees = heuresTravaillees
30.        .JoursTravailles = joursTravailles
31.        .SalaireBase = salaireBase
32.        .Indemnites = indemnites
33.        .CotisationsSociales = cotisationsSociales
34.        .SalaireNet = salaireNet
35.      End With
36.    End Sub
37.  End Class
38. End Namespace
```

Les champs de la classe correspondent aux colonnes du tableau des simulations sauf le champ [_Id]. Celui-ci est mémorisé dans une propriété du composant [DataGrid] qui affiche la liste de simulations.

11.5 La page [Default.aspx]

11.5.1 Vue d'ensemble

La page [Default.aspx] contient plusieurs composants [View], un pour chaque vue. Son squelette est le suivant :

```
1.  <%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb" Inherits="_Default" %>
2.
3.  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4.  <html xmlns="http://www.w3.org/1999/xhtml">
5.  <head id="Head1" runat="server">
6.    <title>Simulateur de paie</title>
7.  </head>
8.  <body background="ressources/standard.jpg">
9.    <form id="form1" runat="server">
10.     <table>
11.       <tr>
12.         <td>
13.           <h2>
14.             Simulateur de calcul de paie</h2>
15.         </td>
16.         <td>
17.           <label>
18.                </label></td>
19.         <td>
20.           <asp:LinkButton ID="LinkButtonFaireSimulation" runat="server"
21.             CausesValidation="False">| Faire la simulation<br /></asp:LinkButton>
22. ...
23.         </td>
24.     </table>
25.     <hr />
26.     <asp:MultiView ID="Vues1" ActiveViewIndex="0" runat="server">
27.       <asp:View ID="VueSaisies" runat="server">
28. ...
29.       </asp:View>
```

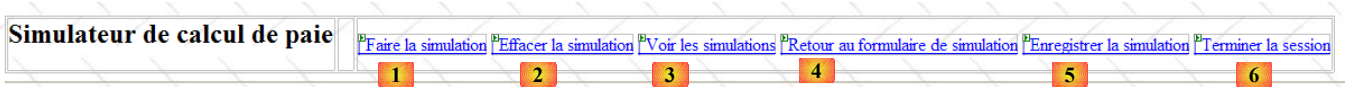
```
30.     </asp:MultiView>
31.     <asp:MultiView ID="Vues2" runat="server">
32.       <asp:View ID="VueSimulation" runat="server">
33. ...
34.       </asp:View>
35.       <asp:View ID="VueSimulations" runat="server">
36. ...
37.       </asp:View>
38.       <asp:View ID="VueSimulationsVides" runat="server">
39.         <h2>
40.           La liste de vos simulations est vide</h2>
41.       </asp:View>
42.       <asp:View ID="VueErreurs" runat="server">
43. ...
44.       </asp:View>
45.     </asp:MultiView>
46.   </form>
47. </body>
48. </html>
```

- lignes 10-25 : code ASP en-dehors de tout composant [Panel]. Il sera présent dans chaque vue dont il formera l'entête. Il présente à l'utilisateur la liste des actions possibles sous la forme d'une liste de liens.
- ligne 21 : on notera l'attribut *CausesValidation="False"* qui fait que les validateurs de la page ne seront pas exécutés lorsque le lien sera cliqué. Lorsque cet attribut est absent, sa valeur par défaut est *True*. La validation de la page sera faite dans le code par *Page.Validate*.
- lignes 26-30 : un composant [MultiView] avec une unique vue [VueSaisies]. Pour cette raison, on a mis en dur, ligne 26, le n° de la vue à afficher : *ActiveViewIndex="0"*
- lignes 31-45 : un composant [MultiView] avec quatre vues : la vue [VueSimulation] lignes 32-34, la vue [VueSimulations] lignes 35-37, la vue [VueSimulationsVides] lignes 38-41, la vue [VueErreurs] lignes 42-44. Un composant [MultiView] n'affiche qu'une vue à la fois. Pour afficher la vue n° i du composant *Vues2*, on écrira le code :

```
Vues2.ActiveViewIndex=i
```

11.5.2 L'entête

L'entête est formé des composants suivants :



N°	Type	Nom	Rôle
1	LinkButton	LinkButtonFaireSimulation	demande le calcul de la simulation
2	LinkButton	LinkButtonEffacerSimulation	efface le formulaire de saisie
3	LinkButton	LinkButtonVoirSimulations	affiche la liste des simulations déjà faites
4	LinkButton	LinkButtonFormulaireSimulation	ramène au formulaire de saisie
5	LinkButton	LinkButtonEnregistrerSimulation	enregistre la simulation courante dans la liste des simulations
6	LinkButton	LinkButtonTerminerSession	abandonne la session courante

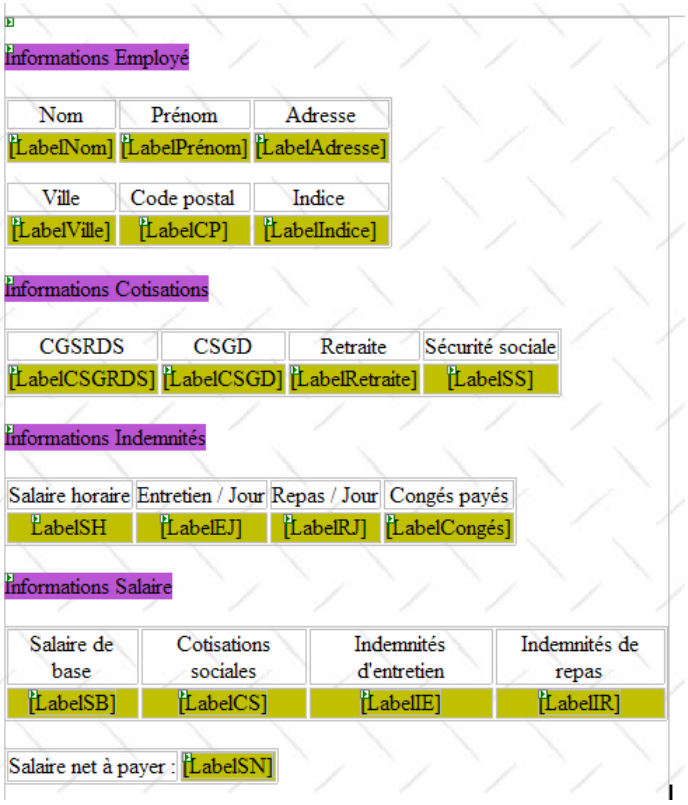
11.5.3 La vue [Saisies]

Le composant [View] nommé [VueSaisies] est le suivant :

N°	Type	Nom	Rôle
1	DropDownList	ComboBoxEmployes	Contient la liste des noms des employés
2	TextBox	TextBoxHeures	Nombre d'heures travaillées – nombre réel
3	TextBox	TextBoxJours	Nombre de jours travaillés – nombre entier
4	RequiredFieldValidator	RequiredFieldValidatorHeures	vérifie que le champ [2] [TextBoxHeures] n'est pas vide
5	RegularExpressionValidator	RegularExpressionValidatorHeures	vérifie que le champ [2] [TextBoxHeures] est un nombre réel >=0
6	RequiredFieldValidator	RequiredFieldValidatorJours	vérifie que le champ [3] [TextBoxJours] n'est pas vide
7	RegularExpressionValidator	RegularExpressionValidatorJours	vérifie que le champ [3] [TextBoxJours] est un nombre entier >=0

11.5.4 La vue [Simulation]

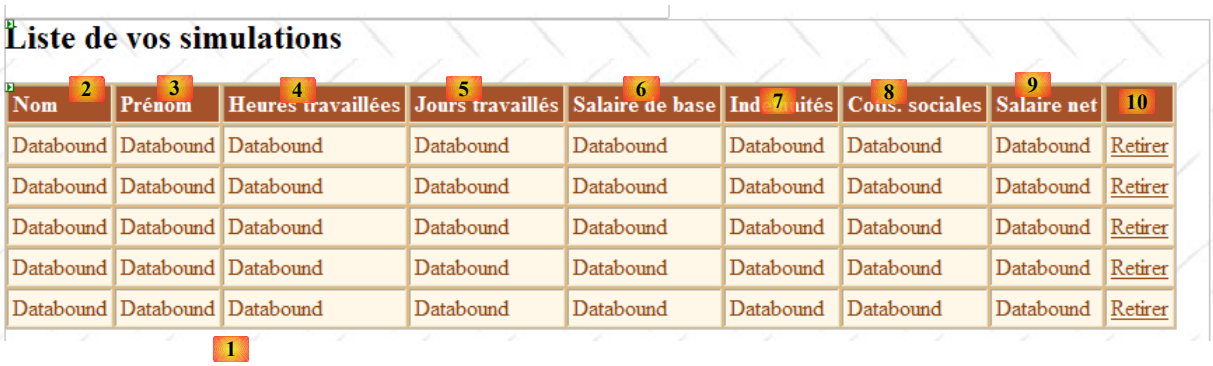
Le composant [View] nommé [VueSimulation] est le suivant :



Il n'est composé que de composants [Label] dont les ID sont indiqués ci-dessus.

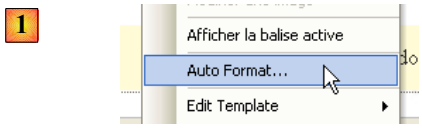
11.5.5 La vue [Simulations]

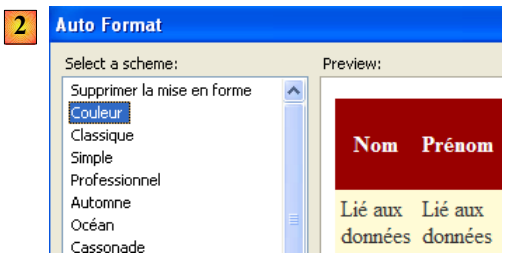
Le composant [View] nommé [VueSimulations] est le suivant :



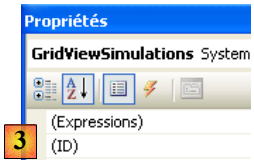
N°	Type	Nom	Rôle
1	GridView	GridViewSimulations	Contient la liste des simulations

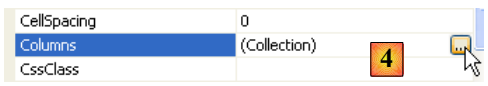
Les propriétés du composant [GridViewSimulations] ont été définies de la façon suivante :

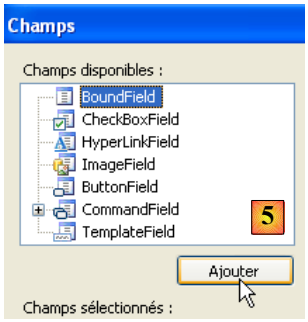




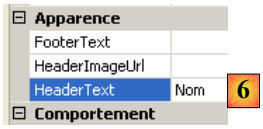
- en [1] : clic droit sur le [GridView] / option [AutoFormat]
- en [2] : choisir un type d'affichage pour le [GridView]

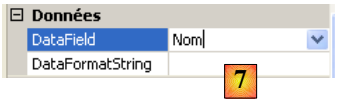






- en [3] : sélectionner les propriétés du [GridView]
- en [4] : éditer les colonnes du [GridView]
- en [5] : ajouter une colonne de type [BoundField] qui sera liée (bound) à l'une des propriétés publiques de l'objet à afficher dans la ligne du [GridView]. L'objet affiché ici, sera un objet de type [Simulation].





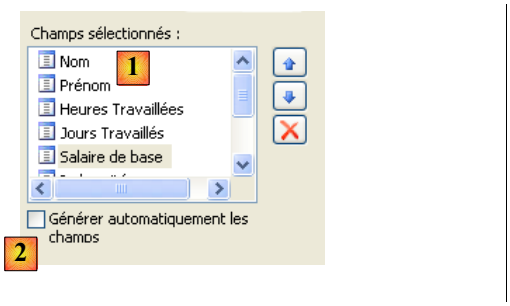
- en [6] : donner le titre de la colonne
- en [7] : donner le nom de la propriété de la classe [Simulation] qui sera associée à cette colonne.
- [DataFormatString] indique comment doivent être formatées les valeurs affichées dans la colonne.

Les colonnes du composant [GridViewSimulations] ont les propriété suivantes :

N°	Propriétés
2	Type : **BoundField**, HeaderText : **Nom**, DataField : **Nom**
3	Type : **BoundField**, HeaderText : **Prénom**, DataField : **Prenom**
4	Type : **BoundField**, HeaderText : **Heures travaillées**, DataField : **HeuresTravaillees**
5	Type : **BoundField**, HeaderText : **Jours travaillés**, DataField : **JoursTravailles**
6	Type : **BoundField**, HeaderText : **Salaire de base**, DataField : **SalaireBase**, DataFormatString : **{0:C}** (format monétaire, C=Currency) – affichera le sigle de l'euro.
7	Type : **BoundField**, HeaderText : **Indemnités**, Data Field : **Indemnites**, DataFormatString : **{0:C}**
8	Type : **BoundField**, HeaderText : **Cotis. sociales**, DataField : **CotisationsSociales**, DataFormatString : **{0:C}**

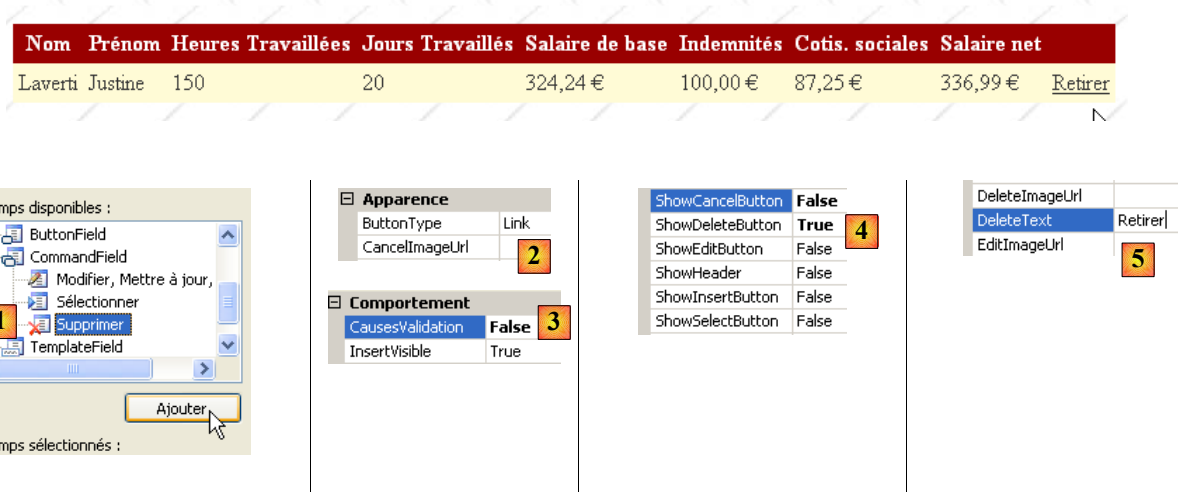
N°	Propriétés
9	Type : **BoundField**, HeaderText : **Salaire net**, DataField : **SalaireNet**, DataFormatString : **{0:C}**

On prêtera attention au fait que le champ [DataField] doit correspondre à une propriété existante de la classe [Simulation]. A l'issue de cette phase, toutes les colonnes de type [BoundField] ont été créées :



- en [1] : les colonnes créées pour le [gridView]
- en [2] : la génération automatique des colonnes doit être inhibée lorsque c'est le développeur qui les définit lui-même comme nous venons de le faire.

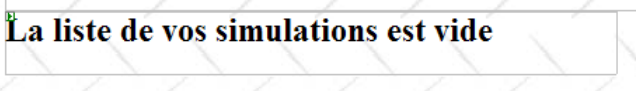
Il nous reste à créer la colonne des liens [Retirer] :



- en [1] : ajouter une colonne de type [CommandField / Supprimer]
- en [2] : *ButtonType=Link* pour avoir un lien dans la colonne plutôt qu'un bouton
- en [3] : *CausesValidation=False*, un clic sur le lien ne provoquera pas l'exécution des contrôles de validation qui peuvent se trouver sur la page. En effet, la suppression d'une simulation ne nécessite aucune vérification de données.
- en [4] : seul le lien de suppression sera visible.
- en [5] : le texte de ce lien

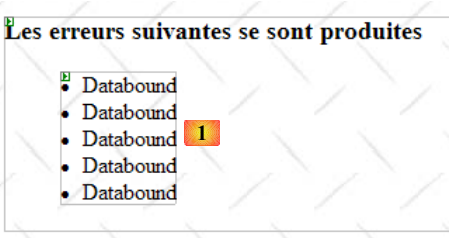
11.5.6 La vue [SimulationsVides]

Le composant [View] nommé [VueSimulationsVides] contient simplement du texte :



11.5.7 La vue [Erreurs]

Le composant [View] nommé [VueErreurs] est le suivant :



N°	Type	Nom	Rôle
1	Repeater	RptErreurs	affiche une liste de messages d'erreur

Le composant [Repeater] permet de répéter un code ASP / HTML pour chaque objet d'une source de données, généralement une collection. Ce code est défini directement dans le code source ASP de la page :

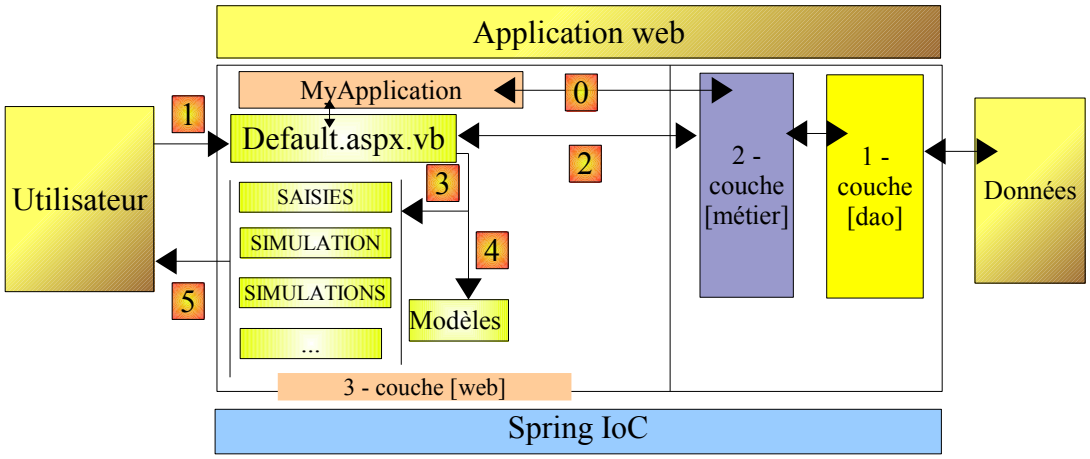
```
1.         <asp:Repeater ID="RptErreurs" runat="server">
2.           <ItemTemplate>
3.             <li>
4.               <%# Container.DataItem %>
5.             </li>
6.           </ItemTemplate>
7.  </asp:Repeater>
```

- ligne 2 : <ItemTemplate> définit le code qui sera répété pour chaque élément de la source de données.
- ligne 4 : affiche la valeur de l'expression *Container.DataItem* qui désigne l'élément courant de la source de données. Cet élément étant un objet, c'est la méthode *ToString* de cet objet qui est utilisée pour inclure celui-ci dans le flux HTML de la page. Notre collection d'objets sera une collection *List(Of String)* contenant des messages d'erreur. Les lignes 3-5 incluront des séquences *
Message
* dans le flux HTML de la page.

11.6 Le contrôleur [Default.aspx.vb]

11.6.1 Vue d'ensemble

Revenons à l'architecture MVC de l'application :



- [Default.aspx.vb] qui est le code de contrôle de la page unique [Default.aspx] est le contrôleur de l'application.
- [MyApplication] est l'objet de type [HttpApplication] qui initialise l'application et qui dispose d'une référence sur la couche [métier].

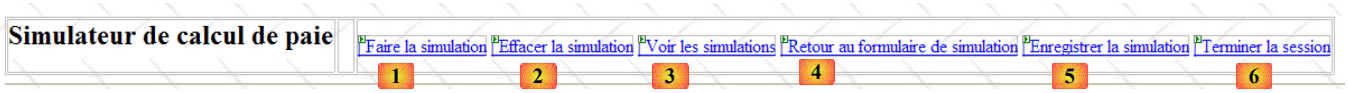
Le squelette du code du contrôleur est le suivant :

```
1.  Imports istia.st.pam.dao.entites
2.  ...
3.
4.  Partial Class _Default
```

```
5.   Inherits System.Web.UI.Page
6.
7.   Protected Sub SetVues(ByVal boolVues1 As Boolean, ByVal boolVues2 As Boolean, ByVal index As
   Integer)
8.     ' on affiche les vues demandées
9.     ' boolVues1 : true si le multivues Vues1 doit être visible
10.    ' boolVues2 : true si le multivues Vues2 doit être visible
11.    ' index : index de la vue de Vues2 à afficher
12. ...
13.  End Sub
14.
15.  Protected Sub SetMenu(ByVal boolFaireSimulation As Boolean, ByVal boolEnregistrerSimulation As
   Boolean, ByVal boolEffacerSimulation As Boolean, ByVal boolFormulaireSimulation As Boolean, ByVal
   boolVoirSimulations As Boolean, ByVal boolTerminerSession As Boolean)
16.    ' on fixe les options de menu
17.    ' chaque booléen est affecté à la propriété Visible du lien correspondant
18. ...
19.  End Sub
20.
21.
22.
23.  ' chargement de la page
24.  Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles MyBase.Load
25.    ' traitement requête initiale
26.    If Not IsPostBack Then
27. ...
28.    End If
29.  End Sub
30.
31.  Protected Sub LinkButtonFaireSimulation_Click(ByVal sender As Object, ByVal e As
   System.EventArgs) Handles LinkButtonFaireSimulation.Click
32. ...
33.  End Sub
34.
35.  Protected Sub LinkButtonEffacerSimulation_Click(ByVal sender As Object, ByVal e As
   System.EventArgs) Handles LinkButtonEffacerSimulation.Click
36. ...
37.  End Sub
38.
39.  Protected Sub LinkButtonVoirSimulations_Click(ByVal sender As Object, ByVal e As
   System.EventArgs) Handles LinkButtonVoirSimulations.Click
40. ...
41.  End Sub
42.
43.     Protected Sub GridViewSimulations_RowDeleting(ByVal sender As Object, ByVal e As
   System.Web.UI.WebControls.GridViewDeleteEventArgs) Handles GridViewSimulations.RowDeleting
44. ...
45.  End Sub
46.
47.  Protected Sub LinkButtonEnregistrerSimulation_Click(ByVal sender As Object, ByVal e As
   System.EventArgs) Handles LinkButtonEnregistrerSimulation.Click
48. ...
49.  End Sub
50.
51.  Protected Sub LinkButtonTerminerSession_Click(ByVal sender As Object, ByVal e As
   System.EventArgs) Handles LinkButtonTerminerSession.Click
52. ...
53.  End Sub
54.
55.  Protected Sub LinkButtonFormulaireSimulation_Click(ByVal sender As Object, ByVal e As
   System.EventArgs) Handles LinkButtonFormulaireSimulation.Click
56. ...
57.  End Sub
58. End Class
```

A la requête initiale (GET) de l'utilisateur, seul l'événement *Load* des lignes 24-29 est traité. Aux requêtes suivantes (POST) faites via les liens du menu, deux événements sont traités :

1. l'événement *Load* (24-29) mais le test du booléen *Page.IsPostback* fait que rien ne sera fait.
2. l'événement lié au lien qui a été cliqué :



- lignes 31-33 : traitent le clic sur le lien [1]
- lignes 35-37 : traitent le clic sur le lien [2]
- lignes 39-41 : traitent le clic sur le lien [3]
- lignes 55-57 : traitent le clic sur le lien [4]
- lignes 47-49 : traitent le clic sur le lien [5]
- lignes 51-53 : traitent le clic sur le lien [6]

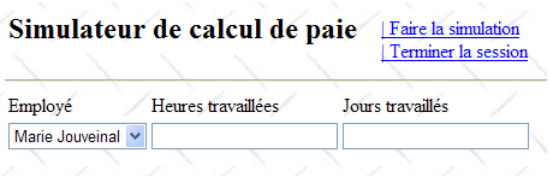
Pour factoriser des séquences de code revenant souvent, deux méthodes internes ont été créées :

- **SetVues**, lignes 7-13 : fixe la ou les vues à afficher
- **SetMenu**, lignes 15-19 : fixe les options de menu à afficher

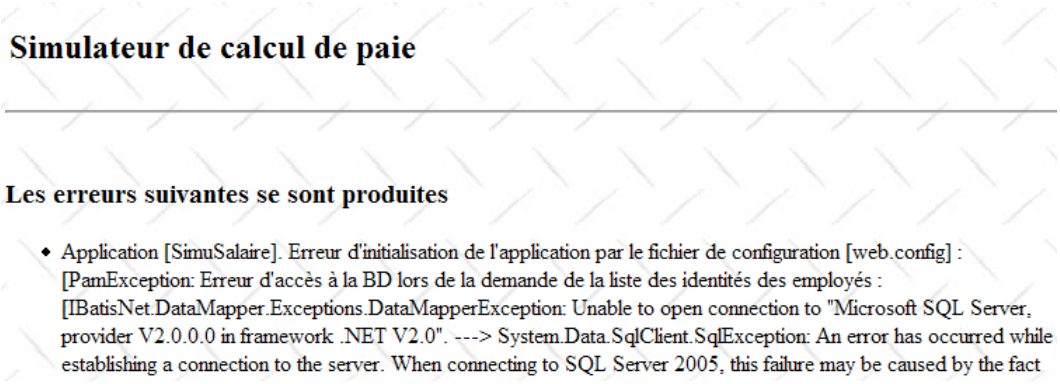
11.6.2 L'événement Load

Lectures conseillées : référence [1], **programmation ASP.NET** vol2 :
- paragraphe 2.8 : Composant [Repeater] et liaison de données.

La première vue présentée à l'utilisateur est celle du formulaire vide :



L'initialisation de l'application nécessite l'accès à une source de données qui peut échouer. Dans ce cas, la première page est une page d'erreurs :



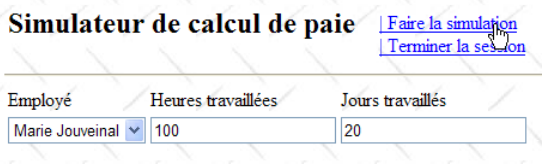
L'événement [Load] est traité de façon analogue à celle des versions ASP.NET précédentes :

```
1.  ' chargement de la page
2.  Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles MyBase.Load
3.    ' traitement requête initiale
4.    If Not IsPostBack Then
5.      ' des erreurs d'initialisation ?
6.      If MyApplication.Erreur Then
7.        ' affichage vue [erreurs]
8....
9.        ' positionnement menu
10....
11.       Exit Sub
12.     End If
13.     ' chargement des noms des employés dans le combo
14....
15.     ' positionnement menu
16....
17.     ' affichage vue [saisie]
18....
19.   End If
20.End Sub
```

Question : compléter le code ci-dessus

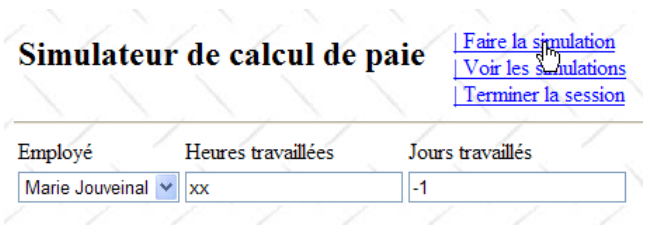
11.6.3 Action : Faire la simulation

Dans ce qui suit, l'écran noté (1) est celui de la demande de l'utilisateur, l'écran noté (2) la réponse qui lui est envoyée par l'application web. A partir de l'écran d'accueil, l'utilisateur peut commencer une simulation :

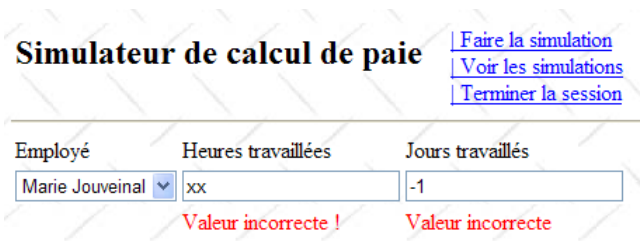


- (1) : l'utilisateur demande une simulation

- (2) : résultat de la simulation



- (1) : on entre des données erronées et on demande la simulation



- (2) : les erreurs ont été signalées

La procédure qui traite cette action pourrait ressembler à ce qui suit :

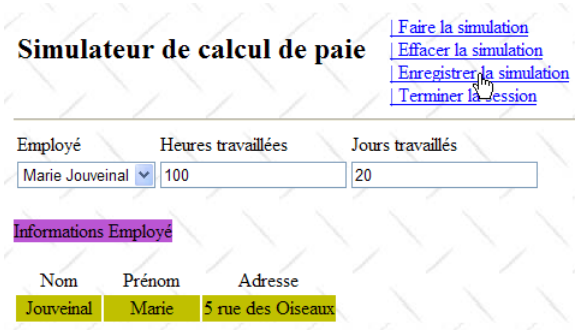
```
1.Protected Sub LinkButtonFaireSimulation_Click(ByVal sender As Object, ByVal e As System.EventArgs)
  Handles LinkButtonFaireSimulation.Click
2.    ' calcul du salaire
3.    ' page valide ?
4.    Page.Validate()
5.    If Not Page.IsValid Then
6.      ' affichage vue [saisie]
7....
8.    End If
9.    ' la page est valide - on calcule le salaire de l'employé
10....    Dim Feuillesalaire As FeuilleSalaire
11.   Try
12....
13.   Catch ex As PamException
14.     ' affichage vue [erreurs]
15....
16.     Exit Sub
17.   End Try
18.   ' on met le résultat dans la session
19....
```

```
20.   ' affichage résultats
21....
22.   ' affichage vue [Simulation]
23....
24.   ' affichage menu
25....
26. End Sub
```

Question : compléter le code ci-dessus

11.6.4 Action : Enregistrer la simulation

Une fois la simulation faite, l'utilisateur peut demander son enregistrement :



- (1) : demande d'enregistrement de la simulation courante

- (2) : la simulation est enregistrée et la liste des simulations faites est présentée

La procédure qui traite cette action pourrait ressembler à ce qui suit :

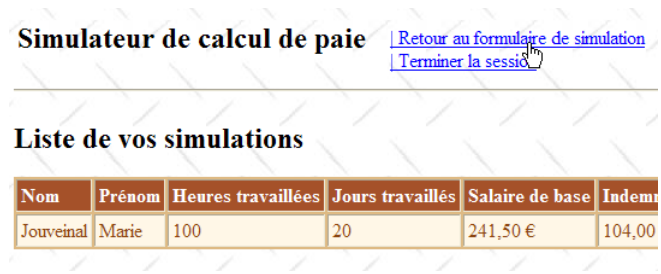
```
1.Protected Sub LinkButtonEnregistrerSimulation_Click(ByVal sender As Object, ByVal e As System.EventArgs)
  Handles LinkButtonEnregistrerSimulation.Click
2.    ' on enregistre la simulation courante dans la liste des simulations présente dans la session
3...
4.    ' on affiche la vue [simulations]
5...
6.  End Sub
```

Question : compléter le code ci-dessus

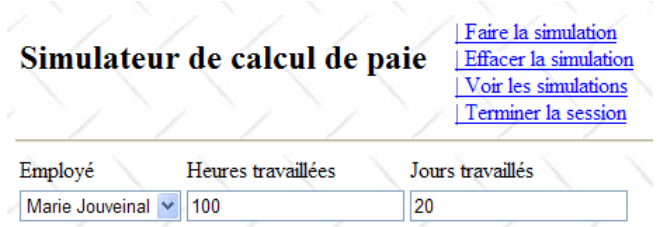
11.6.5 Action : Retour au formulaire de simulation

Lectures conseillées : référence [1], **programmation ASP.NET** vol2 :
- paragraphe 1.6.3 : Le rôle du champ caché _VIEWSTATE

Une fois la liste des simulations présentée, l'utilisateur peut demander à revenir au formulaire de simulation :



- (1) : retour au formulaire de simulation



- (2) : le formulaire de simulation tel qu'il a été saisi initialement

On notera que l'écran (2) présente le formulaire tel qu'il a été saisi. Il faut se rappeler ici que ces différentes vues appartiennent à une même page. Entre les différentes requêtes, les valeurs des composants sont maintenues par le mécanisme du *ViewState* si ces composants ont leur propriété *EnableViewState* à *true*.

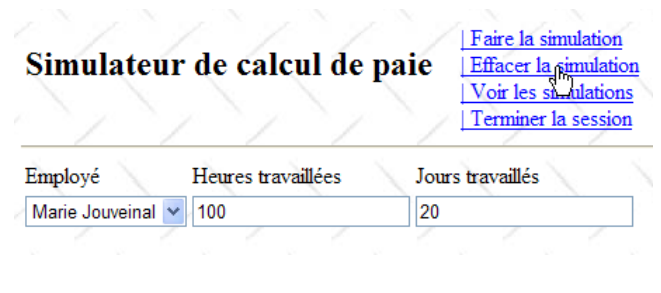
La procédure qui traite cette action pourrait ressembler à ce qui suit :

```
1.Protected Sub LinkButtonFormulaireSimulation_Click(ByVal sender As Object, ByVal e As System.EventArgs)
  Handles LinkButtonFormulaireSimulation.Click
2.    ' affichage vue [saisie]
3...
4.    ' positionnement menu
5....
6.  End Sub
```

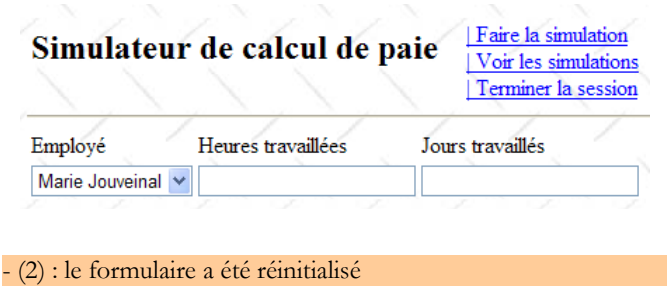
Question : compléter le code ci-dessus

11.6.6 Action : Effacer la simulation

Une fois revenu au formulaire de simulation, l'utilisateur peut demander à effacer les saisies présentes :



- (1) : on efface les données du formulaire



- (2) : le formulaire a été réinitialisé

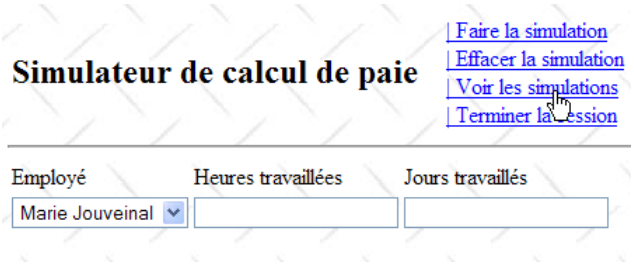
La procédure qui traite cette action pourrait ressembler à ce qui suit :

```
1.Protected Sub LinkButtonEffacerSimulation_Click(ByVal sender As Object, ByVal e As System.EventArgs)
  Handles LinkButtonEffacerSimulation.Click
2.    ' RAZ du formulaire
3....
4.  End Sub
```

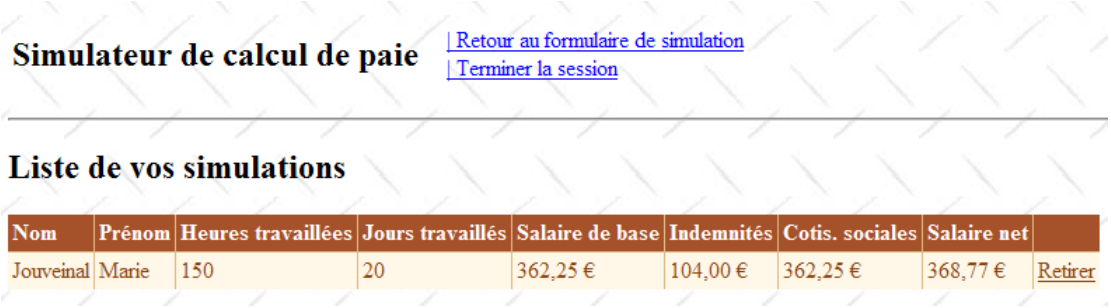
Question : compléter le code ci-dessus

11.6.7 Action : Voir les simulations

L'utilisateur peut demander à voir les simulations qu'il a déjà faites :



- (1) : on demande à voir les simulations



- (2) : la liste des simulations

La procédure qui traite cette action pourrait ressembler à ce qui suit :

```
1.Protected Sub LinkButtonVoirSimulations_Click(ByVal sender As Object, ByVal e As System.EventArgs)
  Handles LinkButtonVoirSimulations.Click
2.    ' on récupère les simulations dans la session
3....
4.    ' y-a-t-il des simulations ?
5.    If ... Then
6.      ' vue [simulations]
7....
8.    Else
9.      ' vue [SimulationsVides]
10....
11.   End If
12.   ' on fixe le menu
13....
14. End Sub
```

Question : compléter le code ci-dessus

11.6.8 Action : Supprimer une simulation

L'utilisateur peut demander à supprimer une simulation :

- (1) : on peut retirer des simulations de la liste

- (2) : la simulation a été retirée

La procédure [GridViewSimulations_RowDeleting] qui traite cette action pourrait ressembler à ce qui suit :

```
1.      Protected Sub GridViewSimulations_RowDeleting(ByVal sender As Object, ByVal e As
    System.Web.UI.WebControls.GridViewDeleteEventArgs) Handles GridViewSimulations.RowDeleting
2.          ' on récupère les simulations dans la session
3.          ...
4.          ' on supprime la simulation désignée par l'argument e passé en paramètre à la procédure
5.          ' e.RowIndex représente le n° de la ligne du [GridView] où le lien [Retirer] a été cliqué
6.          ...
7.          ' reste-t-il des simulations ?
8.          If ... Then
9.              ' on remplit le datagrid
10.             ...
11.         Else
12.             ' vue [SimulationsVides]
13.             ...
14. End If
```

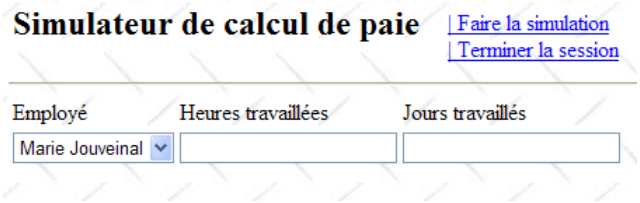
Question : compléter le code ci-dessus

11.6.9 Action : Terminer la session

L'utilisateur peut demander à terminer sa session de simulations. Cela abandonne le contenu de sa session et présente un formulaire vide :

- (1) : on invalide la session courante

- (2) : on revient à la page d'accueil



La procédure qui traite cette action pourrait ressembler à ce qui suit :

```
1.Protected Sub LinkButtonTerminerSession_Click(ByVal sender As Object, ByVal e As System.EventArgs)
  Handles LinkButtonTerminerSession.Click
2.    ' on abandonne la session
3.    Session.Abandon()
4.    ' RAZ formulaire des saisies
5....
6.    ' afficher la vue [saisies]
7....
8.    ' positionnement menu
9....
```

```
10. End Sub
```

Question : compléter le code ci-dessus

Travail pratique : mettre en oeuvre sur machine l'application web précédente

11.7 Ajaxification de l'application

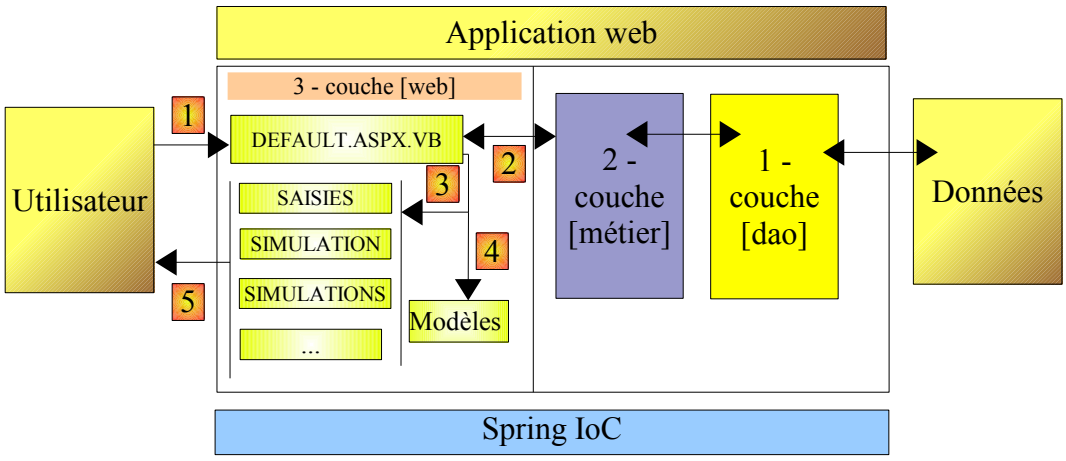
En suivant la démarche décrite pour la version 8 précédente, ajaxifiez le formulaire [Default.aspx].

12 L'application [SimuPaie] – version 10 – ASP.NET / multi-vues / multi-pages

Lectures conseillées : référence [1], **programmation ASP.NET vol1**, paragraphe 5 : **Exemples**

Nous étudions maintenant une version fonctionnellement identique à l'application ASP.NET à trois couches étudiée précédemment mais nous modifions l'architecture de cette dernière de la façon suivante : là où dans la version précédente, les vues étaient implémentées par une unique page ASPX, ici elles seront implémentées par trois pages ASPX.

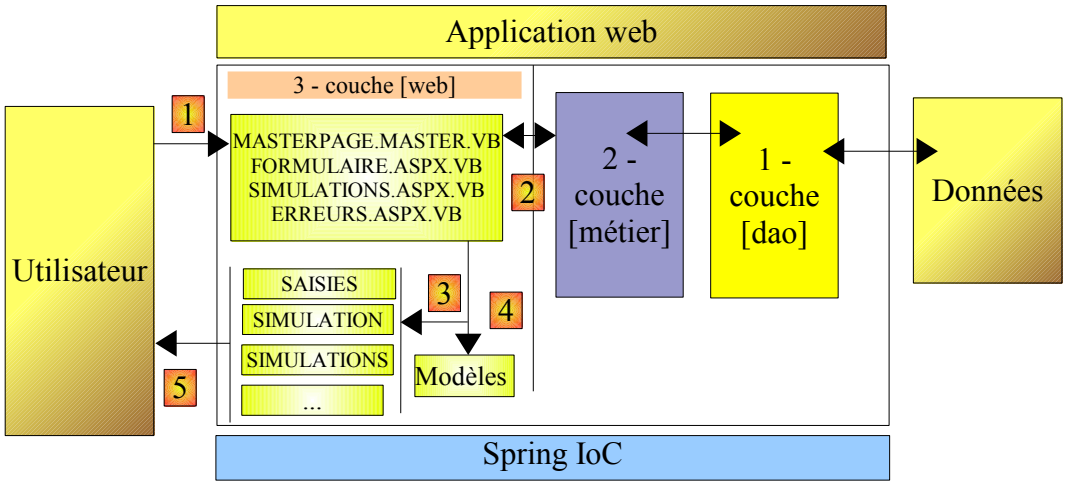
L'architecture de l'application précédente était la suivante :



On a ici une architecture MVC (Modèle – Vue – Contrôleur) :

- [Default.aspx.vb] contient le code du **contrôleur**. La page [Default.aspx] est l'unique interlocuteur du client. Elle voit passer toutes les requêtes de celui-ci.
- [Saisies, Simulation, Simulations, ...] sont les **vues**. Ces vues sont implémentées ici, par des composants [View] de la page [Default.aspx].

L'architecture de la nouvelle version sera elle la suivante :



- seule la couche [web] évolue
- les vues (ce qui est présenté à l'utilisateur) ne changent pas.
- le code du contrôleur, qui était dans la version précédente, tout entier dans [Default.aspx.vb] est désormais réparti sur plusieurs pages :
  - [MasterPage.master] : une page qui factorise ce qui est commun aux différentes vues : le bandeau supérieur avec ses options de menu
  - [Formulaire.aspx] : la page qui présente le formulaire de simulation et gère les actions qui ont lieu sur ce formulaire

- [Simulations.aspx] : la page qui présente la liste des simulations et gère les actions qui ont lieu sur cette même page
- [Erreurs.aspx] : la page qui est affichée lors d'une erreur d'initialisation de l'application. Il n'y a pas d'actions possibles sur cette page.

On peut considérer qu'on a là une architecture MVC à contrôleurs multiples alors que l'architecture de la version précédente était une architecture MVC à contrôleur unique.

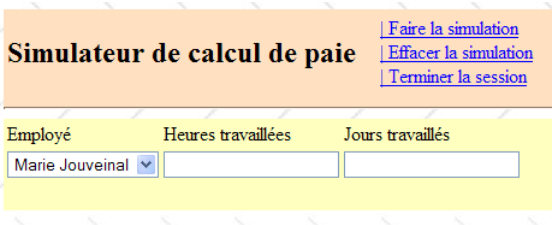
Le traitement d'une demande d'un client se déroule selon les étapes suivantes :

A) le client fait une demande à l'application. Il la fait à l'une des deux pages [Formulaire.aspx, Simulations.aspx].
B) la page demandée traite cette demande. Pour ce faire, elle peut avoir besoin de l'aide de la couche [métier] qui elle-même peut avoir besoin de la couche [dao] si des données doivent être échangées avec la base de données. L'application reçoit une réponse de la couche [métier].
C) selon celle-ci, elle choisit (3) la vue (= la réponse) à envoyer au client en lui fournissant (4) les informations (le modèle) dont elle a besoin.
D) la réponse est envoyée au client (5)

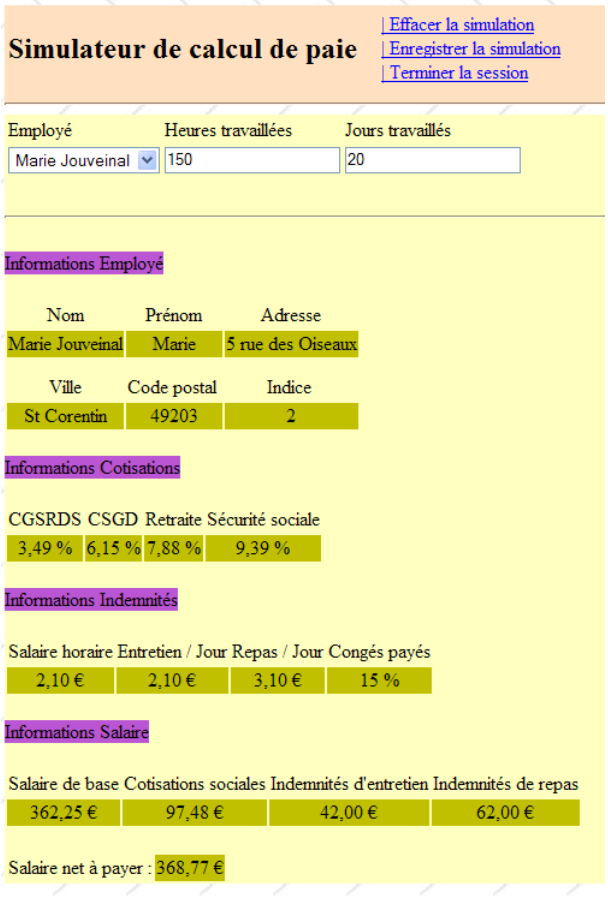
12.1 Les vues de l'application

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

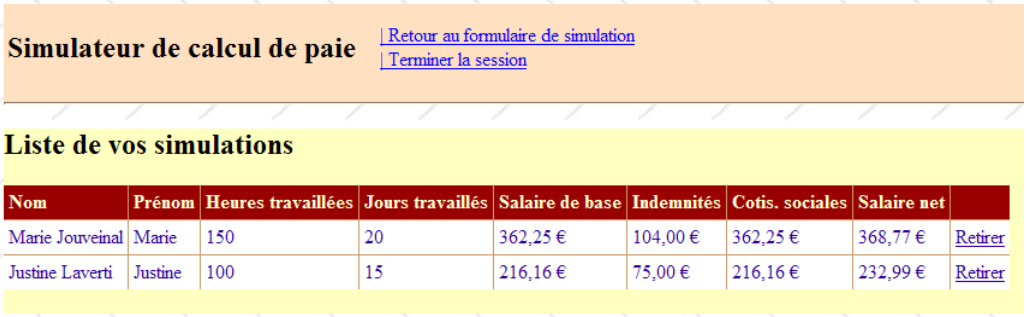
- la vue [VueSaisies] qui présente le formulaire de simulation



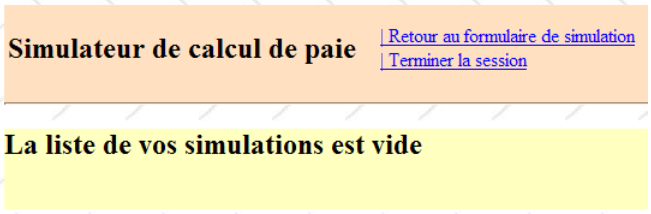
- la vue [VueSimulation] utilisée pour afficher le résultat détaillé de la simulation :



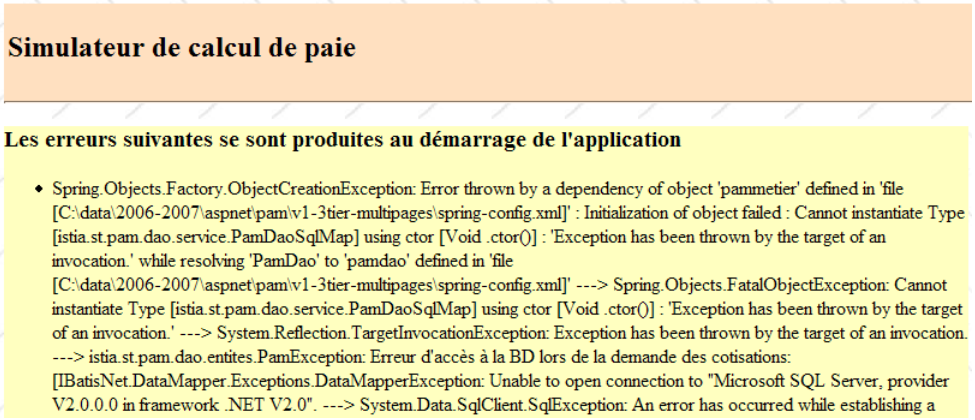
- la vue [VueSimulations] qui donne la liste des simulations faites par le client



- la vue [VueSimulationsVides] qui indique que le client n'a pas ou plus de simulations :



– la vue [VueErreurs] qui indique une erreur d'initialisation de l'application :



12.2 Génération des vues dans un contexte multi-contrôleurs

Dans la version précédente, toutes les vues étaient générées à partir de l'unique page [Default.aspx]. Celle-ci contenait deux composants [MultiView] et les vues étaient composées d'une réunion d'un composant [View] de chacun de ces deux composants.

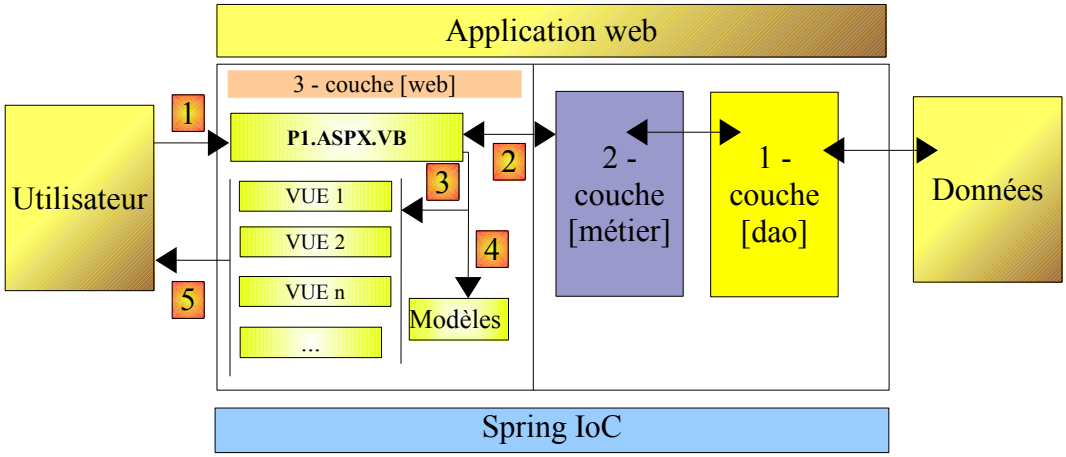
Efficace lorsqu'il y a peu de vues, cette architecture atteint ses limites dès que le nombre des composants formant les différentes vues devient important : en effet, à chaque requête qui est faite à l'unique page [Default.aspx], tous les composants de celle-ci sont instanciés alors même que seuls certains d'entre-eux vont être utilisés pour générer la réponse à l'utilisateur. Un travail inutile est alors fait à chaque nouvelle requête, travail qui devient pénalisant lorsque le nombre total de composants de la page est important.

Une solution est alors de répartir les vues sur différentes pages. C'est ce que nous faisons ici. Etudions deux cas différents de génération de vues :

1. la requête est faite à une page P1 et celle-ci génère la réponse
2. la requête est faite à une page P1 et celle-ci demande à une page P2 de générer la réponse

12.2.1 Cas 1 : une page contrôleur / vue

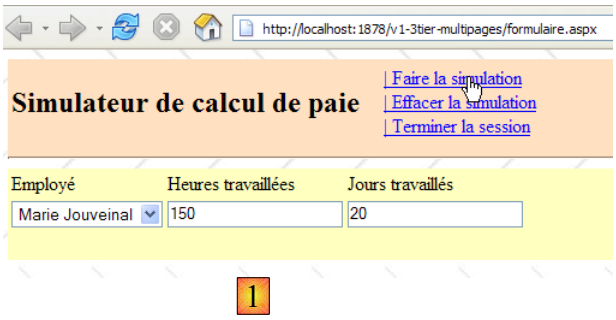
Dans le cas 1, on retombe sur l'architecture mono-contrôleur de la version précédente, où la page [Default.aspx] est la page P1 :



A) le client fait une demande à la page P1 (1)
B) la page P1 traite cette demande. Pour ce faire, elle peut avoir besoin de l'aide de la couche [métier] (2) qui elle-même peut avoir besoin de la couche [dao] si des données doivent être échangées avec la base de données. L'application reçoit une réponse de la couche [métier].
C) selon celle-ci, elle choisit (3) la vue (= la réponse) à envoyer au client en lui fournissant (4) les informations (le modèle) dont elle a besoin. Il s'agit ici, de choisir dans la page P1 les composants [Panel] ou [View] à afficher et d'initialiser les composants qu'ils contiennent.
D) la réponse est envoyée au client (5)

Voici deux exemples pris dans l'application étudiée :

[**page Formulaire.aspx**]



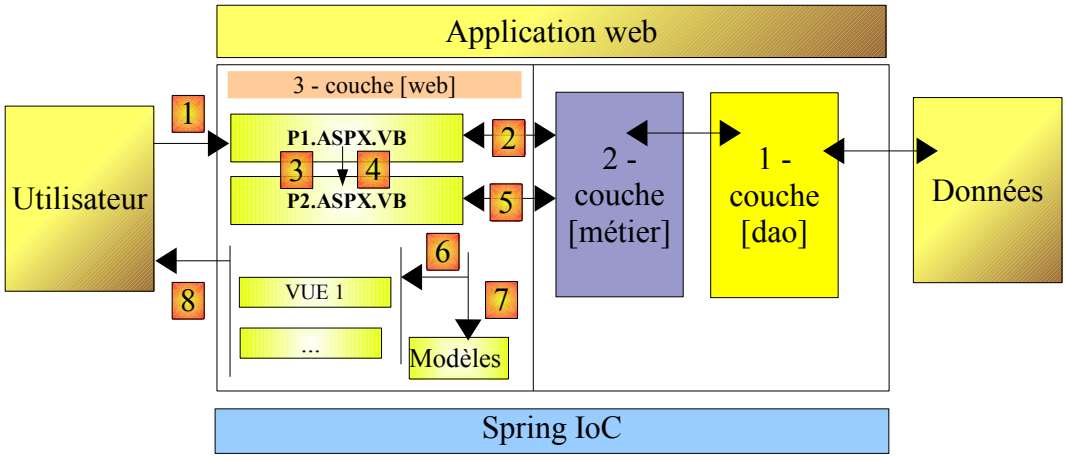
- en [1] : l'utilisateur, après avoir demandé la page [Formulaire.aspx], demande une simulation
- en [2] : la page [Formulaire.aspx] a traité cette demande et généré elle-même la réponse en affichant un composant [View] qui n'avait pas été affiché en [1]

[**page Simulations.aspx**]

- en [1] : l'utilisateur, après avoir demandé la page [Simulations.aspx], veut retirer une simulation
- en [2] : la page [Simulations.aspx] a traité cette demande et généré elle-même la réponse en réaffichant la nouvelle liste de simulations.

12.2.2 Cas 2 : une page 1 contrôleur, une page 2 controleur / vue

Le cas 2 peut recouvrir diverses d'architectures. Nous choisirons la suivante :

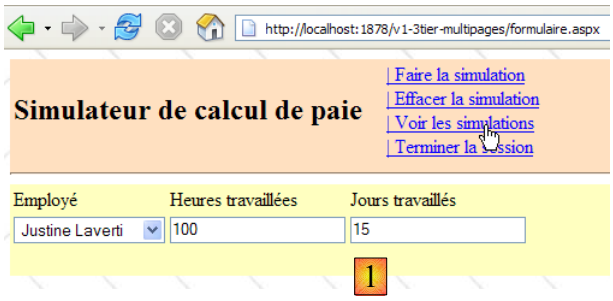


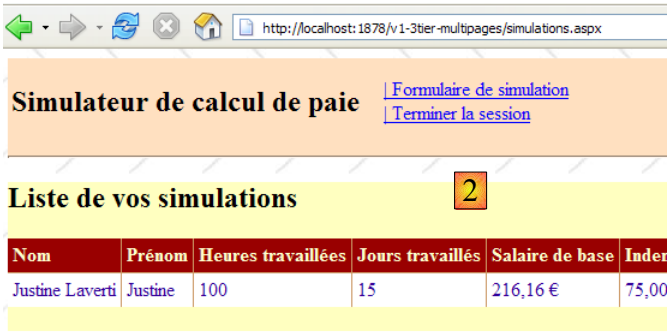
A) le client fait une demande à la page P1 (1)
B) la page P1 traite cette demande. Pour ce faire, elle peut avoir besoin de l'aide de la couche [métier] (2) qui elle-même peut avoir besoin de la couche [dao] si des données doivent être échangées avec la base de données. L'application reçoit une réponse de la couche [métier].
C) selon celle-ci, elle choisit (3) la vue (= la réponse) à envoyer au client en lui fournissant (4) les informations (le modèle) dont elle a besoin. Il se trouve qu'ici, la vue à générer doit l'être par une autre page que P1, la page P2. Pour faire les opérations (3) et (4), la page P1 a deux possibilités :

- ✗ **faire un transfert d'exécution à la page P2** par l'opération [**Server.Transfer(" P2.aspx ")**]. Dans ce cas, elle peut mettre le modèle destiné à la page P2 dans le contexte de la requête [*Context.Items(" clé ")=valeur*] ou dans la session de l'utilisateur [*Session.Items(" clé ")=valeur*]. La page P2 sera alors instanciée et lors du traitement de son événement *Load* par exemple, elle pourra récupérer les informations transmises par la page P1 par les opérations [*valeur=CType(Context.Items(" clé "), Type*] ou bien [*valeur=CType(Session.Items(" clé "), Type*] selon les cas, où *Type* est le type de la valeur associée à la clé.. La transmission de valeurs par le contexte *Context* est la plus appropriée s'il n'y a pas utilité à ce que les valeurs du modèle soient conservées pour une future requête du client.
- ✗ demander au client de **se rediriger vers la page P2** par l'opération [**Response.Redirect(" P2.aspx ")**]. Dans ce cas, la page P1 mettra le modèle destiné à la page P2 dans la session, car le contexte *Context* de requête est supprimé à la fin de chaque requête. Or ici, la redirection va provoquer la fin de la 1ère requête du client vers P1 et l'émission d'une seconde requête de ce même client, vers P2 cette fois. Il y a deux requêtes successives. On sait que la session est l'un des moyens de conserver de la " mémoire " entre requêtes. Il y a d'autres solutions que la session.

D) quelque soit la façon dont P2 prend la main, on retombe ensuite dans le cas 1 : P2 a reçu une requête qu'elle va traiter (5) et elle va générer elle-même la réponse (6, 7). On peut aussi imaginer que la page P2 va après traitement de la requête, passer la main à une page P3, et ainsi de suite.

Voici un exemple pris dans l'application étudiée :

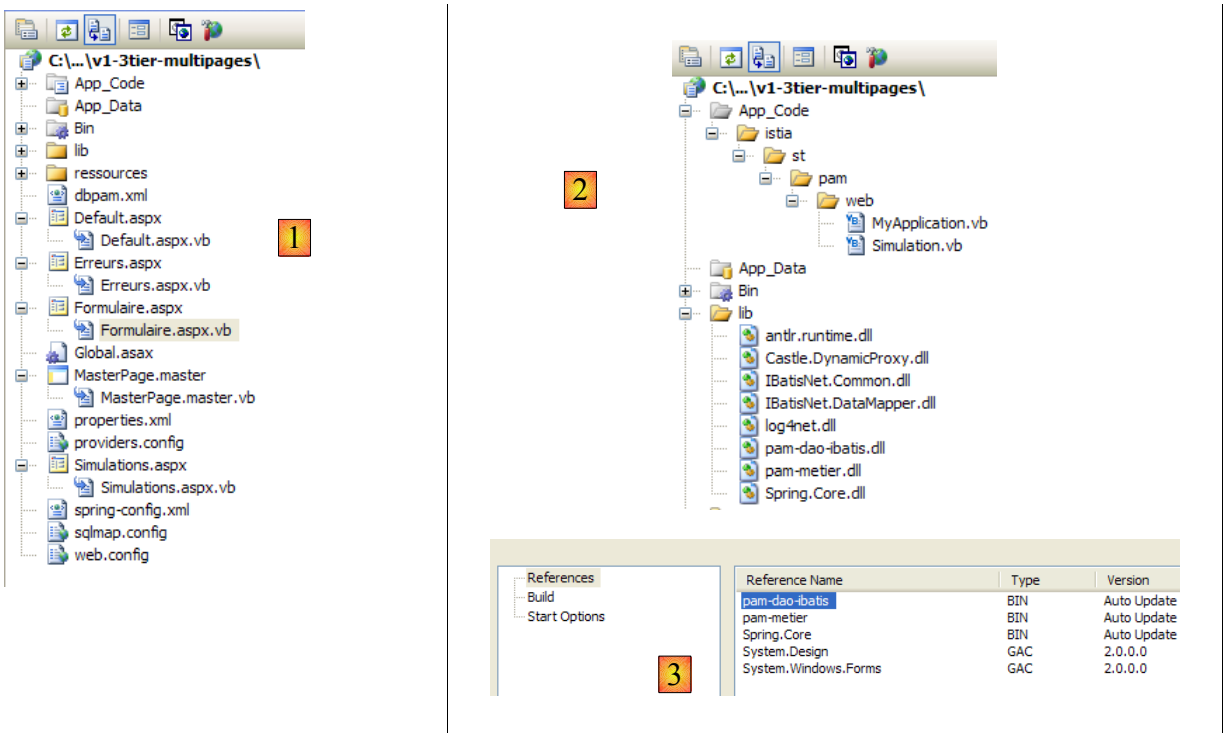




- en [1] : l'utilisateur qui a demandé la page [Formulaire.aspx] demande à voir la liste des simulations
- en [2] : la page [Formulaire.aspx] traite cette demande et redirige le client vers la page [Simulations.aspx]. C'est cette dernière qui fournit la réponse à l'utilisateur. Au lieu de demander au client de se rediriger, la page [Formulaire.aspx] aurait pu transférer la demande du client vers la page [Simulations.aspx]. Dans ce cas en [2], on aurait vu la même Url qu'en [1]. En effet, un navigateur affiche toujours la dernière Url demandée :
  - l'action demandée en [1] est destinée à la page [Formulaire.aspx]. Le navigateur fait un POST vers cette page.
  - si la page [Formulaire.aspx] traite la demande puis la transfère par [Server.Transfer(" Simulations.aspx ")] à la page [Simulations.aspx], on reste dans la même requête. Le navigateur affichera alors en [2], l'Url de [Formulaire.aspx] vers qui a eu lieu le POST.
  - si la page [Formulaire.aspx] traite la demande puis la redirige par [Response.Redirect(" Simulations.aspx ")] vers la page [Simulations.aspx], le navigateur fait alors une 2ième requête, un GET vers [Simulations.aspx]. Le navigateur affichera alors en [2], l'Url de [Simulations.aspx] vers qui a eu lieu le GET. C'est ce que la copie d'écran [2] ci-dessus nous montre.

12.3 Le projet Visual Web Developer de la couche [web]

Le projet Visual Web Developer de la couche [web] est le suivant :

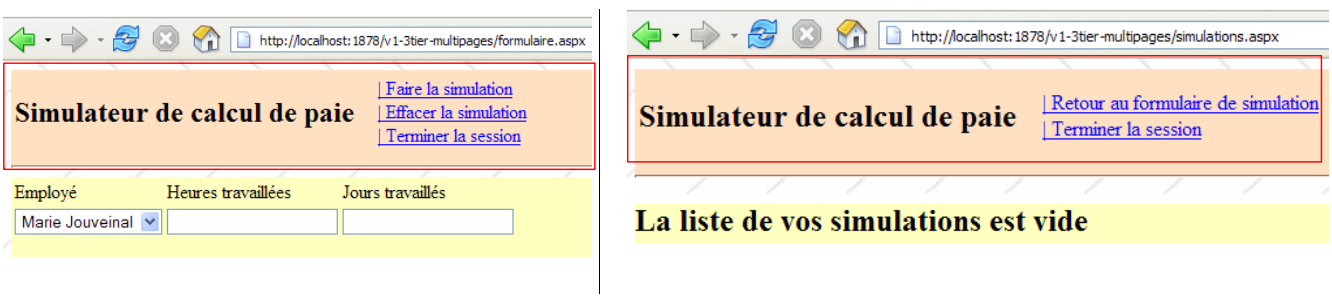


- en [1] on trouve :
  - le fichier de configuration [web.config] de l'application – est identique à celui de l'application précédente.
  - les fichiers de configuration des différentes couches de l'application web : [spring-config.xml, sqlmap.config, providers.config, properties.xml, dbpam.xml] – sont identiques aux fichiers de mêmes noms de l'application précédente
  - la page [Default.aspx] – se contente de rediriger le client vers la page [Formulaire.aspx]
  - la page [Formulaire.aspx] qui présente à l'utilisateur le formulaire de simulation et traite les actions liées à ce formulaire
  - la page [Simulations.aspx] qui présente à l'utilisateur la liste de ses simulations et traite les actions liées à cette page
  - la page [Erreurs.aspx] qui présente à l'utilisateur une page signalant une erreur rencontrée au démarrage de l'application web.
- en [2] on trouve :
  - le fichier [MyApplication.vb] qui gère les événements de l'application web et le fichier [Simulation.vb] de la classe [Simulation]. Ces deux fichiers sont ceux de l'application précédente.
  - le dossier [lib] dans lequel on a mis toutes les DLL nécessaires au projet – son contenu est identique au dossier [lib] de l'application précédente.
- en [3] on voit les références du projet.

12.4 Le code de présentation des pages

12.4.1 La page Maître [MasterPage.master]

Les vues de l'application présentées au paragraphe 12.1, page 103, ont des parties communes qu'on peut factoriser dans une page Maître, appelée la **Master Page** dans Visual Studio. Prenons par exemple, les vues [VueSaisies] et [VueSimulationsVides] ci-dessous, générées respectivement par les pages [Formulaire.aspx] et [Simulations.aspx] :

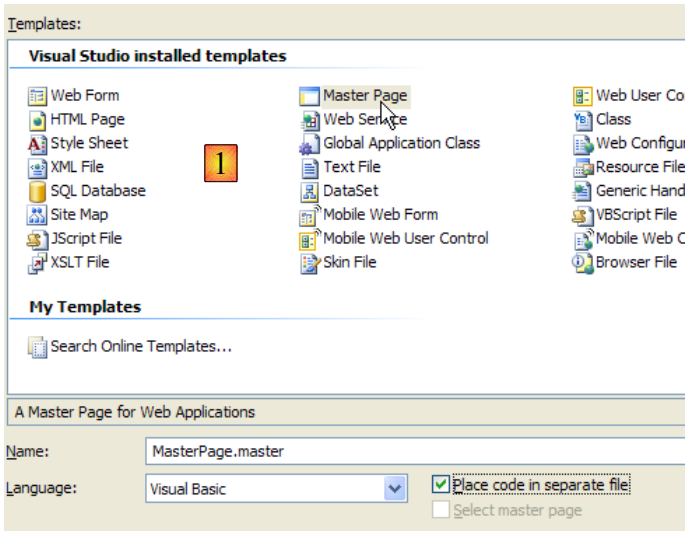


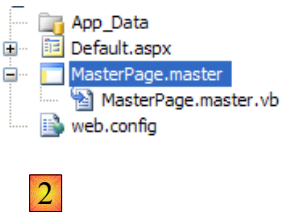
Ces deux vues possèdent en commun, le bandeau supérieur (Titre et Options de menu). Il en est ainsi de toutes les vues qui seront présentées à l'utilisateur : elles auront toutes le même bandeau supérieur. Pour que différentes pages partagent un même fragment de présentation, il existe diverses solutions dont les suivantes :

- mettre ce fragment commun dans un composant utilisateur. C'était la principale technique avec ASP.NET 1.1
- mettre ce fragment commun dans une page Maître. Cette technique est apparue avec ASP.NET 2.0. C'est celle que nous utilisons ici.

Pour créer une page Maître dans une application web, on peut procéder ainsi :

- clic droit sur le projet/ Add New Item / Master Page :





L'ajout d'une page Maître ajoute par défaut deux fichiers à l'application web :

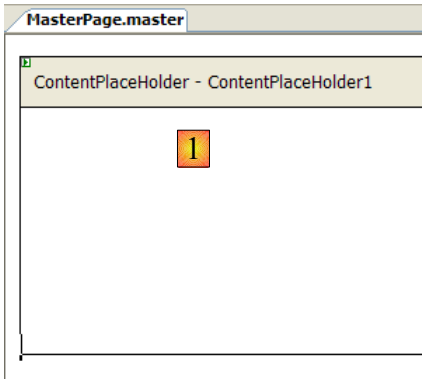
- [MasterPage.master] : le code de présentation de la page Maître
- [MasterPage.master.vb] : le code de contrôle de la page Maître

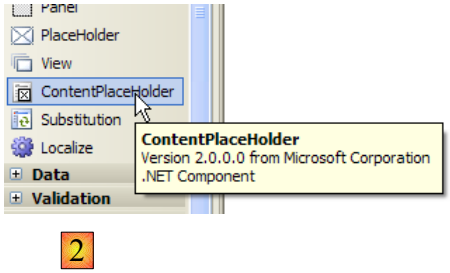
Le code généré par Visual Studio dans [MasterPage.master] est le suivant :

```
1.  <%@ Master Language="VB" CodeFile="MasterPage.master.vb" Inherits="MasterPage" %>
2.
3.  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4.
5.  <html xmlns="http://www.w3.org/1999/xhtml" >
6.  <head runat="server">
7.      <title>Untitled Page</title>
8.  </head>
9.  <body>
10.     <form id="form1" runat="server">
11.     <div>
12.         <asp:contentplaceholder id="ContentPlaceHolder1" runat="server">
13.         </asp:contentplaceholder>
14.     </div>
15.     </form>
16. </body>
17. </html>
```

- ligne 1 : la balise <%@ Master ... %> sert à définir la page comme une page Maître. Le code de contrôle de la page sera dans le fichier défini par l'attribut *CodeFile*, et la page héritera de la classe définie par l'attribut *Inherits*. Cette balise est générée par Visual Studio ainsi que les lignes suivantes 2 à 6. La personnalisation du fichier commencera en ligne 7.
- ligne 10 : le formulaire de la page Maître
- lignes 12-13 : un conteneur vide qui contiendra dans notre application, l'une des pages [Formulaire.aspx, Simulations.aspx, Erreurs.aspx]. Le client reçoit en réponse, toujours la même page, la page Maître, dans laquelle le conteneur [ContentPlaceHolder1] va recevoir un flux HTML fourni par l'une des pages [Formulaire.aspx, Simulations.aspx, Erreurs.aspx]. Ainsi pour changer l'aspect des pages envoyées aux clients, il suffit de changer l'aspect de la page Maître.

La représentation visuelle (onglet Design) de ce code source est présentée en (1) ci-dessous. Par ailleurs, il est possible d'ajouter autant de conteneurs que souhaité, grâce au composant [ContentPlaceHolder] (2) de la barre d'outils [Standard].



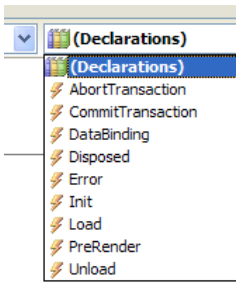


Le code de contrôle généré par Visual Studio dans [MasterPage.master.vb] est le suivant :

```
1.  Partial Class MasterPage
2.      Inherits System.Web.UI.MasterPage
3.  End Class
```

- ligne 1 : la classe référencée par l'attribut [Inherits] de la directive <%@ Master ... %> de la page [MasterPage.master]
- ligne 2 : cette classe dérive de la classe [System.Web.UI.MasterPage]

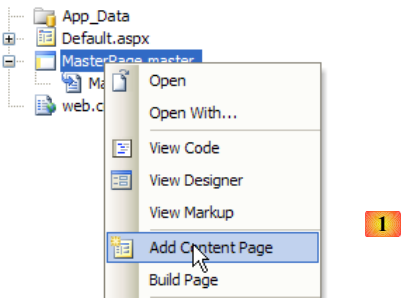
Cette classe permet de gérer les événements de la page [MasterPage.master], ceux classiques d'une page web :

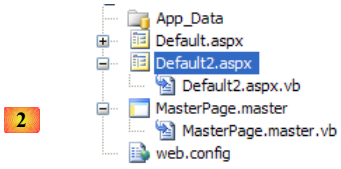


et ceux des composants qui seront placés sur la page, en-dehors des conteneurs de pages. Pour ces derniers, c'est la page encapsulée dans le conteneur qui répondra aux événements de ses propres composants. Nous y reviendrons.

Dans le code de contrôle d'une page, l'événement *Load* est souvent géré. Ici, nous aurons une page maître qui contiendra en son sein une autre page. Dans quel ordre se produisent les événements *Load* des deux pages ? Il s'agit d'une règle générale : l'événement *Load* d'un composant se produit avant celui de son conteneur. Ici, l'événement *Load* de la page insérée dans la page maître aura donc lieu avant celui de la page Maître elle-même.

Pour générer une page qui a pour page maître la page [MasterPage.master] précédente, on pourra procéder comme suit :





- en [1] : clic droit sur la page maître puis option [Add Content Page]
- en [2] : une page par défaut, ici [Default2.aspx] est générée.

Le code de présentation [Default2.aspx] est le suivant :

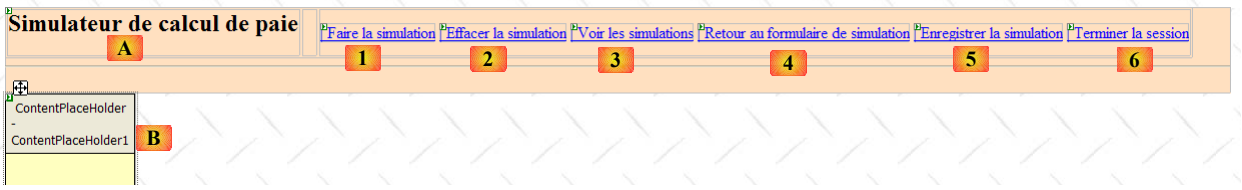
```
1.  <%@ Page Language="VB" MasterPageFile="~/MasterPage.master" AutoEventWireup="false"
    CodeFile="Default2.aspx.vb" Inherits="Default2" title="Untitled Page" %>
```

```
2.  <asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" Runat="Server">
3.  </asp:Content>
```

- ligne 1 : la directive **Page** et ses attributs
  - *MasterPageFile* : désigne le fichier de la page maître de la page décrite par la directive. Le signe ~ désigne le dossier du projet.
  - les autres paramètres sont ceux habituels d'une page web ASP
- lignes 2-3 : les balises <asp:Content> sont reliées une à une aux directives <asp:ContentPlaceHolder> de la page maître via l'attribut *ContentPlaceHolderID*. Les composants placés entre les lignes 2-3 ci-dessus, seront à l'exécution, placés dans le conteneur *ContentPlaceHolder1* de la page maître puis celle-ci sera affichée.

En renommant la page [Default2.aspx] ainsi générée, on peut construire les différentes pages ayant [MasterPage.master] comme page maître.

Pour notre application [SimuPaie], l'aspect visuel de la page Maître sera la suivante :



N°	Type	Nom	Rôle
A	Panel (rose ci-dessus)	entete	entête de la page
B	Panel (jaune ci-dessus)	contenu	contenu de la page
1	LinkButton	LinkButtonFaireSimulation	demande le calcul de la simulation
2	LinkButton	LinkButtonEffacerSimulation	efface le formulaire de saisie
3	LinkButton	LinkButtonVoirSimulations	affiche la liste des simulations déjà faites
4	LinkButton	LinkButtonFormulaireSimulation	ramène au formulaire de saisie
5	LinkButton	LinkButtonEnregistrerSimulation	enregistre la simulation courante dans la liste des simulations
6	LinkButton	LinkButtonTerminerSession	abandonne la session courante

Le code source correspondant est le suivant :

```
1.  <%@ Master Language="VB" CodeFile="MasterPage.master.vb" Inherits="MyMasterPage" %>
2.  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
3.  <html xmlns="http://www.w3.org/1999/xhtml">
4.  <head runat="server">
5.    <title>Untitled Page</title>
6.  </head>
7.  <body background="ressources/standard.jpg">
8.    <form id="form1" runat="server">
9.      <asp:Panel ID="entete" runat="server" BackColor="#FFE0C0">
10.       <table>
11.         <tr>
12.           <td>
13.             <h2>
14.               Simulateur de calcul de paie</h2>
15.           </td>
16.           <td>
17.             <label>
18.                  </label></td>
19.           <td>
20.             <asp:LinkButton ID="LinkButtonFaireSimulation" runat="server" CausesValidation="False">|
    Faire la simulation<br /></asp:LinkButton>
21. ...
22.             <asp:LinkButton ID="LinkButtonTerminerSession" runat="server" CausesValidation="False">|
    Terminer la session<br /></asp:LinkButton>
23.           </td>
24.       </table>
25.       <hr>
26.     </asp:Panel>
27.     <div>
28.       <asp:Panel ID="contenu" runat="server" BackColor="#FFFFC0">
29.         <asp:ContentPlaceHolder ID="ContentPlaceHolder1" runat="server">
```

```
30.         </asp:ContentPlaceHolder>
31.       </asp:Panel>
32.     </div>
33.   </form>
34. </body>
35. </html>
```

- ligne 1 : on notera le nom de la classe de la page maître : **MyMasterPage**
- ligne 7 : on définit une image de fond pour la page.
- lignes 9-26 : le composant Panel [entete]
- lignes 28-31 : le composant Panel [contenu]
- lignes 29-30 : le composant d'ID [ContentPlaceHolder1] qui contiendra la page encapsulée [Formulaire.aspx, Simulations.aspx, Erreurs.aspx]

Pour construire cette page, on pourra insérer dans le panel [entete], le code ASPX de la vue [VueEntete] de la page [Default.aspx] de la version précédente, décrite au paragraphe 11.5.2, page 89.

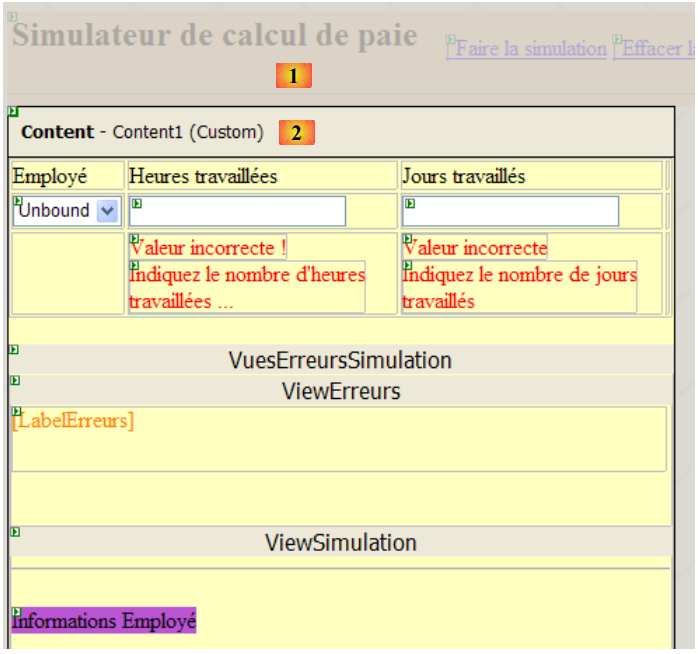
Le squelette du code de contrôle de la page maître sera le suivant :

```
1.  Partial Class MyMasterPage
2.    Inherits System.Web.UI.MasterPage
3.  ...
4.  End Class
```

- ligne 1 : la classe [MyMasterPage] de contrôle de la page maître.

12.4.2 La page [Formulaire.aspx]

Pour générer cette page, on suivra la méthode exposée page 110 et on renommera [Formulaire.aspx] la page [Default2.aspx] ainsi générée. L'aspect visuel de la page [Formulaire.aspx] en cours de construction sera le suivant :



L'aspect visuel de la page [Formulaire.aspx] a deux éléments :

- en [1] la page Maître avec son conteneur [ContentPlaceHolder1] (2)
- en [2] les composants placés dans le conteneur [ContentPlaceHolder1]. Ceux-ci sont identiques à ceux de l'application précédente.

Le code source de cette page est le suivant :

```
1.  <%@ Page Language="VB" MasterPageFile="~/MasterPage.master" AutoEventWireup="false"
2.    CodeFile="Formulaire.aspx.vb" Inherits="Formulaire" title="Simulation de calcul de paie :
    formulaire" %>
3.
4.  <%@ MasterType VirtualPath="~/MasterPage.master" %>
5.  <asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" runat="Server">
6.    <div>
```

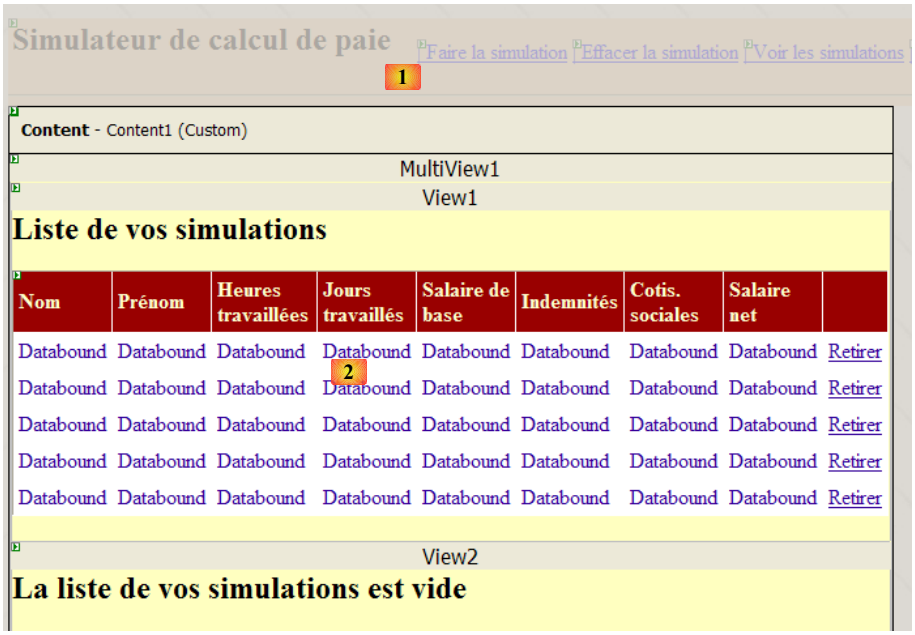
```
7.     <table>
8.       <tr>
9.         <td>
10.          Employé
11.        </td>
12.        <td>
13.          Heures travaillées
14.        </td>
15.        <td>
16.          Jours travaillés
17.        </td>
18.        <td>
19.        </td>
20.      </tr>
21. ....
22. </asp:Content>
```

- ligne 1 : la directive *Page* avec son attribut *MasterPageFile*
- ligne 4 : la classe de contrôle de la page maître peut exposer des champs et propriétés **publics**. Ceux-ci sont accessibles aux pages encapsulées avec la syntaxe **Master.[champ]** ou **Master.[propriété]**. La propriété **Master** de la page désigne la page maître sous la forme d'une instance de type [System.Web.UI.MasterPage]. Aussi dans notre exemple, faudrait-il écrire en réalité *CType(Master,MyMasterPage).[champ]* ou *CType(Master,MyMasterPage).[propriété]*. On peut éviter ce transtypage en insérant dans la page la directive *MasterType* de la ligne 4. L'attribut *VirtualPath* de cette directive indique le fichier de la page maître. Le compilateur peut alors connaître les champs, propriétés et méthodes publics exposés par la classe de la page maître, ici de type [MyMasterPage].
- lignes 5-22 : le contenu qui sera inséré dans le conteneur [ContentPlaceHolder1] de la page maître.

On pourra construire cette page en mettant comme contenu (lignes 6-21), celui de la vue [VueSaisies] décrite au paragraphe 11.5.3 de la page 89 et celui de la vue [VueSimulation] décrite au paragraphe 11.5.4 de la page 90.

12.4.3 La page [Simulations.aspx]

Pour générer cette page, on suivra la méthode exposée page 110 et on renommera [Simulations.aspx] la page [Default2.aspx] ainsi générée. L'aspect visuel de la page [Simulations.aspx] en cours de construction est le suivant :



L'aspect visuel de la page [Simulations.aspx] a deux éléments :

- en [1] la page Maître avec son conteneur [ContentPlaceHolder1]
- en [2] les composants placés dans le conteneur [ContentPlaceHolder1]. Ceux-ci sont identiques à ceux de l'application précédente.

Le code source de cette page est le suivant :

```
1.  <%@ Page Language="VB" MasterPageFile="~/MasterPage.master" AutoEventWireup="false"
2.    CodeFile="Simulations.aspx.vb" Inherits="PageSimulations" title="Simulation de calcul de paie :
    simulations" %>
3.  <%@ MasterType VirtualPath="~/MasterPage.master" %>
4.  <asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" runat="Server">
```

```
5.    <asp:MultiView ID="MultiView1" runat="server">
6.      <asp:View ID="ViewSimulations" runat="server">
7.        <h2>
8.          Liste de vos simulations</h2>
9.        <p>
10.           <asp:GridView ID="GridViewSimulations" runat="server" AutoGenerateColumns="False"
11.             CellPadding="4" ForeColor="#333333" GridLines="None">
12.             ...
13.           </asp:GridView>
14. </p>
15.     </asp:View>
16.     <asp:View ID="ViewSimulationsVides" runat="server">
17.       <h2>
18.         La liste de vos simulations est vide</h2>
19.     </asp:View>
20.   </asp:MultiView><br />
21. </asp:Content>
```

On pourra construire cette page en mettant comme contenu (lignes 5-20), celui de la vue [VueSimulations] décrite au paragraphe 11.5.5 de la page 90 et celui de la vue [VueSimulationsVides] décrite au paragraphe 11.5.6 de la page 92.

12.4.4 La page [Erreurs.aspx]

Pour générer cette page, on suivra la méthode exposée page 110 et on renommera [Erreurs.aspx] la page [Default2.aspx] ainsi générée. L'aspect visuel de la page [Erreurs.aspx] en cours de construction est le suivant :

L'aspect visuel de la page [Erreurs.aspx] a deux éléments :

- en [1] la page maître avec son conteneur [ContentPlaceHolder1]
- en [2] les composants placés dans le conteneur [ContentPlaceHolder1]. Ceux-ci sont identiques à ceux de l'application précédente.

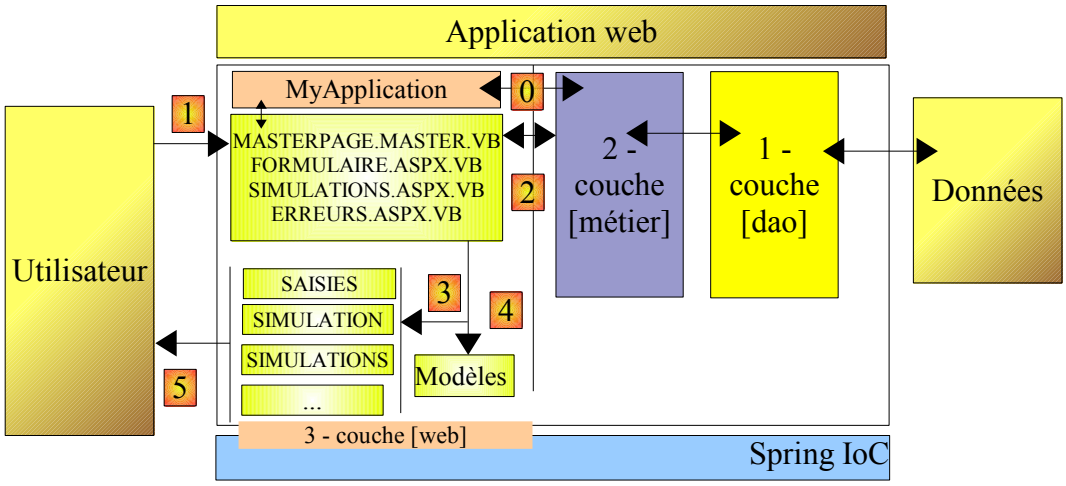
Le code source de cette page est le suivant :

```
1.  <%@ Page Language="VB" MasterPageFile="~/MasterPage.master" AutoEventWireup="false"
    CodeFile="Erreurs.aspx.vb" Inherits="PageErreurs" title="Simulation de calcul de paie : erreurs"
    %>
2.  <%@ MasterType VirtualPath="~/MasterPage.master" %>
3.  <asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" Runat="Server">
4.      <h3>Les erreurs suivantes se sont produites au démarrage de l'application</h3>
5.      <ul>
6.        <asp:Repeater id="rptErreurs" runat="server">
7.          <ItemTemplate>
8.            <li>
9.              <%# Container.DataItem %>
10.           </li>
11.         </ItemTemplate>
12.       </asp:Repeater>
13.     </ul>
14. </asp:Content>
```

12.5 Le code de contrôle des pages

12.5.1 Vue d'ensemble

Revenons à l'architecture de l'application :



- [MyApplication] est l'objet de type [HttpApplication] qui initialise (étape 0) l'application. Cette classe est identique à celle de la version précédente.
- le code du contrôleur, qui était dans la version précédente, tout entier dans [Default.aspx.vb] est désormais réparti sur plusieurs pages :
  - [MasterPage.master] : la page maître des pages [Formulaire.aspx, Simulations.aspx, Erreurs.aspx]. Elle contient le menu.
  - [Formulaire.aspx] : la page qui présente le formulaire de simulation et gère les actions qui ont lieu sur ce formulaire
  - [Simulations.aspx] : la page qui présente la liste des simulations et gère les actions qui ont lieu sur cette même page
  - [Erreurs.aspx] : la page qui est affichée lors d'une erreur d'initialisation de l'application. Il n'y a pas d'actions possibles sur cette page.

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

A) le client fait une demande à l'application. Il la fait normalement à l'une des deux pages [Formulaire.aspx, Simulations.aspx], mais rien ne l'empêche de demander la page [Erreurs.aspx]. Il faudra prévoir ce cas.
B) la page demandée traite cette demande (étape 1). Pour ce faire, elle peut avoir besoin de l'aide de la couche [métier] (étape 2) qui elle-même peut avoir besoin de la couche [dao] si des données doivent être échangées avec la base de données. L'application reçoit une réponse de la couche [métier].
C) selon celle-ci, elle choisit (étape 3) la vue (= la réponse) à envoyer au client et lui fournit (étape 4) les informations (le modèle) dont elle a besoin. Nous avons vu trois possibilités pour générer cette réponse :
   - la page (D) demandée est également la page (R) envoyée en réponse. Construire le modèle de la réponse (R) consiste alors à donner à certains des composants de la page (D), la valeur qu'ils doivent avoir dans la réponse.
   - la page (D) demandée n'est pas la page (R) envoyée en réponse. La page (D) peut alors :
     - transférer le flux d'exécution à la page (R) par l'instruction *Server.Transfer(" R ")*. Le modèle peut alors être placé dans le contexte par *Context.Items(" clé ")=valeur* ou plus rarement dans la session par *Session.Items(" clé ")=valeur*
     - rediriger le client vers la page (R) par l'instruction *Response.redirect(" R ")*. Le modèle peut alors être placé dans la session mais pas dans le contexte.
D) la réponse est envoyée au client (étape 5)

Chacune des pages [MasterPage.master, Formulaire.aspx, Simulations.aspx, Erreurs.aspx] répondra à un ou plusieurs des événements ci-dessous :

- **Init** : premier événement dans le cycle de vie de la page
- **Load** : se produit au chargement de la page
- **Click** : le clic sur l'un des liens du menu de la page maître

Nous traitons les pages les unes après les autres en commençant par la page maître.

12.5.2 Code de contrôle de la page [MasterPage.master]

12.5.2.1 Squelette de la classe

Le code de contrôle de la page maître a le squelette suivant :

```
1.  Imports istia.st.pam.web
2.  Imports System.Collections.Generic
3.
4.
5.  Partial Class MyMasterPage
6.    Inherits System.Web.UI.MasterPage
7.
8.    ' le menu
9.    Public ReadOnly Property OptionFaireSimulation() As LinkButton
10.     Get
11.       Return LinkButtonFaireSimulation
12.     End Get
13.   End Property
14. ....
15.
16.   ' fixer le menu
17.   Public Sub SetMenu(ByVal boolFaireSimulation As Boolean, ByVal boolEnregistrerSimulation As
    Boolean, ByVal boolEffacerSimulation As Boolean, ByVal boolFormulaireSimulation As Boolean, ByVal
    boolVoirSimulations As Boolean, ByVal boolTerminerSession As Boolean)
18. ...
19.   End Sub
20.
21.   ' gestion de l'option [Terminer la session]
22.   Protected Sub LinkButtonTerminerSession_Click(ByVal sender As Object, ByVal e As
    System.EventArgs) Handles LinkButtonTerminerSession.Click
23. ...
24.   End Sub
25.
26.   ' init master page
27.   Protected Sub Page_Init(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Init
28. ...
29.   End Sub
30. End Class
```

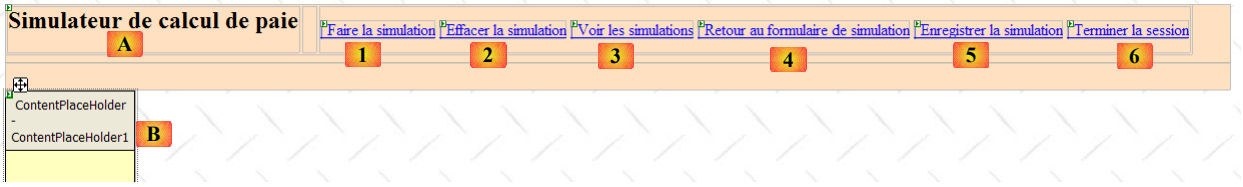
- lignes 5-6 : la classe s'appelle [MyMasterPage] et dérive de la classe système [System.Web.UI.MasterPage].
- lignes 9-15 : les 6 options du menu sont exposées comme propriétés publiques de la classe
- lignes 17-19 : la méthode publique *SetMenu* va permettre aux pages [Formulaire.aspx, Simulations.aspx, Erreurs.aspx] de fixer le menu de la page maître
- lignes 22-24 : la procédure qui va gérer le clic sur le lien [LinkButtonTerminerSession]
- lignes 27-29 : la procédure de gestion de l'événement *Init* de la page maître

12.5.2.2 Propriétés publiques de la classe

```
1.  ...
2.  Partial Class MyMasterPage
3.    Inherits System.Web.UI.MasterPage
4.
5.    ' le menu
6.    Public ReadOnly Property OptionFaireSimulation() As LinkButton
7.      Get
8.        Return LinkButtonFaireSimulation
9.      End Get
10.   End Property
11.
12.   Public ReadOnly Property OptionEffacerSimulation() As LinkButton
13.     Get
14.       Return LinkButtonEffacerSimulation
15.     End Get
16.   End Property
17.
18.   Public ReadOnly Property OptionEnregistrerSimulation() As LinkButton
19.     Get
20.       Return LinkButtonEnregistrerSimulation
21.     End Get
22.   End Property
23.
24.   Public ReadOnly Property OptionVoirSimulations() As LinkButton
25.     Get
26.       Return LinkButtonVoirSimulations
27.     End Get
28.   End Property
29.
30.   Public ReadOnly Property OptionTerminerSession() As LinkButton
```

```
31.     Get
32.       Return LinkButtonTerminerSession
33.     End Get
34.   End Property
35.
36.   Public ReadOnly Property OptionFormulaireSimulation() As LinkButton
37.     Get
38.       Return LinkButtonFormulaireSimulation
39.     End Get
40.   End Property
41. ...
42. End Class
```

Pour comprendre ce code, il faut se rappeler les composants qui forment la page maître :



N°	Type	Nom	Rôle
A	Panel (rose ci-dessus)	entete	entête de la page
B	Panel (jaune ci-dessus)	contenu	contenu de la page
1	LinkButton	LinkButtonFaireSimulation	demande le calcul de la simulation
2	LinkButton	LinkButtonEffacerSimulation	efface le formulaire de saisie
3	LinkButton	LinkButtonVoirSimulations	affiche la liste des simulations déjà faites
4	LinkButton	LinkButtonFormulaireSimulation	ramène au formulaire de saisie
5	LinkButton	LinkButtonEnregistrerSimulation	enregistre la simulation courante dans la liste des simulations
6	LinkButton	LinkButtonTerminerSession	abandonne la session courante

Les composants 1 à 6 ne sont pas accessibles en-dehors de la page qui les contient. Les propriétés des lignes 6 à 40 visent à les rendre accessibles aux classes externes, ici les classes des autres pages de l'application.

12.5.2.3 La méthode SetMenu

La méthode publique *SetMenu* permet aux pages [Formulaire.aspx, Simulations.aspx, Erreurs.aspx] de fixer le menu de la page maître. Son code est basique :

```
1.  Public Sub SetMenu(ByVal boolFaireSimulation As Boolean, ByVal boolEnregistrerSimulation As Boolean,
  ByVal boolEffacerSimulation As Boolean, ByVal boolFormulaireSimulation As Boolean, ByVal
  boolVoirSimulations As Boolean, ByVal boolTerminerSession As Boolean)
2.    ' on fixe les options de menu
3.    LinkButtonFaireSimulation.Visible = boolFaireSimulation
4.    LinkButtonEnregistrerSimulation.Visible = boolEnregistrerSimulation
5.    LinkButtonEffacerSimulation.Visible = boolEffacerSimulation
6.    LinkButtonVoirSimulations.Visible = boolVoirSimulations
7.    LinkButtonFormulaireSimulation.Visible = boolFormulaireSimulation
8.    LinkButtonTerminerSession.Visible = boolTerminerSession
9.End Sub
```

12.5.2.4 La gestion des événements de la page maître

La page maître va gérer deux événements :

- l'événement *Init* qui est le premier événement du cycle de vie de la page
- l'événement *Click* sur le lien [LinkButtonTerminerSession]

La page maître a cinq autres liens : [LinkButtonFaireSimulation, LinkButtonEnregistrerSimulation, LinkButtonEffacerSimulation, LinkButtonVoirSimulations, LinkButtonFormulaireSimulation]. Comme exemple, examinons ce qu'il faudrait faire lors d'un clic sur le lien [LinkButtonFaireSimulation] :

1. vérifier les données saisies (heures, jours) dans la page [Formulaire.aspx]
2. faire le calcul du salaire
3. afficher les résultats dans la page [Formulaire.aspx]

Les opérations 1 et 3 impliquent d'avoir accès aux composants de la page [Formulaire.aspx]. Ce n'est pas le cas. En effet, la page maître n'a aucune connaissance des composants des pages susceptibles d'être insérées dans son conteneur [ContentPlaceHolder1]. Dans notre exemple, c'est à la page [Formulaire.aspx] de gérer le clic sur le lien [LinkButtonFaireSimulation] car c'est elle qui est affichée lorsqu'a lieu cet événement. Comment peut-elle être avertie de celui-ci ?

- le lien [LinkButtonFaireSimulation] ne faisant pas partie de la page [Formulaire.aspx], on ne peut pas écrire dans [Formulaire.aspx] la procédure habituelle :

```
1.   Protected Sub LinkButtonFaireSimulation_Click(ByVal sender As Object, ByVal e As
    System.EventArgs) Handles LinkButtonFaireSimulation.Click
2.
3.  End Sub
```

La clause *Handles* indique que la procédure gère l'événement *Click* du composant *LinkButtonFaireSimulation*. Or à la compilation de la page [Formulaire.aspx], le compilateur indiquera que la page ne contient pas ce composant, ce qui est vrai. On peut contourner le problème avec le code suivant dans [Formulaire.aspx] :

```
1.    ' les options de menu
2.    Private WithEvents OptFaireSimulation As LinkButton
3.    Private WithEvents OptEffacerSimulation As LinkButton
4.    Private WithEvents OptVoirSimulations As LinkButton
5.    Private WithEvents OptEnregistrerSimulation As LinkButton
6.
7.    ' chargement de la page
8.    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles MyBase.Load
9.      ' gestionnaire d'évts
10.     OptFaireSimulation = Master.OptionFaireSimulation
11.     OptEffacerSimulation = Master.OptionEffacerSimulation
12.     OptVoirSimulations = Master.OptionVoirSimulations
13.     OptEnregistrerSimulation = Master.OptionEnregistrerSimulation
14. ...
15.   End Sub
16.
17.   Private Sub DoFaireSimulation(ByVal sender As Object, ByVal e As System.EventArgs) Handles
    OptFaireSimulation.Click
18. End Sub
```

- lignes 2-4 : on déclare des variables pour chacun des liens dont on veut gérer l'événement *Click*. Pour déclarer que ces objets lancent des événements, ils sont déclarés avec l'attribut **WithEvents**.
- lignes 10-13 : lorsque l'événement *Load* de la page [Formulaire.aspx] se produit, la classe [MyMasterPage] de la page maître a été instanciée. Ses champs et propriétés publics sont alors accessibles. Comme dans la classe [MyMasterPage], nous avons créé une propriété publique pour chacun des six liens de la page maître, les variables des lignes 2-4 sont initialisées (lignes 10-13) avec les propriétés publiques correspondantes de la classe [MyMasterPage], avec la syntaxe déjà présentée *Master.propriété*, où **Master** est une propriété de la classe [System.Web.UI.Page] référençant la page maître de la page encapsulée.
- lignes 17-18 : la procédure [DoFaireSimulation] gère l'événement *Click* de l'objet *OptFaireSimulation* (clause Handles). Comme par construction, *OptFaireSimulation* de [Formulaire.aspx] et le lien [LinkButtonFaireSimulation] de [MasterPage.master] sont des références sur le même objet, la procédure [DoFaireSimulation] va au final gérer l'événement *Click* sur le lien [LinkButtonFaireSimulation] de [MasterPage.master].

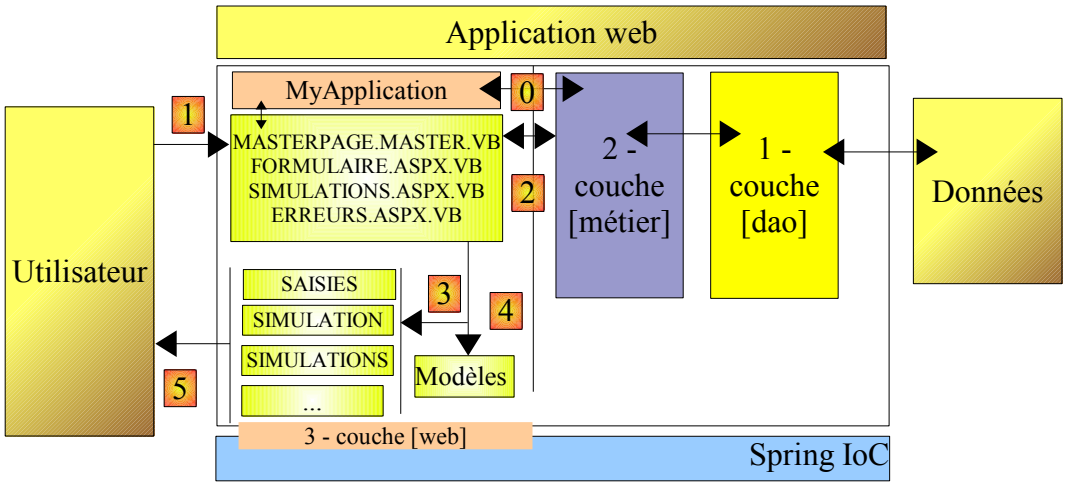
La gestion des événements *Click* sur les six liens du menu sera répartie de la façon suivante :
- la page [Formulaire.aspx] gèrera les liens [LinkButtonFaireSimulation, LinkButtonEnregistrerSimulation, LinkButtonEffacerSimulation, LinkButtonVoirSimulations]
- la page [Simulations.aspx] gèrera le lien [LinkButtonFormulaireSimulation]
- la page maître [MasterPage.master] gèrera le lien [LinkButtonTerminerSession]. Pour cet événement, elle n'a en effet pas besoin de connaître la page qu'elle encapsule.

12.5.2.5 L'événement Init de la page

Les trois pages [Formulaire.aspx, Simulations.aspx, Erreurs.aspx] de l'application ont [MasterPage.master] pour page maître. Appelons M la page maître, E la page encapsulée. Lorsque la page E est demandée par le client, les événements suivants se produisent dans l'ordre :
- E.Init
- M.Init
- E.Load
- M.Load

- ...

Nous allons utiliser l'événement *Init* de la page M pour exécuter du code qu'il serait intéressant d'exécuter le plus tôt possible, ceci quelque soit la page cible E. Pour découvrir ce code, revoyons l'image d'ensemble de l'application :



Ci-dessus, [MyApplication] est l'objet de type [HttpApplication] qui initialise l'application. Cette classe est la même que dans la version précédente :

```
1.  ...
2.  Namespace istia.st.pam.web
3.
4.    Public Class MyApplication
5.      Inherits System.Web.HttpApplication
6.
7.      ' données statiques de l'application
8.      Public Shared Msg As String = String.Empty
9.      Public Shared PamMetier As IPamMetier
10.     Public Shared Employes As RawEmploye()
11.     Public Shared Erreur As Boolean = False
12.
13.     ' démarrage de l'application
14.     Sub Application_Start(ByVal sender As Object, ByVal e As EventArgs)
15. ...
16.     End Sub
17.
18.
19.     ' démarrage de l'application
20.     Sub Session_Start(ByVal sender As Object, ByVal e As EventArgs)
21. ...
22.   End Class
23.
24. End Namespace
```

Si la classe [MyApplication] ne réussit pas à initialiser correctement l'application, elle positionne deux variables publiques statiques :

- le booléen **Erreur** de la ligne 11 est mis à *vrai*
- la variable **Msg** de la ligne 8 contient un message donnant des détails sur l'erreur rencontrée

Lorsque l'utilisateur demande l'une des pages [Formulaire.aspx, Simulations.aspx] alors que l'application ne s'est pas initialisée correctement, cette demande doit être transférée ou redirigée vers la page [Erreurs.aspx], qui affichera le message d'erreur de la classe [MyApplication]. On peut gérer ce cas de diverses façons :

- faire le test d'erreur d'initialisation dans le gestionnaire des événements *Init* ou *Load* de chacune des pages [Formulaire.aspx, Simulations.aspx]
- faire le test d'erreur d'initialisation dans le gestionnaire des événements *Init* ou *Load* de la page maître de ces deux pages. Cette méthode a l'avantage de placer le test d'erreur d'initialisation à un unique endroit.

Nous choisissons de faire le test d'erreur d'initialisation dans le gestionnaire de l'événement *Init* de la page maître :

```
1.    Protected Sub Page_Init(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Init
2.      ' des erreurs d'initialisation ?
3.      If MyApplication.Erreur Then
4.        ' la page encapsulée est-elle la page d'erreurs ?
```

```
5.       Dim IsPageErreurs As Boolean =
    Me.FindControl("form1").FindControl("contenu").FindControl("ContentPlaceHolder1").FindControl("rpt
    Erreurs") IsNot Nothing
6.       ' si c'est la page d'erreurs qui s'affiche, on laisse faire sinon on redirige le client vers
    la page d'erreurs
7.       If Not IsPageErreurs Then Response.Redirect("erreurs.aspx")
8.       Return
9.     End If
10. End Sub
```

Le code ci-dessus va s'exécuter dès que l'une des pages [Formulaire.aspx, Simulations.aspx, Erreurs.aspx] va être demandée. Dans le cas où la page demandée est [Formulaire.aspx, Simulations.aspx], on se contente (ligne 7) de rediriger le client vers la page [Erreurs.aspx], celle-ci se chargeant d'afficher le message d'erreur de la classe [MyApplication]. Dans le cas où la page demandée est [Erreurs.aspx], cette redirection ne doit pas avoir lieu : il faut laisser la page [Erreurs.aspx] s'afficher. Il nous faut donc savoir dans la méthode [Page_Init] de la page maître, quelle est la page que celle-ci encapsule.

Revenons sur l'arbre des composants de la page maître :

```
1.  ...
2.  <body background="ressources/standard.jpg">
3.    <form id="form1" runat="server">
4.      <asp:panel ID="entete" runat="server" BackColor="#FFE0C0" Width="1239px" >
5.  ...
6.      </asp:Panel>
7.      <div>
8.        <asp:Panel ID="contenu" runat="server" BackColor="#FFFFC0">
9.          <asp:ContentPlaceHolder ID="ContentPlaceHolder1" runat="server">
10.         </asp:ContentPlaceHolder>
11.       </asp:Panel>
12.     </div>
13.   </form>
14. </body>
15. </html>
```

- lignes 1-13 : le conteneur d'id "form1"
- lignes 4-6 : le conteneur d'id "entete", inclus dans le conteneur d'id "form1"
- lignes 8-11 : le conteneur d'id "contenu", inclus dans le conteneur d'id "form1"
- lignes 9-10 : le conteneur d'id "ContentPlaceHolder1", inclus dans le conteneur d'id "contenu"

Une page E encapsulée dans la page maître M, l'est dans le conteneur d'id "ContentPlaceHolder1". Pour référencer un composant d'id C de cette page E, on écrira :

```
Me.FindControl("form1").FindControl("contenu").FindControl("ContentPlaceHolder1").FindControl("C")
```

L'arbre des composants de la page [Erreurs.aspx] est lui, le suivant :

```
1.  <%@ Page Language="VB" MasterPageFile="~/MasterPage.master" AutoEventWireup="false"
    CodeFile="Erreurs.aspx.vb" Inherits="PageErreurs" title="Simulation de calcul de paie : erreurs"
    %>
2.  <%@ MasterType VirtualPath="~/MasterPage.master" %>
3.  <asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" Runat="Server">
4.      <h3>Les erreurs suivantes se sont produites au démarrage de l'application</h3>
5.      <ul>
6.        <asp:Repeater id="rptErreurs" runat="server">
7.          <ItemTemplate>
8.            <li>
9.              <%# Container.DataItem %>
10.           </li>
11.         </ItemTemplate>
12.       </asp:Repeater>
13.     </ul>
14. </asp:Content>
```

Lorsque la page [Erreurs.aspx] est fusionnée avec la page maître M, le contenu de la balise <asp:Content> ci-dessus (lignes 3-14), est intégré dans la balise <asp:ContentPlaceHolder> d'id "ContentPlaceholder1" de la page M, l'arbre des composants de celle-ci devenant alors :

```
1.  ...
2.  <body background="ressources/standard.jpg">
3.    <form id="form1" runat="server">
4.      <asp:panel ID="entete" runat="server" BackColor="#FFE0C0" Width="1239px" >
5.  ...
6.      </asp:Panel>
7.      <div>
8.        <asp:Panel ID="contenu" runat="server" BackColor="#FFFFC0">
9.          <asp:ContentPlaceHolder ID="ContentPlaceHolder1" runat="server">
10.           <h3>Les erreurs suivantes se sont produites au démarrage de l'application</h3>
11.           <ul>
```

```
12.           <asp:Repeater id="rptErreurs" runat="server">
13.             <ItemTemplate>
14.               <li>
15.                 <%# Container.DataItem %>
16.               </li>
17.             </ItemTemplate>
18.           </asp:Repeater>
19.         </ul>
20.       </asp:ContentPlaceHolder>
21.     </asp:Panel>
22.   </div>
23.  </form>
24. </body>
25. </html>
```

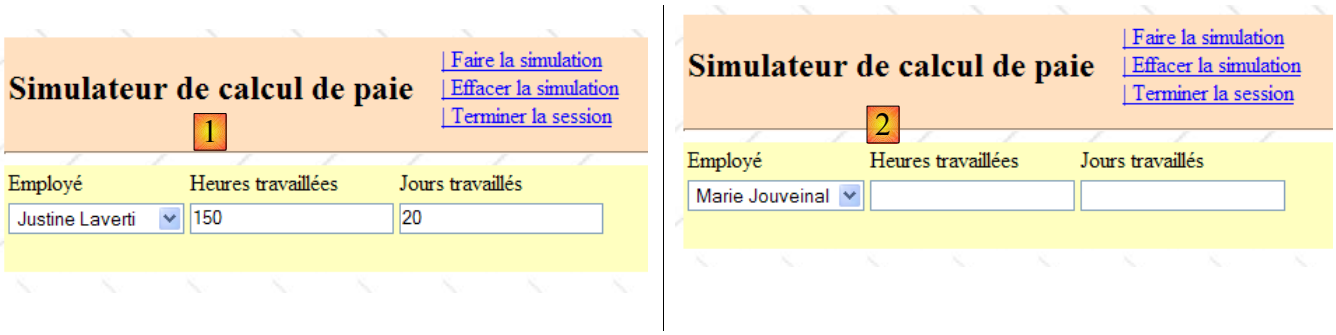
- ligne 12 : le composant [rptErreurs] peut être utilisé pour savoir si la page maître M contient ou non la page [Erreurs.aspx]. En effet, ce composant n'existe que dans cette page.

Ces explications suffisent pour comprendre le code de la procédure [Page_Init] de la page maître :

```
1.   Protected Sub Page_Init(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Init
2.     ' des erreurs d'initialisation ?
3.     If MyApplication.Erreur Then
4.       ' la page encapsulée est-elle la page d'erreurs ?
5.       Dim IsPageErreurs As Boolean =
    Me.FindControl("form1").FindControl("contenu").FindControl("ContentPlaceHolder1").FindControl("rpt
    Erreurs") IsNot Nothing
6.       ' si c'est la page d'erreurs qui s'affiche, on laisse faire sinon on redirige le client vers
    la page d'erreurs
7.       If Not IsPageErreurs Then Response.Redirect("erreurs.aspx")
8.       Return
9.     End If
10. End Sub
```

- ligne 3 : on vérifie si la clase [MyApplication] a positionné son booléen *Erreur*
- ligne 5 : si oui, le booléen *IsPageErreurs* indique si la page encapsulée dans la page maître est la page [Erreurs.aspx]
- ligne 7 : si la page encapsulée dans la page maître n'est pas la page [Erreurs.aspx], alors on redirige le client vers cette page, sinon on ne fait rien.

12.5.2.6 L'événement Click sur le lien [LinkButtonTerminerSession]



Lorsque l'utilisateur clique sur le lien [Terminer la session] dans la vue (1) ci-dessus, il faut vider la session de son contenu et présenter un formulaire vide (2).

Le code du gestionnaire de cet événement pourrait être le suivant :

```
1.  ' gestion de l'option [Terminer la session]
2.  Protected Sub LinkButtonTerminerSession_Click(ByVal sender As Object, ByVal e As System.EventArgs)
  Handles LinkButtonTerminerSession.Click
3.    ' on abandonne la session
4.    Session.Abandon()
5.    ' on affiche la vue [formulaire]
6.    Response.Redirect("Formulaire.aspx")
7.End Sub
```

- ligne 4 : la session courante est abandonnée
- ligne 6 : le client est redirigé vers la page [Formulaire.aspx]

On voit que ce code ne fait intervenir aucun des composants des pages [Formulaire.aspx, Simulations.aspx, Erreurs.aspx]. L'événement peut donc être géré par la page maître elle-même.

12.5.3 Code de contrôle de la page [Erreurs.aspx]

Le code de contrôle de la page [Erreurs.aspx] pourrait être le suivant :

```
1.  Imports System.Collections.Generic
2.  Imports istia.st.pam.web
3.
4.  Partial Class PageErreurs
5.    Inherits System.Web.UI.Page
6.
7.    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Load
8.      ' des erreurs d'initialisation ?
9.      If MyApplication.Erreur Then
10.       ' on prépare le modèle de la page [erreurs]
11.       Dim ErreursInitialisation As New List(Of String)
12.       ErreursInitialisation.Add(MyApplication.Msg)
13.       ' on associe la liste d'erreurs à son composant
14.       With rptErreurs
15.         .DataSource = ErreursInitialisation
16.         .DataBind()
17.       End With
18.     End If
19.     ' on fixe le menu
20.     Master.SetMenu(False, False, False, False, False, False)
21.   End Sub
22.
23. End Class
```

Rappelons que la page [Erreurs.aspx] a pour unique rôle d'afficher une erreur d'initialisation de l'application lorsque celle-ci se produit :

- ligne 9 : on teste si l'initialisation s'est terminée par une erreur
- lignes 11-12 : si oui, le message d'erreur (MyApplication.Msg) est placé dans une liste [ErreursInitialisation]
- lignes 14-17 : on demande au composant [rptErreurs] d'afficher cette liste
- ligne 20 : dans tous les cas (erreur ou pas), les options du menu de la page maître ne sont pas affichées, de sorte que l'utilisateur ne peut débuter aucune nouvelle action à partir de cette page.

Que se passe-t-il si l'utilisateur demande directement la page [Erreurs.aspx] (ce qu'il n'est pas supposé faire dans une utilisation normale de l'application) ? En suivant le code de [MasterPage.master.vb] et de [Erreurs.aspx.vb], on s'apercevra que :

- s'il y a eu erreur d'initialisation, celle-ci est affichée
- s'il n'y a pas eu erreur d'initialisation, l'utilisateur reçoit une page ne contenant que l'entête de [MasterPage.master] avec aucune option de menu affichée.

12.5.4 Code de contrôle de la page [Formulaire.aspx]

12.5.4.1 Squelette de la classe

Le squelette du code de contrôle de la page [Formulaire.aspx] pourrait être le suivant :

```
1.  Imports istia.st.pam.metier.entites
2.  Imports istia.st.pam.metier.service
3.  Imports istia.st.pam.dao.entites
4.  Imports istia.st.pam.dao.service
5.  Imports istia.st.pam.web
6.  Imports System.Collections.Generic
7.
8.  Partial Class PageFormulaire
9.    Inherits System.Web.UI.Page
10.
11.   ' les options de menu
12.   Private WithEvents OptFaireSimulation As LinkButton
13.   Private WithEvents OptEffacerSimulation As LinkButton
14.   Private WithEvents OptVoirSimulations As LinkButton
15.   Private WithEvents OptEnregistrerSimulation As LinkButton
16.
17.   ' chargement de la page
18.   Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles MyBase.Load
19.     ' options du menu de la page maître à gérer
20.     OptFaireSimulation = Master.OptionFaireSimulation
21.     OptEffacerSimulation = Master.OptionEffacerSimulation
22.     OptVoirSimulations = Master.OptionVoirSimulations
23.     OptEnregistrerSimulation = Master.OptionEnregistrerSimulation
24. ...
25.   End Sub
26.
27.   ' calcul de la paie
```

```
28.  Private Sub DoFaireSimulation(ByVal sender As Object, ByVal e As System.EventArgs) Handles
    OptFaireSimulation.Click
29. ...
30.  End Sub
31.
32.  ' effacer la simulation
33.  Private Sub DoEffacerSimulation(ByVal sender As Object, ByVal e As System.EventArgs) Handles
    OptEffacerSimulation.Click
34. ....
35.  End Sub
36.
37.  Protected Sub DoVoirSimulations(ByVal sender As Object, ByVal e As System.EventArgs) Handles
    OptVoirSimulations.Click
38. ...
39.  End Sub
40.
41.  Protected Sub DoEnregistrerSimulation(ByVal sender As Object, ByVal e As System.EventArgs)
    Handles OptEnregistrerSimulation.Click
42. ...
43.  End Sub
44.
45. End Class
```

Le code de contrôle de la page [Formulaire.aspx] gère cinq événements :

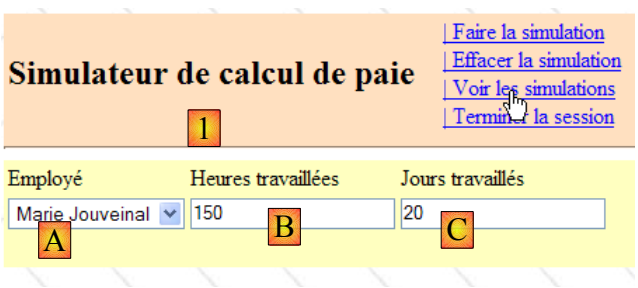
1. l'événement *Load* de la page
2. l'événement *Click* sur le lien [LinkButtonFaireSimulation] de la page maître
3. l'événement *Click* sur le lien [LinkButtonEffacerSimulation] de la page maître
4. l'événement *Click* sur le lien [LinkButtonEnregistrerSimulation] de la page maître
5. l'événement *Click* sur le lien [LinkButtonVoirSimulations] de la page maître

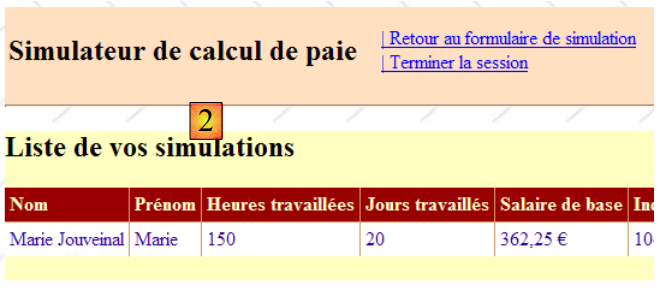
12.5.4.2 Evénement Load de la page

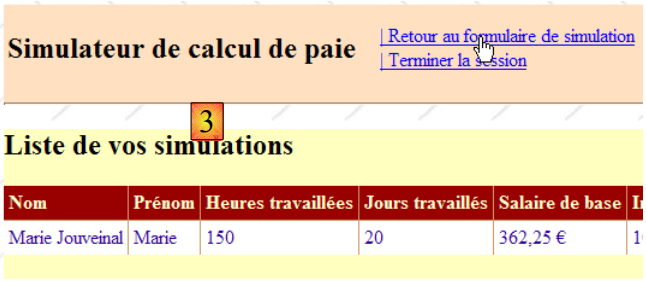
Le squelette du gestionnaire de l'événement *Load* de la page pourrait être le suivant :

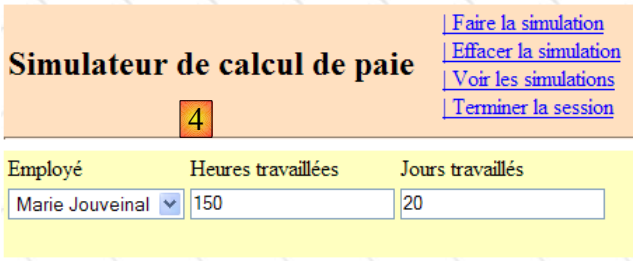
```
1.  ' chargement de la page
2.   Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles MyBase.Load
3.     ' options du menu de la page maître
4.     OptFaireSimulation = Master.OptionFaireSimulation
5.     OptEffacerSimulation = Master.OptionEffacerSimulation
6.     OptVoirSimulations = Master.OptionVoirSimulations
7.     OptEnregistrerSimulation = Master.OptionEnregistrerSimulation
8.     ' affichage vue [saisies]
9.     ...
10.    ' positionnement menu page maître
11.    ...
12.    Master.SetMenu(...)
13.    ' traitement requête GET
14.    If Not IsPostBack Then
15.      ' chargement des noms des employés dans le combo
16. ...
17.      ' init vue [saisies] avec saisies mémorisées dans la session si elles existent
18. ...
19.    End If
20.  End Sub
```

Un exemple pour éclaircir le commentaire de la ligne 17 pourrait être celui-ci :









- en [1], on demande à voir la liste des simulations. Des saisies ont été faites en [A, B, C].
- en [2], on voit la liste
- en [3], on demande à retourner au formulaire
- en [4], on retrouve le formulaire tel qu'on l'a laissé. Comme il y a eu deux requêtes, (1,2) et (3,4), cela signifie que :
  - lors du passage de [1] à [2], les saisies de [1] ont été mémorisées
  - lors du passage de [3] à [4], elles ont été restituées. C'est la procédure [Page_Load] de [Formulaire.aspx] qui opére cette restitution.

Question : compléter la procédure Page_Load en vous aidant des commentaires et du code de la version précédente

12.5.4.3 Gestion des événements *Click* sur les liens du menu

Le squelette des gestionnaires des événements *Click* sur les liens de la page maître est le suivant :

```
1.    ' calcul de la paie
2.    Private Sub DoFaireSimulation(ByVal sender As Object, ByVal e As System.EventArgs) Handles
    OptFaireSimulation.Click
3.      ' page valide ?
4.      Page.Validate()
5.      If Not Page.IsValid Then
6.        ' affichage vue [saisie]
7.        VuesErreursSimulation.Visible = False
8.        Exit Sub
9.      End If
10.     ' la page est valide - on récupère les saisies
11. ...
12.     ' on met le résultat dans la session
13. ...
14.     ' on met les saisies également dans la session
15. ...
16.     ' affichage
17. ...
18.     ' affichage vues
19.     VuesErreursSimulation.Visible = True
20.     VuesErreursSimulation.ActiveViewIndex = 1
21.     ' affichage menu MasterPage
22.     Master.SetMenu(...)
23.   End Sub
24.
25.   ' effacer la simulation
26.   Private Sub DoEffacerSimulation(ByVal sender As Object, ByVal e As System.EventArgs) Handles
    OptEffacerSimulation.Click
27.     ' affichage vue [saisies] réinitialisée
28. ...
29.   End Sub
30.
31.   Protected Sub DoVoirSimulations(ByVal sender As Object, ByVal e As System.EventArgs) Handles
    OptVoirSimulations.Click
32.     ' on met les saisies dans la session
33. ...
34.     ' on affiche la vue [simulations]
35.     Response.Redirect("Simulations.aspx")
36.   End Sub
37.
38.   Protected Sub DoEnregistrerSimulation(ByVal sender As Object, ByVal e As System.EventArgs)
    Handles OptEnregistrerSimulation.Click
39.     ' on enregistre la simulation courante dans la session de l'utilisateur
40. ...
41.     ' on affiche la vue [simulations]
42.     Response.Redirect("Simulations.aspx")
43. End Sub
```

Question : compléter le code des procédures ci-dessus en vous aidant des commentaires et du code de la version précédente

12.5.5 Code de contrôle de la page [Simulations.aspx]

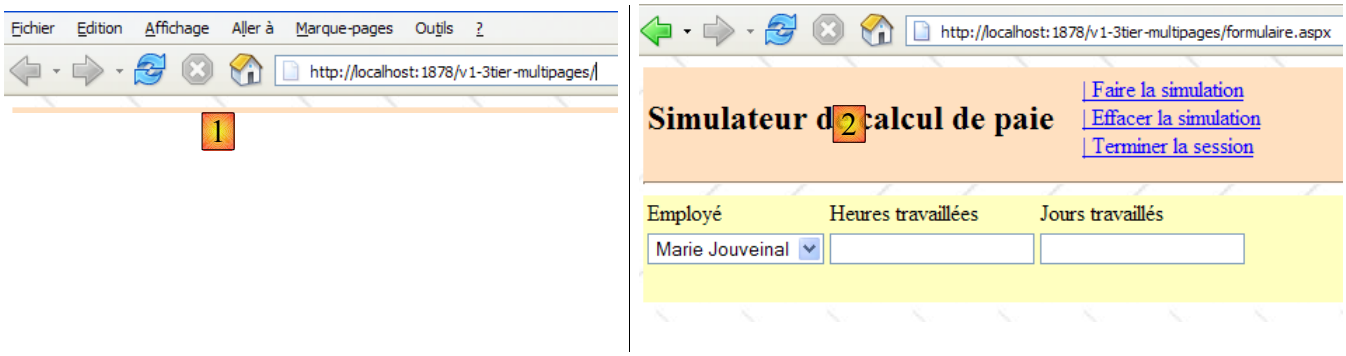
Le squelette du code de contrôle de la page [Simulations.aspx] pourrait être le suivant :

```
1.  Imports istia.st.pam.web
2.  Imports System.Collections.Generic
3.
4.  Partial Class PageSimulations
5.   Inherits System.Web.UI.Page
6.
7.   ' les simulations
8.   Private Simulations As List(Of Simulation)
9.   ' les evts
10.  Private WithEvents OptFormulaireSimulation As LinkButton
11.
12.  Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Load
13.    ' gestionnaire d'évts
14.    OptFormulaireSimulation = Master.OptionFormulaireSimulation
15.    ' on récupère les simulations dans la session
16.    ...
17.    ' y-a-t-il des simulations ?
18.    If Simulations.Count <> 0 Then
19.      ' première vue visible
20.      MultiView1.ActiveViewIndex = 0
21.      ' on remplit le GridView
22.            With GridViewSimulations
23. ...
24.            End With
25.    Else
1.       ' vue [SimulationsVides] visible
26.      ...
27.    End If
28.    ' on fixe le menu
29.    Master.SetMenu(...)
30.  End Sub
31.
32.         Protected Sub GridViewSimulations_RowDeleting(ByVal sender As Object, ByVal e As
    System.Web.UI.WebControls.GridViewDeleteEventArgs) Handles GridViewSimulations.RowDeleting
33.        ' on récupère les simulations dans la session
34. ...
35.        ' on supprime la simulation désignée par e.RowIndex
36.        ...
37.        ' reste-t-il des simulations ?
38.        If Simulations.Count <> 0 Then
39.            ' on lie les données au GridView
40.            With GridViewSimulations
41. ...
42.            End With
43.        Else
44.            ' vue [SimulationsVides]
45.            ...
46.        End If
47.     End Sub
48.
49.  Protected Sub DoFormulaireSimulation(ByVal sender As Object, ByVal e As System.EventArgs) Handles
    OptFormulaireSimulation.Click
50.    ' on affiche la vue [formulaire]
51.    Response.Redirect("formulaire.aspx")
52.  End Sub
53.
54. End Class
```

Question : compléter le code des procédures ci-dessus en vous aidant des commentaires et du code de la version précédente

12.5.6 Code de contrôle de la page [Default.aspx]

On peut prévoir une page [Default.aspx] dans l'application, afin de permettre à l'utilisateur de demander l'url de l'application sans préciser de page, comme ci-dessous :



La demande [1] a reçu en réponse la page [Formulaire.aspx] (2). On sait que la demande (1) est traitée par défaut par la page [Default.aspx] de l'application. Pour obtenir (2), il suffit que [Default.aspx] redirige le client vers la page [Formulaire.aspx]. Cela peut être obtenu avec le code suivant :

```
1.  Partial Class _Default
2.      Inherits System.Web.UI.Page
3.
4.    Protected Sub Page_Init(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Init
5.      ' on redirige vers le formulaire de saisie
6.      Response.Redirect("Formulaire.aspx")
7.    End Sub
8.  End Class
```

La page de présentation [Default.aspx] ne contient elle que la directive qui la relie à [Default.aspx.vb] :

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb" Inherits="_Default" %>
```

12.6 Ajaxification de l'application

En suivant une démarche analogue à celle suivie dans la version précédente, ajaxifiez les différentes pages de cette application.

Table des matières

1 INTRODUCTION 2

2 L'ÉTUDE DE CAS 3

2.1 La base de données 3
2.2 Mode de calcul du salaire d'une assistante maternelle 6

3 L'APPLICATION [SIMUPAIE] – VERSION 1 – VB.NET 8

3.1 Le formulaire 8
3.2 Configuration de l'application 9
3.3 Démarrage de l'application 10
3.4 Calcul du salaire 10

4 L'APPLICATION [SIMUPAIE] – VERSION 2 – VB.NET 12

4.1 Le projet VB.NET 12
4.2 Les objets de l'application 12
4.2.1 La classe [RawEmploye] 12
4.2.2 La classe [RawCotisations] 13
4.2.3 La classe [RawIndemnites] 14
4.3 Configuration et initialisation de l'application 15
4.4 Le code du formulaire [pam.vb] 17

5 L'APPLICATION [SIMUPAIE] – VERSION 3 – ASP.NET 19

5.1 Le projet Visual Web 2005 19
5.2 Configuration de l'application 20
5.3 Le formulaire [Default.aspx] 23
5.3.1 La procédure [Page_Load] 23
5.3.2 La vérification des saisies 24
5.3.3 Le calcul du salaire 25

6 L'APPLICATION [SIMUPAIE] – VERSION 4 – ASP.NET 26

6.1 Le projet Visual Web 2005 26
6.2 La nouvelle architecture de l'application 26
6.3 Les entités de l'application web 28
6.4 Configuration de l'application 30
6.5 La classe [MyApplication] 30
6.6 Le formulaire [Default.aspx.vb] 31

7 L'APPLICATION [SIMUPAIE] – VERSION 5 – AJAX / ASP.NET 32

7.1 Introduction 32
7.2 Le projet Visual Web Developer de la couche [web-ui-ajax] 34
7.3 Tests de la solution Ajax 38
7.4 Conclusion 39

8 L'APPLICATION [SIMUPAIE] – VERSION 6 – VB.NET / 3 COUCHES 40

8.1 Architecture générale de l'application 40
8.2 Les interfaces des différentes couches de l'application 40
8.3 La couche [dao] d'accès aux données 43
8.3.1 Le projet Visual Studio de la couche [dao] 43
8.3.2 La classe [RawEmployeIndemnites] 43
8.3.3 L'interface [IPamDao] de la couche [dao] 44
8.3.4 La classe [PamException] 45
8.4 Implémentation de la couche [dao] avec [Ibatis SqlMap] 46
8.4.1 Le produit iBatis SqlMap 46
8.4.2 Où trouver IBATIS SqlMap ? 47
8.4.3 Les fichiers de configuration d'iBatis SqlMap 48
8.4.4 Les fichiers de configuration du projet [pam-dao-ibatis] 49
8.4.4.1 providers.config 50
8.4.4.2 sqlmap.config 51
8.4.4.3 dbpam.xml 52
8.4.5 L'API de SqlMap 54

8.5 Implémentation et tests de la couche [dao] 57
8.5.1 Le projet Visual Studio 57
8.5.2 Le programme de test 58
8.5.3 Écriture de la classe [PamDaoSqlMap] 59
8.5.4 Génération de la DLL de la couche [dao] 60
8.6 Tests NUnit de la classe [PamDaoSqlMap] 61
8.6.1 Le projet Visual studio des tests Nunit 61
8.6.2 La classe de test [NunitTestPamDao.vb] 62
8.6.3 Mise en oeuvre des tests 63
8.7 La couche métier 64
8.7.1 Le projet Visual Studio de la couche [metier] 65
8.7.2 L'interface [IPamMetier] de la couche [metier] 65
8.7.3 Les entités de la couche [metier] 66
8.7.4 Implémentation de la couche [metier] 67
8.7.5 Un premier programme de test de la couche [metier] 68
8.7.6 Génération de la DLL de la couche [metier] 69
8.8 Tests unitaires de la couche métier 69
8.8.1 Le projet Visual studio des tests Nunit 69
8.8.2 La classe de test [NunitTestPamMetier.vb] 71
8.8.3 Mise en oeuvre des tests 72
8.9 La couche [ui] 73
8.9.1 Le projet Visual Studio de la couche [ui] 73
8.9.2 Configuration de l'application 74
8.9.3 L'interface graphique 77

9 L'APPLICATION [SIMUPAIE] – VERSION 7 – ASP.NET / 3 COUCHES 79

9.1 La couche [ui] 79
9.2 Le projet Visual Web Developer de la couche [web-ui] 79
9.3 Configuration de l'application 80
9.4 Le formulaire [Default.aspx] 80

10 L'APPLICATION [SIMUPAIE] – VERSION 8 – AJAX / ASP.NET 82

10.1 Le projet Visual Web Developer de la couche [web-ui-ajax] 82

11 L'APPLICATION [SIMUPAIE] – VERSION 9 – ASP.NET / MULTI-VUES / MONO-PAGE 84

11.1 Les vues de l'application 84
11.2 Le projet Visual Web Developer de la couche [web] 86
11.3 Le fichier [MyApplication.vb] 87
11.4 La classe [Simulation] 87
11.5 La page [Default.aspx] 88
11.5.1 Vue d'ensemble 88
11.5.2 L'entête 89
11.5.3 La vue [Saisies] 89
11.5.4 La vue [Simulation] 90
11.5.5 La vue [Simulations] 90
11.5.6 La vue [SimulationsVides] 92
11.5.7 La vue [Erreurs] 92
11.6 Le contrôleur [Default.aspx.vb] 93
11.6.1 Vue d'ensemble 93
11.6.2 L'événement Load 95
11.6.3 Action : Faire la simulation 95
11.6.4 Action : Enregistrer la simulation 97
11.6.5 Action : Retour au formulaire de simulation 97
11.6.6 Action : Effacer la simulation 98
11.6.7 Action : Voir les simulations 98
11.6.8 Action : Supprimer une simulation 99
11.6.9 Action : Terminer la session 100
11.7 Ajaxification de l'application 101

12 L'APPLICATION [SIMUPAIE] – VERSION 10 – ASP.NET / MULTI-VUES / MULTI-PAGES 102

12.1 Les vues de l'application 103
12.2 Génération des vues dans un contexte multi-contrôleurs 104
12.2.1 Cas 1 : une page contrôleur / vue 104
12.2.2 Cas 2 : une page 1 contrôleur, une page 2 controleur / vue 106
12.3 Le projet Visual Web Developer de la couche [web] 107

12.4 Le code de présentation des pages 108
12.4.1 La page Maître [MasterPage.master] 108
12.4.2 La page [Formulaire.aspx] 112
12.4.3 La page [Simulations.aspx] 113
12.4.4 La page [Erreurs.aspx] 114
12.5 Le code de contrôle des pages 115
12.5.1 Vue d'ensemble 115
12.5.2 Code de contrôle de la page [MasterPage.master] 116
12.5.2.1 Squelette de la classe 116
12.5.2.2 Propriétés publiques de la classe 116
12.5.2.3 La méthode SetMenu 117
12.5.2.4 La gestion des événements de la page maître 117
12.5.2.5 L'événement Init de la page 118
12.5.2.6 L'événement Click sur le lien [LinkButtonTerminerSession] 121
12.5.3 Code de contrôle de la page [Erreurs.aspx] 122
12.5.4 Code de contrôle de la page [Formulaire.aspx] 122
12.5.4.1 Squelette de la classe 122
12.5.4.2 Evénement Load de la page 123
12.5.4.3 Gestion des événements Click sur les liens du menu 124
12.5.5 Code de contrôle de la page [Simulations.aspx] 125
12.5.6 Code de contrôle de la page [Default.aspx] 125
12.6 Ajaxification de l'application 126