Construction d'applications à trois couches avec

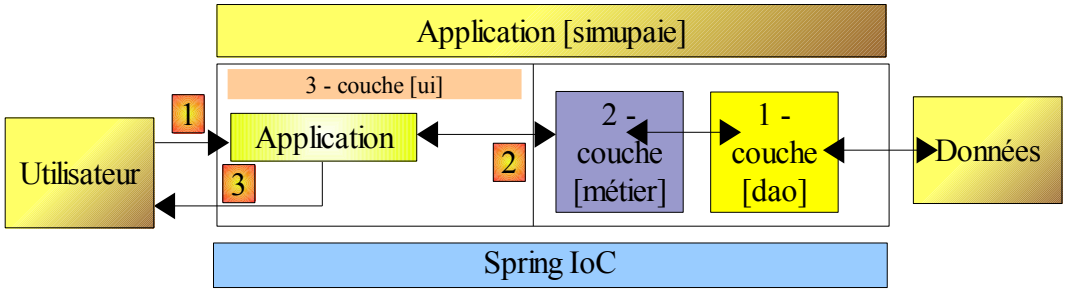
C# et ASP.Net 2008
Spring.Net, iBatis.Net

serge.tahe@istia.univ-angers.fr, mai 2008

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 neuf versions de cette application :

1. une versions windows C# 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. de nouveau une versions windows C# qui utilise des classes pour encapsuler les informations de la base de données et met en cache les données les plus demandées de la base de données afin d'améliorer les performances.
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 version ASP.NET 4 avec des extensions Ajax
6. une versions windows C# 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. une version web ASP.NET multi-vues et mono-page.
9. 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 C# 2008** : [http://tahe.developpez.com/dotnet/csharp/] : 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 3 couches implémente le modèle MVC.

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

Outils

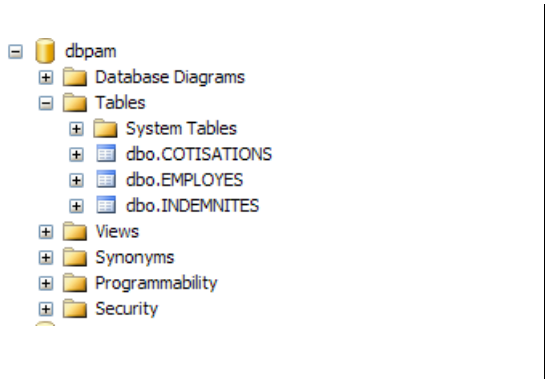
Les outils utilisés dans cet étude de cas sont librement disponibles sur le web. Ce sont les suivants :

- **Visual C# 2008, Visual Web Developer Express 2008**, **SQL Server Express 2005** disponibles à l'url [http://www.microsoft.com/france/msdn/vstudio/express/default.mspx].
- **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.

2 L'étude de cas

2.1 La base de données

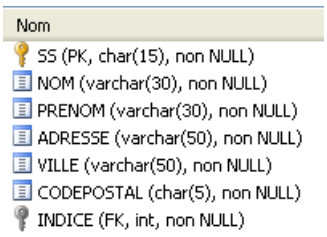
Les données statiques utiles pour construire la fiche de paie sont placées dans une base de données SQL Server Express nommée **dbpam** (**pam**=Paie Assistante Maternelle). Cette base a un administrateur appelé **sa** ayant le mot de passe **msde**.



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 :



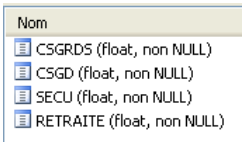
SS	numéro de sécurité sociale de l'employé - clé primaire
NOM	nom de l'employé
PRENOM	son prénom
ADRESSE	son adresse
VILLE	sa ville
CODEPOSTAL	son code postal
INDICE	son indice de traitement - clé étrangère sur le champ [INDICE] de la table [INDEMNITES]

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 :



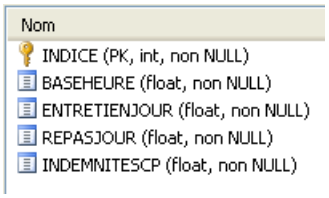
CSGRDS	pourcentage : contribution sociale généralisée + contribution au remboursement de la dette sociale
CSGD	pourcentage : contribution sociale généralisée déductible
SECU	pourcentage : sécurité sociale
RETRAITE	pourcentage : retraite complémentaire + assurance chômage

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



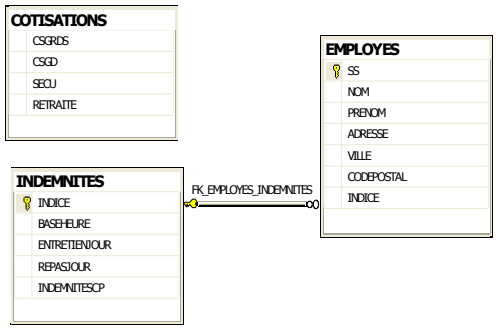
INDICE	indice de traitement - clé primaire
BASEHEURE	prix net en euro d'une heure de garde
ENTRETIENJOUR	indemnité d'entretien en euro par jour de garde
REPASJOUR	indemnité de repas en euro par jour de garde
INDEMNITESCP	indemnité de congés payés. C'est un pourcentage à appliquer au salaire de base.

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

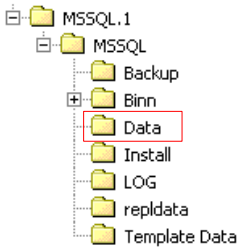
On notera que les indemnités peuvent varier d'une assistante maternelle à une autre. Elles sont en effet associées à une assistante maternelle précise via l'indice de traitement de celle-ci. Ainsi *Mme Marie Jouveinal* qui a un indice de traitement de 2 (table EMPLOYES) a un salaire horaire de 2,1 euros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

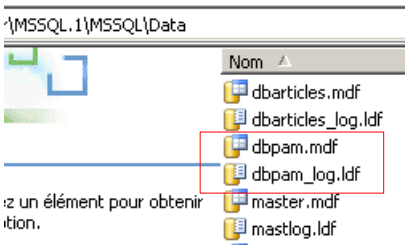
Les relations entre les trois tables sont les suivantes :



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

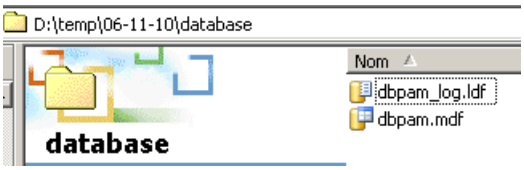
La base de données [dbpam] ainsi créée donne naissance à deux fichiers dans le dossier de SQL Server Express :



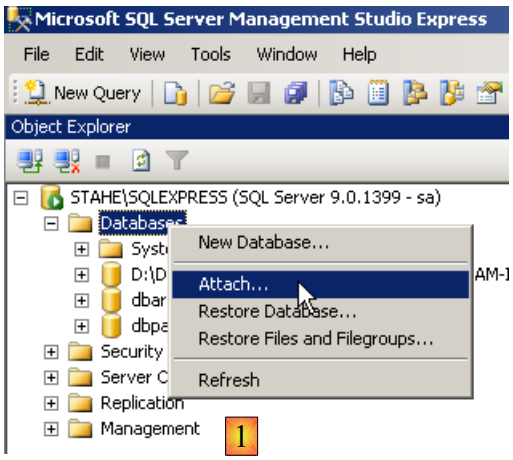


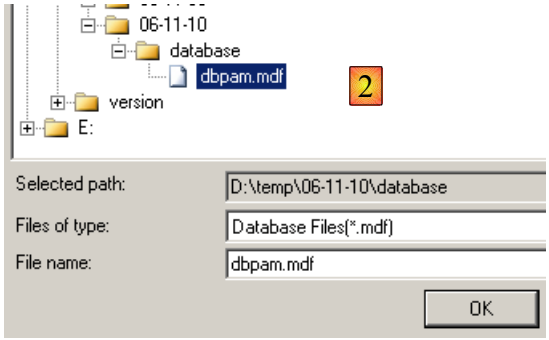
Les fichiers [dbpam.mdf, dbpam_log.ldf] peuvent être transportés sur une autre machine et être réattachés au SGBD SQL Server Express de cette machine. Voici comment procéder :

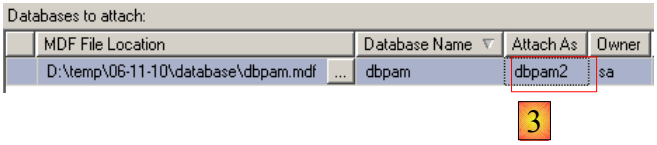
- les fichiers de la BD [dbpam] sont dupliqués dans un dossier

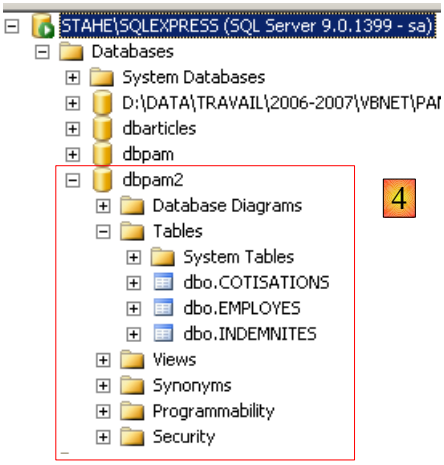


- on lance SQL Server Express
- avec le client SQL Server Management Studio Express, on attache le fichier [dbpam.mdf] au SGBD :









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 utile 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]	CSGRDS : 12,64 CSGD : 22,28

Les éléments suivants sont pris en compte : 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 :	[TOTALHEURES]: total des heures travaillées dans le mois [TOTALJOURS]: total des jours travaillés dans le mois Contribution sociale généralisée déductible : [SALAIREBASE]*[CSGD/100] Sécurité sociale, veuvage, vieillesse : [SALAIREBASE]*[SECU/100] Retraite Complémentaire + AGPF + Assurance Chômage : [SALAIREBASE]*[RETRAITE/100] [INDEMNITÉS]=[TOTALJOURS]*(ENTRETIENJOUR+REPASJOUR)	[TOTALHEURES]=150 [TOTALJOURS]= 20 Sécurité sociale : 34,02 Retraite : 28,55 [INDEMNITES]=104
Total des cotisations sociales :	[COTISATIONSSOCIALES]=[SALAIREBASE]*(CSGRDS+CSGD+SECU+RETRAITE)/100	[COTISATIONSSOCIALES]=97,48
Au final, le salaire net à payer à l'assistante maternelle est le suivant :	[SALAIREBASE]-[COTISATIONSSOCIALES]+[INDEMNITÉS]	[salaire NET]=368,77

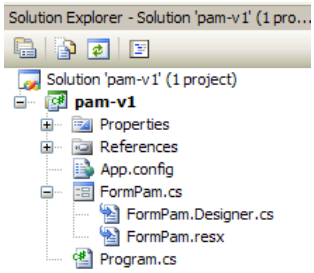
3 L'application [SimuPaie] – version 1 – C#

Lectures conseillées : **Langage C# 2008** : [http://tahe.developpez.com/dotnet/csharp/], chapitre 6 : " Accès aux bases de données "

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

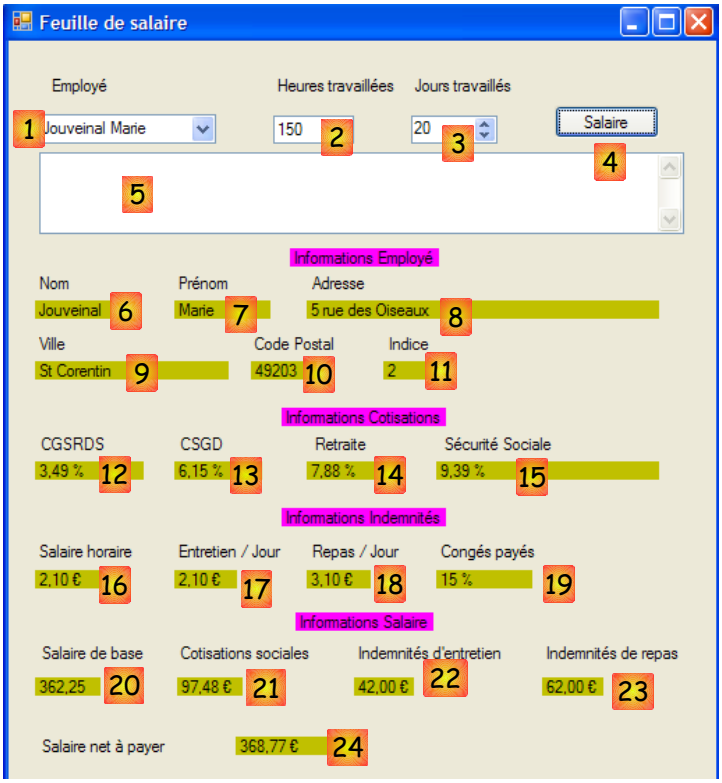
3.1 Le formulaire

Le projet C# pourrait être le suivant :



- **FormPam.cs** : 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	NumericUpDown	numericUpDownJours	Nombre de jours travaillés – nombre entier
4	Button	buttonSalaire	Demande le calcul du salaire
5	TextBox	textBoxMsg	Messages d'informations à destination de l'utilisateur MultiLine=True, ScrollBars=Both, Locked=True
6	Label	labelNom	Nom de l'employé sélectionné en (1)
7	Label	lLabelPrénom	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="dbpamSqlServer2005" connectionString="Data Source=.\SQLEXPRESS;AttachDbFilename=C:\
    data\2007-2008\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="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"/>
9.      <add key="selectEmployes" value="select PRENOM, NOM, SS from EMPLOYES"/>
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.

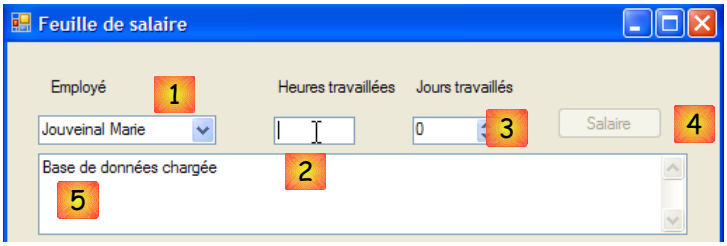
- ligne 8 : l'ordre SELECT pour obtenir l'identité de tous les employés
- ligne 9 : l'ordre SELECT pour obtenir les informations concernant un employé particulier
- ligne 10 : l'ordre SELECT pour obtenir les taux de cotisations

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

```
1.  ...
2.  public partial class FormPam : Form {
3.
4.      // données du formulaire
5.      private string connectionString;
6.      private string selectEmploye;
7.      private string selectEmployes;
8.      private string selectCotisations;
9.      private string msgErreur = null;
10.     ...
11.
12.     public FormPam() {
13.        InitializeComponent();
14.        // configuration
15.        try {
16.           // chaîne de connexion
17.           connectionString =
    ConfigurationManager.ConnectionStrings["dbPamSqlServer2005"].ConnectionString;
18.           // ordres select
19.           selectEmploye = ConfigurationManager.AppSettings["selectEmploye"];
20.           selectEmployes = ConfigurationManager.AppSettings["selectEmployes"];
21.           selectCotisations = ConfigurationManager.AppSettings["selectCotisations"];
22.        } catch {
23.           // configuration erronée
24.           msgErreur = "Erreur de configuration dans [App.config]";
25.        }
26.     }
27. ...
```

3.3 Démarrage de l'application

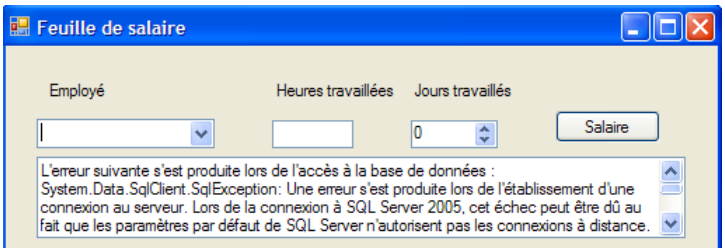
Lorsque l'application VB démarre, elle va chercher dans la base de données décrite au paragraphe 2.1, page 3, les noms des employés pour les mettre dans le ComboBox [1] :



- le ComboBox [1] a été rempli
- le bouton [4] est désactivé. Il sera activé lorsque le champs [2] sera non vide.

- le champ [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 champ [5] l'indique :



Question 1 : écrire la méthode *FormPam_Load* qui, exécutée au démarrage de l'application, garantit le fonctionnement précédent.

3.4 Calcul du salaire

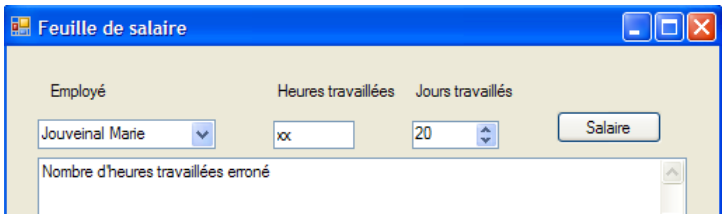
Pour calculer un salaire, l'utilisateur :

- 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 void buttonSalaire_Click(object sender, System.EventArgs e) {...}
```

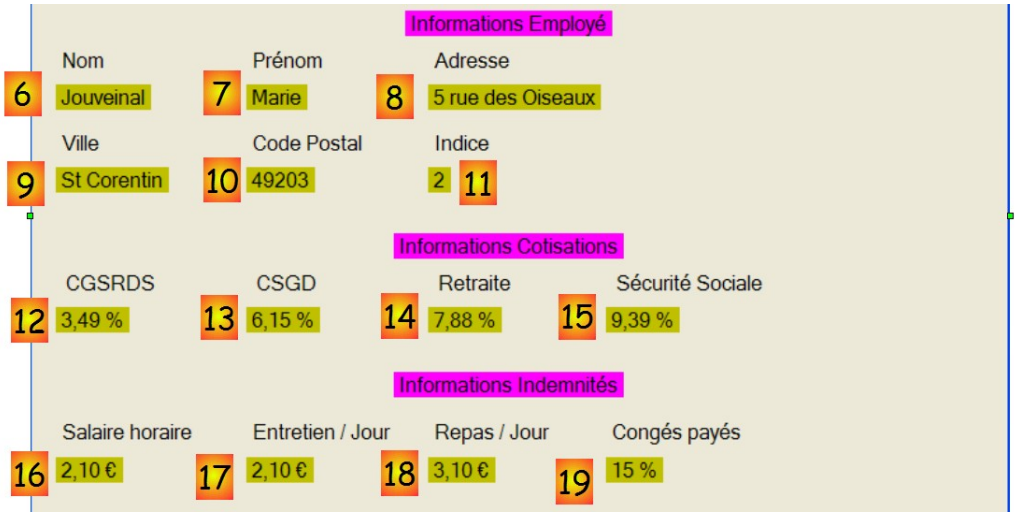
Le gestionnaire [buttonSalaire_Click] commence par vérifier la validité de la saisie faite en [2]. Si elle est incorrecte, l'erreur est signalée comme ci-dessous :



Question : écrire le code de la procédure [buttonSalaire_Click] qui vérifie la validité du nombre d'heures travaillées.

Une fois la saisie [2] trouvée 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 : 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 : écrire le code de la procédure [buttonSalaire_Click] qui va permettre d'obtenir les informations [20] à [24] ci-dessous.

Question : écrire le gestionnaires d'événement permettant de gérer l'état du bouton [Salaire] qui doit être inactif lorsque le champs (1) est vide ou blanc.

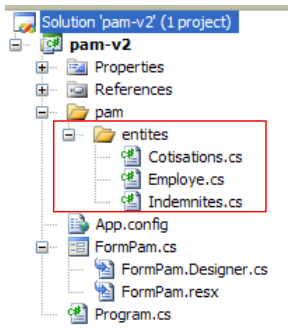
4 L'application [SimuPaie] – version 2 – C#

Cette version modifie la version précédente de la façon suivante :

- elle utilise des classes pour encapsuler les informations de la base de données
- elle met en cache les données les plus demandées de la base de données afin d'améliorer les performances

4.1 Le projet C#

Le nouveau projet C# 2008 est le suivant :



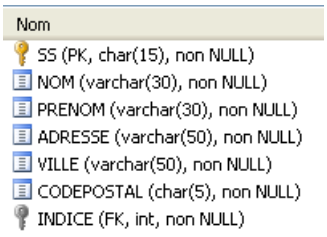
- **FormPam.cs** : le formulaire de l'application précédente
- **App.config** : le fichier de configuration de l'application
- [**Cotisations.cs**, **Employe.cs**, **Indemnites.cs**] : des classes destinées à encapsuler les données des tables [COTISATIONS, EMPLOYES, INDEMNITES]

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 [Employe]

Un objet de type [Employe] 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 [Employe] est défini comme suit :

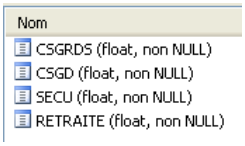
```
1.  namespace Pam.Entites {
2.   public class Employe {
3.      // propriétés automatiques
4.      public string SS { get; set; }
5.      public string Nom { get; set; }
```

```
6.      public string Prenom { get; set; }
7.      public string Adresse { get; set; }
8.      public string Ville { get; set; }
9.      public string CodePostal { get; set; }
10.     public int Indice { get; set; }
11.
12.     // constructeurs
13.     public Employe() {
14.     }
15.
16.     public Employe(string ss, string nom, string prenom, string adresse, string codePostal, string
    ville, int indice) {
17.         this.SS = ss;
18.         this.Nom = nom;
19.         this.Prenom = prenom;
20.         this.Adresse = adresse;
21.         this.CodePostal = codePostal;
22.         this.Ville = ville;
23.         this.Indice = indice;
24.     }
25.
26.     // ToString
27.     public override string ToString() {
28.         return string.Format("[{0},{1},{2},{3},{4},{5},{6}]", SS, Nom, Prenom, Adresse, Ville,
    CodePostal, Indice);
29.     }
30.  }
31. }
```

- ligne 1 : les entités de l'application appartiendront toutes à l'espace de noms [Pam.Entites]
- lignes 4-10 : les propriétés publiques qui reçoivent les valeurs des colonnes de même nom d'une ligne de la table [EMPLOYES]
- lignes 13-24 : les deux constructeurs
- lignes 27-29 : la méthode [ToString] de la classe

4.2.2 La classe [Cotisations]

Un objet de type [Cotisations] 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 [Cotisations] est défini comme suit :

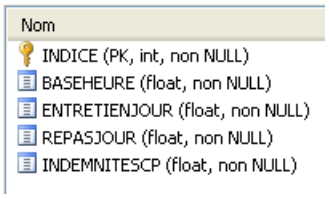
```
1.  namespace Pam.Entites {
2.   public class Cotisations {
3.      // propriétés automatiques
4.      public double CsgRds { get; set; }
5.      public double Csgd { get; set; }
6.      public double Secu { get; set; }
7.      public double Retraite { get; set; }
8.
9.      // constructeurs
10.     public Cotisations() {
11.
12.     }
13.
14.     public Cotisations(double csgRds, double csgd, double secu, double retraite) {
15.         this.CsgRds = csgRds;
16.         this.Csgd = csgd;
17.         this.Secu = secu;
18.         this.Retraite = retraite;
19.     }
20.
21.     // ToString
22.     public override string ToString() {
23.         return string.Format("[{0},{1},{2},{3}]", CsgRds, Csgd, Secu, Retraite);
```

```
24.     }
25.  }
26.
27. }
```

- ligne 1 : la classe appartient à l'espace de noms [Pam.Entites]
- lignes 4-7 : les propriétés publiques qui reçoivent les valeurs des colonnes de même nom d'une ligne de la table [COTISATIONS]
- lignes 10-19 : les deux constructeurs
- lignes 22-24 : la méthode [ToString] de la classe

4.2.3 La classe [Indemnites]

Un objet de type [Indemnites] 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 [Indemnites] est défini comme suit :

```
1.  namespace Pam.Entites {
2.   public class Indemnites {
3.
4.      // champs privés
5.      public int Indice { get; set; }
6.      public double BaseHeure { get; set; }
7.      public double EntretienJour { get; set; }
8.      public double RepasJour { get; set; }
9.      public double IndemnitesCp { get; set; }
10.
11.     // constructeurs
12.     public Indemnites() {
13.     }
14.
15.     public Indemnites(int indice, double baseHeure, double entretienJour, double repasJour, double
    indemnitesCP) {
16.         this.Indice = indice;
17.         this.BaseHeure = baseHeure;
18.         this.EntretienJour = entretienJour;
19.         this.RepasJour = repasJour;
20.         this.IndemnitesCp = indemnitesCP;
21.     }
22.
23.     // identité
24.     public override string ToString() {
25.         return string.Format("[{0}, {1}, {2}, {3}, {4}]", Indice, BaseHeure, EntretienJour,
    RepasJour, IndemnitesCp);
26.     }
27.  }
28. }
```

- ligne 1 : la classe appartient à l'espace de noms [Pam.Entites]
- lignes 5-9 : les propriétés publiques qui reçoivent les valeurs des colonnes de même nom d'une ligne de la table [INDEMNITES]
- lignes 12-21 : les deux constructeurs
- lignes 24-26 : 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.   <connectionStrings>
4.      <add name="dbpamSqlServer2005" connectionString="Data Source=.\SQLEXPRESS;AttachDbFilename=C:\
    data\...\dbpam.mdf;User Id=sa;Password=msde;Connect Timeout=30;"
5.           providerName="System.Data.SqlClient" />
6.   </connectionStrings>
7.   <appSettings>
8.      <add key="selectEmploye" value="select NOM,PRENOM,ADRESSE,VILLE,CODEPOSTAL,INDICE from
    EMPLOYES where SS=@SS"/>
9.      <add key="selectEmployes" value="select PRENOM, NOM, SS from EMPLOYES"/>
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>
```

L'initialisation du formulaire est faite dans la méthode [FormPam_Load] :

```
1.  namespace Pam {
2.   public partial class FormPam : Form {
3.
4.      // données du formulaire
5.      private Employe[] employes;
6.      private Cotisations cotisations;
7.      private Dictionary<int, Indemnites> indemnites = new Dictionary<int, Indemnites>();
8.      private string connectionString = null;
9.
10.     public FormPam() {
11.        InitializeComponent();
12.     }
13.
14.
15.     // chargement formulaire
16.     private void FormPam_Load(object sender, EventArgs e) {
17. ...
18.        try {
19.              // chargement des identités des employés dans un tableau
20. ...
21.              // informations sur les taux de cotisation dans l'objet cotisations
22. ...
23.              // dictionnaire des indemnites
24. ...
25.        } catch (Exception ex) {
26.           // on note l'erreur
27. ...
28.          return;
29.        }
30.        // chargement des noms des employés dans le combo
31. ...
32.        // message de réussite
33. ...
34.     }
35.
36.     // calcul du salaire
37.     private void buttonSalaire_Click(object sender, System.EventArgs e) {
38. ...
39.     }
40.
41.     private void textBoxHeures_TextChanged(object sender, EventArgs e) {
42. ...
43.     }
44.
45.  }
46. }
```

Le rôle de la procédure [FormPam_Load] 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

- ligne 5 : le tableau d'objets de type [Employe] qui mémorisera la liste simplifiée (SS, NOM, PRENOM) de tous les employés

- ligne 6 : l'objet de type [Cotisations] qui mémorisera les taux de cotisations
- ligne 7 : 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 [Indemnites]
- ligne 8 : la chaîne de connexion à la base de données.

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

4.4 Le code du formulaire [FormPam.cs]

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

- dans la version précédente, le constructeur faisait un certain nombre d'initialisations désormais faites dans la procédure [FormPam_Load]. 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 gérons. C'est elle qui assure l'initialisation de l'application. Si celle-ci se termine par une erreur, un message d'erreur est affiché et la gestion de l'événement *Load* s'arrête là. S'il y a pas d'erreur, le combo doit être rempli avec le contenu du tableau [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 [FormPam_Load] et [buttonSalaire_Click]

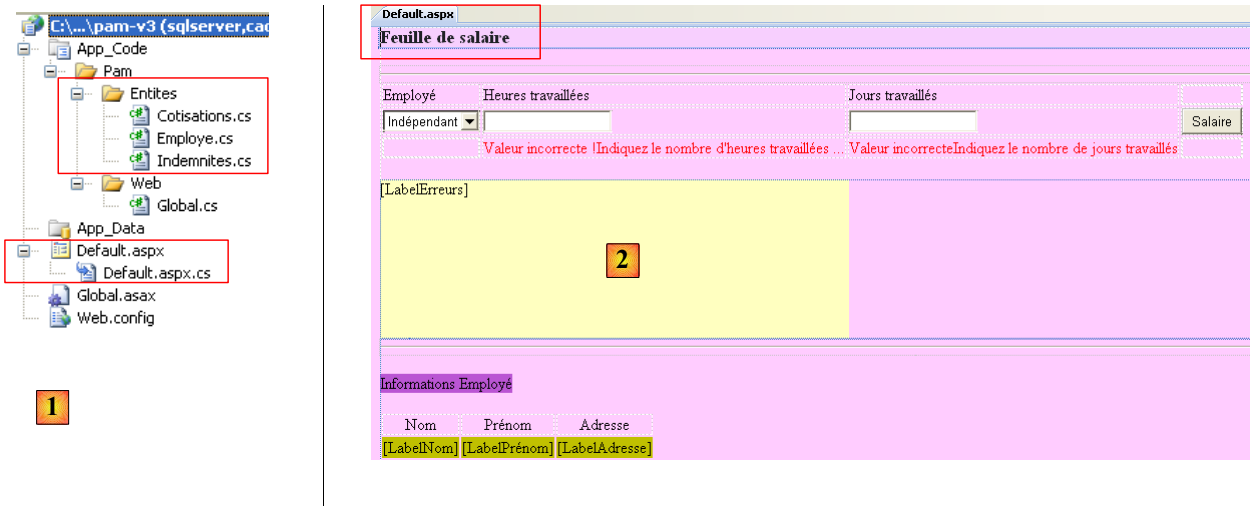
Travail pratique : mettre en oeuvre sur machine cette application.

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

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

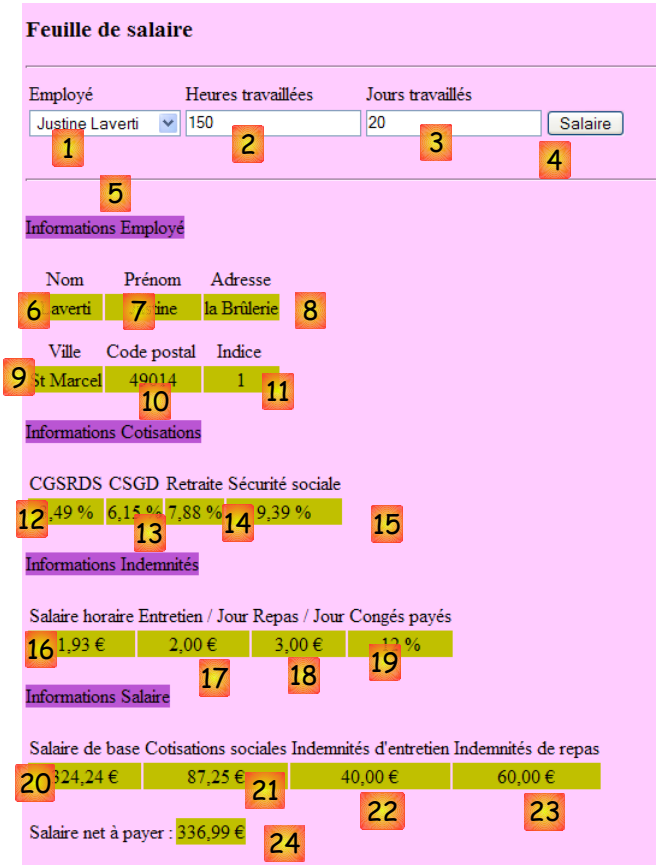
Nous reprenons l'application précédente en lui donnant maintenant une interface web.

5.1 Le projet Visual Web Developer 2008



- en [1], le projet Visual Web Developer
- en [2], la page [Default.aspx] en mode conception

Le formulaire [Default.aspx] est le suivant :



Les composants ont gardé le même nom que dans l'application C# 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)

N°	Type	Nom	Rôle
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*.

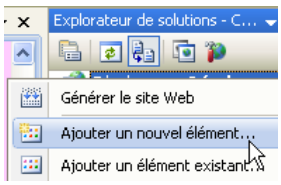
5.2 Configuration de l'application

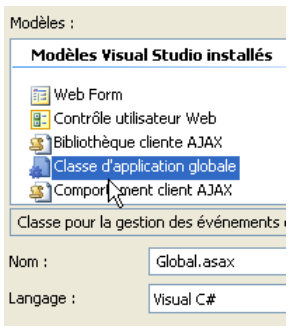
Lectures conseillées : référence [1], paragraphe 3.1 " La notion d'application web ASP.NET " dans ASP.NET, vol1 [1], pages 49-70.

Le fichier [Web.config] qui configure l'application est analogue au fichier [App.config] configurant l'application C# de la version 2 étudiée au paragraphe 4.3, page 15. Il est exploité au démarrage de l'application par le code du couple [Global.asax, Global.asax.cs] :

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="C#" %>
2.
3.  <script runat="server">
4.
5.      void Application_Start(object sender, EventArgs e)
6.      {
7.          // Code qui s'exécute au démarrage de l'application
8.
9.      }
10.
11.     void Application_End(object sender, EventArgs e)
12.     {
13.         //  Code qui s'exécute à l'arrêt de l'application
14.
15.     }
```

```
16.
17.     void Application_Error(object sender, EventArgs e)
18.     {
19.         // Code qui s'exécute lorsqu'une erreur non gérée se produit
20.
21.     }
22.
23.     void Session_Start(object sender, EventArgs e)
24.     {
25.         // Code qui s'exécute lorsqu'une nouvelle session démarre
26.
27.     }
28.
29.     void Session_End(object sender, EventArgs e)
30.     {
31.         // Code qui s'exécute lorsqu'une session se termine.
32.         // Remarque : l'événement Session_End est déclenché uniquement lorsque le mode
    sessionstate
33.         // a la valeur InProc dans le fichier Web.config. Si le mode de session a la valeur
    StateServer
34.         // ou SQLServer, l'événement n'est pas déclenché.
35.
36.     }
37.
38. </script>
```

- lignes 5-9 : la procédure exécutée au démarrage de l'application
- lignes 11-15 : la procédure exécutée à la fin de l'application, lorsque le serveur web la décharge de la mémoire
- lignes 17-21 : la procédure exécutée lorsqu'une exception remonte jusqu'au serveur web
- lignes 23-27 : la procédure exécutée lors de l'arrivée d'un nouvel utilisateur (absence du cookie de session)
- lignes 29-36 : 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 (lignes 1, 3) et code C# (lignes 5-9). Ecrire du code dans ce fichier est malaisé. Ainsi on écrira :

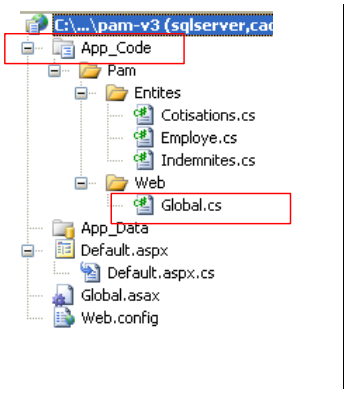
```
1.  <%@ Application Language="C#" %>
2.  <%@ Import Namespace="System.Collections.Generic" %>
3.
4.  <script runat="server">
5.
6.    List<String> liste = new List<String>();
7.
8.      void Application_Start(object sender, EventArgs e)
9.      {
10.         // Code qui s'exécute au démarrage de l'application
11.
12.     }
13. ...
```

- 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="C#" inherits="Pam.Web.Global"  %>
```

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



La classe [Pam.Web.Global] a été placée dans le fichier [Global.cs] ci-dessus, lequel est dans un dossier dont le chemin est identique à l'espace de noms de la classe. Il n'y a aucune obligation à cela. La classe [Pam.Web.Global] est la suivante :

```
1.  using System;
2.  using System.Collections.Generic;
3.  ...
4.  using Pam.Entites;
5.
6.  namespace Pam.Web
7.  {
8.      public class Global : HttpApplication
9.      {
10.
11.         // --- données statiques de l'application ---
12.         public static Employe[] Employes;
13.         public static Cotisations Cotisations;
14.         public static Dictionary<int, Indemnites> Indemnites = new Dictionary<int, Indemnites>();
15.         public static string Msg = string.Empty;
16.         public static bool Erreur = false;
17.         public static string ConnectionString = null;
18.
19.         // démarrage de l'application
20.         public void Application_Start(object sender, EventArgs e)
21.         {
22. ....
23.         }
24.     }
25. }
```

- ligne 8 : la classe [Global] (on aurait pu l'appeler autrement) dérive de la classe [HttpApplication]. Les pages web d'une application ASP.NET ont 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="Pam.Web.Global" %>
```

- lignes 20-23 : 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 [Global] sont créées, elles partagent toutes le même champ statique [Employes], accessible via la référence [Global.Employes], c.a.d. *[NomDeClasse].ChampStatique*. On notera la différence entre les deux notations suivantes :
  - [Global] 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 par défaut dans toute application ASP.NET une référence sur une instance de la classe [Global] ci-dessus.

Revenons à notre classe [Global]. 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 [Global], 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 quelconque de l'application web de la façon suivante :

```
Employe[] employes=Application.Item["employes"] as Employe;
```

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 mai partager des données via des champs statiques typés de l'objet [Global] 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.

Les données partagées par tous les utilisateurs sont les suivantes :

- ligne 12 : le tableau d'objets de type [Employe] qui mémorisera la liste simplifiée (SS, NOM, PRENOM) de tous les employés
- ligne 13 : l'objet de type [Cotisations] qui mémorisera les taux de cotisations
- ligne 14 : 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 [Indemnites]
- ligne 15 : un message indiquant comment s'est terminée l'initialisation (bien ou avec erreur)
- ligne 16 : un booléen indiquant si l'initialisation s'est terminée par une erreur ou non.
- ligne 17 : la chaîne de connexion à la base de données.

Le code de la méthode [Application_Start] est identique à la méthode [FormPam_Load] de la version précédente aux détails près suivants :

- la méthode [Application_Start] ne remplit pas le combo des employés
- elle n'affiche pas de message dans l'interface mais met ce message dans la variable statique [Msg] de la ligne 15

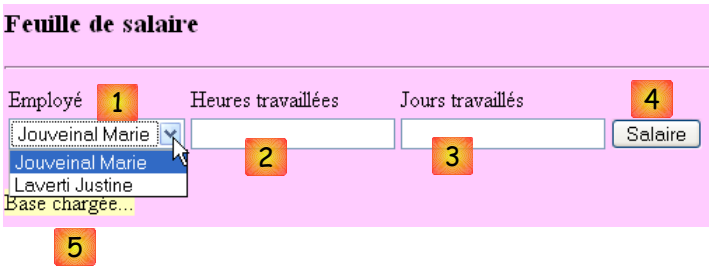
Ces deux différences viennent du fait que la classe [Global] n'est pas associée à une interface visuelle.

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

5.3 Le formulaire [Default.aspx]

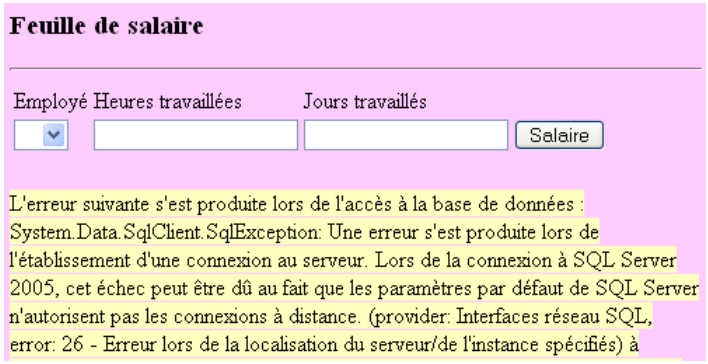
5.3.1 La procédure [Page_Load]

Lorsque le formulaire [Default.aspx] est chargé, il va chercher dans le tableau [Global.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 :



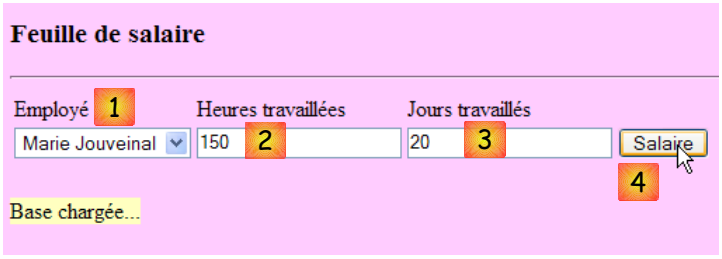
Question : écrire la procédure [Page_Load] de la page web [Default.aspx] qui, exécutée au démarrage de l'application, garantit le fonctionnement précédent. On s'inspirera de la procédure analogue de l'application C# de la version 2.

5.3.2 La vérification des saisies

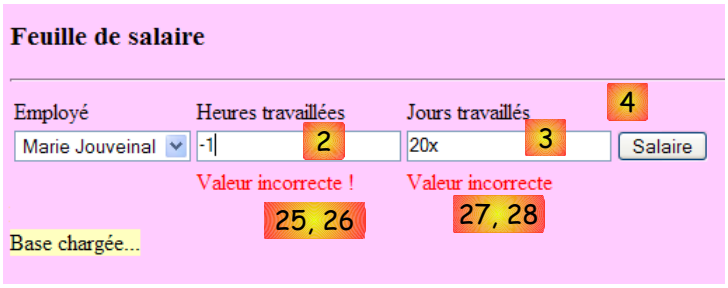
Lectures conseillées : référence [1], paragraphe 2.2 de ASP.NET, vol2 : " Les composants de validation de données ".

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]

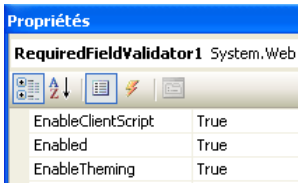


Lorsque l'utilisateur entre des données erronées dans [2] et [3], celles-ci sont vérifiées dès que l'utilisateur change de champ de saisie. Ainsi, la copie d'écran ci-dessous a été obtenue avant même que l'utilisateur ne clique sur le bouton [4] :



Il faut deux conditions pour avoir le comportement précédent :

- le composant de validation doit avoir sa propriété EnableClientScript à vrai :



- le navigateur affichant la page doit être capable d'exécuter le code Javascript embarqué dans une page HTML.

Si le navigateur client ne vérifie pas lui-même la validité des données, celles-ci ne seront vérifiées que lorsque le navigateur postera les saisies du formulaire au serveur. C'est alors le code situé sur ce dernier et qui traite la demande du navigateur qui vérifiera 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 : 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 void ButtonSalaire_Click(object sender, System.EventArgs e)
```

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.

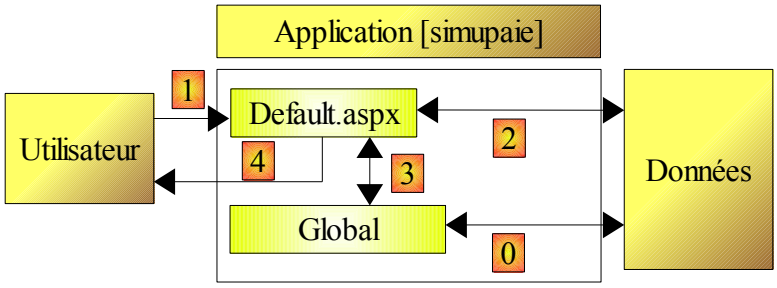
Question : écrire le code de la procédure [ButtonSalaire_Click]. On s'inspirera de la procédure analogue de l'application C# de la version 2.

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

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

6.1 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 [Global] 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 [Global] 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 [Global]. 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.

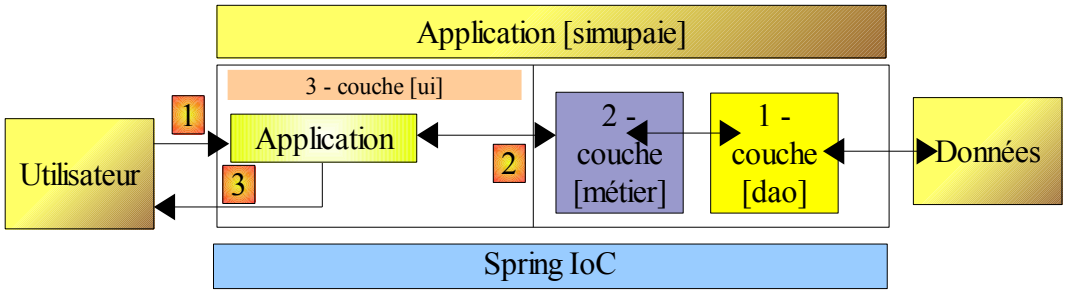
On constate que le code de la page [Default.aspx] fait diverses choses :

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 :

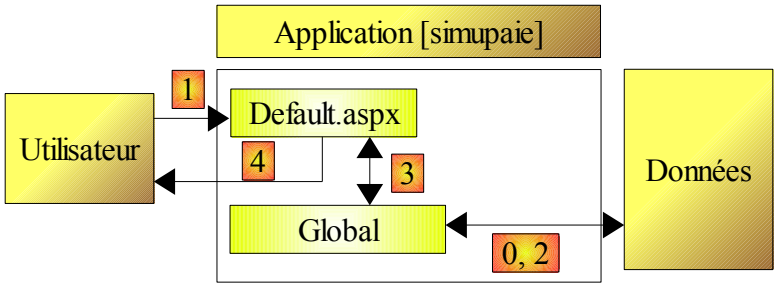


- la couche [1-dao] (dao=Data Access Object) s'occupera de l'accès aux données.
- la couche [2-métier] s'occupera de l'aspect métier de l'application, le calcul de la paie.
- la couche [3-ui] (ui=User Interface) s'occupera de la présentation des données à l'utilisateur et de l'exécution de ses requêtes. 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.

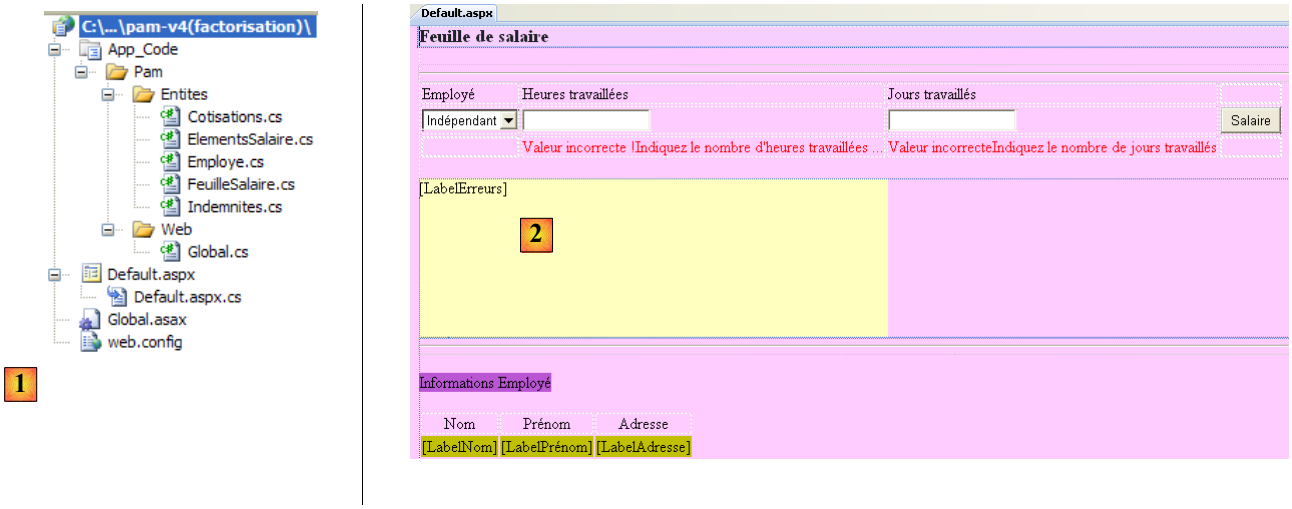
L'architecture précédente est assez complexe et n'est pas préconisée pour une application simple qui doit être développée rapidement par un unique développeur. Nous allons ici adopter une architecture moins complexe :



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 statiques de l'objet [Global]. 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 [Global] 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.

6.2 Le projet Visual Web Developer 2008



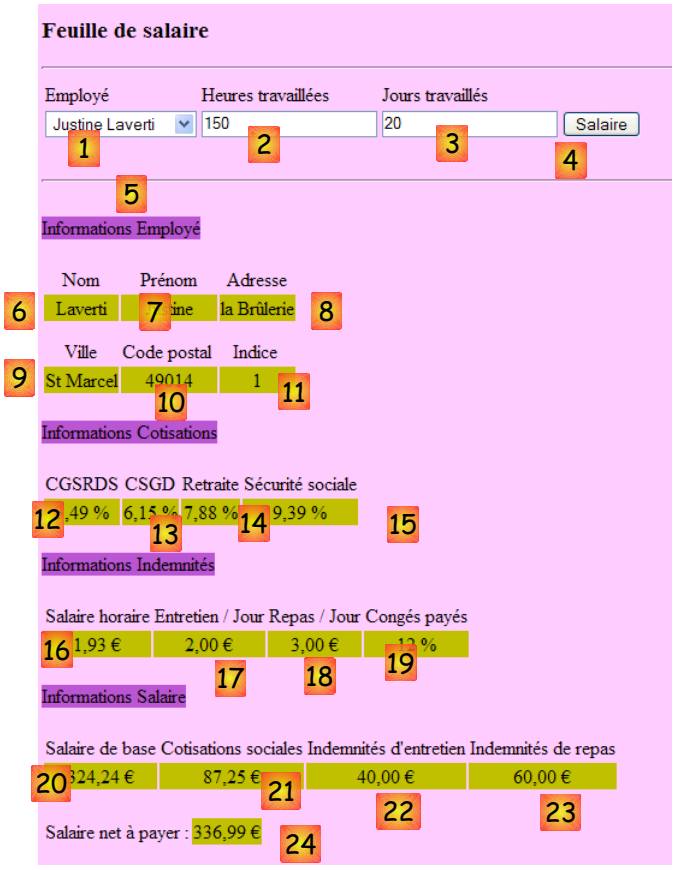
- en [1], apparaissent deux nouvelles entités.
- en [2], le formulaire [Default.aspx] ne change pas d'aspect.

6.2.1 Les entités de l'application web

Le dossier [entites] du projet Visual Studio contient les objets manipulés par :

- les procédures d'accès aux données : [Employe, Cotisations, Indemnites]
- 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.  namespace Pam.Entites
2.  {
3.
4.      public class FeuilleSalaire
5.      {
6.
7.          // propriétés automatiques
8.          public Employe Employe { get; set; }
9.          public Cotisations Cotisations;
10.         public Indemnites Indemnites;
11.         public ElementsSalaire ElementsSalaire;
12.
13.         // constructeurs
14.         public FeuilleSalaire()
15.         {
16.
17.         }
18.
19.         public FeuilleSalaire(Employe employe, Cotisations cotisations, Indemnites indemnites,
    ElementsSalaire elementsSalaire)
20.         {
21.             {
22.                 this.Employe = employe;
23.                 this.Cotisations = cotisations;
24.                 this.Indemnites = indemnites;
25.                 this.ElementsSalaire = elementsSalaire;
26.             }
27.         }
28.
29.         // ToString
30.         public override string ToString()
31.         {
32.             return string.Format("[{0},{1},{2},{3}", Employe, Cotisations, Indemnites,
    ElementsSalaire);
33.         }
34.     }
35. }
```

- 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 14-27 : les constructeurs
- lignes 30-33 : la méthode [ToString]

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

```
1.  namespace Pam.Entites
2.  {
3.      public class ElementsSalaire
4.      {
5.          // propriétés automatiques
6.          public double SalaireBase { get; set; }
7.          public double CotisationsSociales { get; set; }
8.          public double IndemnitesEntretien { get; set; }
9.          public double IndemnitesRepas { get; set; }
10.         public double SalaireNet { get; set; }
11.
12.
13.         // constructeurs
14.         public ElementsSalaire()
15.         {
16.
17.         }
18.
19.         public ElementsSalaire(double salaireBase, double cotisationsSociales, double
    indemnitesEntretien, double indemnitesRepas, double salaireNet)
20.         {
21.                 this.SalaireBase = salaireBase;
22.                 this.CotisationsSociales = cotisationsSociales;
23.                 this.IndemnitesEntretien = indemnitesEntretien;
24.                 this.IndemnitesRepas = indemnitesRepas;
25.                 this.SalaireNet = salaireNet;
26.         }
27.
```

```
28.         // ToString
29.         public override string ToString()
30.         {
31.             return string.Format("[{0} : {1} : {2} : {3} : {4} ]", SalaireBase,
    CotisationsSociales, IndemnitesEntretien, IndemnitesRepas, SalaireNet);
32.         }
33.     }
34. }
```

- lignes 6-10 : les éléments du salaire tels qu'expliqués dans les règles métier décrites page 6.
- ligne 6 : le salaire de base de l'employé, fonction du nombre d'heures travaillées
- ligne 7 : les cotisations prélevées sur ce salaire de base
- lignes 8 et 9 : les indemnités à ajouter au salaire de base, fonction de l'indice de l'employé et du nombre de jours travaillés
- ligne 10 : le salaire net à payer
- lignes 14-26 : les constructeurs
- lignes 29-32 : la méthode [ToString] de la classe.

6.2.2 Configuration de l'application

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

6.2.3 La classe [Global]

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

```
1.  using System;
2.  using System.Collections.Generic;
3.  using System.Configuration;
4.  using System.Data.SqlClient;
5.  using System.Web;
6.  using Pam.Entites;
7.
8.  namespace Pam.Web
9.  {
10.     public class Global : HttpApplication
11.     {
12.
13.         // --- données statiques de l'application ---
14.         public static Employe[] Employes;
15.         public static Cotisations Cotisations;
16.         public static Dictionary<int, Indemnites> Indemnites = new Dictionary<int, Indemnites>();
17.         public static string MsgErreur = string.Empty;
18.         public static bool Erreur = false;
19.         public static string ConnectionString = null;
20.
21.         // démarrage de l'application
22.         public void Application_Start(object sender, EventArgs e)
23.         {
24. ...
25.         }
26.
27.         // calcul du salaire
28.         public static FeuilleSalaire GetSalaire(string ss, double heuresTravaillées, int
    joursTravaillés)
29.         {
30.             // ss : n° SS de l'employé
31.             // heuresTravaillées : le nombre d'heures travaillés
32.             // jours Travaillés : nbre de jours travaillés
33. ...
34.         }
35.
36.         // obtenir un employé
37.         private static Employe getEmploye(string ss)
38.         {
39. ....
40.             }
41.         }
42.     }
43. }
```

- lignes 14-19 : 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 22-25 : la procédure [Application_Start] qui gère l'initialisation de l'application n'a pas changé
- lignes 28-34 : le calcul du salaire n'est plus fait par la page [Default.aspx.vb] mais par la fonction statique [Global.GetSalaire] qui rend un objet de type [FeuilleSalaire].
- lignes 37-41 : 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 [Global].

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

6.2.4 Le formulaire [Default.aspx.cs]

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

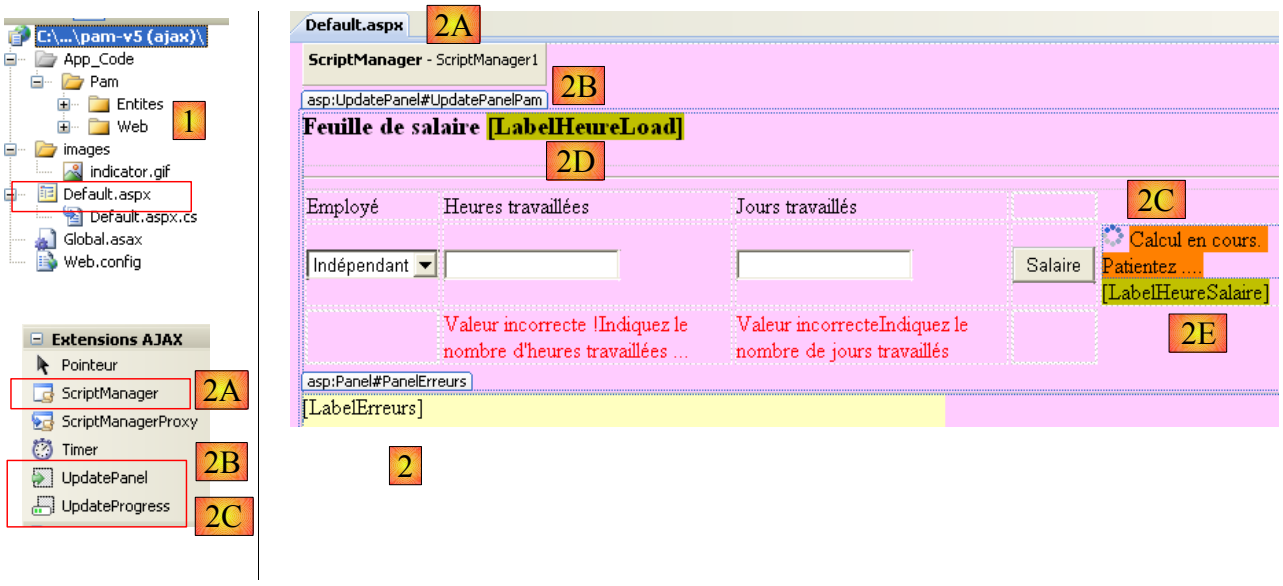
La principale difficulté dans AJAX est le langage Javascript. Créé dès les débuts du web,

- 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, .NET, PHP, Ruby, ... AJAX est intégré dans Web Developer 2008. Pour les versions précédentes, il faut télécharger l'extension AJAX de Microsoft [http://ajax.asp.net] :

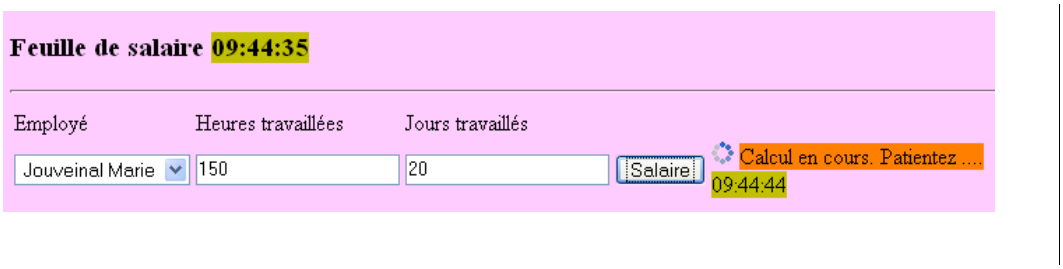
7.2 Le projet Visual Web Developer de la couche [web-ui-ajax]

La nouvelle version est obtenue par duplication de la version précédente. Seule la page [Default.aspx] va être modifiée.



La page [Default.aspx] va être modifiée de la façon suivante :

- 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 comme montré ci-dessous :



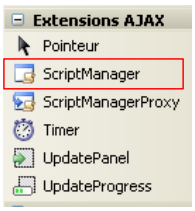
  - 2D : un type *Label* nommé *LabelHeureLoad* qui va afficher l'heure à laquelle s'exécute le gestionnaire de l'événement *Load* de la page.
  - 2E : un type *Label* nommé *LabelHeureSalaire* qui va afficher l'heure à laquelle s'exécute le gestionnaire de l'événement *Click* sur le bouton [Salaire]. On veut montrer qu'Ajax ne recharge pas toute la page lors du clic sur le bouton [Salaire]. C'est ce que montre la copie d'écran précédente où on peut voir deux heures différentes dans les deux *Labels*.

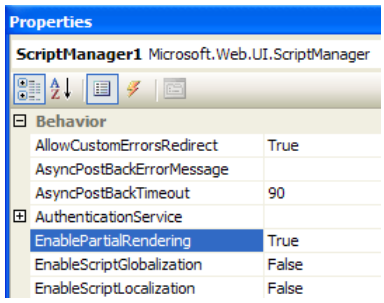
L'Ajaxification précédente du formulaire peut se faire directement dans le code source de [Default.aspx] par l'ajout de balises d'extensions Ajax :

```
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 :



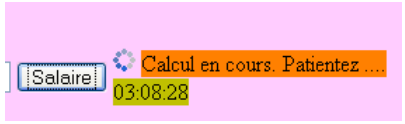


- la balise <*asp:ScriptManager*> peut être obtenue par double-clic sur le composant [ScriptManager] ci-dessus. Il faut prendre soin de vérifier que cette balise est contenue dans la balise <form> du code source. L'attribut [EnablePartialRendering= "true "] de la ligne 8 est absent pas défaut. La valeur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

Le code de la page [Default.aspx.cs] évolue de la façon suivante :

```
1.      protected void Page_Load(object sender, System.EventArgs e)
2.      {
3.          // heure de chaque chargement de page
4.          LabelHeureLoad.Text = DateTime.Now.ToString("hh:mm:ss");
5.          // traitement requête initiale
6.          if (!IsPostBack)
7.          {
8.  ....
9.  }
```

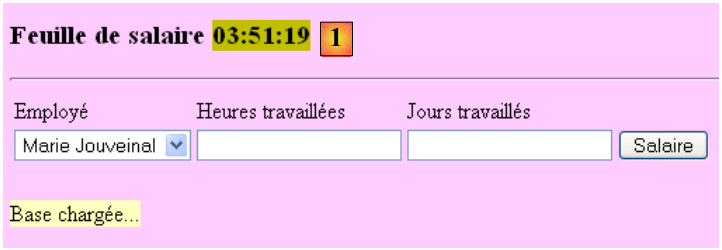
- ligne 4 : mise à jour de l'heure de chargement de la page

```
1.     protected void ButtonSalaire_Click(object sender, System.EventArgs e)
2.      {
3.          // heure calcul salaire
4.          LabelHeureSalaire.Text = DateTime.Now.ToString("hh:mm:ss");
5.          // on vérifie les données
6.  ....
7.  }
```

- ligne 4 : mise à jour de l'heure de calcul du salaire

7.3 Tests de la solution Ajax

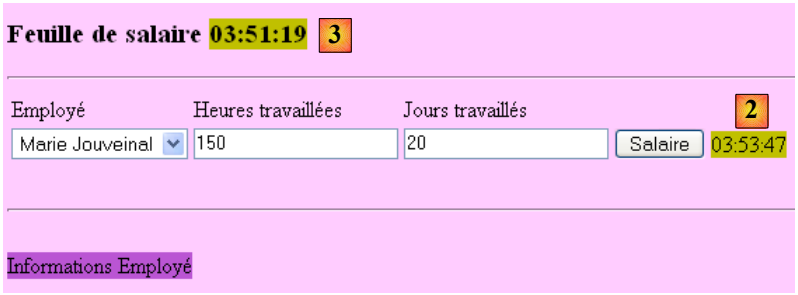
Lorsqu'on demande l'exécution du projet (CTRL-F5), on obtient la page suivante :



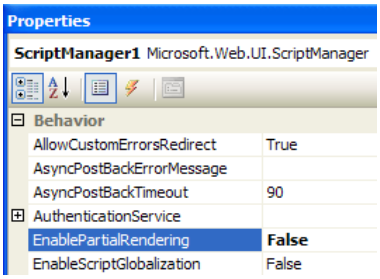
- en [1], l'heure de chargement de la page

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* :

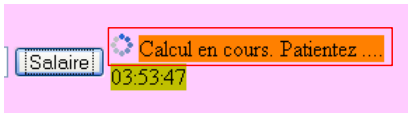


Constater qu'alors on retrouve le comportement d'une page non ajaxifiée. Il y a rechargement total de la page lors du clic sur le bouton [Salaire]. Refaire les tests avec un autre navigateur. Le test multi-navigateurs a pour but de montrer que le javascript généré par les composants serveur ASP / AJAX 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.cs], ajoutons une instruction qui arrête la procédure pendant 3 secondes :

```
1.  using System.Threading;
2.      ...
3.  protected void ButtonSalaire_Click(object sender, System.EventArgs e)
4.      {
5.          // heure calcul salaire
6.          LabelHeureSalaire.Text = DateTime.Now.ToString("hh:mm:ss");
7.          // attente
8.          Thread.Sleep(3000);
9.          // on vérifie les données
10. ....
11. }
```

La ligne 8 fait attendre 5 secondes le thread qui exécute la procédure [ButtonSalaire_Click]. Ceci fait, refaisons des tests (après avoir remis l'attribut *EnablePartialRendering* du composant *ScriptManager1* à True). Cette fois, on voit le texte de [UpdateProgress] ainsi que l'image animée pendant le calcul du salaire :



7.4 Conclusion

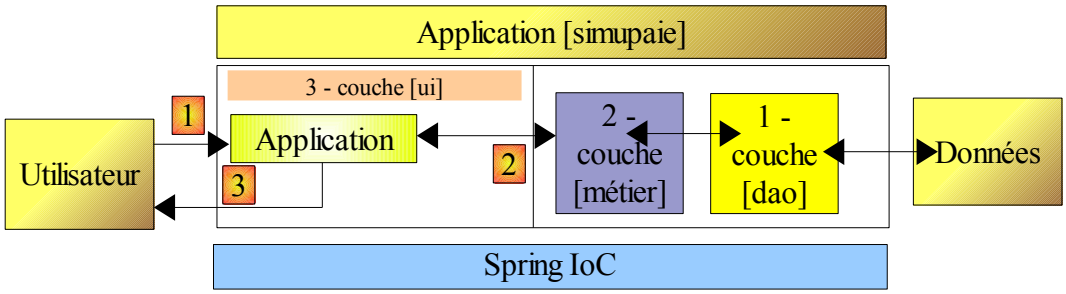
L'étude précédente nous a montré qu'il était possible d'ajaxifier des applications ASP.NET existantes. Les extensions AJAX d'ASP.NET vont beaucoup plus loin que ce qui vient d'être vu. Le lecteur est invité à consulter le site [http://ajax.asp.net].

8 L'application [SimuPaie] – version 6 – C# / 3 couches

Lectures conseillées : "**Langage C# 2008, Chapitre 4 : Architectures 3 couches, tests NUnit, framework Spring**".

8.1 Architecture générale de l'application

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

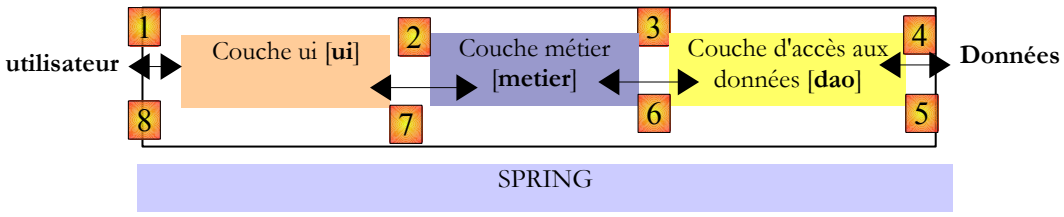


- la couche [1-dao] (dao=Data Access Object) s'occupera de l'accès aux données.
- la couche [2-métier] s'occupera de l'aspect métier de l'application, le calcul de la paie.
- la couche [3-ui] (ui=User Interface) s'occupera de la présentation des données à l'utilisateur et de l'exécution de ses requêtes. 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.

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].

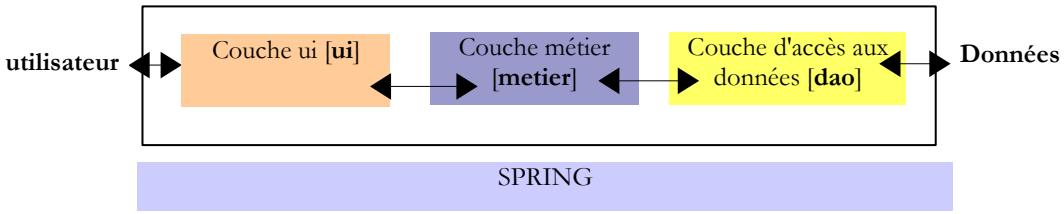
(h) la couche [ui] va mettre en forme ces résultats pour les présenter à l'utilisateur sous une forme appropriée puis les présenter. C'est le chemin [8].

(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.

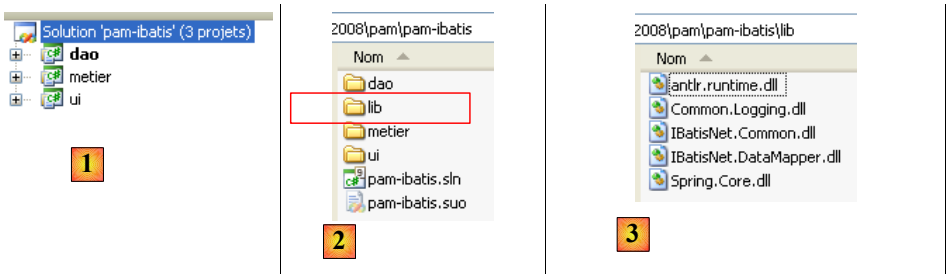
On voit dans cette description qu'une couche est amenée à utiliser les ressources de la couche qui est à sa droite, jamais de celle qui est à sa gauche.

Notre première implémentation de cette architecture 3 couches sera une application C# où la couche [ui] est implémentée par le formulaire de la version 1 (cf page 8).

8.2 La solution Visual Studio C#



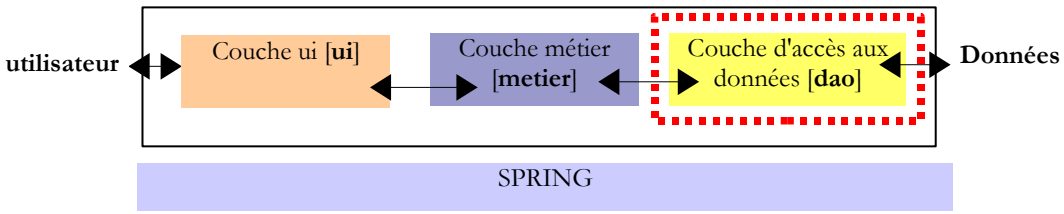
Pour implémenter l'architecture à trois couches ci-dessus, nous utiliserons la solution Visual studio suivante :



- en [1], la solution Visual Studio aura trois projets, un pour chacune des couches de l'application.
- en [2], le dossier de la solution a un dossier [lib] dans lequel ont été rassemblées les DLL nécessaires aux différents projets [3]. Nous reviendrons sur le rôle de celles-ci.

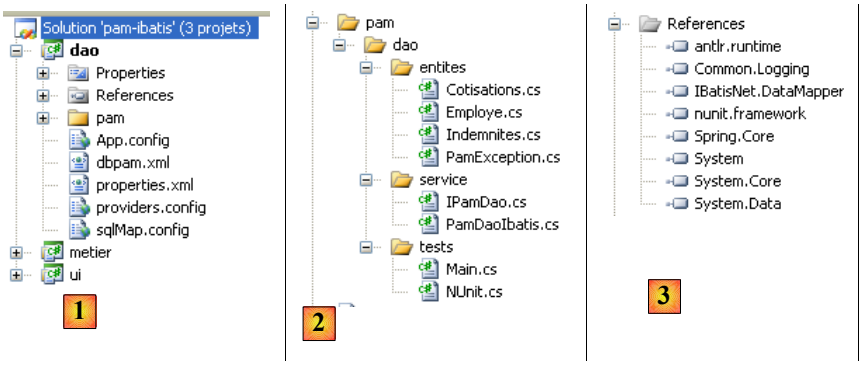
Nous allons construire et tester successivement chacune des trois couches de l'application. Nous commençons par la couche [dao].

8.3 La couche [dao] d'accès aux données



8.3.1 Le projet Visual Studio C# de la couche [dao]

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



- en [1], le projet dans sa globalité
- en [2], les classes du projet
- en [3], les références du projet.

Les entités (objets) nécessaires à la couche [dao] ont été rassemblées dans le dossier [entites] du projet. Certaines nous sont déjà connues : [Cotisations] décrite page 14, [Employe] décrite page 13, [Indemnites] décrite page 15. Elles sont toutes dans l'espace de noms [Pam.Dao.Entites].

La classe [Employe] évolue de la façon suivante :

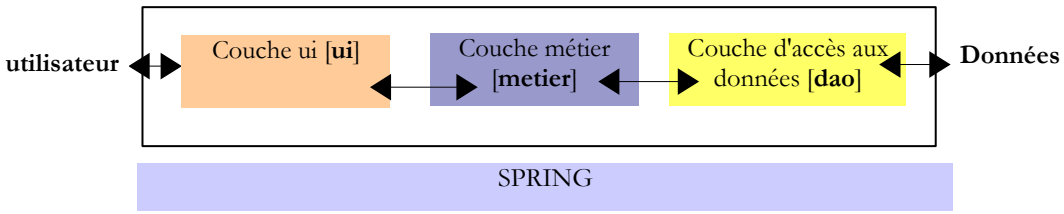
```
1.  namespace Pam.Dao.Entites {
2.   public class Employe {
3.      // propriétés automatiques
4.      public string SS { get; set; }
5.      public string Nom { get; set; }
6.      public string Prenom { get; set; }
7.      public string Adresse { get; set; }
8.      public string Ville { get; set; }
9.      public string CodePostal { get; set; }
10.     public int Indice { get; set; }
11.     public Indemnites Indemnites { get; set; }
12.
13.     // constructeurs
14.     public Employe() {
15.
16.     }
17.
18.     public Employe(string ss, string nom, string prenom, string adresse, string codePostal, string
    ville, int indice) {
19.         this.SS = ss;
20.         this.Nom = nom;
21.         this.Prenom = prenom;
22.         this.Adresse = adresse;
23.         this.CodePostal = codePostal;
24.         this.Ville = ville;
25.         this.Indice = indice;
26.     }
27.
28.     // ToString
29.     public override string ToString() {
```

```
30.         return string.Format("[{0},{1},{2},{3},{4},{5},{6}]", SS, Nom, Prenom, Adresse, Ville,
    CodePostal, Indemnites);
31.     }
32.  }
33. }
```

- ligne 11 : la classe [Employe] embarque désormais les indemnités de l'employé. On aurait pu choisir de dériver la classe originelle décrite page 13 pour arriver au même résultat.

8.3.2 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.  using Pam.Dao.Entites;
2.
3.  namespace Pam.Dao.Service {
4.   public interface IPamDao {
5.      // liste de toutes les identités des employés
6.      Employe[] GetAllIdentitesEmployes();
7.      // un employé particulier avec ses indemnités
8.      Employe GetEmploye(string ss);
9.      // liste de toutes les cotisations
10.     Cotisations GetCotisations();
11.  }
12. }
```

- 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 [Pam.Dao.Service]. Les éléments de l'espace de noms [Pam.Dao.Entites] peuvent être créés en plusieurs exemplaires. Les éléments de l'espace de noms [Pam.Dao.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 :
  - ligne 6, [GetAllIdentitesEmployes] rend un tableau d'objets de type [Employe] qui représente la liste des assistantes maternelles sous une forme simplifiée (nom, prénom, SS).
  - ligne 8, [GetEmploye] rend un objet [Employe] : l'employé qui a le n° de sécurité sociale passé en paramètre à la méthode avec les indemnités liées à son indice.

- ligne 10, [GetCotisations] rend l'objet [Cotisations] qui encapsule les taux des différentes cotisations sociales à prélever sur le salaire brut.

8.3.3 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, GetEmploye, 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 (Exception1 ex1){
...on gère un type d'exceptions
}catch (Exception2 ex2){
...on gère un autre type d'exceptions
}finally{
...
}
```

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

```
1.  using System;
2.  namespace Pam.Dao.Entites {
3.
4.   public class PamException : Exception {
5.
6.      // le code de l'erreur
7.      public int Code { get; set; }
8.
9.      // constructeurs
10.     public PamException() {
11.     }
12.
13.     public PamException(int Code)
14.        : base() {
15.        this.Code = Code;
16.     }
17.
18.     public PamException(string message, int Code)
19.        : base(message) {
20.        this.Code = Code;
21.     }
22.
23.     public PamException(string message, Exception ex, int Code)
24.        : base(message, ex) {
25.        this.Code = Code;
26.     }
27.  }
28. }
```

- ligne 2 : la classe appartient à l'espace de noms [Pam.Dao.Entites]
- ligne 4 : la classe dérive de la classe [Exception]
- ligne 7 : elle a une propriété publique [Code] qui est un code d'erreur
- nous utiliserons dans notre couche [dao] deux sortes de constructeur :
  - celui des lignes 18-21 qu'on peut utiliser comme montré ci-dessous :

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

  - ou celui des lignes 23-26 destiné à faire remonter une exception déjà survenue en l'encapsulant dans une exception de type [PamException] :

```
try{
....
```

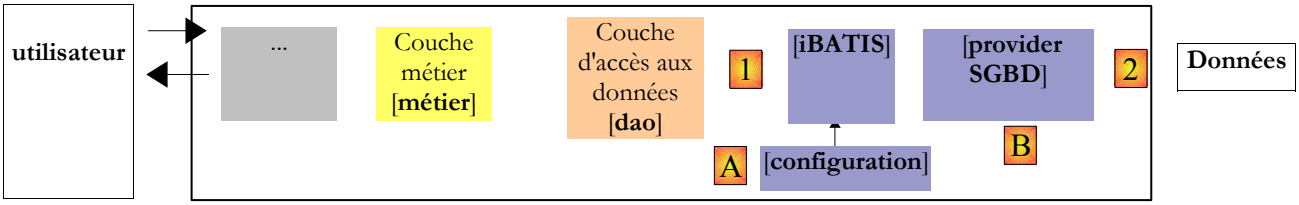
```
}catch (IOException ex){
  // on encapsule l'exception
  throw new PamException("Problème d'accès aux données",ex,10);
}
```

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]

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.



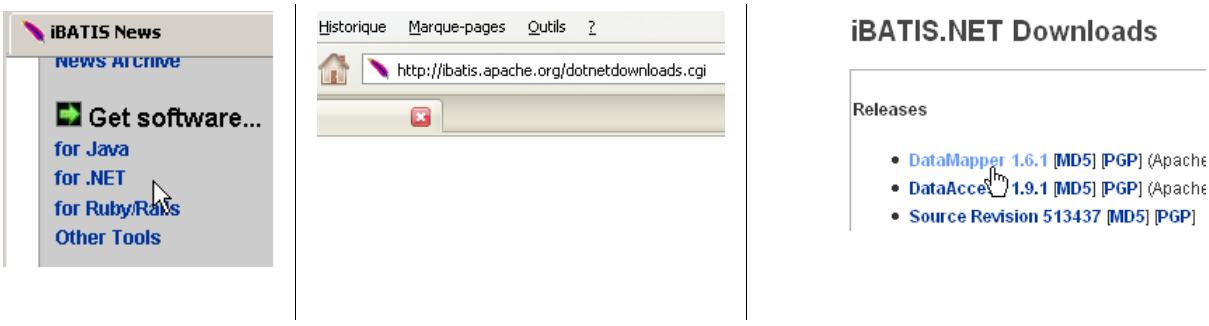
L'accès aux données est assuré à l'aide :

- [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

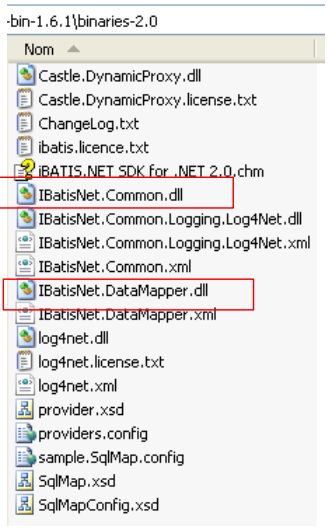
L'outil Ibatis SqlMap a été développé initialement pour la plate-forme Java puis a été porté sur la plate-forme .NET.

8.4.2 Où trouver IBATIS SqlMap ?

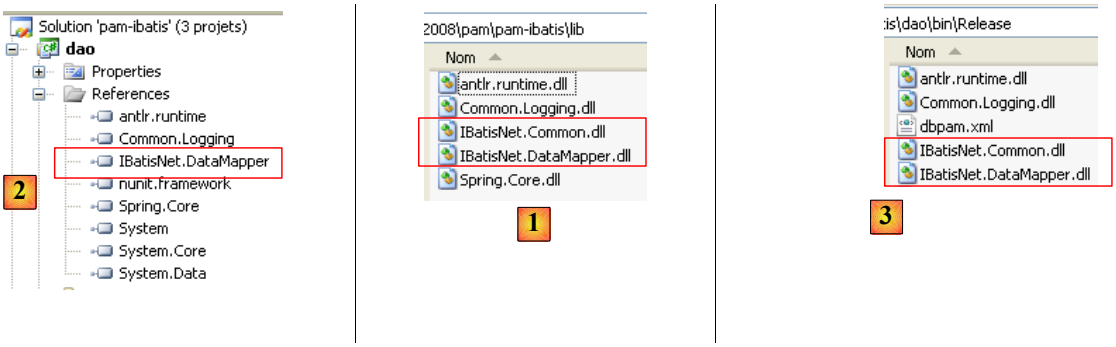
Le site principal d'Ibatis est [http://ibatis.apache.org/]. La page de téléchargements offre les liens suivants :



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]. Celle-ci a une dépendance sur la DLL [IBatisNet.Common]. Celle-ci doit être disponible dans le même dossier que la DLL [IbatisNet.DataMapper]. C'est ce qui est fait pour le projet [dao] :



- en [1], les DLL [IbatisNet.DataMapper] et [IBatisNet.Common] ont été placées dans le dossier [lib] de la solution [pam-ibatis].
- en [2], une référence sur la DLL [IbatisNet.DataMapper] a été créée. Il n'y a pas utilité à créer une référence sur la DLL [IBatisNet.Common] mais celle-ci doit être présente dans le même dossier que la DLL [IbatisNet.DataMapper].
- en [3], Visual studio copie ces deux DLL dans le dossier [bin/Release] du projet.

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.

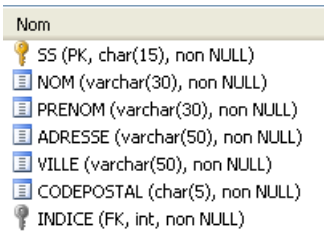
  La connexion à la base sera ouverte par instanciation de la classe [Connection] définie par le *provider* choisi, au constructeur duquel sera passée la chaîne de connexion définie dans [sqlmap.config].
- les **fichiers de mapping** définissent :
  - des associations entre lignes de tables de données 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 :



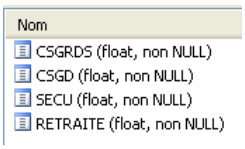
SS	numéro de sécurité sociale de l'employé - clé primaire
NOM	nom de l'employé
PRENOM	son prénom
ADRESSE	son adresse
VILLE	sa ville
CODEPOSTAL	son code postal
INDICE	son indice de traitement - clé étrangère sur le champ [INDICE] de la table [INDEMNITES]

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 :



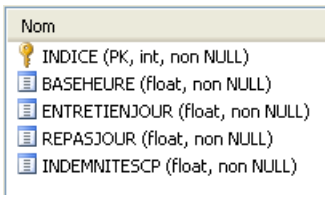
CSGRDS	pourcentage : contribution sociale généralisée + contribution au remboursement de la dette sociale
CSGD	pourcentage : contribution sociale généralisée déductible
SECU	pourcentage : sécurité sociale
RETRAITE	pourcentage : retraite complémentaire + assurance chômage

Son contenu pourrait être le suivant :

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

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

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>
```

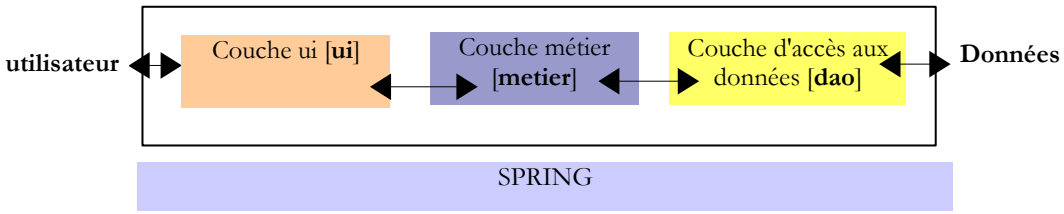
ligne 3 - la clé [provider] est définie. Sa valeur est le nom de la balise <provider> à utiliser dans [providers.config]. Ici, on utilise le provider [sqlServer2.0] qui permet de gérer des sources de données SQL Server Express 2005.
ligne 4 - la clé [connectionString] est définie. Sa valeur est la chaîne 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

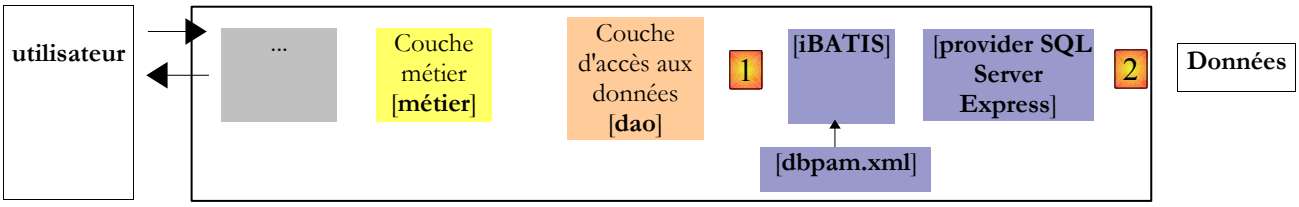
Rappelons l'interface [IPamDao] de la couche [dao] de notre architecture à trois couches :




```
1.  using Pam.Dao.Entites;
2.
3.  namespace Pam.Dao.Service {
4.   public interface IPamDao {
5.      // liste de toutes les identités des employés
6.      Employe[] GetAllIdentitesEmployes();
7.      // un employé particulier avec ses indemnités
8.      Employe GetEmploye(string ss);
9.      // liste de toutes les cotisations
10.     Cotisations GetCotisations();
11.  }
12. }
```

La couche [métier] interagit avec la couche [dao] via les méthodes de l'interface [IPamDao]. Ces interactions amènent des échanges d'objets de type [Cotisations, Employe, Indemnites] 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).



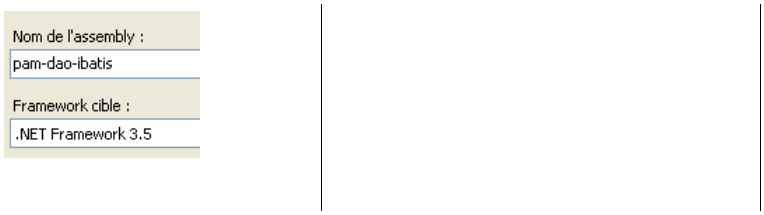
Le fichier [dbpam.xml] est référencé par le fichier principal [**sqlmap.config**] comme définissant des relations entre tables de données et objets .NET ainsi que les ordres SQL à exécuter sur la base. Son contenu sera le suivant :

```
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="Pam.Dao.Entites.Employe, 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="Pam.Dao.Entites.Employe, 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="Pam.Dao.Entites.Cotisations, 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="Pam.Dao.Entites.Indemnites, 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 :



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

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.  Employe[] GetAllIdentitesEmployes();
```

Cette méthode rend un tableau d'objets de type [Employe] 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 [Employe].

```
1.        // liste de toutes les cotisations
2.  Cotisations GetCotisations();
```

Cette méthode rend une liste d'objets de type [Cotisations]. 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 [Cotisations].

```
1.        // un employé particulier avec ses indemnités
2.  Employe GetEmploye(string ss);
```

Cette méthode rend un objet de type [Employe]. 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 [Employe].

La classe [Employe] contient des informations 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 [Indemnites]. La table obtenue sera placée dans un dictionnaire de type *Dictionary<int,Indemnites>* indexé par l'indice. C'est là qu'on trouvera les indemnités d'un employé particulier.

8.4.5 L'API de SqlMap

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

```
using IBatisNet.DataMapper;
```

Toutes les opérations SQL se font au travers d'un singleton de type [IBatisNet.DataMapper.ISqlMapper] où *ISqlMapper* est une interface. Ce singleton peut être obtenu de la façon suivante :

```
    private ISqlMapper mappeur = 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 [ISqlMapper]. 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="Pam.Dao.Entites.Employe, 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="Pam.Dao.Entites.Employe, 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.  </statements>
36. </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 "*Pam.Dao.Entites.Employe, pam-dao-ibatis*" définie ligne 8.

Pour exécuter cette requête *select* nommée "*getAllIdentitesEmployes*", on pourra écrire dans le code C# :

```
          IList listEmployes = mappeur.QueryForList("getAllIdentitesEmployes", null);
```

- [mappeur] est le singleton de type [ISqlMapper] obtenu par la ligne :

```
      private ISqlMapper mappeur = 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 *null*.
- 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 [Employe]. Nous obtenons dans la variable [listEmployes] 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 dans la requête sous la forme #*param*#, 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 (int, 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 :

```
Employe employe=mappeur.QueryForObject("getEmploye", "1451053072022") as Employe;
```

- [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 [Employe]. Si la requête SELECT ne rend aucune ligne, la méthode [QueryForObject] rend le pointeur *null*. 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="Pam.Dao.Entites.Employe">
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 [Pam.Dao.Entites.Employe]
- ligne 4 : utilise les champs de l'objet [Employe] passé en paramètre. L'exploitation de cet ordre *SqlMap* dans le code C# pourrait se faire de la façon suivante :

```
Employe empoye=new Employe("1451053072022", "TOUCHON", "Paul", "2 rue des acacias", "Segré", "49400",2);
int nbLignesInsérées=mappeur.Insert("insertEmploye", Employe);
```

- [mappeur] est le singleton de type [ISqlMapper] 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 C# pourrait se faire alors de la façon suivante :

```
Dictionary<String,String> infosEmploye=new Dictionary<String,String>();
    infosEmploye["SS"]= "1451053072022";
    infosEmploye["Nom"]= "TOUCHON";
    infosEmploye["Prenom"]= "Paul";
    infosEmploye["Adresse"]="2 rue des acacias";
    infosEmploye["Ville"]="Segré";
    infosEmploye["CodePostal"]="49400";
    infosEmploye["Indice"]="2";
    int nbLignesInsérées=mappeur.Insert("insertEmploye", infosEmploye);
```

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 C# pourrait se faire de la façon suivante :

```
      int nbLignesDétruites=mappeur.Delete("deleteEmploye", "1451053072022");
```

- [mappeur] est le singleton de type [ISqlMapper] 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="Pam.Dao.Entites.Employe">
3.          update EMPLOYES set ADRESSE= #Adresse# where SS= #SS#
4.  </update>
```

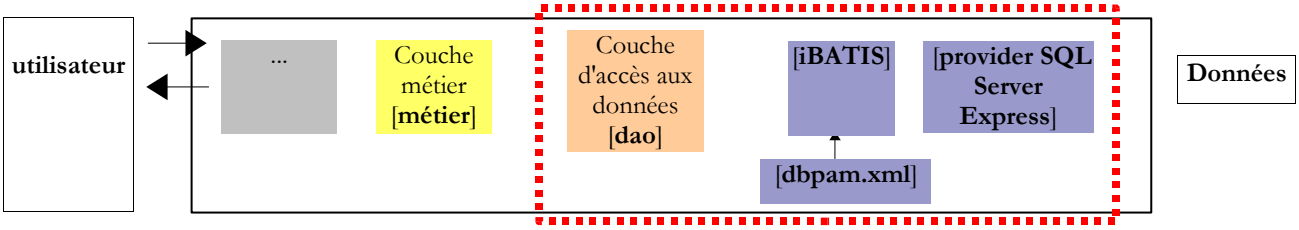
- ligne 2 : définit une instruction SQL nommée "modifyEmploye" qui admet pour paramètre un objet de type [Pam.Dao.Entites.Employe]
- ligne 3 : définit une instruction SQL UPDATE paramétrée par les champs de l'objet [Employe] passé en paramètre.

L'exploitation de cet ordre *SqlMap* dans le code C# pourrait se faire de la façon suivante :

```
1.  Employe employe=mappeur.QueryForObject("getEmploye", "1451053072022")as Employe;
2.  employe.Adresse="2 rue des oiseaux";
3.  int nbLignesModifiées=mappeur.Update("modifyEmploye", employe);
```

- ligne 1 : [mappeur] est le singleton de type [ISqlMapper] 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]

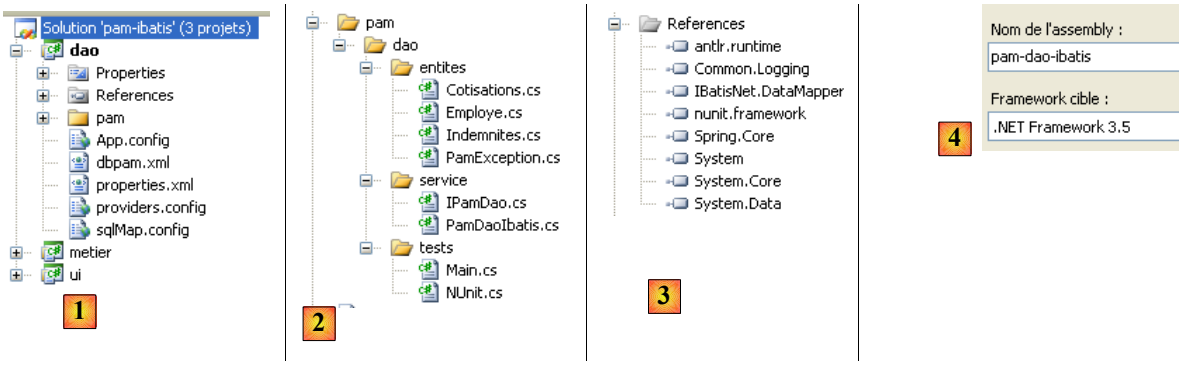


L'interface [IPamDao] implémentée par la couche [dao] est la suivante :

```
1.  using Pam.Dao.Entites;
2.
3.  namespace Pam.Dao.Service {
4.   public interface IPamDao {
5.      // liste de toutes les identités des employés
6.      Employe[] GetAllIdentitesEmployes();
7.      // un employé particulier avec ses indemnités
8.      Employe GetEmploye(string ss);
9.      // liste de toutes les cotisations
10.     Cotisations GetCotisations();
11.  }
12. }
```

8.5.1 Le projet Visual Studio

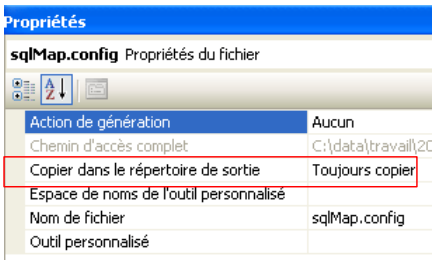
Le projet Visual Studio a déjà été présenté. Rappelons-le :



- dans [2] on voit que :

- le dossier [entites] contient les objets manipulés par la couche [dao]. Ils ont été présentés au paragraphe 8.3.1, page 39.
- le dossier [service] implémente l'interface [IPamDao] définie au paragraphe 8.3.2, page 40.
- le dossier [tests] contient deux programmes 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]. Ces fichiers sont à la racine du projet VS. On a configuré leur propriété [Copier dans le répertoire de sortie] à "Toujours copier" :



  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 [4] 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 console

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

```
1.  using System;
2.  using Pam.Dao.Entites;
3.  using Pam.Dao.Service;
4.  using Spring.Context.Support;
5.
6.  namespace Pam.Dao.Tests {
7.   public class MainPamDaoTests {
8.      public static void Main() {
9.         try {
10.           // instanciation couche [dao]
11.           IPamDao pamDao = (IPamDao)ContextRegistry.GetContext().GetObject("pamdao");
12.           // liste des identités des employés
13.           foreach (Employe Employe in pamDao.GetAllIdentitesEmployes()) {
14.              Console.WriteLine(Employe.ToString());
15.           }
16.           // un employé avec ses indemnités
17.           Console.WriteLine("-------------------------------------");
18.           Console.WriteLine(pamDao.GetEmploye("254104940426058"));
19.           Console.WriteLine("-------------------------------------");
20.           // liste des cotisations
21.           Cotisations cotisations = pamDao.GetCotisations();
22.           Console.WriteLine(cotisations.ToString());
23.        } catch (Exception ex) {
24.           // affichage exception
25.           Console.WriteLine(ex.ToString());
26.        }
27.        //pause
28.        Console.ReadLine();
29.     }
30.  }
31. }
```

- ligne 11 : on demande à Spring une référence sur la couche [dao].
- lignes 13-15 : test de la méthode [GetAllIdentitesEmployes] de l'interface [IPamDao]
- ligne 18 : test de la méthode [GetEmploye] de l'interface [IPamDao]
- ligne 21 : test de la méthode [GetCotisations] de l'interface [IPamDao]

Spring est configuré par le fichier [App.config] suivant :

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

```
3.
4.   <configSections>
5.      <sectionGroup name="spring">
6.         <section name="context" type="Spring.Context.Support.ContextHandler, Spring.Core" />
7.         <section name="objects" type="Spring.Context.Support.DefaultSectionHandler, Spring.Core" />
8.      </sectionGroup>
9.   </configSections>
10.
11.  <spring>
12.     <context>
13.        <resource uri="config://spring/objects" />
14.     </context>
15.     <objects xmlns="http://www.springframework.net">
16.        <object id="pamdao" type="Pam.Dao.Service.PamDaoIbatis, pam-dao-ibatis"/>
17.     </objects>
18.  </spring>
19. </configuration>
```

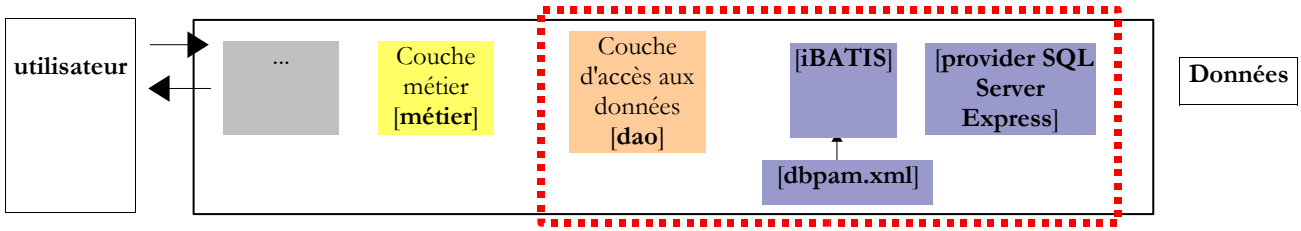
- ligne 16 : l'objet d'id "pamdao" a le type [Pam.Dao.Service.PamDaoIbatis] et est trouvé dans l'assembly [pam-dao-ibatis].

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

```
1.  [254104940426058,Jouveinal,Marie,,,,]
2.  [260124402111742,Laverti,Justine,,,,]
3.  ------------------------------------
4.  [254104940426058,Jouveinal,Marie,5 rue des Oiseaux,St Corentin,49203,[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 [Employe] avec les seules informations [SS, Nom, Prenom]
- ligne 4 : l'employé de type [Employe] ayant le n° de sécurité sociale [254104940426058]
- ligne 5 : les taux de cotisations

8.5.3 Écriture de la classe [PamDaoIbatis]



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

```
1.  using Pam.Dao.Entites;
2.
3.  namespace Pam.Dao.Service {
4.   public interface IPamDao {
5.      // liste de toutes les identités des employés
6.      Employe[] GetAllIdentitesEmployes();
7.      // un employé particulier avec ses indemnités
8.      Employe GetEmploye(string ss);
9.      // liste de toutes les cotisations
10.     Cotisations GetCotisations();
11.  }
12. }
```

Question : écrire le code de la classe [PamDaoIbatis] 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 43).

Spécifications :

On supposera que les données demandées à la couche [dao] peuvent entièrement tenir en mémoire. Ainsi, pour améliorer les performances, la classe [PamDaoIbatis] 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 [Employe]
- la table [COTISATIONS] sous la forme d'un unique objet de type [Cotisations]

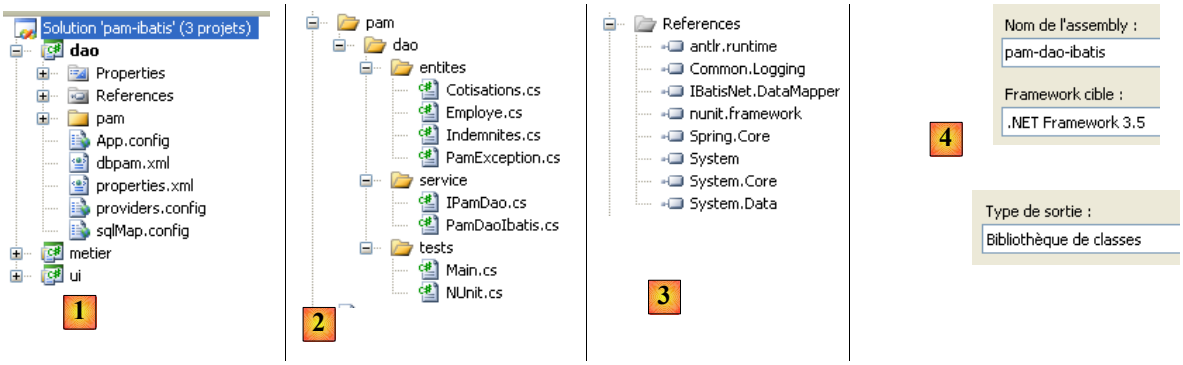
- la table [INDEMNITES] sous la forme d'un dictionnaire indexé par le champ [INDICE] et dont les valeurs seront des objets de type [Indemnites]

8.5.4 Tests unitaires avec NUnit

Lectures conseillées : **"Langage C# 2008, Chapitre 4 : Architectures 3 couches, tests NUnit, framework Spring"**.

Le test précédent avait été visuel : 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/].

Revenons sur le projet Visual Studio de la couche [dao] :



- en [2], le programme de test [NUnit.cs]
- en [4], le projet va générer une DLL nommé [pam-dao-ibatis.dll]
- en [3], la référence à la DLL du framework NUnit : [nunit.framework.dll]

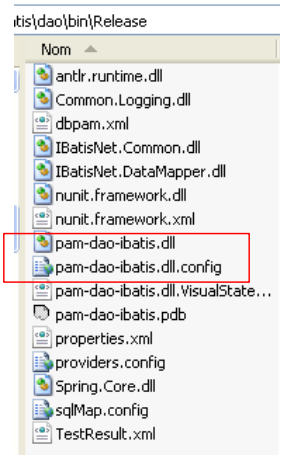
La classe de test NUnit est la suivante :

```
1.  using System.Collections;
2.  using NUnit.Framework;
3.  using Pam.Dao.Service;
4.  using Pam.Dao.Entites;
5.  using Spring.Objects.Factory.Xml;
6.  using Spring.Core.IO;
7.  using Spring.Context.Support;
8.
9.  namespace Pam.Dao.Tests {
10.
11.  [TestFixture]
12.  public class NunitPamDao : AssertionHelper {
13.     // la couche [dao] à tester
14.     private IPamDao pamDao = null;
15.
16.     // constructeur
17.     public NunitPamDao() {
18.        // instanciation couche [dao]
19.        pamDao = (IPamDao)ContextRegistry.GetContext().GetObject("pamdao");
20.     }
21.
22.     // init
23.     [SetUp]
24.     public void Init() {
25.
26.     }
27.
28.     [Test]
29.     public void GetAllIdentitesEmployes() {
30.        // vérification nbre d'employes
31.        Expect(2, EqualTo(pamDao.GetAllIdentitesEmployes().Length));
32.     }
33.
34.     [Test]
35.     public void GetCotisations() {
```

```
36.         // vérification taux de cotisations
37.         Cotisations cotisations = pamDao.GetCotisations();
38.         Expect(3.49, EqualTo(cotisations.CsgRds).Within(1E-06));
39.         Expect(6.15, EqualTo(cotisations.Csgd).Within(1E-06));
40.         Expect(9.39, EqualTo(cotisations.Secu).Within(1E-06));
41.         Expect(7.88, EqualTo(cotisations.Retraite).Within(1E-06));
42.     }
43.
44.     [Test]
45.     public void GetEmployeIdemnites() {
46.         // vérification individus
47.         Employe employe1 = pamDao.GetEmploye("254104940426058");
48.         Employe employe2 = pamDao.GetEmploye("260124402111742");
49.         Expect("Jouveinal", EqualTo(employe1.Nom));
50.         Expect(2.1, EqualTo(employe1.Indemnites.BaseHeure).Within(1E-06));
51.         Expect("Laverti", EqualTo(employe2.Nom));
52.         Expect(1.93, EqualTo(employe2.Indemnites.BaseHeure).Within(1E-06));
53.     }
54.
55.     [Test]
56.     public void GetEmployeIdemnites2() {
57.         // vérification individu inexistant
58.         bool erreur = false;
59.         try {
60.             Employe employe1 = pamDao.GetEmploye("xx");
61.         } catch {
62.             erreur = true;
63.         }
64.         Expect(erreur, True);
65.     }
66. }
67. }
```

- ligne 11 : la classe a l'attribut [*TestFixture*] qui en fait une classe de test [NUnit].
- ligne 12 : la classe dérive de la classe utilitaire *AssertionHelper* du framework *NUnit* (à partir de la version 2.4.6).
- ligne 14 : 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 23-26, une méthode avec l'attribut [*SetUp*].
- lignes 17-20 : le constructeur de la classe initialise le champ privé [pamDao] à l'aide de Spring et [App.config].
- lignes 29-32 : testent la méthode [GetAllIdentitesEmployes]
- lignes 35-42 : testent la méthode [GetCotisations]
- lignes 45-53 : testent la méthode [GetEmploye]
- lignes 56-65 : testent la méthode [GetEmploye] lors d'une exception.

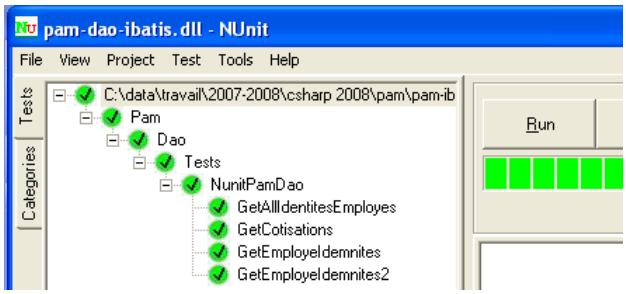
Le projet ci-dessus génère la DLL [pam-dao-ibatis.dll] dans le dossier [bin/Release].



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 : [IbatisNet.DataMapper, Spring.Core.dll]. Ces DLL sont accompagnées des copies des DLL qu'elles utilisent elles-mêmes :
  - [IbatisNet.Common.dll] pour l'outil iBatis
  - [antlr.runtime.dll, Common.Logging.dll] pour l'outil Spring
- les fichiers de configuration du projet qui ont leur attribut [Copier dans le répertoire de sortie] à vrai : [dbpam.xml, properties.xml, providers.config, sqlmap.config]
- le fichier [pam-dao-ibatis.dll.config] est une copie du fichier de configuration [App.config]. C'est VS qui opère cette duplication. A l'exécution c'est le fichier [pam-dao-ibatis.dll.config] qui est utilisé et non [App.config].

On charge la DLL [pam-dao-ibatis.dll] avec l'outil [NUnit-Gui], version 2.4.6 et on exécute les tests :



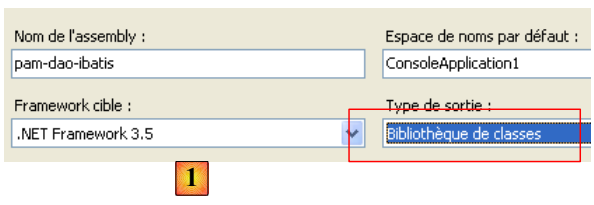
Ci-dessus, les tests ont été réussis.

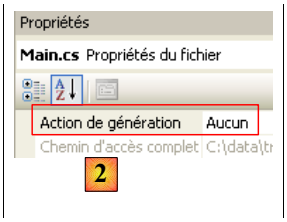
Travail pratique :

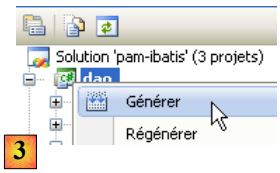
- mettre en oeuvre sur machine les tests de la classe [PamDaoIbatis].
- utiliser différents fichiers de configuration SqlMap afin d'utiliser des SGBD différents (Firebird, MySQL, Postgres, SQL Server)

8.5.5 Génération de la DLL de la couche [dao]

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

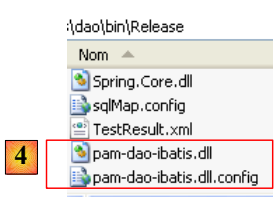






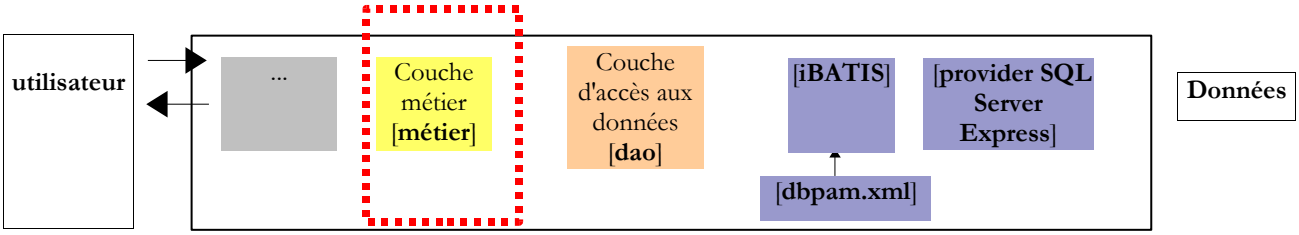
- [1], le type de sortie doit être "Bibliothèque de classes":
- [2], exclure du projet généré les deux programme de test (Main.cs et NUnit.cs) :
- [3], générer le projet

- la DLL est générée dans le dossier [bin/Release] [4]



8.6 La couche métier

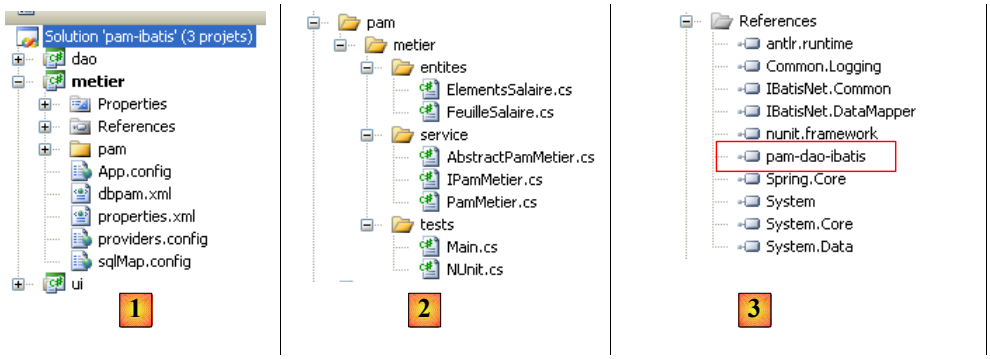
Revenons sur l'architecture générale de l'application [SimuPaie] :



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.

8.6.1 Le projet Visual Studio de la couche [metier]

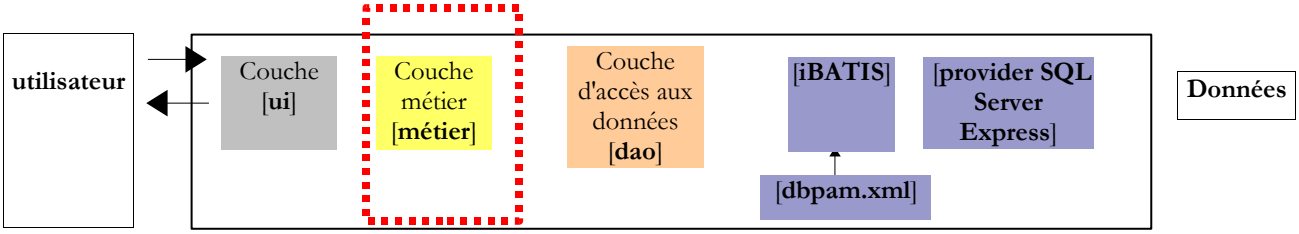
Le projet Visual Studio de la couche métier pourrait ressembler à ce qui suit :



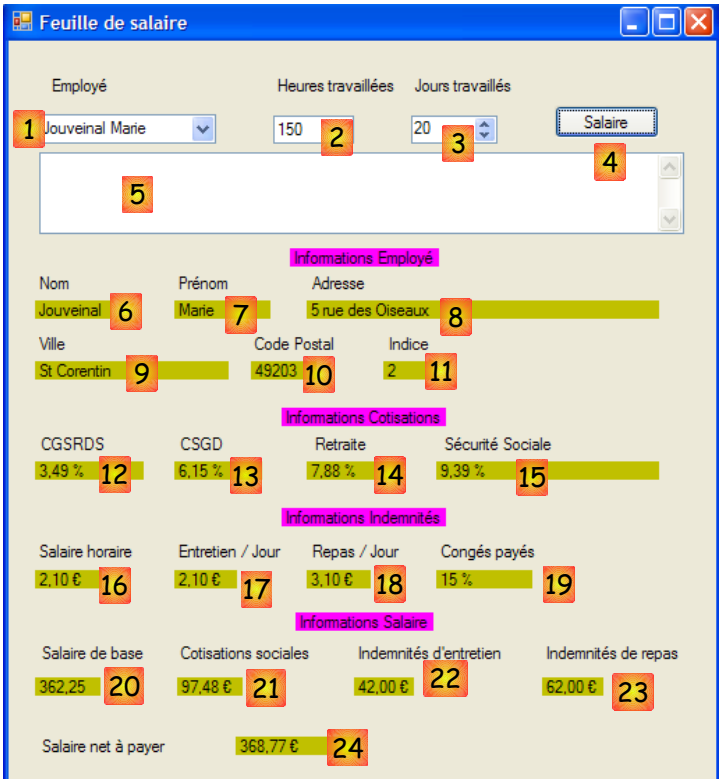
- en [1] l'ensemble du projet. On retrouve l'ensemble des fichiers de configuration de la couche [dao]. Ceux-ci vont être nécessaires pour tester l'intégration des couches [metier] et [dao].
- en [2], la couche [metier] est formée des deux dossiers [entites, service]. Le dossier [tests] contient un programme de test console (Main.cs) et un programme de test NUnit (NUnit.cs).
- en [3] les références utilisées par le projet. On notera la DLL [pam-dao-ibatis] de la couche [dao] étudiée précédemment.

8.6.2 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.  using Pam.Dao.Entites;
2.  using Pam.Metier.Entites;
3.
4.  namespace Pam.Metier.Service {
5.   public interface IPamMetier {
6.      // liste de toutes les identités des employés
7.      Employe[] GetAllIdentitesEmployes();
8.
9.      // ------- le calcul du salaire
10.     FeuilleSalaire GetSalaire(string ss, double heuresTravaillées, int joursTravaillés);
11.  }
```

```
12. }
```

- ligne 7 : la méthode qui permettra le remplissage du combo [1]
- ligne 10 : la méthode qui permettra d'obtenir les renseignements 6 à 24. Ceux-ci ont été rassemblés dans un objet de type [FeuilleSalaire].

8.6.3 Les entités de la couche [metier]

Le dossier [entites] du projet Visual Studio (cf paragraphe 8.6.1, page 59) 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.

8.6.4 Implémentation de la couche [metier]

Nous allons implémenter l'interface [IPamMetier] 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].
- [PamMetier] 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.  using Pam.Dao.Entites;
2.  using Pam.Dao.Service;
3.  using Pam.Metier.Entites;
4.
5.  namespace Pam.Metier.Service {
6.   public abstract class AbstractBasePamMetier : IPamMetier {
7.
8.      // l'objet d'accès aux données
9.      public IPamDao PamDao { get; set; }
10.
11.     // liste de toutes les identités des employés
12.     public Employe[] GetAllIdentitesEmployes() {
13.        return PamDao.GetAllIdentitesEmployes();
14.     }
15.
16.     // un employé particulier avec ses indemnités
17.     protected Employe GetEmploye(string ss) {
18.        return PamDao.GetEmploye(ss);
19.     }
20.
21.     // les cotisations
22.     protected Cotisations GetCotisations() {
23.        return PamDao.GetCotisations();
24.     }
25.
26.     // le calcul du salaire
27.     public abstract FeuilleSalaire GetSalaire(string ss, double heuresTravaillées, int
    joursTravaillés);
28.  }
29. }
```

- ligne 5 : la classe appartient à l'espace de noms [Pam.Metier.Service] comme toutes les classes et interfaces de la couche [metier].
- ligne 6 : la classe est abstraite (attribut *abstract*) et implémente l'interface [IPamMetier]
- ligne 9 : la classe détient une référence sur la couche [dao] sous la forme d'une propriété publique
- lignes 12-14 : implémentation de la méthode [GetAllIdentitesEmployes] de l'interface [IPamMetier] – utilise la méthode de même nom de la couche [dao]
- lignes 17-19 : méthode interne (protected) [GetEmploye] 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 22-24 : méthode interne (protected) [GetCotisations] qui fait appel à la méthode de même nom de la couche [dao]
- ligne 27 : implémentation abstraite (attribut *abstract*) de la méthode [GetSalaire] de l'interface [IPamMetier].

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

```
1.  using System;
```

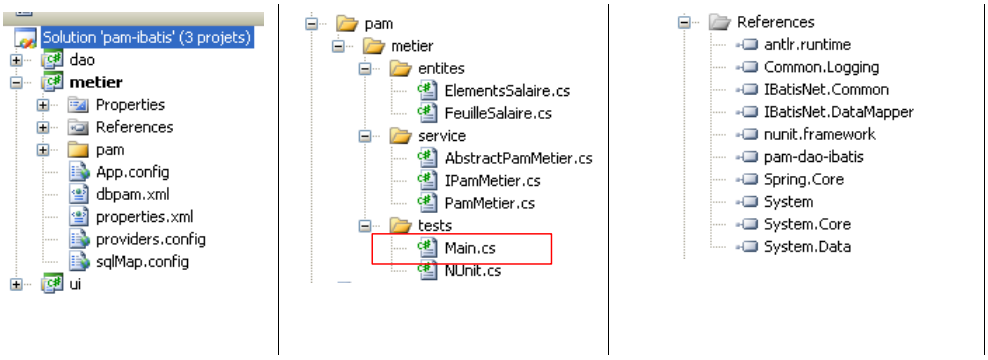
```
2.  using Pam.Dao.Entites;
3.  using Pam.Metier.Entites;
4.
5.  namespace Pam.Metier.Service {
6.
7.   public class PamMetier : AbstractBasePamMetier {
8.
9.      // calcul du salaire
10.     public override FeuilleSalaire GetSalaire(string ss, double heuresTravaillées, int
    joursTravaillés) {
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.        ...
20.        // on rend la feuille de salaire
21.        return ...;
22.     }
23.  }
24. }
```

- 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].

8.6.5 Le test console de la couche [metier]

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



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

```
1.  using System;
2.  using Pam.Dao.Entites;
3.  using Pam.Metier.Service;
4.  using Spring.Context.Support;
5.
6.  namespace Pam.Metier.Tests {
7.   class MainPamMetierTests {
8.      public static void Main() {
9.         try {
10.           // instanciation couche [metier]
11.           IPamMetier pamMetier = ContextRegistry.GetContext().GetObject("pammetier") as
    IPamMetier;
12.           // calculs de feuilles de salaire
13.           Console.WriteLine(pamMetier.GetSalaire("260124402111742", 30, 5));
14.           Console.WriteLine(pamMetier.GetSalaire("254104940426058", 150, 20));
15.           try {
16.              Console.WriteLine(pamMetier.GetSalaire("xx", 150, 20));
17.           } catch (PamException ex) {
18.              Console.WriteLine(string.Format("PamException : {0}", ex.Message));
19.           }
```

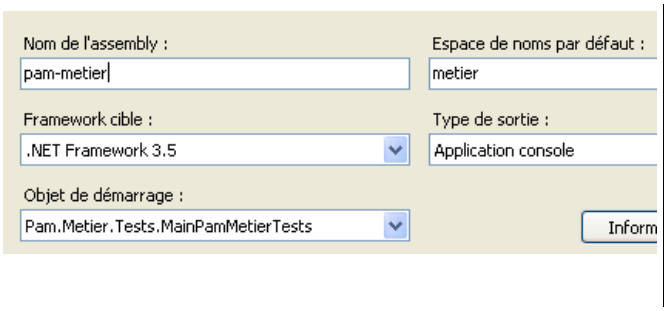
```
20.         } catch (Exception ex) {
21.            Console.WriteLine(string.Format("Exception : {0}", ex.ToString()));
22.         }
23.         // pause
24.         Console.ReadLine();
25.     }
26.  }
27. }
```

- ligne 11 : instanciation par Spring de la couche [metier].
- lignes 13-14 : tests de la méthode [GetSalaire] de l'interface [IPamMetier]
- lignes 15-22 : test de la méthode [GetSalaire] lorsqu'il se produit une exception

Le programme de test utilise le fichier de configuration [App.config] suivant :

```
1.  <?xml version="1.0" encoding="utf-8" ?>
2.  <configuration>
3.
4.   <configSections>
5.      <sectionGroup name="spring">
6.         <section name="context" type="Spring.Context.Support.ContextHandler, Spring.Core" />
7.         <section name="objects" type="Spring.Context.Support.DefaultSectionHandler, Spring.Core" />
8.      </sectionGroup>
9.   </configSections>
10.
11.  <spring>
12.     <context>
13.        <resource uri="config://spring/objects" />
14.     </context>
15.     <objects xmlns="http://www.springframework.net">
16.        <object id="pamdao" type="Pam.Dao.Service.PamDaoIbatis, pam-dao-ibatis"/>
17.        <object id="pammetier" type="Pam.Metier.Service.PamMetier, pam-metier">
18.           <property name="PamDao" ref="pamdao"/>
19.        </object>
20.     </objects>
21.  </spring>
22. </configuration>
```

- ligne 16 : l'objet d'id "pamdao" a le type [Pam.Dao.Service.PamDaoIbatis] et est trouvé dans l'assembly [pam-dao-ibatis]. La couche [dao] est celle étudiée précédemment, c.a.d. l'implémentation iBatis.
- lignes 17-19 : l'objet d'id "pammetier" a le type [Pam.Metier.Service.PamMetier] et est trouvé dans l'assembly [pam-metier]. Il faut configurer le projet en ce sens :



- ligne 18 : l'objet [PamMetier] 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 :

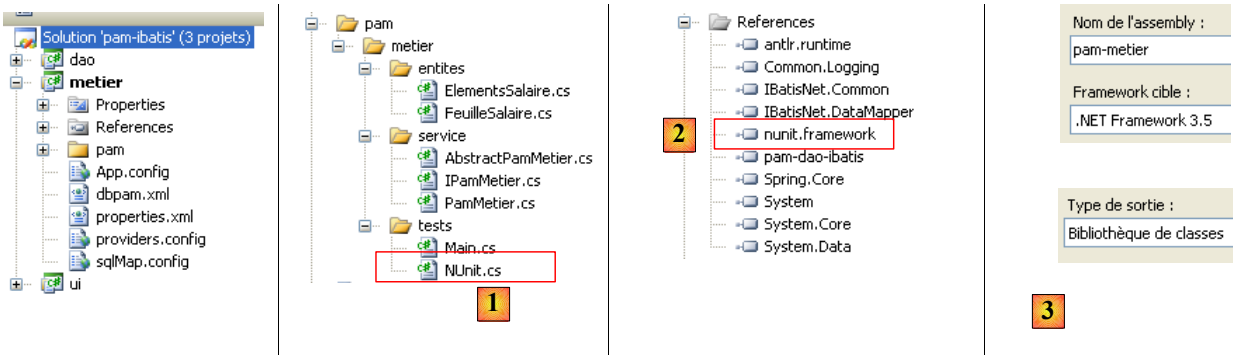
```
1.  [[260124402111742,Laverti,Justine,la Brûlerie,St Marcel,49014,[1, 1,93, 2, 3, 12]],
    [3,49,6,15,9,39,7,88],[1, 1,93, 2, 3, 12],[64,85 : 17,45 : 10 : 15 : 72,4 ]
2.  [[254104940426058,Jouveinal,Marie,5 rue des Oiseaux,St Corentin,49203,[2, 2,1, 2,1, 3,1, 15]],
    [3,49,6,15,9,39,7,88],[2, 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.

8.6.6 Tests unitaires de la couche métier

Le test précédent était visuel : on vérifiait à l'écran qu'on obtenait bien les résultats attendus. Nous passons maintenant aux tests non visuels NUnit.

Revenons au projet Visual Studio du projet [metier] :



- en [1], le programme de test NUnit
- en [2], la référence sur la DLL [nunit.framework]
- en [3], la génération du projet va produire la DLL [pam-metier.dll].

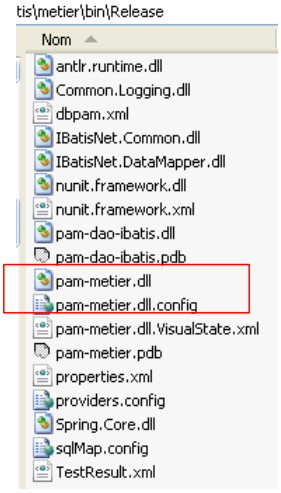
La classe de test NUnit est la suivante :

```
1.  using NUnit.Framework;
2.  using Pam.Dao.Entites;
3.  using Pam.Metier.Entites;
4.  using Pam.Metier.Service;
5.  using Spring.Context.Support;
6.
7.  namespace Pam.Metier.Tests {
8.
9.   [TestFixture()]
10.  public class NunitTestPamMetier : AssertionHelper {
11.
12.     // la couche [metier] à tester
13.     private IPamMetier pamMetier;
14.
15.     // constructeur
16.     public NunitTestPamMetier() {
17.        // instanciation couche [dao]
18.        pamMetier = ContextRegistry.GetContext().GetObject("pammetier") as IPamMetier;
19.     }
20.
21.
22.     [Test]
23.     public void GetAllIdentitesEmployes() {
24.        // vérification nbre d'employes
25.        Expect(2, EqualTo(pamMetier.GetAllIdentitesEmployes().Length));
26.     }
27.
28.     [Test]
29.     public void GetSalaire1() {
30.        // calcul d'une feuille de salaire
31.        FeuilleSalaire feuilleSalaire = pamMetier.GetSalaire("254104940426058", 150, 20);
32.        // vérifications
33.        Expect(368.77, EqualTo(feuilleSalaire.ElementsSalaire.SalaireNet).Within(1E-06));
34.        // feuille de salaire d'un employé inexistant
35.        bool erreur = false;
36.        try {
37.           feuilleSalaire = pamMetier.GetSalaire("xx", 150, 20);
38.        } catch (PamException) {
39.           erreur = true;
```

```
40.         }
41.         Expect(erreur, True);
42.     }
43.
44.  }
45. }
```

- ligne 13 : 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 16-19 : le constructeur de la classe initialise le champ privé [pamMetier] à l'aide de Spring et du fichier de configuration [App.config].
- lignes 23-26 : testent la méthode [GetAllIdentitesEmployes]
- lignes 29-42 : testent la méthode [GetSalaire]

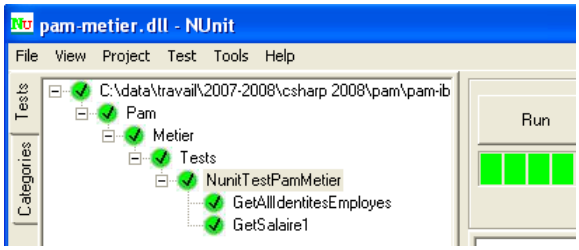
Le projet ci-dessus génère la DLL [pam-metier.dll] dans le dossier [bin/Release].



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 : [IbatisNet.DataMapper, Spring.Core.dll, pam-dao-ibatis]. Ces DLL sont accompagnées des copies des DLL qu'elles utilisent elles-mêmes :
  - [IbatisNet.Common.dll] pour l'outil iBatis
  - [antlr.runtime.dll, Common.Logging.dll] pour l'outil Spring
- les fichiers de configuration du projet qui ont leur attribut [Copier dans le répertoire de sortie] à vrai : [dbpam.xml, properties.xml, providers.config, sqlmap.config]
- le fichier [pam-metier.dll.config] est une copie du fichier de configuration [App.config].

On charge la DLL [pam-metier.dll] avec l'outil [NUnit-Gui, version 2.4.6] et on exécute les tests :



Ci-dessus, les tests ont été réussis.

Travail pratique :

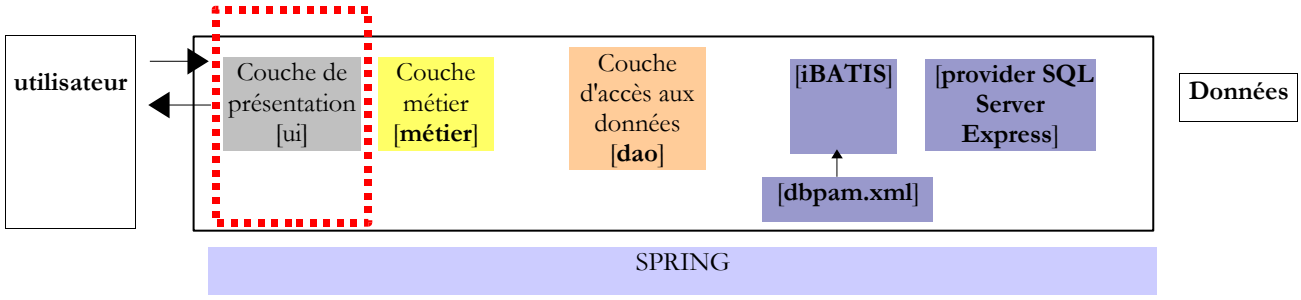
- mettre en oeuvre sur machine les tests de la classe [PamMetier].
- utiliser différents fichiers de configuration SqlMap afin d'utiliser des SGBD différents (Firebird, MySQL, Postgres, SQL Server)

8.6.7 Génération de la DLL de la couche [metier]

Une fois écrite et testée la classe [PamMetier], on génèrera la DLL [pam-metier.dll] de la couche [metier] en suivant la méthode décrite au paragraphe 8.5.5, page 58. On prendra soin de ne pas inclure dans la DLL les programmes de test [Main.cs] et [NUnit.cs].

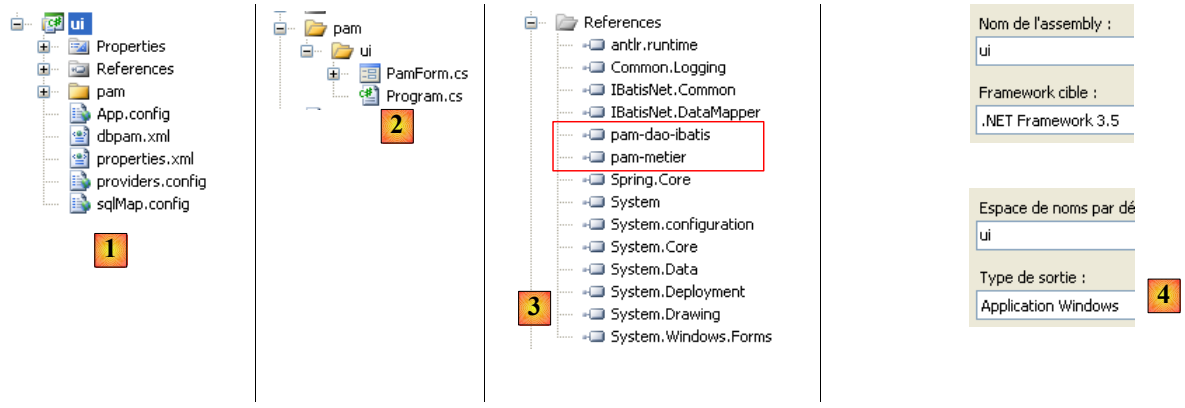
8.7 La couche [ui]

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



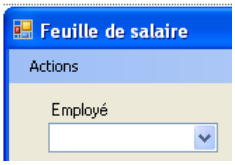
Nous considérons désormais que la couche [métier] est acquise et qu'elle a été encapsulée dans la DLL [pam-metier.dll]. Nous nous intéressons maintenant à la couche [ui] implémentée ici sous la forme d'une interface graphique windows.

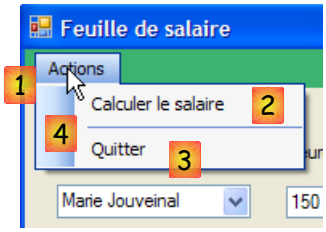
Le projet Visual Studio de la couche [ui] pourrait ressembler à ce qui suit :



- en [1] l'ensemble du projet. On retrouve les fichiers de configuration de la couche [dao] : [App.config, dbpam.xml, properties.xml, providers.config, sqlmap.config] et de la couche [metier] : [App.config].
- en [2], la couche [ui] est formée de deux classes :
  - [PamForm.cs] : l'interface graphique du projet
  - [Program.cs] : le lanceur de l'application
- en [3] les références utilisées par le projet. On notera les DLL [pam-dao-ibatis, pam-metier] des couches [dao] et [métier] étudiées précédemment.
- en [4] le type du projet (Application windows) et le nom de l'assembly généré (ui).

L'interface graphique est celle décrit au paragraphe 3.1, page 8 au détail près qu'elle a désormais un menu au lieu du bouton [Salaire] :





N°	Type	Nom	Rôle
1	ToolStripMenuItem	actionsToolStripMenuItem	liste d'actions possibles
2	ToolStripMenuItem	calculerLeSalaireToolStripMenuItem	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 [PamForm.cs] en vous inspirant des versions déjà écrites. Ici, la couche [ui] doit s'appuyer sur la couche [metier].

Travail pratique :

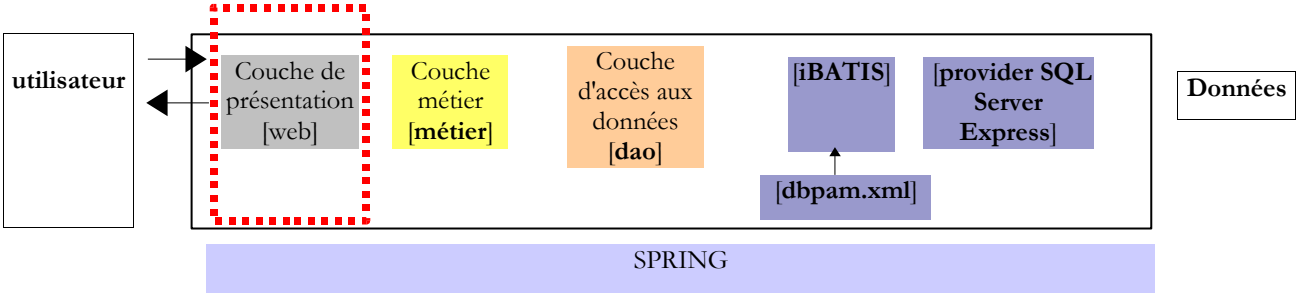
- mettre en oeuvre sur machine les tests de l'application C# précédente
- utiliser différents fichiers de configuration *SqlMap* afin d'utiliser des SGBD différents (Firebird, MySQL, Postgres, SQL Server)

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

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

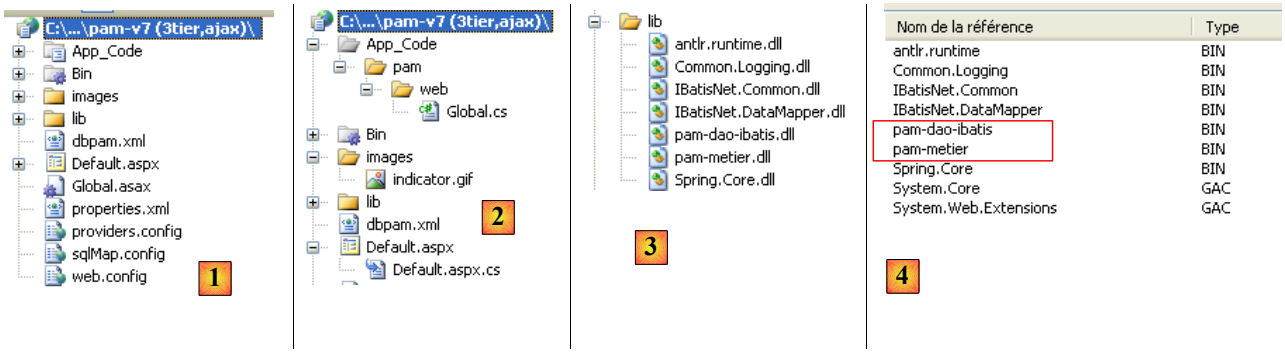
9.1 La couche [ui]

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



La couche [ui] est désormais 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]



- en [1], le projet dans son ensemble. Les fichiers de configuration [dbpam.xml, properties.xml, providers.config, sqlmap.config] sont identiques à ceux de la version précédente. Ils configurent la couche [dao] encapsulée dans la DLL [pam-dao-ibatis.dll] [4].
- en [2], le code de la couche [web] :
  - [Global.asax, Global.cs] : la classe instanciée au démarrage de l'application web et qui assure l'initialisation de l'application
  - [Default.aspx, Default.aspx.cs] : la page du formulaire web et son code de contrôle
  - [lib] : les DLL nécessaires à l'application web et détaillées dans [3] :
    - IbatisNet.*.dll : les DLL du framework iBatis
    - Spring.Core, Common.Logging, antlr.runtime : les DLL du framework Spring
    - pam-dao-ibatis, pam-metier : les DLL des couches [dao] et [metier] construites dans la version précédente

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 C# étudiée précédemment. Celles-ci doivent prendre place dans le code pré-généré du fichier [web.config] :

```
1.  <?xml version="1.0"?>
2.
3.  <configuration>
4.   <configSections>
5.      <sectionGroup name="system.web.extensions"...>
6.  ...
```

```
7.      </sectionGroup>
8.      <!-- spring -->
9.      <sectionGroup name="spring">
10.        <section name="context" type="Spring.Context.Support.ContextHandler, Spring.Core"/>
11.        <section name="objects" type="Spring.Context.Support.DefaultSectionHandler, Spring.Core"/>
12.     </sectionGroup>
13.  </configSections>
14.  <!-- spring objects -->
15.  <spring>
16.     <context>
17.        <resource uri="config://spring/objects"/>
18.     </context>
19.     <objects xmlns="http://www.springframework.net">
20.        <object id="pamdao" type="Pam.Dao.Service.PamDaoIbatis, pam-dao-ibatis"/>
21.        <object id="pammetier" type="Pam.Metier.Service.PamMetier, pam-metier">
22.           <property name="PamDao" ref="pamdao"/>
23.        </object>
24.     </objects>
25.  </spring>
26.  <appSettings/>
27.  <connectionStrings/>
28.  <system.web>
29.  ....
30.  </system.web>
31.  ...
32.  </configuration>
```

On retrouve lignes 9-12 et 15-25 la configuration Spring décrite dans le fichier [App.config] de la version précédente (cf page 63)

Global.asax

```
<%@ Application Language="C#" inherits="Pam.Web.Global"  %>
```

Global.cs

```
1.  using System;
2.  using System.Web;
3.  using Pam.Dao.Entites;
4.  using Pam.Metier.Service;
5.  using Spring.Context.Support;
6.
7.  namespace Pam.Web
8.  {
9.      public class Global : HttpApplication
10.     {
11.
12.         // --- données statiques de l'application ---
13.         public static Employe[] Employes;
14.         public static string Msg = string.Empty;
15.         public static bool Erreur = false;
16.         public static IPamMetier PamMetier = null;
17.
18.         // démarrage de l'application
19.         public void Application_Start(object sender, EventArgs e)
20.         {
21.             // configuration
22.             try
23.             {
24.                 // instanciation couche [metier]
25.                 PamMetier = ContextRegistry.GetContext().GetObject("pammetier") as IPamMetier;
26.                 // on récupère le tableau simplifié des employés
27.                 Employes = PamMetier.GetAllIdentitesEmployes();
28.                 // on note le succès
29.                 Msg = "Base de données chargée ...";
30.             }
31.             catch (Exception ex)
32.             {
33.                 // erreur d'initialisation
34.                 Msg = String.Format("Erreur lors de l'initialisation de l'application : {0}",
    ex.ToString());
35.                 Erreur = true;
36.             }
37.         }
38.     }
39. }
```

On rappelle que :

- la classe [Global.cs] est instanciée au démarrage de l'application et que cette instance est accessible à toutes les requêtes de tous les utilisateurs. Les champs statiques des lignes 13-16 sont ainsi partagés entre tous les utilisateurs.
- que la méthode [Application_Start] est exécutée une unique fois après l'instanciation de la classe. C'est la méthode où est faite en général l'initialisation de l'application.

Les données partagées par tous les utilisateurs sont les suivantes :

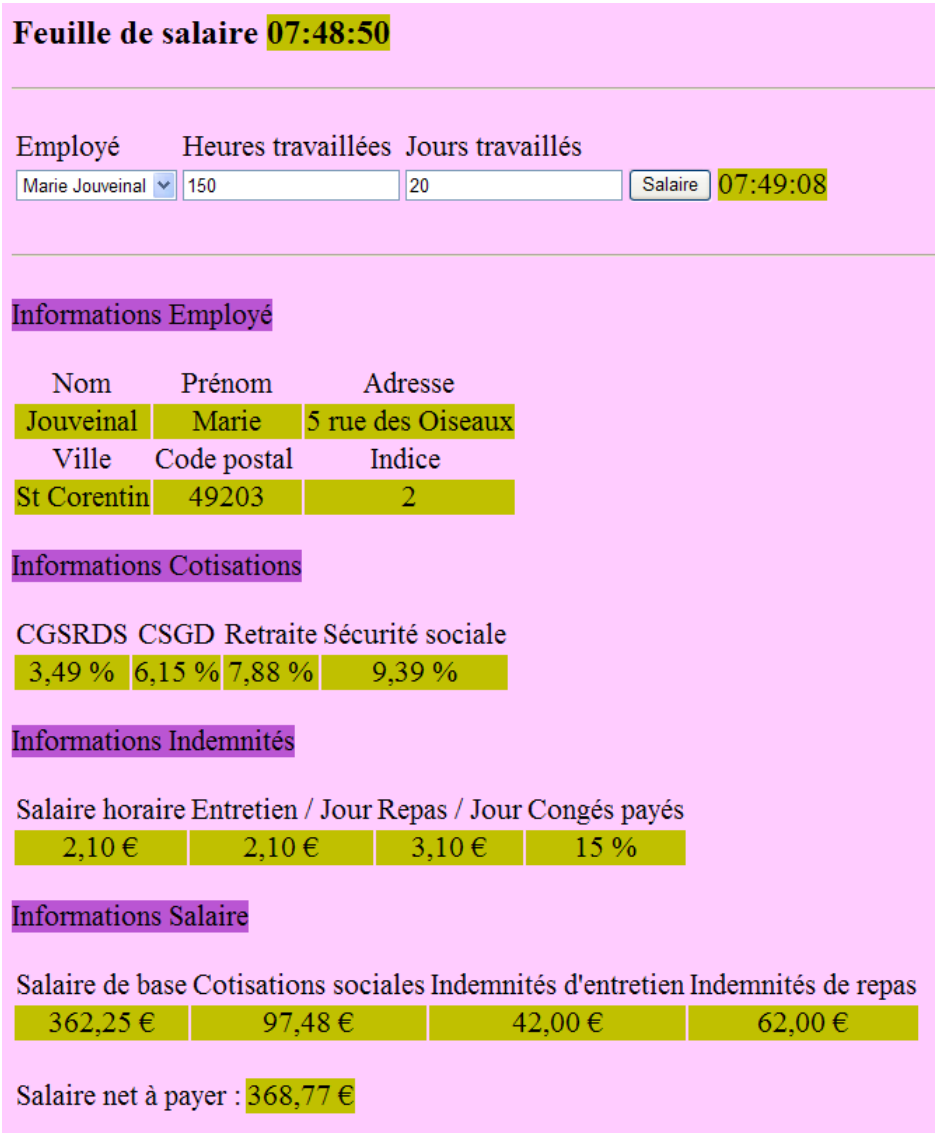
- ligne 13 : le tableau d'objets de type [Employe] qui mémorisera la liste simplifiée (SS, NOM, PRENOM) de tous les employés
- ligne 14 : un message indiquant comment s'est terminé l'initialisation (bien ou avec erreur)
- ligne 15 : un booléen indiquant si l'initialisation s'est terminée par une erreur ou non.
- ligne 16 : une référence sur la couche [metier] encapsulée dans la DLL [pam-metier.dll]

Dans [Application_Start] :

- ligne 25 : Spring instancie les couches [metier] et [dao] et rend une référence sur la couche [metier]. Celle-ci est mémorisée dans le champ statique [PamMetier] de la ligne 16.
- ligne 27 : le tableau des employés est demandé à la couche [metier]
- ligne 29 : le message en cas de réussite
- ligne 34 : le message en cas d'erreur

9.4 Le formulaire [Default.aspx]

Le formulaire est celui de la version 3 et décrit au paragraphe 5.1, page 18, complété par la partie Ajax de la version 5 décrite au paragraphe 7.2, page 33.



Question : Le fonctionnement du formulaire a été décrit au paragraphe 5.3.1, page 23 et sa partie Ajax au paragraphe 7.2, page 33. En vous inspirant du code C# écrit précédemment, de la page [Default.aspx.cs] de la version 3, de celle de la version 5 pour la partie Ajax, écrire le code [Default.aspx.cs].

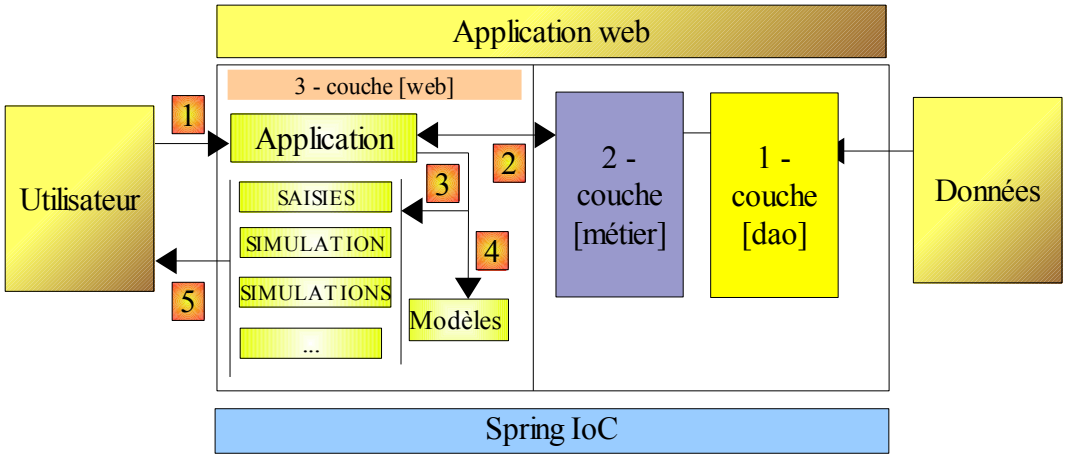
Travail pratique :

- 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 – 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.

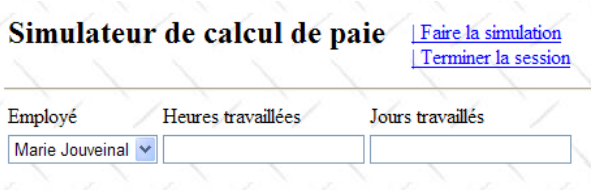
Nous allons ici implémenter le modèle (design pattern) MVC de la façon suivante :
- les vues seront des composants [View] au sein d'une page unique [Default.aspx]
- le contrôleur est alors le code [Default.aspx.cs] de cette page unique.

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.

10.1 Les vues de l'application

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

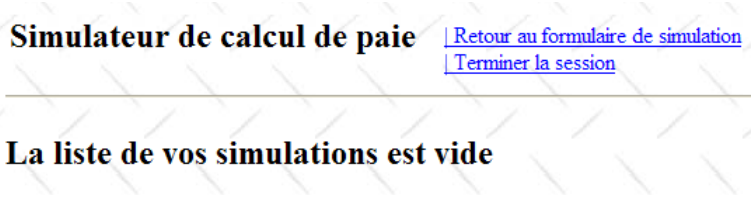
- la vue [VueSaisies] qui présente le formulaire de simulation



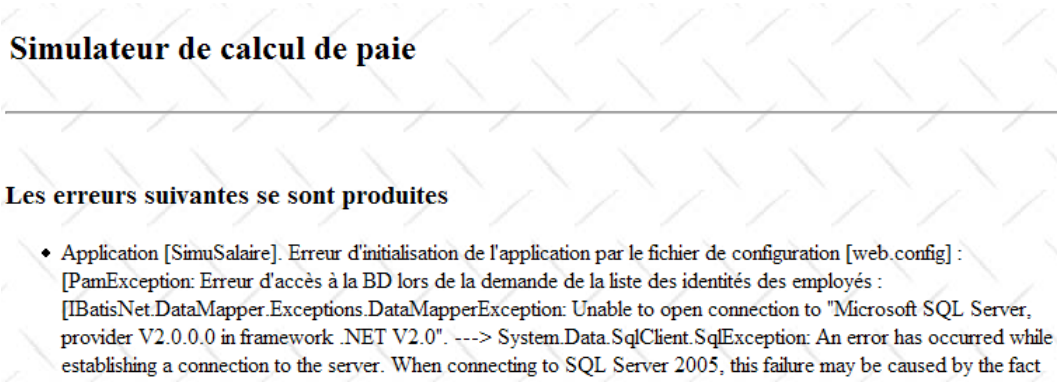
- la vue [VueSimulation] utilisée pour afficher le résultat détaillé de la simulation :

- 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 :

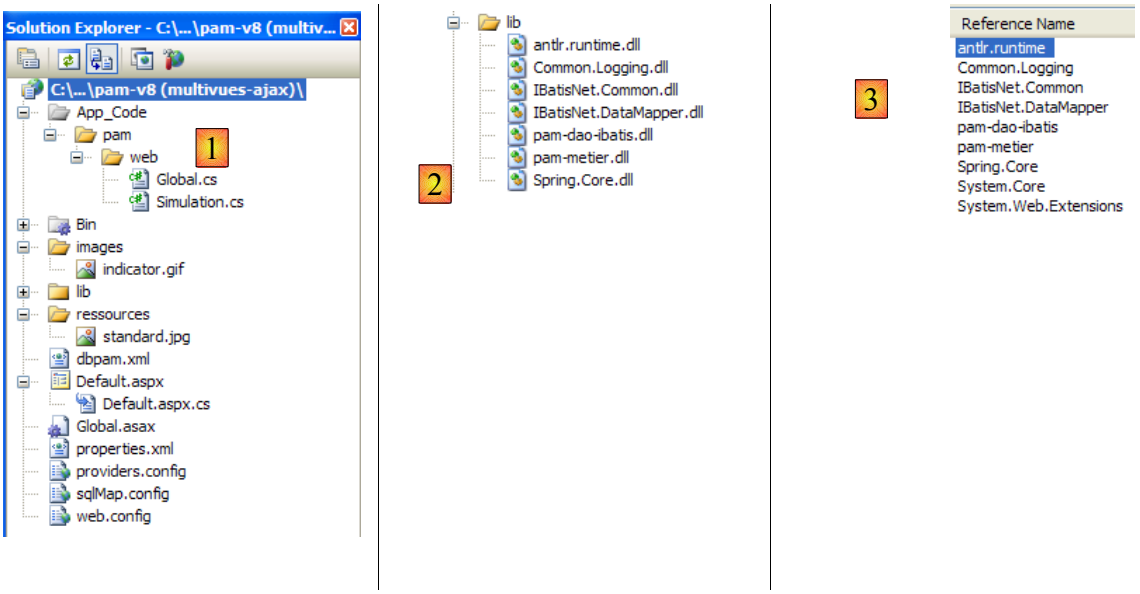


– la vue [VueErreurs] qui indique une ou plusieurs erreurs :



10.2 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 [Global.cs] 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 : [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.

10.3 Le fichier [Global.cs]

Le fichier [Global.cs] 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="C#" inherits="Pam.Web.Global"  %>
```

Global.cs

```
1.  using System;
2.  using System.Web;
3.  using Pam.Dao.Entites;
4.  using Pam.Metier.Service;
5.  using Spring.Context.Support;
6.  using System.Collections.Generic;
7.  using istia.st.pam.web;
8.
9.  namespace Pam.Web
10. {
11.     public class Global : HttpApplication
12.     {
13.
14.         // --- données statiques de l'application ---
15.         public static Employe[] Employes;
16.         public static string Msg = string.Empty;
17.         public static bool Erreur = false;
18.         public static IPamMetier PamMetier = null;
19.
20.         // démarrage de l'application
21.         public void Application_Start(object sender, EventArgs e)
22.         {
23. ...
24.         }
25.
26.         // démarrage de la session d'un utilisateur
27.         public void Session_Start(object sender, EventArgs e)
28.         {
29.             // on met une liste de simulations vide dans la session
30.             List<Simulation> simulations = new List<Simulation>();
31.             Session["simulations"] = simulations;
32.         }
33.
34.     }
35. }
```

- lignes 27-34 : on gère le démarrage de la session. Nous mettrons dans celle-ci la liste des simulations faites par l'utilisateur.
- ligne 30 : une liste de simulations vide est créée. Une simulation est un objet de type [Simulation] que nous allons détailler prochainement.
- ligne 31 : la liste de simulations est placée dans la session associée à la clé " simulations "

10.4 La classe [Simulation]

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

Son code est le suivant :

```
1.  namespace Pam.Web
2.  {
3.
4.      public class Simulation
5.      {
6.          // données d'une simulation
7.          public string Nom { get; set; }
8.          public string Prenom { get; set; }
9.          public double HeuresTravaillees { get; set; }
10.         public int JoursTravailles { get; set; }
11.         public double SalaireBase { get; set; }
12.         public double Indemnites { get; set; }
13.         public double CotisationsSociales { get; set; }
14.         public double SalaireNet { get; set; }
15.
16.         // constructeurs
17.         public Simulation()
18.         {
19.
20.         }
21.
22.         public Simulation(string nom, string prenom, double heuresTravaillees, int
    joursTravailles, double salaireBase, double indemnites, double cotisationsSociales, double
    salaireNet)
23.         {
24.             {
25.                 this.Nom = nom;
26.                 this.Prenom = prenom;
27.                 this.HeuresTravaillees = heuresTravaillees;
28.                 this.JoursTravailles = joursTravailles;
29.                 this.SalaireBase = salaireBase;
30.                 this.Indemnites = indemnites;
31.                 this.CotisationsSociales = cotisationsSociales;
32.                 this.SalaireNet = salaireNet;
33.             }
34.         }
35.     }
36. }
```

Les champs de la classe correspondent aux colonnes du tableau des simulations.

10.5 La page [Default.aspx]

10.5.1 Vue d'ensemble

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

```
1.  <%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="PagePam" %>
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">
```

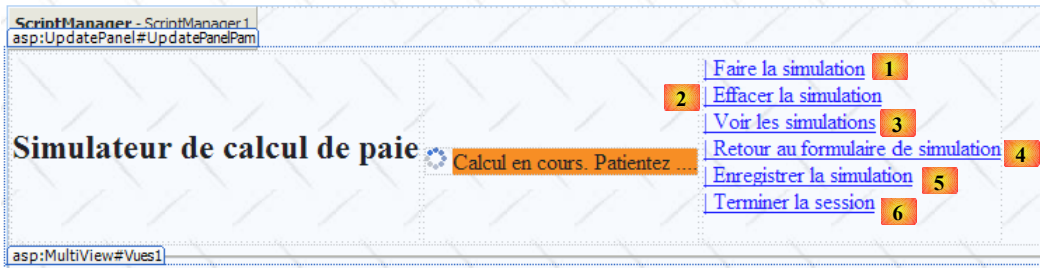
```
10.     <asp:ScriptManager ID="ScriptManager1" runat="server" EnablePartialRendering="true" />
11.     <asp:UpdatePanel runat="server" ID="UpdatePanelPam" UpdateMode="Conditional">
12.         <ContentTemplate>
13.             <table>
14.                 <tr>
15.                     <td>
16.                         <h2>
17.                             Simulateur de calcul de paie</h2>
18.                     </td>
19.                     <td>
20.                         <label>
21.                                </label>
22.                         <asp:UpdateProgress ID="UpdateProgress1" runat="server">
23.                             <ProgressTemplate>
24.                                 <img src="images/indicator.gif" />
25.                                 <asp:Label ID="Label5" runat="server" BackColor="#FF8000"
26.                                  EnableViewState="False" Text="Calcul en cours. Patientez ....">
27.                                 </asp:Label>
28.                             </ProgressTemplate>
29.                         </asp:UpdateProgress>
30.                     </td>
31.                     <td>
32.                        <asp:LinkButton ID="LinkButtonFaireSimulation" runat="server"
33.                         CausesValidation="False" OnClick="LinkButtonFaireSimulation_Click">
34.                             | Faire la simulation<br /></asp:LinkButton>
35.                        <asp:LinkButton ID="LinkButtonEffacerSimulation" runat="server"
36.                                 CausesValidation="False" OnClick="LinkButtonEffacerSimulation_Click">
37.                             | Effacer la simulation<br /></asp:LinkButton>
38.                        <asp:LinkButton ID="LinkButtonVoirSimulations" runat="server"
39.                         CausesValidation="False" OnClick="LinkButtonVoirSimulations_Click">
40.                             | Voir les simulations<br /></asp:LinkButton>
41.                         <asp:LinkButton ID="LinkButtonFormulaireSimulation" runat="server"
42.                         CausesValidation="False" OnClick="LinkButtonFormulaireSimulation_Click">
43.                         | Retour au formulaire de simulation<br /></asp:LinkButton>
44.                         <asp:LinkButton ID="LinkButtonEnregistrerSimulation" runat="server"
45.                         CausesValidation="False" OnClick="LinkButtonEnregistrerSimulation_Click">
46.                             | Enregistrer la simulation<br /></asp:LinkButton>
47.                         <asp:LinkButton ID="LinkButtonTerminerSession" runat="server"
48.                         CausesValidation="False" OnClick="LinkButtonTerminerSession_Click">
49.                             | Terminer la session<br /></asp:LinkButton>
50.                     </td>
51.             </table>
52.             <hr />
53.             <asp:MultiView ID="Vues1" ActiveViewIndex="0" runat="server">
54.                 <asp:View ID="VueSaisies" runat="server">
55. ...
56.                 </asp:View>
57.             </asp:MultiView>
58.             <asp:MultiView ID="Vues2" runat="server">
59.                 <asp:View ID="VueSimulation" runat="server">
60. ...
61.                 </asp:View>
62.                 <asp:View ID="VueSimulations" runat="server">
63. ...
64.                 </asp:View>
65.                 <asp:View ID="VueSimulationsVides" runat="server">
66. ...
67.                 </asp:View>
68.                 <asp:View ID="VueErreurs" runat="server">
69. ...
70.                 </asp:View>
71.             </asp:MultiView>
72.         </ContentTemplate>
73.     </asp:UpdatePanel>
74.     </form>
75. </body>
76. </html>
```

- ligne 10 : la balise pour disposer des extensions Ajax
- lignes 11-73 : le conteneur *UpdatePanel* mis à jour par des appels Ajax
- lignes 12-72 : le contenu du coneneur *UpdatePanel*
- lignes 13-52 : l'entête qui sera présent dans chaque vue. Il présente à l'utilisateur la liste des actions possibles sous la forme d'une liste de liens.
- ligne 33 : on notera l'attribut *CausesValidation="False"* qui fait que les validateurs de la page ne seront pas exécutés **implicitement** lorsque le lien sera cliqué. Lorsque cet attribut est absent, sa valeur par défaut est *True*. La validation de la page sera faite **explicitement** dans le code par l'opération *Page.Validate*.
- lignes 53-57 : un composant [MultiView] avec une unique vue [VueSaisies]. Pour cette raison, on a mis en dur, ligne 53, le n° de la vue à afficher : *ActiveViewIndex="0"*
- lignes 53-67 : un composant [MultiView] avec quatre vues : la vue [VueSimulation] lignes 59-61, la vue [VueSimulations] lignes 62-64, la vue [VueSimulationsVides] lignes 65-67, la vue [VueErreurs] lignes 68-70. 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
```

10.5.2 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

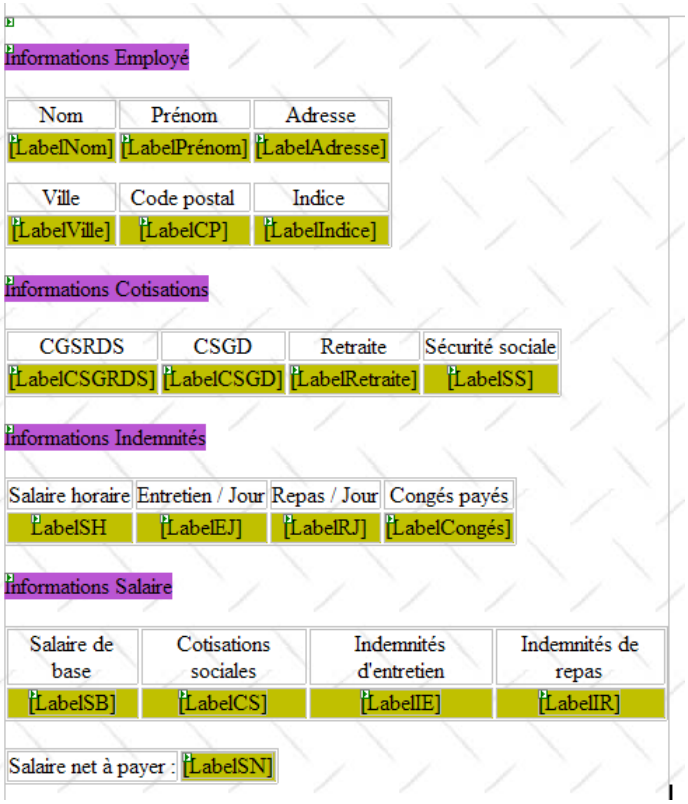
10.5.3 La vue [Saisies]

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

10.5.4 La vue [Simulation]

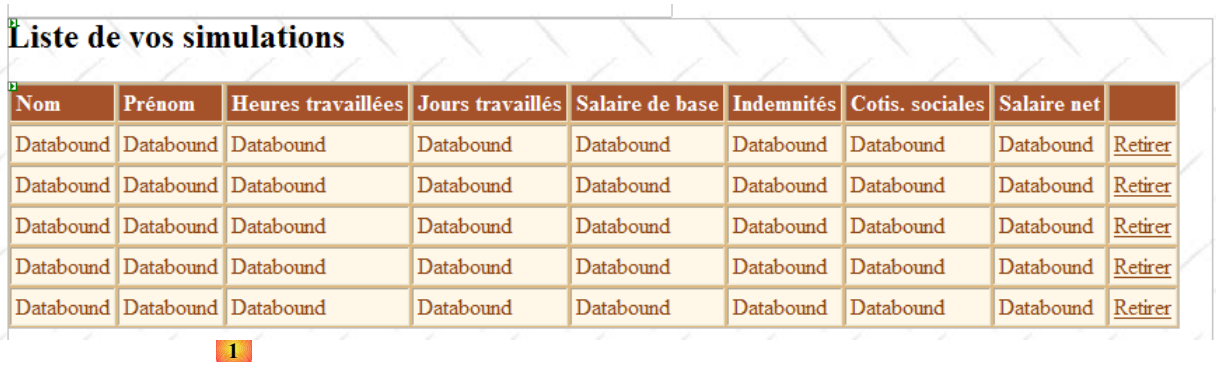
Le composant [View] nommé [VueSimulation] est le suivant :



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

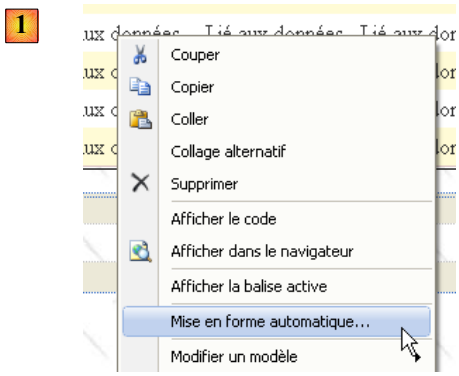
10.5.5 La vue [Simulations]

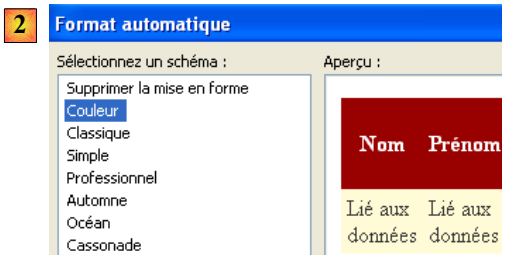
Le composant [View] nommé [VueSimulations] est le suivant :



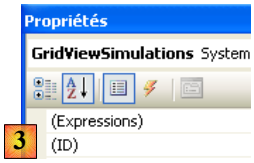
N°	Type	Nom	Rôle
1	GridView	GridViewSimulations	Contient la liste des simulations

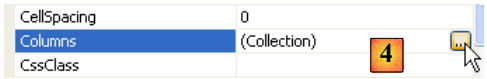
Les propriétés du composant [GridViewSimulations] ont été définies de la façon suivante :

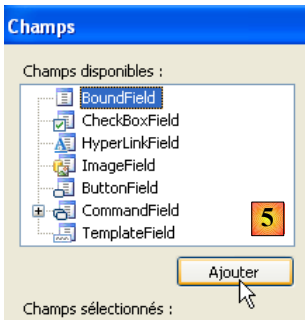




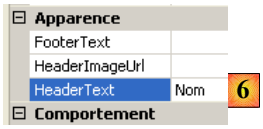
- en [1] : clic droit sur le [GridView] / option [Mise en forme automatique]
- en [2] : choisir un type d'affichage pour le [GridView]







- 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].



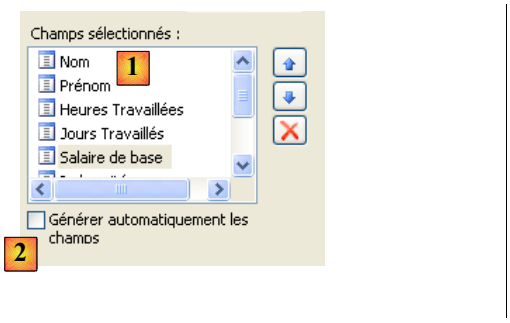
- en [6] : donner le titre de la colonne
- en [7] : donner le nom de la propriété de la classe [Simulation] qui sera associée à cette colonne.
- [DataFormatString] indique comment doivent être formatées les valeurs affichées dans la colonne.

Les colonnes du composant [GridViewSimulations] ont les propriétés 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}**

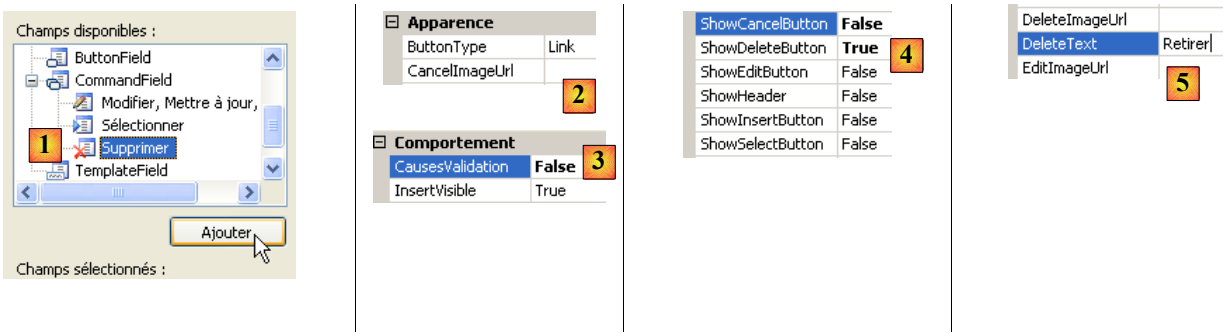
On prêtera attention au fait que le champ [DataField] doit correspondre à une propriété existante de la classe [Simulation]. A l'issue de cette phase, toutes les colonnes de type [BoundField] ont été créées :



- 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] :

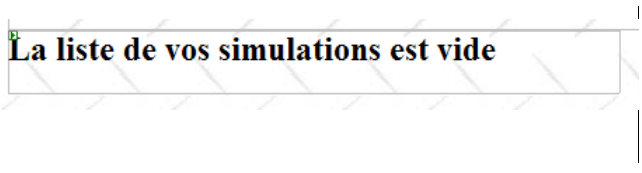
Nom	Prénom	Heures Travaillées	Jours Travaillés	Salaire de base	Indemnités	Cotis. sociales	Salaire net	
Laverti	Justine	150	20	324,24 €	100,00 €	87,25 €	336,99 €	Retirer



- 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

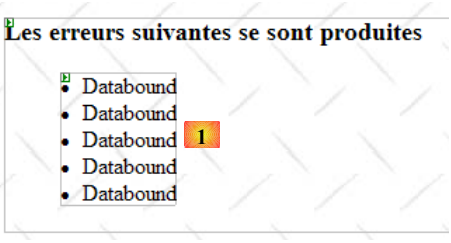
10.5.6 La vue [SimulationsVides]

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



10.5.7 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 :

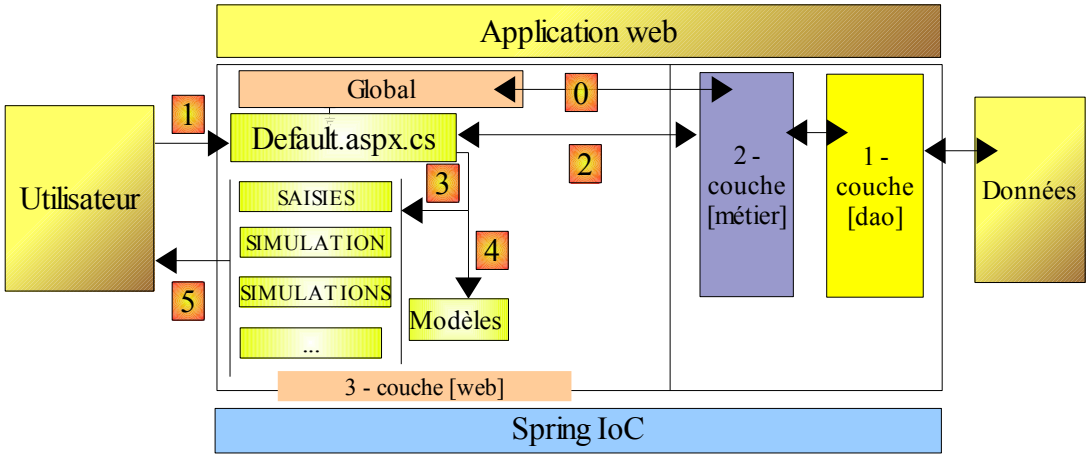
```
1.         <asp:Repeater ID="RptErreurs" runat="server">
2.           <ItemTemplate>
3.             <li>
4.               <%# Container.DataItem %>
5.             </li>
6.           </ItemTemplate>
7.  </asp:Repeater>
```

- 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 <*li*>*Message*</*li*> dans le flux HTML de la page.

10.6 Le contrôleur [Default.aspx.cs]

10.6.1 Vue d'ensemble

Revenons à l'architecture MVC de l'application :



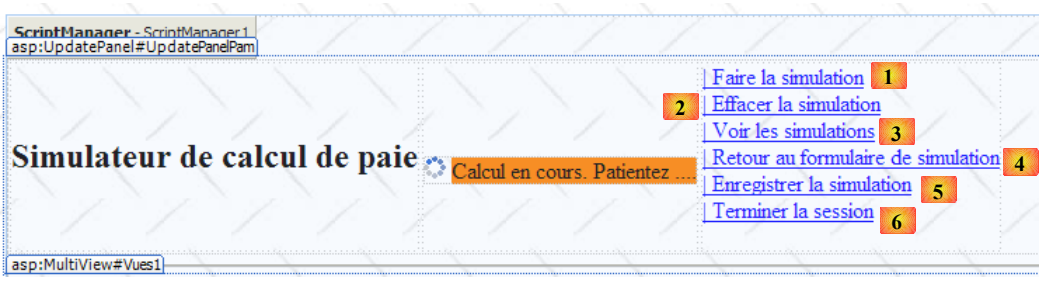
- [Default.aspx.cs] qui est le code de contrôle de la page unique [Default.aspx] est le contrôleur de l'application.
- [Global] est l'objet de type [HttpApplication] qui initialise l'application et qui dispose d'une référence sur la couche [métier].

Le squelette du code du contrôleur [Default.aspx.cs] est le suivant :

```
1.  using System.Collections.Generic;
2.  ...
3.
4.  public partial class PagePam : Page
5.  {
6.
7.      private void setVues(bool boolVues1, bool boolVues2, int index)
8.      {
9.          // on affiche les vues demandées
10.         // boolVues1 : true si le multivues Vues1 doit être visible
11.         // boolVues1 : true si le multivues Vues2 doit être visible
12.         // index : index de la vue de Vues2 à afficher
13. ...
14.     }
15.
16.     private void setMenu(bool boolFaireSimulation, bool boolEnregistrerSimulation, bool
    boolEffacerSimulation, bool boolFormulaireSimulation, bool boolVoirSimulations, bool
    boolTerminerSession)
17.     {
18.         // on fixe les options de menu
19.         // chaque booléen est affecté à la propriété Visible du lien correspondant
20. ...
21.     }
22.
23.     // chargement de la page
24.     protected void Page_Load(object sender, System.EventArgs e)
25.     {
26.         // traitement requête initiale
27.         if (!IsPostBack)
28.         {
29. ...
30.         }
31.     }
32.
33.     protected void LinkButtonFaireSimulation_Click(object sender, System.EventArgs e)
34.     {
35. ...
36.     }
37.
38.     protected void LinkButtonEffacerSimulation_Click(object sender, System.EventArgs e)
39.     {
40. ....
41.     }
42.
43.     protected void LinkButtonVoirSimulations_Click(object sender, System.EventArgs e)
44.     {
45. ...
46.     }
47.
48.     protected void LinkButtonEnregistrerSimulation_Click(object sender, System.EventArgs e)
49.     {
50. ...
51.     }
52.
53.     protected void LinkButtonTerminerSession_Click(object sender, System.EventArgs e)
54.     {
55. ...
56.     }
57.
58.     protected void LinkButtonFormulaireSimulation_Click(object sender, System.EventArgs e)
59.     {
60. ...
61.     }
62.
63.
64.     protected void GridViewSimulations_RowDeleting(object sender, GridViewDeleteEventArgs e)
65.     {
66. ...
67.     }
68. }
```

A la requête initiale (GET) de l'utilisateur, seul l'événement *Load* des lignes 24-31 est traité. Aux requêtes suivantes (POST) faites via les liens du menu, deux événements sont traités :

1. l'événement *Load* (24-31) mais le test du booléen *Page.IsPostback* (ligne 27) fait que rien ne sera fait.
2. l'événement lié au lien qui a été cliqué :



- lignes 33-36 : traitent le clic sur le lien [1]
- lignes 38-41 : traitent le clic sur le lien [2]
- lignes 43-46 : traitent le clic sur le lien [3]
- lignes 58-61 : traitent le clic sur le lien [4]
- lignes 48-51 : traitent le clic sur le lien [5]
- lignes 53-56 : traitent le clic sur le lien [6]

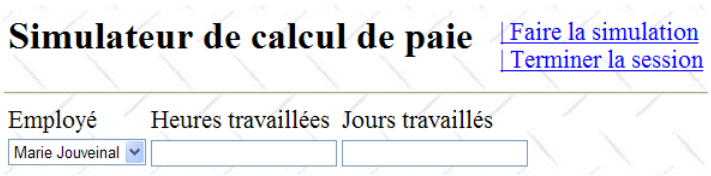
Pour factoriser des séquences de code revenant souvent, deux méthodes internes ont été créées :

- **setVues**, lignes 7-14 : fixe la ou les vues à afficher
- **setMenu**, lignes 16-21 : fixe les options de menu à afficher

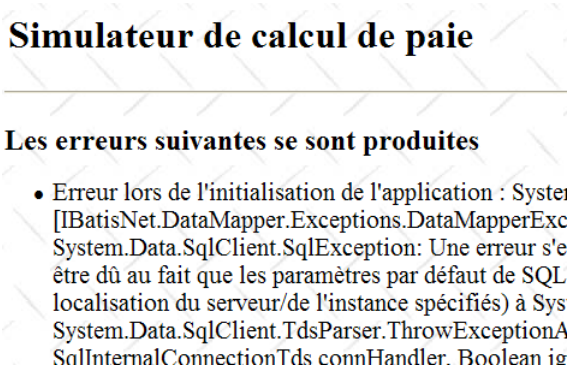
10.6.2 L'événement Load

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 :



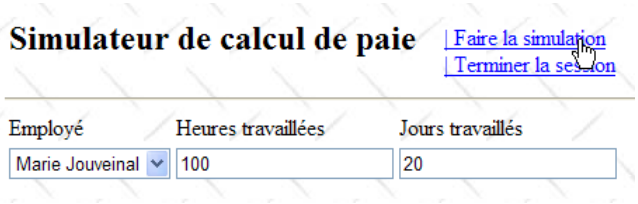
L'événement [Load] est traité de façon analogue à celle des versions ASP.NET précédentes :

```
1.      // chargement de la page
2.      protected void Page_Load(object sender, System.EventArgs e)
3.      {
4.          // traitement requête initiale
5.          if (!IsPostBack)
6.          {
7.              // des erreurs d'initialisation ?
8.              if (Global.Erreur)
9.              {
10.                 // affichage vue [erreurs]
11. ...
12.                 // positionnement menu
13.                 ...
14.                 return;
15.             }
16.             // chargement des noms des employés dans le combo
17. ...
18.             // positionnement menu
19. ...
20.             // affichage vue [saisie]
21.  ...
22.         }
23. }
```

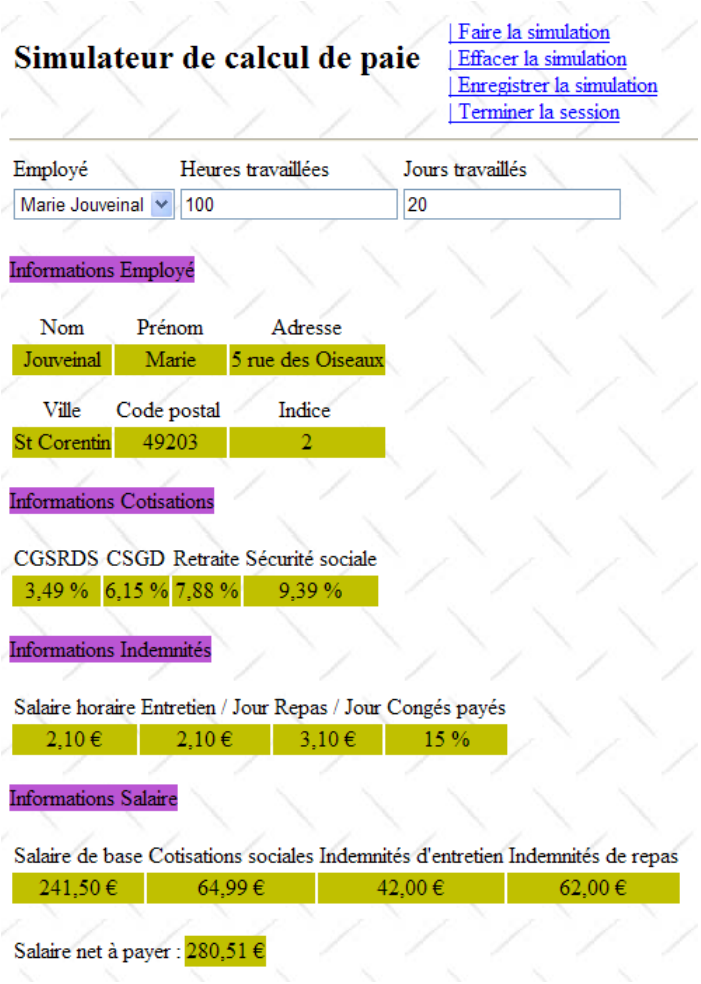
Question : compléter le code ci-dessus

10.6.3 Action : Faire la simulation

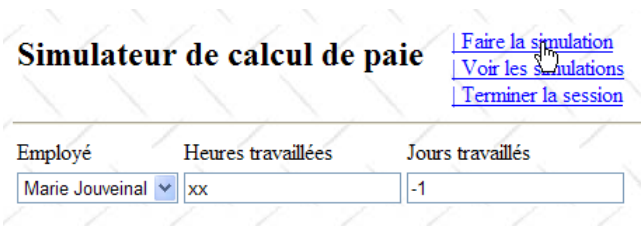
Dans ce qui suit, l'écran noté (1) est celui de la demande de l'utilisateur, l'écran noté (2) la réponse qui lui est envoyée par l'application web. A partir de l'écran d'accueil, l'utilisateur peut commencer une simulation :



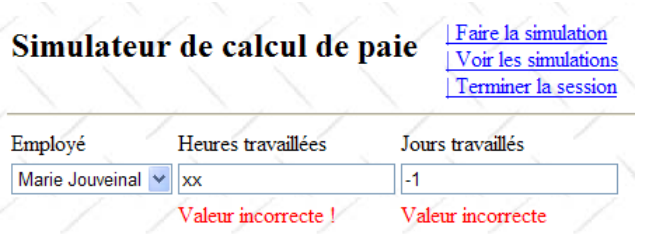
- (1) : l'utilisateur demande une simulation



- (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 :

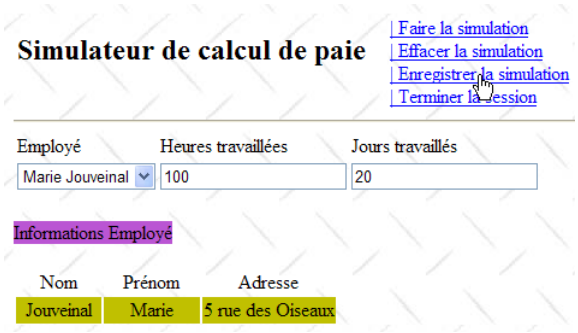
```
1.      protected void LinkButtonFaireSimulation_Click(object sender, System.EventArgs e)
2.      {
3.          // calcul du salaire
4.          // page valide ?
5.          Page.Validate();
6.          if (!Page.IsValid)
7.          {
8.              // affichage vue [saisie]
9.              ...
10.             return;
11.         }
```

```
12.         // la page est valide - on récupère les saisies
13.         double HeuresTravaillées = ...;
14.         int JoursTravaillés = ...;
15.         // on calcule le salaire de l'employé
16.         FeuilleSalaire feuillesalaire;
17.         try
18.         {
19.             feuillesalaire = ...
20.         }
21.         catch (PamException ex)
22.         {
23.             // affichage vue [erreurs]
24. ...
25.             return;
26.         }
27.         // on met le résultat dans la session
28.         Session["simulation"] = ...
29.         // affichage résultats
30. ...
31.         // affichage vues [saisie, employe, salaire]
32.         ...
33.         // affichage menu
34.         ...
35. }
```

Question : compléter le code ci-dessus

10.6.4 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 :

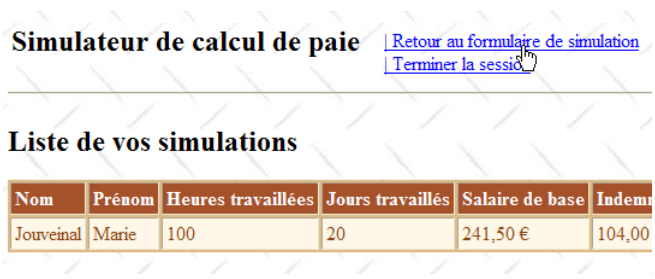
```
1.      protected void LinkButtonEnregistrerSimulation_Click(object sender, System.EventArgs e)
2.      {
3.          // on enregistre la simulation courante dans la liste des simulations présente dans la
    session
4.          ...
5.          // on affiche la vue [simulations]
6.          ...
7.  }
```

Question : compléter le code ci-dessus

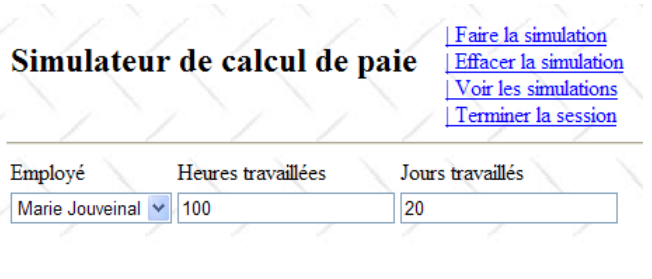
10.6.5 Action : Retour au formulaire de simulation

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*.

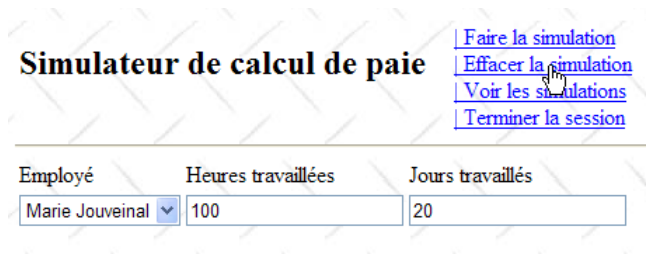
La procédure qui traite cette action pourrait ressembler à ce qui suit :

```
1.      protected void LinkButtonFormulaireSimulation_Click(object sender, System.EventArgs e)
2.      {
3.          // affichage vue [saisie]
4.          ...
5.          // positionnement menu
6.          ...
7.  }
```

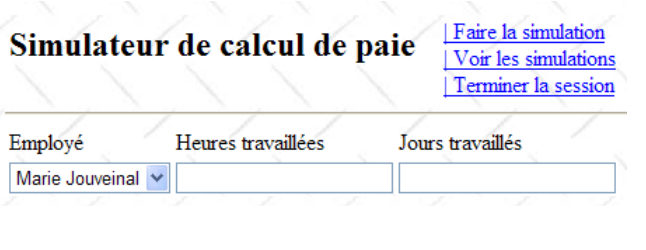
Question : compléter le code ci-dessus

10.6.6 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

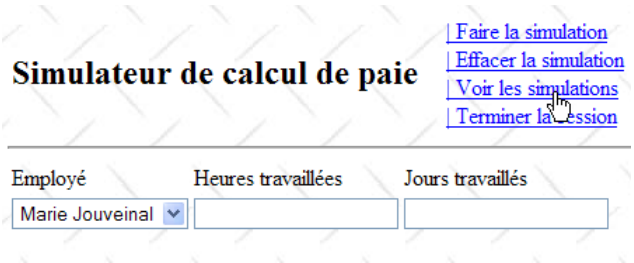
La procédure qui traite cette action pourrait ressembler à ce qui suit :

```
1.      protected void LinkButtonEffacerSimulation_Click(object sender, System.EventArgs e)
2.      {
3.          // RAZ du formulaire
4.          ...
5.  }
```

Question : compléter le code ci-dessus

10.6.7 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 void LinkButtonVoirSimulations_Click(object sender, System.EventArgs e)
2.      {
3.          // on récupère les simulations dans la session
4.          ...
5.          // y-a-t-il des simulations ?
6.          if (...)
7.          {
8.              // vue [simulations] visible
9.  ...
10.         }
11.         else
12.         {
13.             // vue [SimulationsVides]
14.             ...
15.         }
16.         // on fixe le menu
17.         ...
18.     }
```

Question : compléter le code ci-dessus

10.6.8 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 :

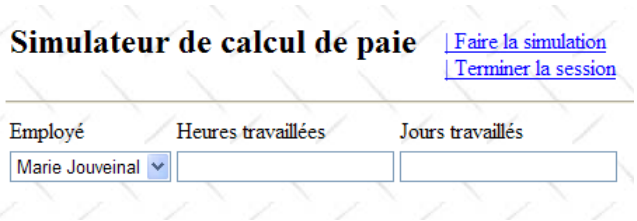
```
1.      protected void GridViewSimulations_RowDeleting(object sender, GridViewDeleteEventArgs e)
2.      {
3.          // on récupère les simulations dans la session
4.          ...
5.          // on supprime la simulation désignée (e.RowIndex est le n° de la ligne supprimée)
6.          ...
7.          // reste-t-il des simulations ?
8.          if (...)
9.          {
10.             // on remplit le GridView
11.  ...
12.         }
13.         else
14.         {
15.             // vue [SimulationsVides]
16.             ...
17.         }
18. }
```

Question : compléter le code ci-dessus

10.6.9 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 void LinkButtonTerminerSession_Click(object sender, System.EventArgs e)
2.      {
3.          // on abandonne la session
4.          ...
5.          // effacer la simulation
6.          ...
7.          // afficher la vue [saisies]
8.          ...
9.          // positionnement menu
10.         ...
11. }
```

Question : compléter le code ci-dessus

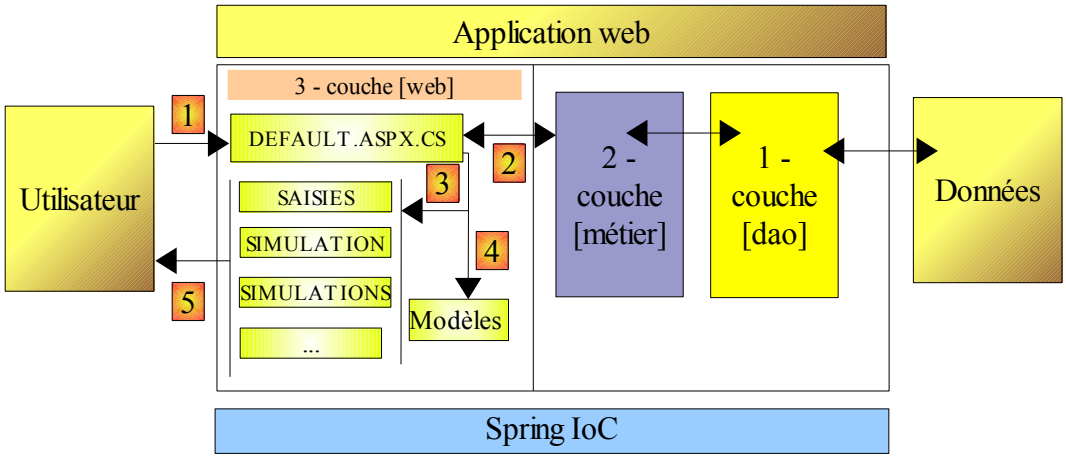
Travail pratique : mettre en oeuvre sur machine l'application web précédente. Ajoutez-lui un comportement Ajax.

11 L'application [SimuPaie] – version 9 – 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.

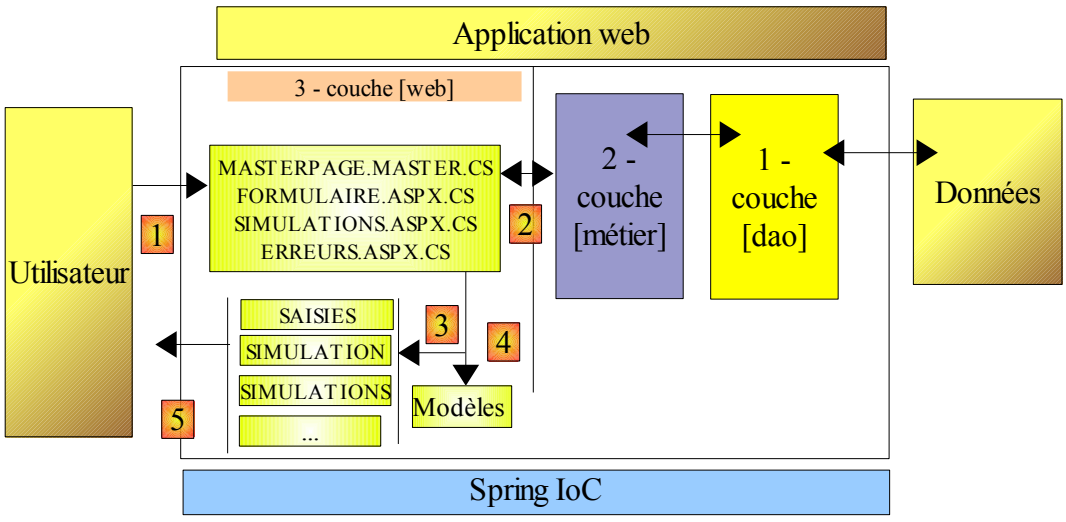
L'architecture de l'application précédente était la suivante :



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

- [Default.aspx.cs] 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.

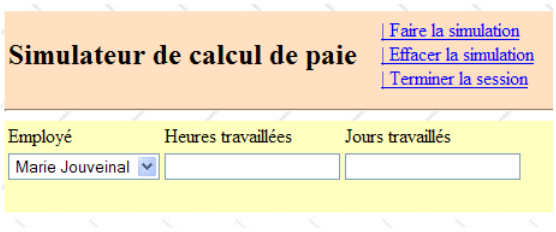
Le traitement d'une demande d'un client se déroule selon les étapes suivantes :

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)

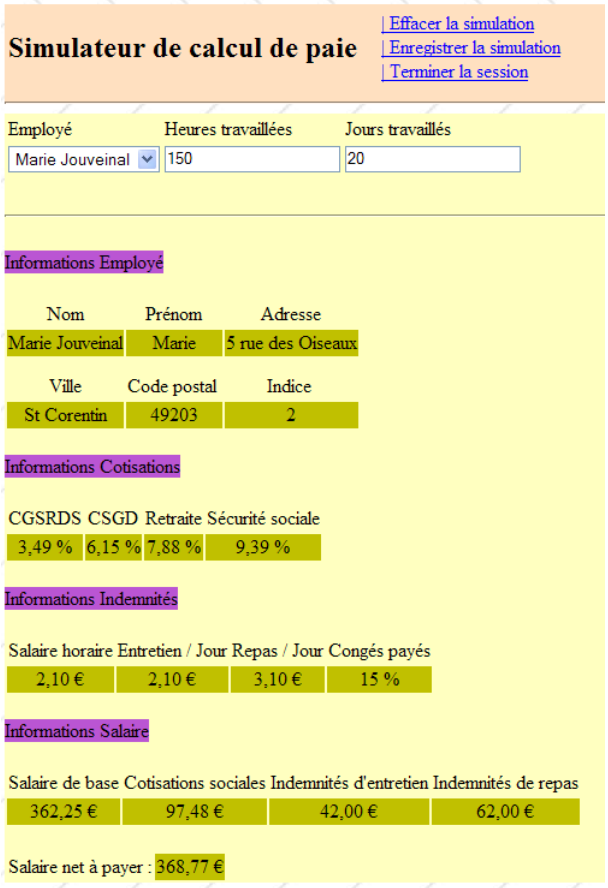
11.1 Les vues de l'application

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

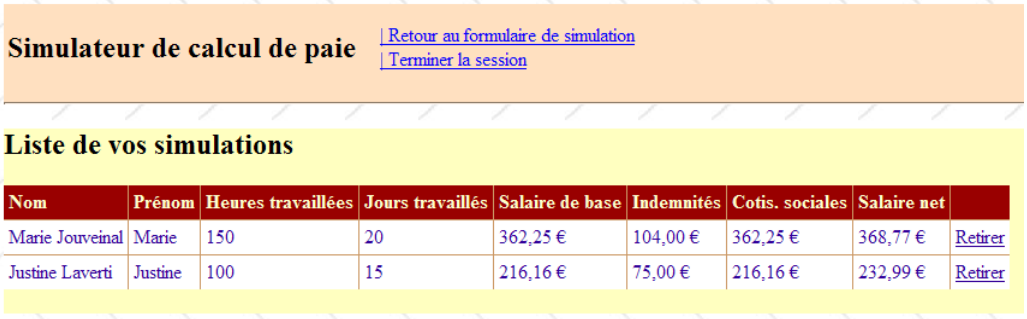
- la vue [VueSaisies] qui présente le formulaire de simulation



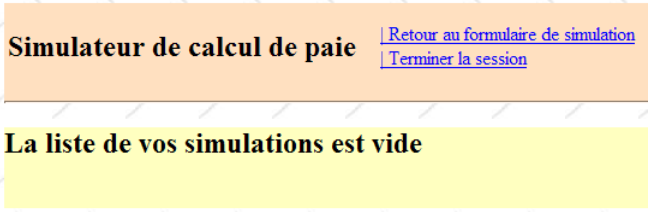
- la vue [VueSimulation] utilisée pour afficher le résultat détaillé de la simulation :



- 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 :

Simulateur de calcul de paie

Les erreurs suivantes se sont produites au démarrage de l'application

- Spring.Objects.Factory.ObjectCreationException: Error thrown by a dependency of object 'pammetier' defined in 'file [C:\data\2006-2007\aspnet\pam\v1-3tier-multipages\spring-config.xml]' : Initialization of object failed : Cannot instantiate Type [istia.st.pam.dao.service.PamDaoSqlMap] using ctor [Void .ctor()] : 'Exception has been thrown by the target of an invocation.' while resolving 'PamDao' to 'pamdao' defined in 'file [C:\data\2006-2007\aspnet\pam\v1-3tier-multipages\spring-config.xml]' ---> Spring.Objects.FatalObjectException: Cannot instantiate Type [istia.st.pam.dao.service.PamDaoSqlMap] using ctor [Void .ctor()] : 'Exception has been thrown by the target of an invocation.' ---> System.Reflection.TargetInvocationException: Exception has been thrown by the target of an invocation. ---> istia.st.pam.dao.entites.PamException: Erreur d'accès à la BD lors de la demande des cotisations: [IBatisNet.DataMapper.Exceptions.DataMapperException: Unable to open connection to "Microsoft SQL Server, provider V2.0.0.0 in framework .NET V2.0". ---> System.Data.SqlClient.SqlException: An error has occurred while establishing a

11.2 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 ou deux composants [View] appartenant à ces deux composants [MultiView].

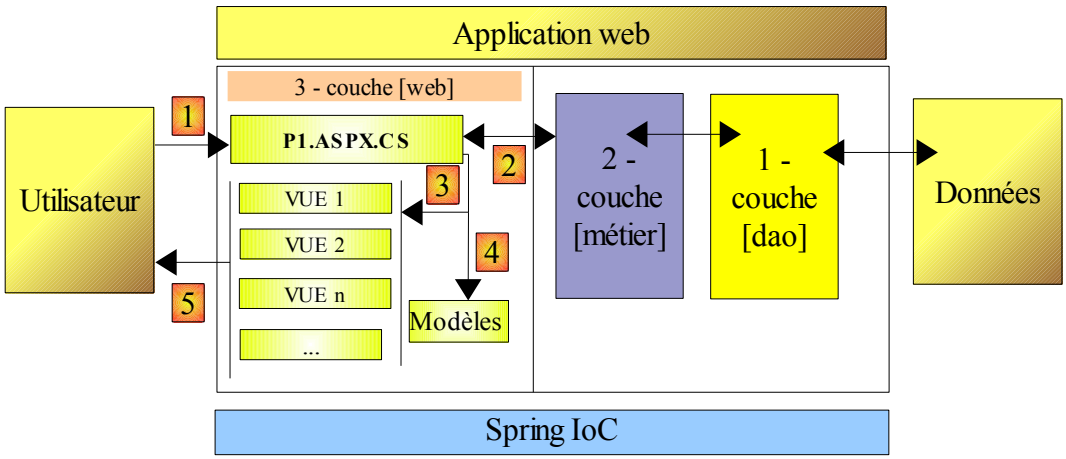
Efficace lorsqu'il y a peu de vues, cette architecture atteint ses limites dès que le nombre des composants formant les différentes vues devient important : en effet, à chaque requête qui est faite à l'unique page [Default.aspx], tous les composants de celle-ci sont instanciés alors même que seuls certains 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

11.2.1 Cas 1 : une page contrôleur / vue

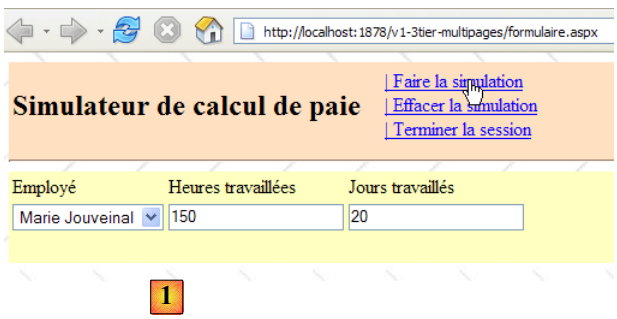
Dans le cas 1, on retombe sur l'architecture mono-contrôleur de la version précédente, où la page [Default.aspx] est la page P1 :



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**]



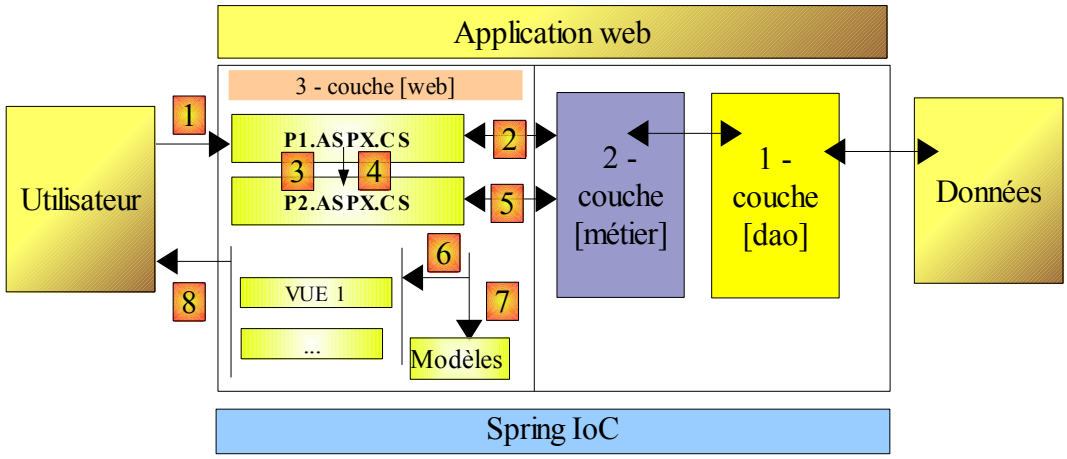
- en [1] : l'utilisateur, après avoir demandé la page [Formulaire.aspx], demande une simulation
- en [2] : la page [Formulaire.aspx] a traité cette demande et généré elle-même la réponse en affichant un composant [View] qui n'avait pas été affiché en [1]

[**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.

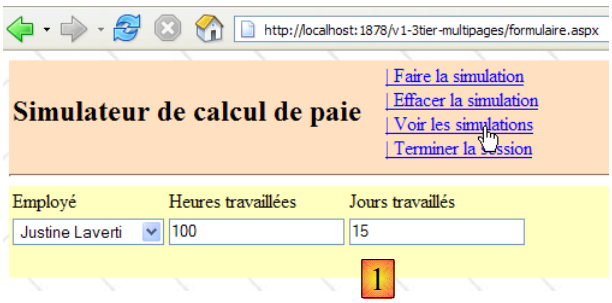
11.2.2 Cas 2 : une page 1 contrôleur, une page 2 controleur / vue

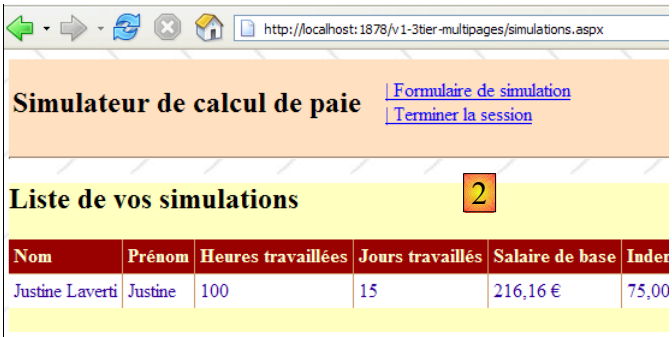
Le cas 2 peut recouvrir diverses d'architectures. Nous choisirons la suivante :



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.[" 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=(Type)Context.Items[" clé "]*] ou bien [*valeur=(Type)Session[" clé "]*] 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 :

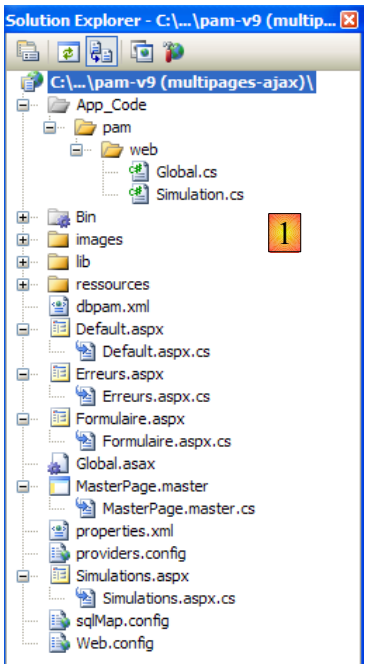


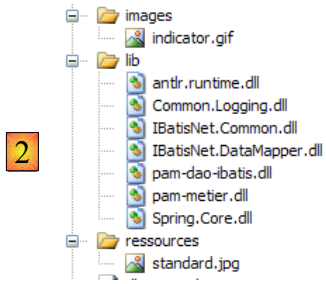


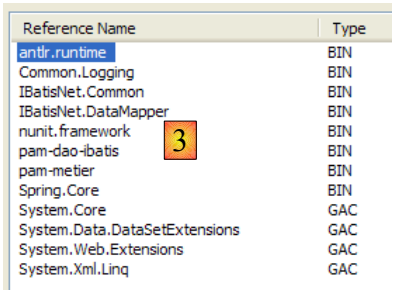
- en [1] : l'utilisateur qui a demandé la page [Formulaire.aspx] demande à voir la liste des simulations
- en [2] : la page [Formulaire.aspx] traite cette demande et redirige le client vers la page [Simulations.aspx]. C'est cette dernière qui fournit la réponse à l'utilisateur. Au lieu de demander au 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.

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

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





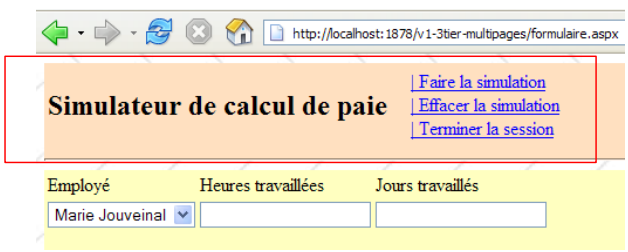


- en [1] on trouve :
  - le fichier de configuration [web.config] de l'application – est identique à celui de l'application précédente.
  - les fichiers de configuration des différentes couches de l'application web : [sqlmap.config, providers.config, properties.xml, dbpam.xml] – sont identiques aux fichiers 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.
  - le fichier [Global.cs] qui gère les événements de l'application web et le fichier [Simulation.cs] de la classe [Simulation]. Ces deux fichiers sont ceux 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.

11.4 Le code de présentation des pages

11.4.1 La page maître [MasterPage.master]

Les vues de l'application présentées au paragraphe 11.1, page 94, 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] :

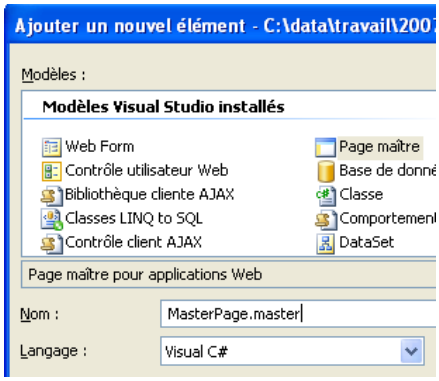


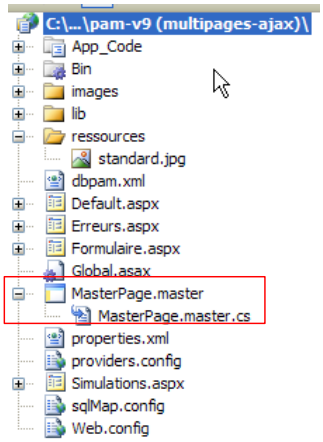
Ces deux vues possèdent en commun, le bandeau supérieur (Titre et Options de menu). Il en est ainsi de toutes les vues qui seront présentées à l'utilisateur : elles auront toutes le même bandeau supérieur. Pour que différentes pages partagent un même fragment de présentation, il existe diverses solutions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





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.cs] : le code de contrôle de la page Maître

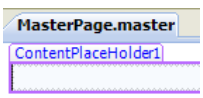
Le code généré par Visual Studio dans [MasterPage.master] est le suivant :

```
1.  <%@ Master Language="C#" AutoEventWireup="true" CodeFile="MasterPage.master.cs"
    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.      <asp:ContentPlaceHolder id="head" runat="server">
9.      </asp:ContentPlaceHolder>
10. </head>
11. <body>
12.     <form id="form1" runat="server">
13.     <div>
14.         <asp:ContentPlaceHolder id="ContentPlaceHolder1" runat="server">
15.
16.         </asp:ContentPlaceHolder>
17.     </div>
18.     </form>
19. </body>
20. </html>
```

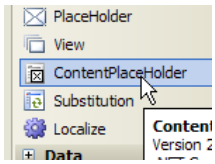
- ligne 1 : la balise <%@ Master ... %> sert à définir la page comme une page Maître. Le code de contrôle de la page sera dans le fichier défini par l'attribut *CodeFile*, et la page héritera de la classe définie par l'attribut *Inherits*.
- lignes 12-18 : le formulaire de la page Maître
- lignes 14-16 : 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.
- lignes 8-9 : un conteneur vide avec lequel les pages "fille" pourront personnaliser l'entête <head>...</head>.

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].

1



2



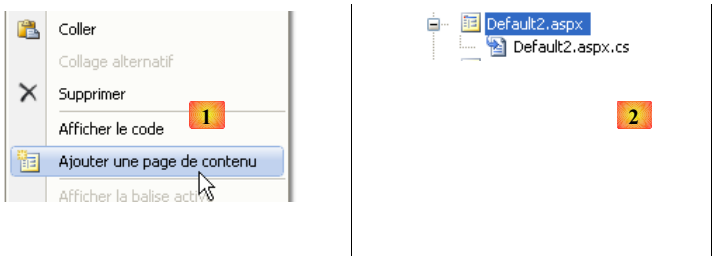
Le code de contrôle généré par Visual Studio dans [MasterPage.master.vb] est le suivant :

```
1.  using System;
2.
3.  public partial class MasterPage : System.Web.UI.MasterPage
4.  {
5.      protected void Page_Load(object sender, EventArgs e)
6.      {
7.
8.      }
9.  }
```

- ligne 3 : la classe référencée par l'attribut [Inherits] de la directive <%@ Master ... %> de la page [MasterPage.master] dérive de la classe [System.Web.UI.MasterPage]

Ci-dessus, nous voyons la présence de la méthode *Page_Load* qui gère l'événement *Load* de la page maître. La page maître 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 [Ajouter une page de contenu]
- en [2] : une page par défaut, ici [Default2.aspx] est générée.

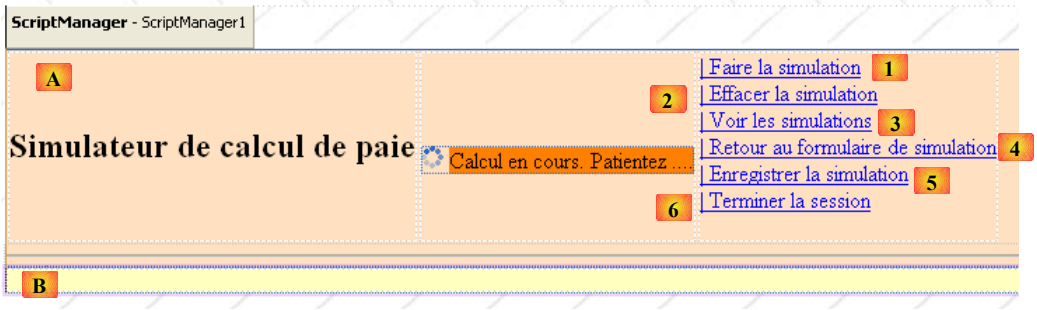
Le code de présentation [Default2.aspx] est le suivant :

```
1.  <%@ Page Language="C#" MasterPageFile="~/MasterPage.master" AutoEventWireup="true"
    CodeFile="Default2.aspx.cs" Inherits="Default2" Title="Page sans titre" %>
2.
3.  <asp:Content ID="Content1" ContentPlaceHolderID="head" Runat="Server">
4.  </asp:Content>
5.  <asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder1" Runat="Server">
6.  </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 3-4 : 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 3-4 ci-dessus, seront à l'exécution, placés dans le conteneur *head* de la page maître.
- lignes 5-6 : les composants à l'intérieur de ces lignes, seront à l'exécution, placés dans le conteneur *ContentPlaceHolder1* de la page maître.

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="C#" AutoEventWireup="true" CodeFile="MasterPage.master.cs"
    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.  <html xmlns="http://www.w3.org/1999/xhtml">
5.  <head id="Head1" runat="server">
6.    <title>Application PAM</title>
7.  </head>
8.  <body background="ressources/standard.jpg">
9.    <form id="form1" runat="server">
10.   <asp:ScriptManager ID="ScriptManager1" runat="server" EnablePartialRendering="true" />
11.   <asp:UpdatePanel runat="server" ID="UpdatePanelPam" UpdateMode="Conditional">
12.     <ContentTemplate>
13.       <asp:Panel ID="entete" runat="server" BackColor="#FFE0C0">
14.         <table>
15.           <tr>
16.             <td>
17.               <h2>
18.                 Simulateur de calcul de paie</h2>
19.             </td>
20.             <td>
21.               <label>
22.                    </label>
23.               <asp:UpdateProgress ID="UpdateProgress1" runat="server">
24.                 <ProgressTemplate>
25.                   <img alt="" src="images/indicator.gif" />
26.                   <asp:Label ID="Label5" runat="server" BackColor="#FF8000"
27.                    EnableViewState="False" Text="Calcul en cours. Patientez ....">
28.                 </asp:Label>
29.                 </ProgressTemplate>
30.               </asp:UpdateProgress>
31.             </td>
32.             <td>
33.               <asp:LinkButton ID="LinkButtonFaireSimulation" runat="server"
34.                   CausesValidation="False">| Faire la simulation<br />
35.              </asp:LinkButton>
36.               <asp:LinkButton ID="LinkButtonEffacerSimulation" runat="server"
37.                   CausesValidation="False">| Effacer la simulation<br />
38.              </asp:LinkButton>
39.               <asp:LinkButton ID="LinkButtonVoirSimulations" runat="server"
```

```
40.                 CausesValidation="False">| Voir les simulations<br />
41.               </asp:LinkButton>
42.                 <asp:LinkButton ID="LinkButtonFormulaireSimulation" runat="server"
43.                     CausesValidation="False">| Retour au formulaire de simulation<br />
44.               </asp:LinkButton>
45.                 <asp:LinkButton ID="LinkButtonEnregistrerSimulation" runat="server"
46.                     CausesValidation="False">| Enregistrer la simulation<br />
47.               </asp:LinkButton>
48.                 <asp:LinkButton ID="LinkButtonTerminerSession" runat="server"
49.                     CausesValidation="False">| Terminer la session<br />
50.               </asp:LinkButton>
51.             </td>
52.           </tr>
53.         </table>
54.         <hr />
55.       </asp:Panel>
56.       <div>
57.         <asp:Panel ID="contenu" runat="server" BackColor="#FFFFC0">
58.           <asp:ContentPlaceHolder ID="ContentPlaceHolder1" runat="server">
59.           </asp:ContentPlaceHolder>
60.         </asp:Panel>
61.       </div>
62.     </ContentTemplate>
63.   </asp:UpdatePanel>
64.   </form>
65. </body>
66. </html>
```

- ligne 1 : on notera le nom de la classe de la page maître : **MasterPage**
- ligne 8 : on définit une image de fond pour la page.
- lignes 9-64 : le formulaire
- ligne 10 : le composant ScriptManager nécessaire aux effets Ajax
- lignes 11-63 : le conteneur AJax
- lignes 12-62 : le contenu ajaxifié
- lignes 13-55 : le composant Panel [entete]
- lignes 57-60 : le composant Panel [contenu]
- lignes 58-59 : 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 10.5.2, page 78.

11.4.2 La page [Formulaire.aspx]

Pour générer cette page, on suivra la méthode exposée page 102 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 :

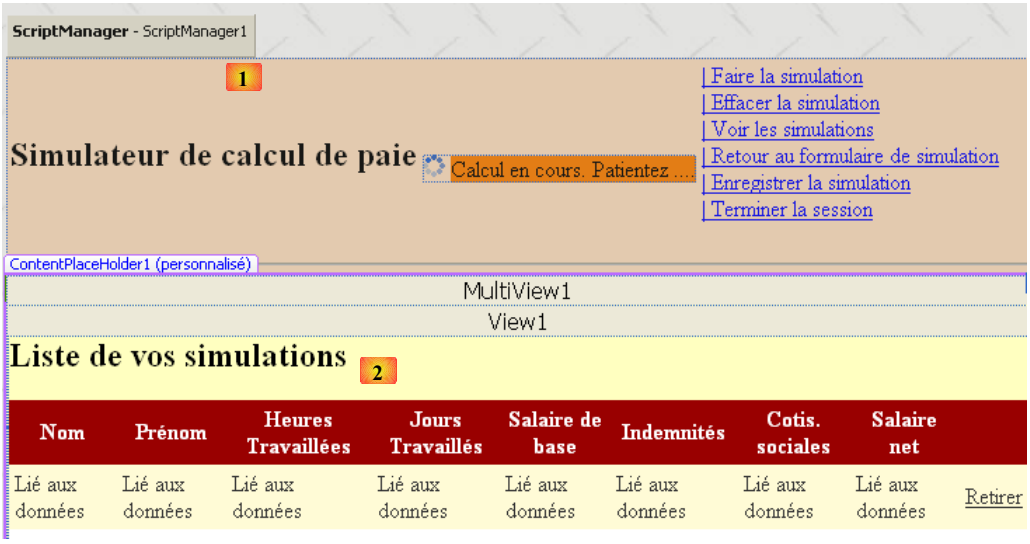
```
1.  <%@ Page Language="C#" MasterPageFile="~/MasterPage.master" AutoEventWireup="true"
2.    CodeFile="Formulaire.aspx.cs" Inherits="PageFormulaire" Title="Simulation de calcul de paie :
    formulaire" %>
3.
4.  <%@ MasterType VirtualPath="~/MasterPage.master" %>
5.  <asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" 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 *(MasterPage)(Master).[champ]* ou *(MasterPage)(Master).[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 [MasterPage].
- 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 10.5.3 de la page 78 et celui de la vue [VueSimulation] décrite au paragraphe 10.5.4 de la page 78.

11.4.3 La page [Simulations.aspx]

Pour générer cette page, on suivra la méthode exposée page 102 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.

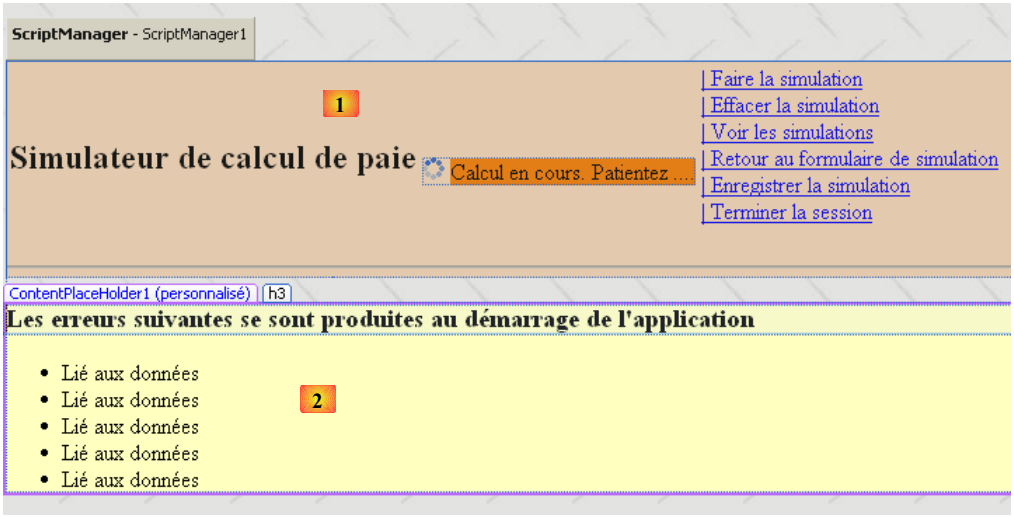
Le code source de cette page est le suivant :

```
1.  <%@ Page Language="C#" MasterPageFile="~/MasterPage.master" AutoEventWireup="true"
2.    CodeFile="Simulations.aspx.cs" Inherits="PageSimulations" Title="Pam : liste des simulations" %>
3.
4.  <%@ MasterType VirtualPath="~/MasterPage.master" %>
5.  <asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" runat="Server">
6.    <asp:MultiView ID="MultiView1" runat="server">
7.      <asp:View ID="View1" runat="server">
8.        <h2>
9.          Liste de vos simulations</h2>
10.       <p>
11.         <asp:GridView ID="GridViewSimulations" runat="server" ...>
12. ...
13.         </asp:GridView>
14.       </p>
15.     </asp:View>
16.     <asp:View ID="View2" 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 5-21), celui de la vue [VueSimulations] décrite au paragraphe 10.5.5 de la page 79 et celui de la vue [VueSimulationsVides] décrite au paragraphe 10.5.6 de la page 81.

11.4.4 La page [Erreurs.aspx]

Pour générer cette page, on suivra la méthode exposée page 102 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.

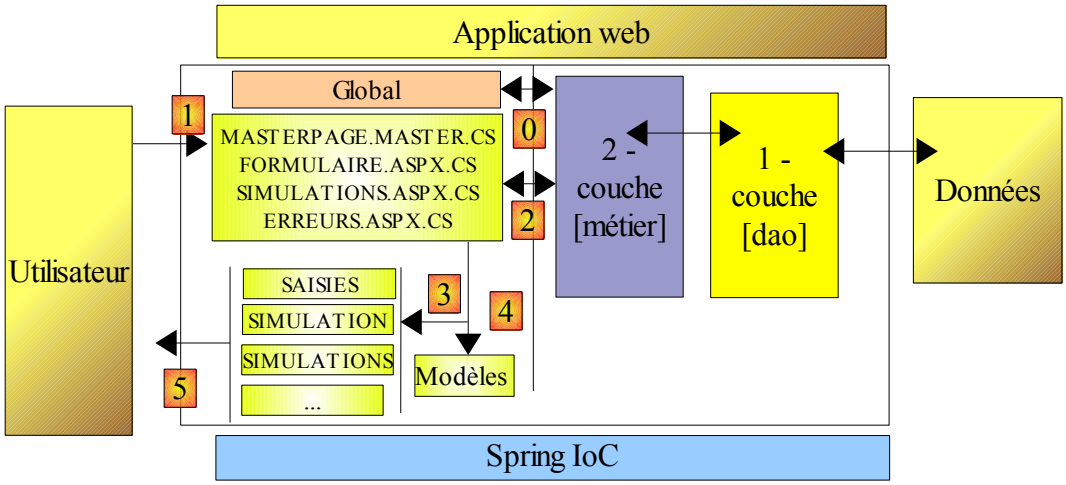
Le code source de cette page est le suivant :

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

11.5 Le code de contrôle des pages

11.5.1 Vue d'ensemble

Revenons à l'architecture de l'application :



- [Global] 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.cs] 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.

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

11.5.2.1 Squelette de la classe

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

```
1.  using System.Collections.Generic;
2.  using System.Web.UI.WebControls;
3.  using Pam.Web;
4.
5.      partial class MasterPage : System.Web.UI.MasterPage
6.      {
7.
8.          // le menu
9.          public LinkButton OptionFaireSimulation
10.         {
11.             get { return LinkButtonFaireSimulation; }
12.         }
13. ...
14.
15.         // fixer le menu
16.         public void SetMenu(bool boolFaireSimulation, bool boolEnregistrerSimulation, bool
    boolEffacerSimulation, bool boolFormulaireSimulation, bool boolVoirSimulations, bool
    boolTerminerSession)
17.         {
18. ...
19.         }
20.
21.         // gestion de l'option [Terminer la session]
22.         protected void LinkButtonTerminerSession_Click(object sender, System.EventArgs e)
23.         {
24. ...
25.         }
26.
27.         // init master page
28.         protected void Page_Init(object sender, System.EventArgs e)
29.         {
30.  ...
31.         }
32. }
```

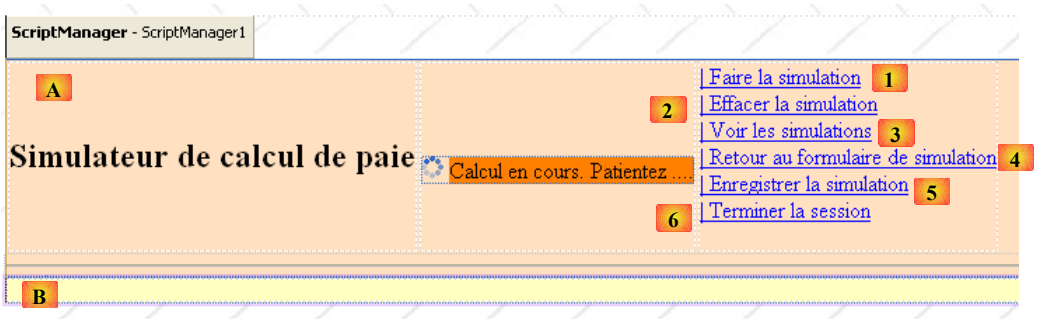
- ligne 5 : la classe s'appelle [MasterPage] et dérive de la classe système [System.Web.UI.MasterPage].
- lignes 9-14 : les 6 options du menu sont exposées comme propriétés publiques de la classe
- lignes 16-19 : la méthode publique *SetMenu* va permettre aux pages [Formulaire.aspx, Simulations.aspx, Erreurs.aspx] de fixer le menu de la page maître
- lignes 22-25 : la procédure qui va gérer le clic sur le lien [LinkButtonTerminerSession]
- lignes 28-31 : la procédure de gestion de l'événement *Init* de la page maître

11.5.2.2 Propriétés publiques de la classe

```
1.  using System.Collections.Generic;
2.  using System.Web.UI.WebControls;
3.  using Pam.Web;
4.
5.      partial class MasterPage : System.Web.UI.MasterPage
6.      {
7.
8.          // le menu
9.          public LinkButton OptionFaireSimulation
10.         {
11.             get { return LinkButtonFaireSimulation; }
12.         }
13.
14.         public LinkButton OptionEffacerSimulation
15.         {
16.             get { return LinkButtonEffacerSimulation; }
17.         }
18.
19.         public LinkButton OptionEnregistrerSimulation
20.         {
21.             get { return LinkButtonEnregistrerSimulation; }
22.         }
23.
24.         public LinkButton OptionVoirSimulations
25.         {
26.             get { return LinkButtonVoirSimulations; }
27.         }
28.
29.         public LinkButton OptionTerminerSession
```

```
30.         {
31.             get { return LinkButtonTerminerSession; }
32.         }
33.
34.         public LinkButton OptionFormulaireSimulation
35.         {
36.             get { return LinkButtonFormulaireSimulation; }
37.         }
38.
39. ...}
```

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 9 à 37 visent à les rendre accessibles aux classes externes, ici les classes des autres pages de l'application.

11.5.2.3 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.          // fixer le menu
2.          public void SetMenu(bool boolFaireSimulation, bool boolEnregistrerSimulation, bool
    boolEffacerSimulation, bool boolFormulaireSimulation, bool boolVoirSimulations, bool
    boolTerminerSession)
3.          {
4.              // on fixe les options de menu
5.              LinkButtonFaireSimulation.Visible = boolFaireSimulation;
6.              LinkButtonEnregistrerSimulation.Visible = boolEnregistrerSimulation;
7.              LinkButtonEffacerSimulation.Visible = boolEffacerSimulation;
8.              LinkButtonVoirSimulations.Visible = boolVoirSimulations;
9.              LinkButtonFormulaireSimulation.Visible = boolFormulaireSimulation;
10.             LinkButtonTerminerSession.Visible = boolTerminerSession;
11. }
```

11.5.2.4 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.      private void LinkButtonFaireSimulation_Click(object sender, System.EventArgs e)
2.      {
3.  ...
4.  }
```

On peut contourner le problème avec le code suivant dans [Formulaire.aspx] :

```
1.  partial class PageFormulaire : System.Web.UI.Page
2.  {
3.
4.      // chargement de la page
5.      protected void Page_Load(object sender, System.EventArgs e)
6.      {
7.          // gestionnaire d'évts
8.          Master.OptionFaireSimulation.Click += OptFaireSimulation_Click;
9.          Master.OptionEffacerSimulation.Click += OptEffacerSimulation_Click;
10.         Master.OptionVoirSimulations.Click += OptVoirSimulations_Click;
11.         Master.OptionEnregistrerSimulation.Click += OptEnregistrerSimulation_Click;
12. ...
13.         }
14.     }
15.
16.     // calcul de la paie
17.     private void OptFaireSimulation_Click(object sender, System.EventArgs e)
18.     {
19. ...
20.     }
21.
22.     // effacer la simulation
23.     private void OptEffacerSimulation_Click(object sender, System.EventArgs e)
24.     {
25. ...
26.     }
27.
28.     protected void OptVoirSimulations_Click(object sender, System.EventArgs e)
29.     {
30. ...
31.     }
32.
33.     protected void OptEnregistrerSimulation_Click(object sender, System.EventArgs e)
34.     {
35. ...
36.     }
37.
38. }
```

- lignes 8-11 : lorsque l'événement *Load* de la page [Formulaire.aspx] se produit, la classe [MasterPage] de la page maître a été instanciée. Les propriétés publiques *Optxx* sont accessibles et sont de type *LinkButton*, un composant qui supporte l'événement *Click*. Nous associons à ces événements *Click* les méthodes :
  - *OptFaireSimulation_Click* pour l'événement *Click* sur le lien *LinkButtonFaireSimulation*
  - *OptEffacerSimulation_Click* pour l'événement *Click* sur le lien *LinkButtonEffacerSimulation*
  - *OptVoirSimulations_Click* pour l'événement *Click* sur le lien *LinkButtonVoirSimulations*
  - *OptEnregistrerSimulation_Click* pour l'événement *Click* sur le lien *LinkButtonEnregistrerSimulation*

La gestion des événements *Click* sur les six liens du menu sera répartie de la façon suivante :

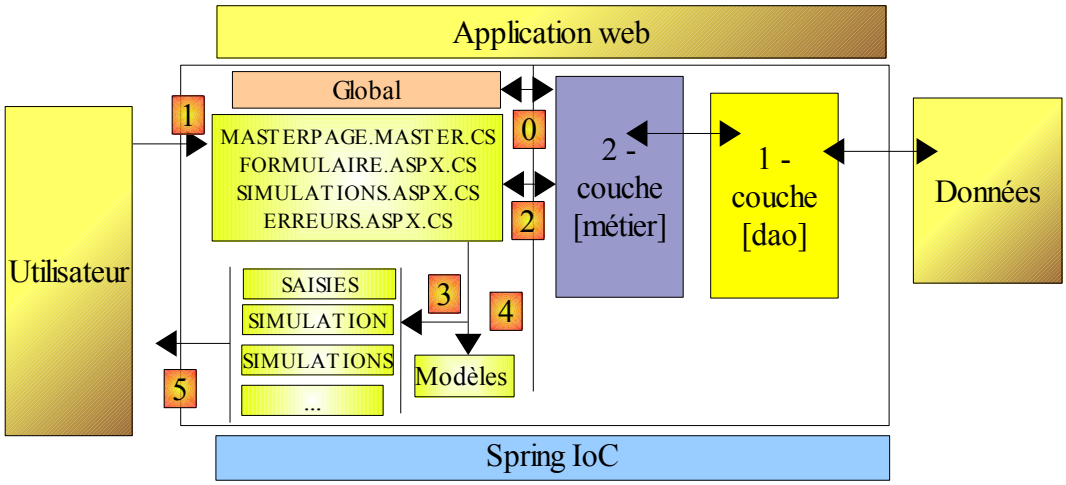
- la page [Formulaire.aspx] gèrera les liens [LinkButtonFaireSimulation, LinkButtonEnregistrerSimulation, LinkButtonEffacerSimulation, LinkButtonVoirSimulations]
- la page [Simulations.aspx] gèrera le lien [LinkButtonFormulaireSimulation]
- la page maître [MasterPage.master] gèrera le lien [LinkButtonTerminerSession]. Pour cet événement, elle n'a en effet pas besoin de connaître la page qu'elle encapsule.

11.5.2.5 L'événement Init de la page maître

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

```
1.  using System;
2.  ...
3.
4.  namespace Pam.Web
5.  {
6.      public class Global : HttpApplication
7.      {
8.
9.          // --- données statiques de l'application ---
10.         public static Employe[] Employes;
11.         public static string Msg = string.Empty;
12.         public static bool Erreur = false;
13.         public static IPamMetier PamMetier = null;
14.
15.         // démarrage de l'application
16.         public void Application_Start(object sender, EventArgs e)
17.         {
18. ...
19.         }
20.
21.         // démarrage de la session d'un utilisateur
22.         public void Session_Start(object sender, EventArgs e)
```

```
23.         {
24. ...
25.         }
26.
27.     }
28. }
```

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

- le booléen **Erreur** de la ligne 12 est mis à *vrai*
- la variable **Msg** de la ligne 11 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 [Global]. 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 void Page_Init(object sender, System.EventArgs e)
2.          {
3.              // gestionnaire d'évts
4.              LinkButtonTerminerSession.Click += LinkButtonTerminerSession_Click;
5.              // des erreurs d'initialisation ?
6.              if (Global.Erreur)
7.              {
8.                  // la page encapsulée est-elle la page d'erreurs ?
9.                  bool isPageErreurs =...;
10.                 // si c'est la page d'erreurs qui s'affiche, on laisse faire sinon on redirige le
    client vers la page d'erreurs
11.                 if (!isPageErreurs)
12.                     Response.Redirect("Erreurs.aspx");
13.                 return;
14.             }
15. }
```

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 [Global]. 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 :

```
this.FindControl("form1").FindControl("contenu").FindControl("ContentPlaceHolder1").FindControl("C");
```

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

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

Lorsque la page [Erreurs.aspx] est fusionnée avec la page maître M, le contenu de la balise <*asp:Content*> ci-dessus (lignes 5-16), 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.

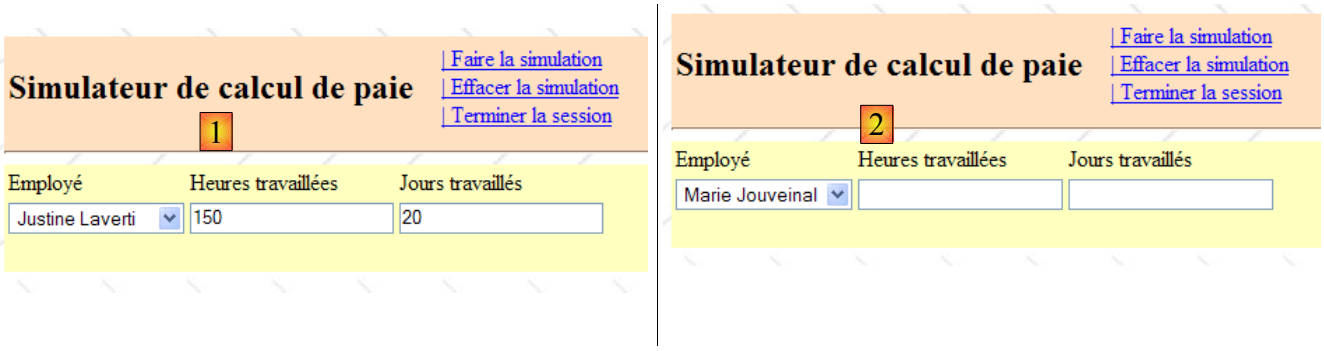
Ces explications suffisent pour comprendre le code de la procédure [Page_Init] de la page maître :

```
1.  protected void Page_Init(object sender, System.EventArgs e)
2.          {
3.              // gestionnaire d'évts
4.              LinkButtonTerminerSession.Click += LinkButtonTerminerSession_Click;
5.              // des erreurs d'initialisation ?
6.              if (Global.Erreur)
7.              {
8.                  // la page encapsulée est-elle la page d'erreurs ?
9.                  bool isPageErreurs =
    this.FindControl("form1").FindControl("contenu").FindControl("ContentPlaceHolder1").FindControl("
    rptErreurs") != null;
10.                 // si c'est la page d'erreurs qui s'affiche, on laisse faire sinon on redirige le
    client vers la page d'erreurs
11.                 if (!isPageErreurs)
12.                     Response.Redirect("Erreurs.aspx");
```

```
13.                 return;
14.             }
15.         }
```

- ligne 4 : on associe un gestionnaire d'événement à l'événement *Click* sur le lien *LinkButtonTerminerSession*. Ce gestionnaire est dans la classe *MasterPage*.
- ligne 6 : on vérifie si la classe [Global] a positionné son booléen *Erreur*
- ligne 9 : si oui, le booléen *IsPageErreurs* indique si la page encapsulée dans la page maître est la page [Erreurs.aspx]
- ligne 12 : 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.

11.5.2.6 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.          protected void LinkButtonTerminerSession_Click(object sender, System.EventArgs e)
2.          {
3.              // on abandonne la session
4.              Session.Abandon();
5.              // on affiche la vue [formulaire]
6.              Response.Redirect("Formulaire.aspx");
7.  }
```

- 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.

11.5.3 Code de contrôle de la page [Erreurs.aspx]

Le code de contrôle de la page [Erreurs.aspx] pourrait être le suivant :

```
1.  using System.Collections.Generic;
2.  using Pam.Web;
3.
4.  partial class PageErreurs : System.Web.UI.Page
5.  {
6.
7.      protected void Page_Load(object sender, System.EventArgs e)
8.      {
9.          // des erreurs d'initialisation ?
10.         if (Global.Erreur)
11.         {
12.             // on prépare le modèle de la page [erreurs]
13.             List<string> erreursInitialisation = new List<string>();
14.             erreursInitialisation.Add(Global.Msg);
15.             // on associe la liste d'erreurs à son composant
16.             rptErreurs.DataSource = erreursInitialisation;
17.             rptErreurs.DataBind();
```

```
18.         }
19.         // on fixe le menu
20.         Master.SetMenu(false, false, false, false, false, false);
21.     }
22.
23. }
```

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 10 : on teste si l'initialisation s'est terminée par une erreur
- lignes 13-14 : si oui, le message d'erreur (Global.Msg) est placé dans une liste [ErreursInitialisation]
- lignes 16-17 : 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.cs] et de [Erreurs.aspx.cs], 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.

11.5.4 Code de contrôle de la page [Formulaire.aspx]

11.5.4.1 Squelette de la classe

Le squelette du code de contrôle de la page [Formulaire.aspx] pourrait être le suivant :

```
1.  using Pam.Metier.Entites;
2.  ...
3.
4.  partial class PageFormulaire : System.Web.UI.Page
5.  {
6.
7.      // chargement de la page
8.      protected void Page_Load(object sender, System.EventArgs e)
9.      {
10.         // gestionnaire d'évts
11.         Master.OptionFaireSimulation.Click += OptFaireSimulation_Click;
12.         Master.OptionEffacerSimulation.Click += OptEffacerSimulation_Click;
13.         Master.OptionVoirSimulations.Click += OptVoirSimulations_Click;
14.         Master.OptionEnregistrerSimulation.Click += OptEnregistrerSimulation_Click;
15. ....
16.     }
17.
18.     // calcul de la paie
19.     private void OptFaireSimulation_Click(object sender, System.EventArgs e)
20.     {
21. ....
22.     }
23.
24.     // effacer la simulation
25.     private void OptEffacerSimulation_Click(object sender, System.EventArgs e)
26.     {
27. ...
28.     }
29.
30.     protected void OptVoirSimulations_Click(object sender, System.EventArgs e)
31.     {
32. ....
33.     }
34.
35.     protected void OptEnregistrerSimulation_Click(object sender, System.EventArgs e)
36.     {
37. ...
38.     }
39.
40. }
```

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

11.5.4.2 Evénement Load de la page

Le squelette du gestionnaire de l'événement *Load* de la page pourrait être le suivant :

```
1.      protected void Page_Load(object sender, System.EventArgs e)
2.      {
3.          // gestionnaire d'évts
4.          Master.OptionFaireSimulation.Click += OptFaireSimulation_Click;
5.          Master.OptionEffacerSimulation.Click += OptEffacerSimulation_Click;
6.          Master.OptionVoirSimulations.Click += OptVoirSimulations_Click;
7.          Master.OptionEnregistrerSimulation.Click += OptEnregistrerSimulation_Click;
8.          // affichage vue [saisies]
9.          ...
10.         // positionnement menu page maître
11.         ...
12.         // traitement requête GET
13.         if (!IsPostBack)
14.         {
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.         }
20. }
```

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

---

11.5.4.3 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 void OptFaireSimulation_Click(object sender, System.EventArgs e)
3.      {
4.          // effet Ajax
5.          Thread.Sleep(3000);
6.          // page valide ?
7.          Page.Validate();
8.          if (!Page.IsValid)
9.          {
10.             // affichage vue [saisie]
11. ...
12.         }
13.         // la page est valide - on récupère les saisies
14. ...
15.         // on calcule le salaire de l'employé
16.         FeuilleSalaire feuillesalaire;
17.         try
18.         {
19.             feuillesalaire = ...;
20.         }
21.         catch (PamException ex)
22.         {
23.             // on a rencontré un problème
24. ...
25.             return;
26.         }
27.         // on met le résultat dans la session
28.         Session["simulation"] = ...;
29.         // on met les saisies dans la session
30. ...
31.         // affichage
32. ...
33.         // affichage vues
34. ...
35.         // affichage menu MasterPage
36. ...
37.     }
38.
39.     // effacer la simulation
40.     private void OptEffacerSimulation_Click(object sender, System.EventArgs e)
41.     {
42.         // affichage panel [saisie]
43. ...
44.         // sélection 1er employé
45. ...
46.     }
47.
48.     protected void OptVoirSimulations_Click(object sender, System.EventArgs e)
49.     {
50.         // on met les saisies dans la session
51. ...
52.         // on affiche la vue [simulations]
53.         Response.Redirect("simulations.aspx");
54.     }
55.
56.     protected void OptEnregistrerSimulation_Click(object sender, System.EventArgs e)
57.     {
58.         // on enregistre la simulation courante dans la session de l'utilisateur
59. ...
60.         // on affiche la vue [simulations]
61.         Response.Redirect("simulations.aspx");
62. }
```

Question : compléter le code des procédures ci-dessus en vous aidant des commentaires et du code de la version précédente

11.5.5 Code de contrôle de la page [Simulations.aspx]

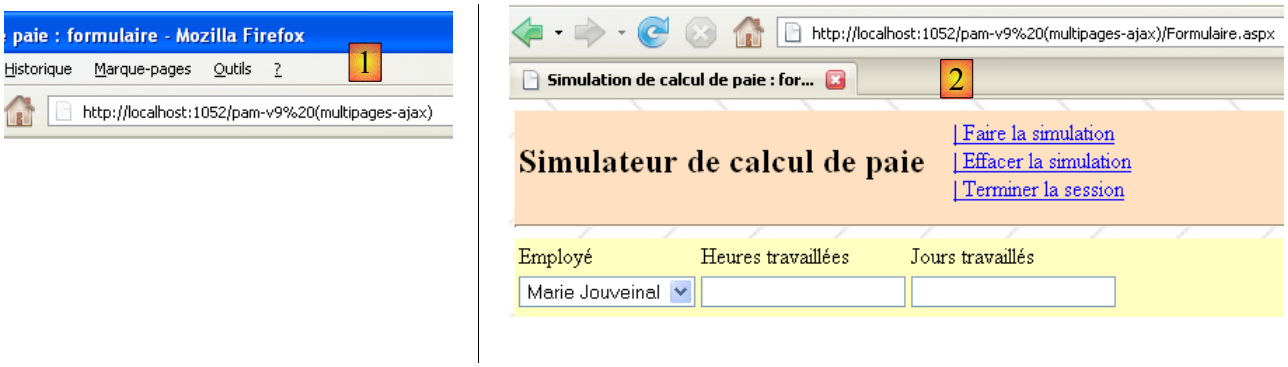
Le squelette du code de contrôle de la page [Simulations.aspx] pourrait être le suivant :

```
1.  using System.Collections.Generic;
2.  using Pam.Web;
3.  using System.Web.UI.WebControls;
4.
5.  partial class PageSimulations : System.Web.UI.Page
6.  {
7.
8.      // les simulations
9.      private List<Simulation> simulations;
10.
11.     // chargement de la page
12.     protected void Page_Load(object sender, System.EventArgs e)
13.     {
14.         // gestionnaire d'évts
15.         Master.OptionFormulaireSimulation.Click += OptFormulaireSimulation_Click;
16.         GridViewSimulations.RowDeleting += GridViewSimulations_RowDeleting;
17.         // on récupère les simulations dans la session
18.         simulations = ...;
19.         // y-a-t-il des simulations ?
20.         if (simulations.Count != 0)
21.         {
22.             // première vue visible
23.             ...
24.             // on remplit le gridview
25.             ...
26.         }
27.         else
28.         {
29.             // seconde vue
30.             ...
31.         }
32.         // on fixe le menu
33.         ...
34.     }
35.
36.     protected void GridViewSimulations_RowDeleting(object sender,
    System.Web.UI.WebControls.GridViewDeleteEventArgs e)
37.     {
38.         // on récupère les simulations dans la session
39.         List<Simulation> simulations = ...;
40.         // on supprime la simulation désignée (e.RowIndex représente le n° de la ligne supprimée
    dans le gridview)
41.         ..
42.         // reste-t-il des simulations ?
43.         if (simulations.Count != 0)
44.         {
45.             // on remplit le gridview
46.             ...
47.         }
48.         else
49.         {
50.             // vue [SimulationsVides]
51.             ...
52.         }
53.     }
54.
55.     protected void OptFormulaireSimulation_Click(object sender, System.EventArgs e)
56.     {
57.         // on affiche la vue [formulaire]
58.         Response.Redirect("formulaire.aspx");
59.     }
60.
61. }
```

Question : compléter le code des procédures ci-dessus en vous aidant des commentaires et du code de la version précédente

11.5.6 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 : System.Web.UI.Page
2.  {
3.
4.      protected void Page_Init(object sender, System.EventArgs e)
5.      {
6.          // on redirige vers le formulaire de saisie
7.          Response.Redirect("Formulaire.aspx");
8.      }
9.  }
```

La page de présentation [Default.aspx] ne contient elle que la directive qui la relie à [Default.aspx.cs] :

```
<%@ Page Language="C#" AutoEventWireup="true"
  CodeFile="Default.aspx.cs" Inherits="_Default" Title="Untitled Page" %>
```

Table des matières